Developer Guide

AWS SDK for .NET (V4)

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS SDK for .NET (V4): Developer Guide

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

..... x

What is the AWS SDK for .NET 1

About this version 1

Maintenance and support for SDK major versions 2

Common use cases 2

Additional topics in this section 2

Related AWS tools 3

Tools for Windows PowerShell and Tools for PowerShell Core 3

Toolkit for VS Code 3

Toolkit for Visual Studio 3

Toolkit for Azure DevOps 4

Supported platforms 4

.NET Core 4

.NET Standard 2.0 4

.NET Framework 4.5 4

.NET Framework 3.5 4

Portable Class Library and Xamarin 5

Unity support 5

SDKs and Tools Reference 5

Revision history 5

What's new 5

Additional resources 7

Getting started 10

Installing and configuring your toolchain 10

Cross-platform development 10

Windows with Visual Studio and .NET Core 11

Next step 11

Authenticating with AWS 11

Login using console credentials 12

Enable and configure IAM Identity Center 13

Configure the SDK to use IAM Identity Center. 13

Start an AWS access portal session 14

Additional information 15

Creating a simple application 15

Simple cross-platform app ... 16
Simple Windows-based app ... 22
Next steps ... 28
Configuring ... 29
Start a new project ... 29
Install AWSSDK packages with NuGet ... 31
Using NuGet from the Command prompt or terminal ... 31
Using NuGet from Visual Studio Solution Explorer ... 32
Using NuGet from the Package Manager Console ... 33
Install AWSSDK assemblies without NuGet ... 33
AWS Region ... 35
Create a service client with a particular Region ... 35
Specify a Region for all service clients ... 36
Region resolution ... 37
Special information about the China (Beijing) Region ... 38
Special information about new AWS services ... 38
Credential and profile resolution ... 38
Profile resolution ... 39
Using federated user account credentials ... 40
Specifying roles or temporary credentials ... 40
Using proxy credentials ... 41
Retries and timeouts ... 41
Retries ... 42
Timeouts ... 43
Observability ... 46
Additional resources ... 46
Configure a `TelemetryProvider` ... 46
Metrics ... 48
Telemetry providers ... 50
Users and roles ... 51
Users and permission sets ... 51
Service roles ... 52
Advanced configuration ... 53
AWSSDK.Extensions.NETCore.Setup and IConfiguration ... 53
Configuring Other Application Parameters ... 58
Configuration Files Reference for AWS SDK for .NET ... 66

Using legacy credentials ... 78
Important warnings and guidance for credentials ... 79
Using the shared AWS credentials file ... 80
Using the SDK Store (Windows only) ... 83
Using the SDK ... 87
Asynchronous programming ... 87
Pagination ... 89
Where do I find paginators? ... 89
What do paginators give me? ... 89
Synchronous vs. asynchronous pagination ... 90
Example ... 90
Additional considerations for paginators ... 94
Support for HTTP 2 ... 94
Additional considerations ... 97
Additional tools ... 97
AWS Deploy Tool ... 97
AWS Message Processing Framework for .NET ... 97
Integrations with .NET Aspire ... 97
Advanced auth ... 98
Single sign-on ... 98
Prerequisites ... 99
Setting up an SSO profile ... 99
Generating and using SSO tokens ... 101
Additional resources ... 105
Tutorials ... 106
Tutorial: .NET application only ... 106
Tutorial: AWS CLI and .NET application ... 115
Deploy to AWS ... 124
Deploy from the .NET CLI ... 124
Deploy from the IDE toolkits ... 124
Use cases ... 125
ASP.NET Core apps ... 125
.NET Console apps ... 126
Blazor WebAssembly apps ... 127
AWS Lambda projects ... 127
Prerequisites ... 128

Available Lambda commands ... 128
Steps to deploy ... 129
Calling AWS services ... 130
Guided code examples ... 130
CloudFormation ... 131
Amazon Cognito ... 135
DynamoDB ... 143
Amazon EC2 ... 173
IAM ... 234
Amazon S3 ... 254
Amazon SNS ... 264
Amazon SQS ... 268
AWS Lambda ... 301
APIs ... 301
Prerequisites ... 301
Additional information ... 301
Topics ... 301
Lambda Annotations ... 302
High-level libraries and frameworks ... 303
Message Processing Framework ... 304
Integrating AWS with .NET Aspire ... 324
Other services and configuration ... 325
Code examples ... 327
Aurora ... 328
Basics ... 330
Actions ... 354
Auto Scaling ... 369
Basics ... 330
Actions ... 354
Amazon Bedrock ... 404
Actions ... 354
Amazon Bedrock Runtime ... 408
Anthropic Claude ... 409
Cohere Command ... 414
Meta Llama ... 418
Mistral AI ... 422

CloudFormation ... 425
CloudWatch ... 429
Basics ... 330
Actions ... 354
CloudWatch Logs ... 484
Actions ... 354
Scenarios ... 488
Amazon Cognito Identity Provider ... 510
Actions ... 354
AWS Control Tower ... 511
Basics ... 330
Actions ... 354
DynamoDB ... 550
Basics ... 330
Actions ... 354
Amazon EC2 ... 590
Amazon ECS ... 592
AWS IoT ... 594
Basics ... 330
Actions ... 354
AWS IoT data ... 652
Actions ... 354
Amazon Redshift ... 655
Basics ... 330
Actions ... 354
Amazon S3 ... 685
Actions ... 354
Scenarios ... 488
Migrating your project ... 696
Migrating to version 4 ... 696
.NET Framework ... 696
Value types ... 696
Collections ... 697
AWS Security Token Service (STS) ... 697
Changes related to `ClientConfig` ... 698
The AWSSDK.Extensions.NETCore.Setup NuGet package ... 698

CookieSigner and UrlSigner ... 699
DateTime versus UTC DateTime ... 700
DateTime parsing ... 700
ConvertFromUnixEpochSeconds and ConvertFromUnixEpochMilliseconds ... 701
Logging ... 701
Support for HTTP 2 ... 702
Single sign-on ... 702
Changes specific to DynamoDB ... 702
Changes specific to EC2 ... 706
Changes specific to S3 ... 707
Programming elements that were removed ... 710
Migrating from .NET Standard 1.3 ... 713
Security ... 715
Data protection ... 715
Identity and Access Management ... 716
Audience ... 717
Authenticating with identities ... 717
Managing access using policies ... 719
How AWS services work with IAM ... 720
Troubleshooting AWS identity and access ... 721
Compliance Validation ... 722
Resilience ... 723
Infrastructure Security ... 723
Enforcing a minimum TLS version ... 724
.NET Core ... 724
.NET Framework ... 725
AWS Tools for PowerShell ... 726
Xamarin ... 727
Unity ... 728
Browser (for Blazor WebAssembly) ... 728
S3 Encryption Client Migration (V1 to V2) ... 728
Migration Overview ... 729
Update Existing Clients to V1-transitional Clients to Read New Formats ... 729
Migrate V1-transitional Clients to V2 Clients to Write New Formats ... 730
Update V2 Clients to No Longer Read V1 Formats ... 733
S3 Encryption Client Migration (V2 to V4) ... 734

Understanding V4 Concepts 734
Update Existing Clients to Read V4 Formats 736
Migrate to V4 Clients 737
Additional Configuration Examples 741
Special considerations 743
Obtaining AWSSDK assemblies 743
Download and extract ZIP files 743
Accessing credentials and profiles in an application 744
Examples for class CredentialProfileStoreChain 744
Examples for classes SharedCredentialsFile and AWSCredentialsFactory 746
Unity support 747
Xamarin support 748
API reference 749
About API reference versions 749
Document history 751

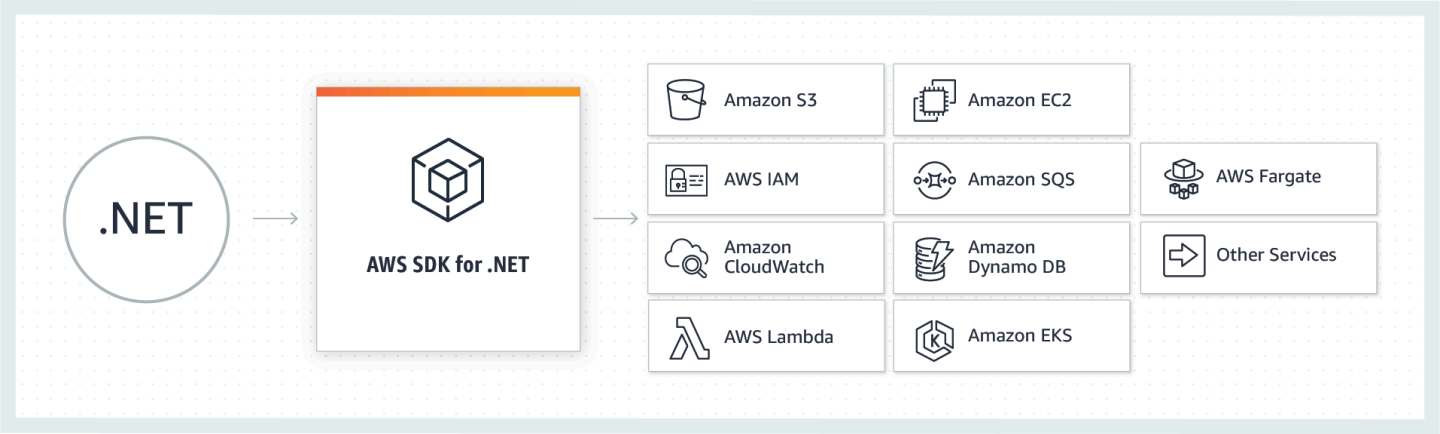
Version 4 (V4) of the AWS SDK for .NET has been released!

For information about breaking changes and migrating your applications, see the migration topic.

Click here for details

What is the AWS SDK for .NET

The AWS SDK for .NET makes it easier to build .NET applications that tap into cost-effective, scalable, and reliable AWS services such as Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2). The SDK simplifies the use of AWS services by providing a set of libraries that are consistent and familiar for .NET developers..



(**OK, got it!** I'm ready to set up and take a quick tour.)

About this version

Current and past documentation

This documentation is for version 4.0 and later (V4) of the AWS SDK for .NET. It's mostly centered around .NET Core and ASP.NET Core, but also contains information about .NET Framework and ASP.NET 4.*x*. In addition to Windows and Visual Studio, it gives equal consideration to cross-platform development.
For information about migrating from earlier versions of the SDK, see Migrating your project.
For information about the previous version of the SDK, V3, which will be deprecated in the future, see the AWS SDK for .NET (V3) Developer Guide.
To find deprecated content for earlier versions of the SDK, see the following item(s):

- AWS SDK for .NET (version 2, deprecated) Developer Guide
- Deprecated API references for the AWS SDK for .NET

Maintenance and support for SDK major versions

For information about maintenance and support for SDK major versions and their underlying dependencies, see the following in the AWS SDKs and Tools Reference Guide:

- AWS SDKs and tools maintenance policy
- AWS SDKs and tools version support matrix

Common use cases

The AWS SDK for .NET helps you realize several compelling use cases, including the following:

- Manage users and roles with AWS Identity and Access Management (IAM).
- Access Amazon Simple Storage Service (Amazon S3) to create buckets and store objects.
- Manage Amazon Simple Notification Service (Amazon SNS) HTTP subscriptions to topics.
- Use the S3 transfer utility to transfer files to Amazon S3 from your Xamarin applications.
- Use Amazon Simple Queue Service (Amazon SQS) to process messages and workflows between components in a system.
- Perform efficient Amazon S3 transfers by sending SQL statements to Amazon S3 Select.
- Create and launch Amazon EC2 instances, and configure and request Amazon EC2 spot instances.

Additional topics in this section

- AWS tools related to the AWS SDK for .NET
- Platforms supported by the AWS SDK for .NET
- AWS SDKs and Tools Reference Guide
- Revision history
- What's new in the AWS SDK for .NET
- Additional resources

AWS tools related to the AWS SDK for .NET

Tools for Windows PowerShell and Tools for PowerShell Core

The AWS Tools for Windows PowerShell and AWS Tools for PowerShell Core are PowerShell modules that are built on the functionality exposed by the AWS SDK for .NET. The AWS PowerShell tools enable you to script operations on your AWS resources from the PowerShell prompt. Although the cmdlets are implemented using the service clients and methods from the SDK, the cmdlets provide an idiomatic PowerShell experience for specifying parameters and handling results.

To get started, see AWS Tools for Windows PowerShell.

Toolkit for VS Code

The AWS Toolkit for Visual Studio Code is a plugin for the Visual Studio Code (VS Code) editor. The toolkit makes it easier for you to develop, debug, and deploy applications that use AWS.

With the toolkit, you can do such things as the following:

- Create serverless applications that contain AWS Lambda functions, and then deploy the applications to an AWS CloudFormation stack.
- Work with Amazon EventBridge schemas.
- Use IntelliSense when working with Amazon ECS task-definition files.
- Visualize an AWS Cloud Development Kit (AWS CDK) application.

Toolkit for Visual Studio

The AWS Toolkit for Visual Studio is a plugin for the Visual Studio IDE that makes it easier for you to develop, debug, and deploy .NET applications that use Amazon Web Services. The Toolkit for Visual Studio provides Visual Studio templates for services such as Lambda and deployment wizards for web applications and serverless applications. You can use the AWS Explorer to manage Amazon EC2 instances, work with Amazon DynamoDB tables, publish messages to Amazon Simple Notification Service (Amazon SNS) queues, and more, all within Visual Studio.

To get started, see Setting up the AWS Toolkit for Visual Studio.

Toolkit for Azure DevOps

The AWS Toolkit for Microsoft Azure DevOps adds tasks to easily enable build and release pipelines in Azure DevOps and Azure DevOps Server to work with AWS services. You can work with Amazon S3, AWS Elastic Beanstalk, AWS CodeDeploy, Lambda, AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS), and Amazon SNS. You can also run commands using the Windows PowerShell module and the AWS Command Line Interface (AWS CLI).

To get started with the AWS Toolkit for Azure DevOps, see the AWS Toolkit for Microsoft Azure DevOps User Guide.

Platforms supported by the AWS SDK for .NET

The AWS SDK for .NET provides distinct groups of assemblies for developers to target different platforms. However, not all SDK functionality is the same on each of these platforms. This topic describes the differences in support for each platform.

.NET Core

The AWS SDK for .NET supports applications written for .NET Core (.NET Core 3.1, .NET 5, .NET 6, and so on). AWS service clients only support asynchronous calling patterns in .NET core. This also affects many of the high level abstractions built on top of service clients, like the Amazon S3 `TransferUtility`, which will only support asynchronous calls in the .NET Core environment.

.NET Standard 2.0

Non-Framework variations of the AWS SDK for .NET comply with .NET Standard 2.0. The AWS SDK for .NET provides only asynchronous methods for applications written against .NET Standard.

.NET Framework 4.5

This version of the AWS SDK for .NET is compiled against .NET Framework 4.7.2 and runs in the .NET 4.0 runtime. AWS service clients support synchronous and asynchronous calling patterns and use the async and await keywords introduced in C# 5.0.

.NET Framework 3.5

Version 4 of the AWS SDK for .NET doesn't support .NET Framework 3.5.

Portable Class Library and Xamarin

The AWS SDK for .NET also contains a Portable Class Library implementation. The Portable Class Library implementation can target multiple platforms, including Universal Windows Platform (UWP) and Xamarin on iOS and Android. See the Mobile SDK for .NET and Xamarin for more details. AWS service clients only support asynchronous calling patterns.

Unity support

For information about Unity support, see Special considerations for Unity support.

AWS SDKs and Tools Reference Guide

The AWS SDKs and Tools Reference Guide contains information that's relevant and important for many of the AWS SDKs and toolkits and the AWS CLI. The following are some examples of the information that the reference contains:

- Information about the shared AWS `config` and `credentials` files and their location.
- Setting up AWS accounts, users, and roles
- Configuration and authentication settings reference
- AWS Common Runtime (CRT) libraries
- AWS SDKs and tools maintenance policy
- AWS SDKs and tools version support matrix

Revision history

To find out what has changed in various releases, see the following:

- SDK change logs
- What's new in the AWS SDK for .NET
- Document history

What's new in the AWS SDK for .NET

For high-level information about new developments related to the AWS SDK for .NET, see the product page at https://aws.amazon.com/sdk-for-net/ and the SDK change logs.

The following is what's new in the AWS SDK for .NET.

October 17, 2025: Version 2.0 of the AWS Deploy Tool

Version 2.0 of the AWS Deploy Tool for .NET CLI has been released. For information about the Deploy Tool, see Deploy applications to AWS. For information about version 2.0 of the Deploy Tool, see the blog post What's New in the AWS Deploy Tool for .NET.

August 21, 2025: Upcoming end-of-support for version 3 of the AWS SDK for .NET

The end-of-support for version 3 (V3) of the AWS SDK for .NET has been announced. See the migration guide to migrate your V3 applications and avoid disruptions. For more information, see the blog post Announcing the end-of-support for the AWS SDK for .NET v3.

July 2, 2025: General availability of the Distributed Cache Provider

The AWS .NET Distributed Cache Provider for Amazon DynamoDB is generally available! This provider implements the ASP.NET Core IDistributedCache interface, letting you integrate the fully managed and durable infrastructure of DynamoDB into your caching layer with minimal code changes. The Distributed Cache Provider is available through the AWS.AspNetCore.DistributedCacheProvider package on NuGet. For additional information, see the blog post AWS .NET Distributed Cache Provider for Amazon DynamoDB now Generally Available.

> **Note**
>
> The released version of the Distributed Cache Provider is supported only on V4 of the AWS SDK for .NET. If you tried the preview version of the Distributed Cache Provider, you might need to update package dependencies in your application.

May 5, 2025: General availability of the AWS Message Processing Framework for .NET

The AWS Message Processing Framework for .NET is generally available! The framework is an AWS-native framework that simplifies the development of .NET message-processing applications that use AWS services such as Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS), and Amazon EventBridge.

April 28, 2025: Version 4 of the AWS SDK for .NET

Version 4 of the AWS SDK for .NET is generally available! For information about migrating your applications to V4, see Migrating to version 4. Also see the blog post General Availability of AWS SDK for .NET V4.0 that announces general availability.

February 15, 2025: Integrations with .NET Aspire

Integrations with .NET Aspire to improve the inner dev loop have been released. For information, see Integrating AWS with .NET Aspire in the AWS SDK for .NET.

February 10, 2025: GA release for observability

Observability is the extent to which a system's current state can be inferred from the data it emits. Observability has been added to the AWS SDK for .NET, including an implementation of a telemetry provider. For more information, see Observability in this guide and the blog post Announcing the general availability of AWS .NET OpenTelemetry libraries.

January 15, 2025: New default behavior for integrity protection

Beginning with version 3.7.412.0 of the AWS SDK for .NET, the SDK provides default integrity protections by automatically calculating a CRC32 checksum for uploads. For more information, see the announcement on GitHub at https://github.com/aws/aws-sdk-net/issues/3610. The SDK also provides global settings for data integrity protections that you can set externally, which you can read about in Data Integrity Protections in the AWS SDKs and Tools Reference Guide.

November 15, 2024: Blog post for preview 4 release for version 4

Preview 4 of the AWS SDK for .NET Version 4 was released on November 15, 2024. For more information about this preview, see the blog post Preview 4 of AWS SDK for .NET V4.

August 16, 2024: Blog post for preview 1 release for version 4

The AWS SDK for .NET Version 4 was released as a first preview on August 16, 2024. For more information about this preview, see the blog post Preview 1 of AWS SDK for .NET V4.

Additional resources

Supported services

The AWS SDK for .NET supports most AWS infrastructure products, and more services are added frequently. For a list of the AWS services supported by the SDK, see the SDK README file.

Home page for the AWS SDK for .NET

For more information about the AWS SDK for .NET, see the home page for the SDK at https://aws.amazon.com/sdk-for-net/.

SDK reference documentation

The SDK reference documentation gives you the ability to browse and search across all code included with the SDK. It provides thorough documentation and usage examples. For more information, see the AWS SDK for .NET API Reference.

AWS re:Post (formerly AWS forums)

Visit AWS re:Post, specifically the topic for the AWS SDK for .NET, to ask questions or provide feedback about AWS. Each documentation page has a **Try AWS re:Post** link at the bottom of the page that takes you to the associated re:Post topic. AWS engineers monitor the topics and respond to questions, feedback, and issues.

If you're signed in to re:Post, you can also follow a topic. To follow the topic for the AWS SDK for .NET, go to the **All Topics** page, find ".NET on AWS", and select the **Follow** button.

Toolkits

- AWS Toolkit for Visual Studio: If you use the Microsoft Visual Studio IDE, you should check out the AWS Toolkit for Visual Studio User Guide.
- AWS Toolkit for Visual Studio Code: If you use the Microsoft Visual Studio IDE, you should check out the AWS Toolkit for Visual Studio Code User Guide.

Helpful libraries, extensions and tools

Visit the aws/dotnet and aws/aws-sdk-net repositories on the GitHub website for links to libraries, tools, and resources you can use to help build .NET applications and services on AWS.

The following are some examples:

- AWS .NET Configuration Extension for Systems Manager
- AWS Extensions .NET Core Setup
- AWS Logging .NET
- Amazon Cognito Authentication Extension Library

- AWS X-Ray SDK for .NET

Other resources

The following are other resources that might prove useful:

- Developer net
- .NET Development Environment on the AWS Cloud - Quick Start Reference Deployment
- Hello, Cloud! blog
- AWS Whitepaper: Developing and Deploying .NET Applications on AWS
- AWS Microservice Extractor for .NET
- Porting Assistant for .NET
- AWS SDKs and Tools Reference Guide

Getting started with the AWS SDK for .NET

Learn how to install, set up, and use the AWS SDK for .NET. You must install your toolchain and set up a number of essential things that your application needs to access AWS services. These include:

- An appropriate user account or role
- Authentication information for that user account or to assume that role

For information about how to configure projects and other critical pieces, see Configuring the AWS SDK for .NET.

Topics

- Installing and configuring your toolchain for the AWS SDK for .NET
- Authenticating the AWS SDK for .NET with AWS
- Creating a simple application using the AWS SDK for .NET

Installing and configuring your toolchain for the AWS SDK for .NET

To use the AWS SDK for .NET, you must have certain development tools installed.

Cross-platform development

The following are required for cross-platform .NET development on Windows, Linux, or macOS:

- Microsoft .NET Core SDK, version 2.1, 3.1, or later, which includes the .NET command line interface (CLI) (**`dotnet`**) and the .NET Core runtime.

- A code editor or integrated development environment (IDE) that is appropriate for your operating system and requirements. This is typically one that provides some support for .NET Core.

  Examples include Microsoft Visual Studio Code (VS Code), JetBrains Rider, and Microsoft Visual Studio.

- (Optional) An AWS toolkit if one is available for the editor you chose and your operating system.

  Examples include the AWS Toolkit for Visual Studio Code, AWS Toolkit for JetBrains, and AWS Toolkit for Visual Studio.

Windows with Visual Studio and .NET Core

The following are required for development on Windows with Visual Studio and .NET Core:

- Microsoft Visual Studio

- Microsoft .NET Core 2.1, 3.1 or later

  This is typically included by default when installing a recent version of Visual Studio.

- (Optional) The AWS Toolkit for Visual Studio, which is a plugin that provides a user interface for managing your AWS resources and local profiles from Visual Studio. To install the toolkit, see Setting up the AWS Toolkit for Visual Studio.

  For more information, see the AWS Toolkit for Visual Studio User Guide.

Next step

Authenticating the AWS SDK for .NET with AWS

Authenticating the AWS SDK for .NET with AWS

You must establish how your code authenticates with AWS when developing with AWS services. There are different ways in which you can configure programmatic access to AWS resources, depending on the environment and the AWS access available to you.

To see various methods of authentication for the SDK, see Authentication and access in the *AWS SDKs and Tools Reference Guide*.

This topic assumes that a new user is developing locally, has not been given a method of authentication by their employer, and will be using AWS IAM Identity Center to obtain temporary credentials. If your environment doesn't fall under these assumptions, some of the information in

this topic might not apply to you, or some of the information might have already been given to you.

Configuring this environment requires several steps, which are summarized as follows:

1. Login using console credentials
2. Enable and configure IAM Identity Center
3. Configure the SDK to use IAM Identity Center.
4. Start an AWS access portal session

Login using console credentials

You can use your existing AWS Management Console sign-in credentials for programmatic access to AWS services. After a browser-based authentication flow, AWS generates temporary credentials that work across local development tools like the AWS CLI, AWS Tools for PowerShell and the AWS SDK for .NET v4.

With this process, you can authenticate using root credentials created during initial account set up, an IAM user, or a federated identity from your identity provider, and AWS CLI or AWS Tools for PowerShell automatically manages the temporary credentials for you. This approach enhances security by eliminating the need to store long-term credentials locally. When you run the `Invoke-AWSLogin` cmdlet from AWS Tools for PowerShell or the `aws login` command from AWS CLI, you can select from your active console sessions, or sign in through the browser-based authentication flow and this will automatically generate temporary credentials. The temporary credential token will expire in 15 minutes, but the AWS SDK for .NET v4 automatically refresh the token when needed during your requests.

> **Important**
>
> If you're using the console credentials for authentication, your application must reference the following NuGet packages so that AWS Signin resolution can work:
>
> - **AWSSDK.Signin**
>
> Failure to reference these packages will result in a runtime exception.

- AWS Tools for PowerShell login creds Developer Guide.

- Login for AWS local development using console credentials using AWS CLI.

The `config` file would look like this once you've login following the steps given above for AWS Tools for PowerShell or AWS CLI.

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

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

Enable and configure IAM Identity Center

To use IAM Identity Center, it must first be enabled and configured. To see details about how to do this for the SDK, look at **Step 1** in the topic for IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*. Specifically, follow any necessary instructions under **I do not have established access through IAM Identity Center**.

Configure the SDK to use IAM Identity Center.

Information about how to configure the SDK to use IAM Identity Center is in **Step 2** in the topic for IAM Identity Center authentication in the *AWS SDKs and Tools Reference Guide*. After you complete this configuration, your system should contain the following elements:

- The AWS CLI, which you use to start an AWS access portal session before you run your application.
- The shared AWS `config` file that contains a `[default]` profile with a set of configuration values that can be referenced from the SDK. To find the location of this file, see Location of the shared files in the *AWS SDKs and Tools Reference Guide*. The AWS SDK for .NET uses the profile's SSO token provider to acquire credentials before sending requests to AWS. The `sso_role_name` value, which is an IAM role connected to an IAM Identity Center permission set, should allow access to the AWS services used in your application.

  The following sample `config` file shows a default profile set up with SSO token provider. The profile's `sso_session` setting refers to the named `sso-session` section. The `sso-session` section contains settings to initiate an AWS access portal session.

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

Important

If you're using AWS IAM Identity Center for authentication, your application must reference the following NuGet packages so that SSO resolution can work:

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

Failure to reference these packages will result in a *runtime* exception.

Start an AWS access portal session

Before running an application that accesses AWS services, you need an active AWS access portal session for the SDK to use IAM Identity Center authentication to resolve credentials. Depending on your configured session lengths, your access will eventually expire and the SDK will encounter an authentication error. To sign in to the AWS access portal, run the following command in the AWS CLI.

```
aws sso login
```

Since you have a default profile setup, you do not need to call the command with a `--profile` option. If your SSO token provider configuration is using a named profile, the command is `aws sso login --profile named-profile`.

To test if you already have an active session, run the following AWS CLI command.

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

The response to this command should report the IAM Identity Center account and permission set configured in the shared `config` file.

> **Note**
>
> If you already have an active AWS access portal session and run `aws sso login`, you will not be required to provide credentials.
> The sign-in process might prompt you to allow the AWS CLI access to your data. Because the AWS CLI is built on top of the SDK for Python, permission messages may contain variations of the `botocore` name.

Additional information

- For additional information about using IAM Identity Center and SSO in a development environment, see Single sign-on in the Advanced auth section. This information includes alternative and more advanced methods, as well as tutorials that show you how to use these methods.
- For more options on authentication for the SDK, such as the use of profiles and environment variables, see the configuration chapter in the *AWS SDKs and Tools Reference Guide*.
- To learn more about best practices, see Security best practices in IAM in the *IAM User Guide*.
- To create short-term AWS credentials, see Temporary Security Credentials in the *IAM User Guide*.
- To learn about other credential providers, see Standardized credential providers in the *AWS SDKs and Tools Reference Guide*.

Creating a simple application using the AWS SDK for .NET

This section provides basic tutorials for developers who are new to the AWS SDK for .NET.

> **Note**
>
> Before you use these tutorials, you must have first installed your toolchain and configured SDK authentication.

For information about developing software for specific AWS services along with code examples, see Calling AWS services. For additional code examples, see SDK for .NET (v4) code examples.

Topics

- Simple cross-platform application using the AWS SDK for .NET
- Simple Windows-based application using the AWS SDK for .NET
- Next steps

Simple cross-platform application using the AWS SDK for .NET

This tutorial uses the AWS SDK for .NET and .NET Core for cross-platform development. The tutorial shows you how to use the SDK to list the Amazon S3 buckets that you own and, optionally, create a bucket.

You'll perform this tutorial using cross-platform tools such as the .NET command line interface (CLI). For other ways to configure your development environment, see Installing and configuring your toolchain for the AWS SDK for .NET.

Required for cross-platform .NET development on Windows, Linux, or macOS:

- Microsoft .NET Core SDK, version 2.1, 3.1, or later, which includes the .NET command line interface (CLI) (**`dotnet`**) and the .NET Core runtime.

- A code editor or integrated development environment (IDE) that is appropriate for your operating system and requirements. This is typically one that provides some support for .NET Core.

  Examples include Microsoft Visual Studio Code (VS Code), JetBrains Rider, and Microsoft Visual Studio.

> **Note**
>
> Before you use these tutorials, you must have first installed your toolchain and configured SDK authentication.

Steps

- Create the project
- Create the code
- Run the application
- Cleanup

Create the project

1. Open the command prompt or terminal. Find or create an operating system folder under which you can create a .NET project.
2. In that folder, run the following command to create the .NET project.

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

3. Go to the newly created `S3CreateAndList` folder and run the following commands:

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

The preceding commands install the NuGet packages from the NuGet package manager. Because we know exactly what NuGet packages we need for this tutorial, we can perform this step now. It's also common that the required packages become known during development. When this happens, a similar command can be run at that time.

Create the code

1. In the `S3CreateAndList` folder, find and open `Program.cs` in your code editor.

2. Replace the contents with the following code and save the file.

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

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

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

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

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

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

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

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

        //
        // Method to parse the command line.
        private static Boolean GetBucketName(string[] args, out String bucketName)
        {
            Boolean retval = false;
            bucketName = String.Empty;
            if (args.Length == 0)
            {
                Console.WriteLine("\nNo arguments specified. Will simply list your
Amazon S3 buckets." +
```

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

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

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

```
            return response.Arn;
        }
    }
}
```

Run the application

1. Run the following command.

```
dotnet run
```

2. Examine the output to see the number of Amazon S3 buckets that you own, if any, and their names.
3. Choose a name for a new Amazon S3 bucket. Use "dotnet-quicktour-s3-1-cross-" as a base and add something unique to it, such as a GUID or your name. Be sure to follow the rules for bucket names, as described in Rules for bucket naming in the Amazon S3 User Guide.
4. Run the following command, replacing *amzn-s3-demo-bucket* with the name of the bucket that you chose.

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

5. Examine the output to see the new bucket that was created.

Cleanup

While performing this tutorial, you created some resources that you can choose to clean up at this time.

- If you don't want to keep the bucket that the application created in an earlier step, delete it by using the Amazon S3 console at https://console.aws.amazon.com/s3/.
- If you don't want to keep your .NET project, remove the `S3CreateAndList` folder from your development environment.

Where to go next

Go back to the quick-tour menu or go straight to the end of this quick tour.

Simple Windows-based application using the AWS SDK for .NET

This tutorial uses the AWS SDK for .NET on Windows with Visual Studio and .NET Core. The tutorial shows you how to use the SDK to list the Amazon S3 buckets that you own and optionally create a bucket.

You'll perform this tutorial on Windows using Visual Studio and .NET Core. For other ways to configure your development environment, see Installing and configuring your toolchain for the AWS SDK for .NET.

Required for development on Windows with Visual Studio and .NET Core:

- Microsoft Visual Studio
- Microsoft .NET Core 2.1, 3.1 or later

  This is typically included by default when installing a recent version of Visual Studio.

> **Note**
>
> Before you use these tutorials, you must have first installed your toolchain and configured SDK authentication.

Steps

- Create the project
- Create the code
- Run the application
- Cleanup

Create the project

1. Open Visual Studio and create a new project that uses the C# version of the **Console App** template; that is, with description: "...for creating a command-line application that can run on .NET...". Name the project `S3CreateAndList`.

> **Note**
>
> Don't choose the .NET Framework version of the console app template, or, if you do, be sure to use .NET Framework 4.7.2 or later.

2. With the newly created project loaded, choose **Tools**, **NuGet Package Manager**, **Manage NuGet Packages for Solution**.
3. Browse for the following NuGet packages and install them into the project: `AWSSDK.S3`, `AWSSDK.SecurityToken`, `AWSSDK.SSO`, and `AWSSDK.SSOOIDC`

   This process installs the NuGet packages from the NuGet package manager. Because we know exactly what NuGet packages we need for this tutorial, we can perform this step now. It's also common that the required packages become known during development. When this happens, follow a similar process to install them at that time.
4. If you intend to run the application from the command prompt, open a command prompt now and navigate to the folder that will contain the build output. This is typically something like `S3CreateAndList\S3CreateAndList\bin\Debug\net6.0`, but will depend on your environment.

Create the code

1. In the `S3CreateAndList` project, find and open `Program.cs` in the IDE.
2. Replace the contents with the following code and save the file.

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

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

namespace S3CreateAndList
{
    class Program
```

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

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

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

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

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

```
                Console.WriteLine("Caught exception when creating a bucket:");
                Console.WriteLine(e.Message);
            }
        }

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

    //
    // Method to parse the command line.
    private static Boolean GetBucketName(string[] args, out String bucketName)
    {
        Boolean retval = false;
        bucketName = String.Empty;
        if (args.Length == 0)
        {
            Console.WriteLine("\nNo arguments specified. Will simply list your
Amazon S3 buckets." +
              "\nIf you wish to create a bucket, supply a valid, globally
unique bucket name.");
            bucketName = String.Empty;
            retval = false;
        }
        else if (args.Length == 1)
        {
            bucketName = args[0];
            retval = true;
        }
        else
        {
            Console.WriteLine("\nToo many arguments specified." +
              "\n\ndotnet_tutorials - A utility to list your Amazon S3 buckets
and optionally create a new one." +
              "\n\nUsage: S3CreateAndList [bucket_name]" +
              "\n - bucket_name: A valid, globally unique bucket name." +
```

```
                    "\n - If bucket_name isn't supplied, this utility simply lists
 your buckets.");
                Environment.Exit(1);
            }
            return retval;
        }

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

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

3. Build the application.

> **Note**
>
> If you're using an older version of Visual Studio, you might get a build error similar to the following:
> "Feature 'async main' is not available in C# 7.0. Please use language version 7.1 or greater."
> If you get this error, set up your project to use a later version of the language. This is typically done in the project properties, **Build**, **Advanced**.

Run the application

1. Run the application with no command line arguments. Do this either in the command prompt (if you opened one earlier) or from the IDE.
2. Examine the output to see the number of Amazon S3 buckets that you own, if any, and their names.
3. Choose a name for a new Amazon S3 bucket. Use "dotnet-quicktour-s3-1-winvs-" as a base and add something unique to it, such as a GUID or your name. Be sure to follow the rules for bucket names, as described in Rules for Bucket Naming in the Amazon S3 User Guide.
4. Run the application again, this time supplying the bucket name.

   In the command line, replace *amzn-s3-demo-bucket* in the following command with the name of the bucket that you chose.

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

   Or, if you are running the application in the IDE, choose **Project**, **S3CreateAndList Properties**, **Debug** and enter the bucket name there.
5. Examine the output to see the new bucket that was created.

Cleanup

While performing this tutorial, you created some resources that you can choose to clean up at this time.

- If you don't want to keep the bucket that the application created in an earlier step, delete it by using the Amazon S3 console at https://console.aws.amazon.com/s3/.
- If you don't want to keep your .NET project, remove the `S3CreateAndList` folder from your development environment.

Where to go next

Go back to the quick-tour menu or go straight to the end of this quick tour.

Next steps

Be sure to clean up any leftover resources that you created while performing these tutorials. These might be AWS resources or resources in your development environment such as files and folders.

Now that you've toured the AWS SDK for .NET, you might want to start your project.

Configuring the AWS SDK for .NET

To use the AWS SDK for .NET, you need to set up your environment and also configure a number of essential things that your application needs to access AWS services. These include:

- Specification of the AWS Region
- AWSSDK packages or assemblies

Some of the topics in this section provide information about how to configure these essential things.

Other topics in this section and other sections provide information about more advanced ways that you can configure your project.

Topics

- Start a new project
- Install AWSSDK packages with NuGet
- Install AWSSDK assemblies without NuGet
- Setting the AWS Region for the AWS SDK for .NET
- Credential and profile resolution
- Retries and timeouts
- Observability
- Additional information about users and roles
- Advanced configuration for your AWS SDK for .NET project
- Using legacy credentials

Start a new project

There are several techniques you can use to start a new project to access AWS services. The following are some of those techniques:

- If you're new to .NET development on AWS or at least new to the AWS SDK for .NET, you can see complete examples in Creating a simple application. It gives you an introduction to the SDK.

- You can start a basic project by using the .NET CLI. To see an example of this, open a command prompt or terminal, create a folder or directory and navigate to it, and then enter the following.

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

  An empty project is created to which you can add code and NuGet packages. For more information, see the .NET Core guide.

  To see a list of project templates, use the following: `dotnet new --list`

- The AWS Toolkit for Visual Studio includes C# project templates for a variety of AWS services. After you install the toolkit in Visual Studio, you can access the templates while creating a new project.

  To see this, go to Working with AWS services in the AWS Toolkit for Visual Studio User Guide. Several of the examples in that section create new projects.

- If you develop with Visual Studio on Windows but without the AWS Toolkit for Visual Studio, use your typical techniques for creating a new project.

  To see an example, open Visual Studio and choose **File**, **New**, **Project**. Search for ".net core" and choose the C# version of the **Console App (.NET Core)** or **WPF App (.NET Core)** template. An empty project is created to which you can add code and NuGet packages.

You can find some examples of how to work with AWS services in Guided code examples.

> **Important**
>
> If you're using AWS IAM Identity Center for authentication, your application must reference the following NuGet packages so that SSO resolution can work:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> Failure to reference these packages will result in a *runtime* exception.

Install AWSSDK packages with NuGet

NuGet is a package management system for the .NET platform. With NuGet, you can install the AWSSDK packages, as well as several other extensions, to your project. For additional information, see the aws/dotnet repository on the GitHub website.

NuGet always has the most recent versions of the AWSSDK packages, as well as previous versions. NuGet is aware of dependencies between packages and installs all required packages automatically.

> **Warning**
>
> The list of NuGet packages might include one named simply "AWSSDK" (with no appended identifier). Do NOT install this NuGet package; it is legacy and should not be used for new projects.

Packages installed with NuGet are stored with your project instead of in a central location. This enables you to install assembly versions specific to a given application without creating compatibility issues for other applications. For more information about NuGet, see the NuGet documentation.

> **Note**
>
> If you can't or aren't allowed to download and install NuGet packages on a per-project basis, you can obtain the AWSSDK assemblies and store them locally (or on premises). If this applies to you and you haven't already obtained the AWSSDK assemblies, see Obtaining AWSSDK assemblies. To learn how to use the locally stored assemblies, see Install AWSSDK assemblies without NuGet.

Using NuGet from the Command prompt or terminal

1. Go to the AWSSDK packages on NuGet and determine which packages you need in your project; for example, **AWSSDK.S3**.
2. Copy the .NET CLI command from that package's webpage, as shown in the following example.

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

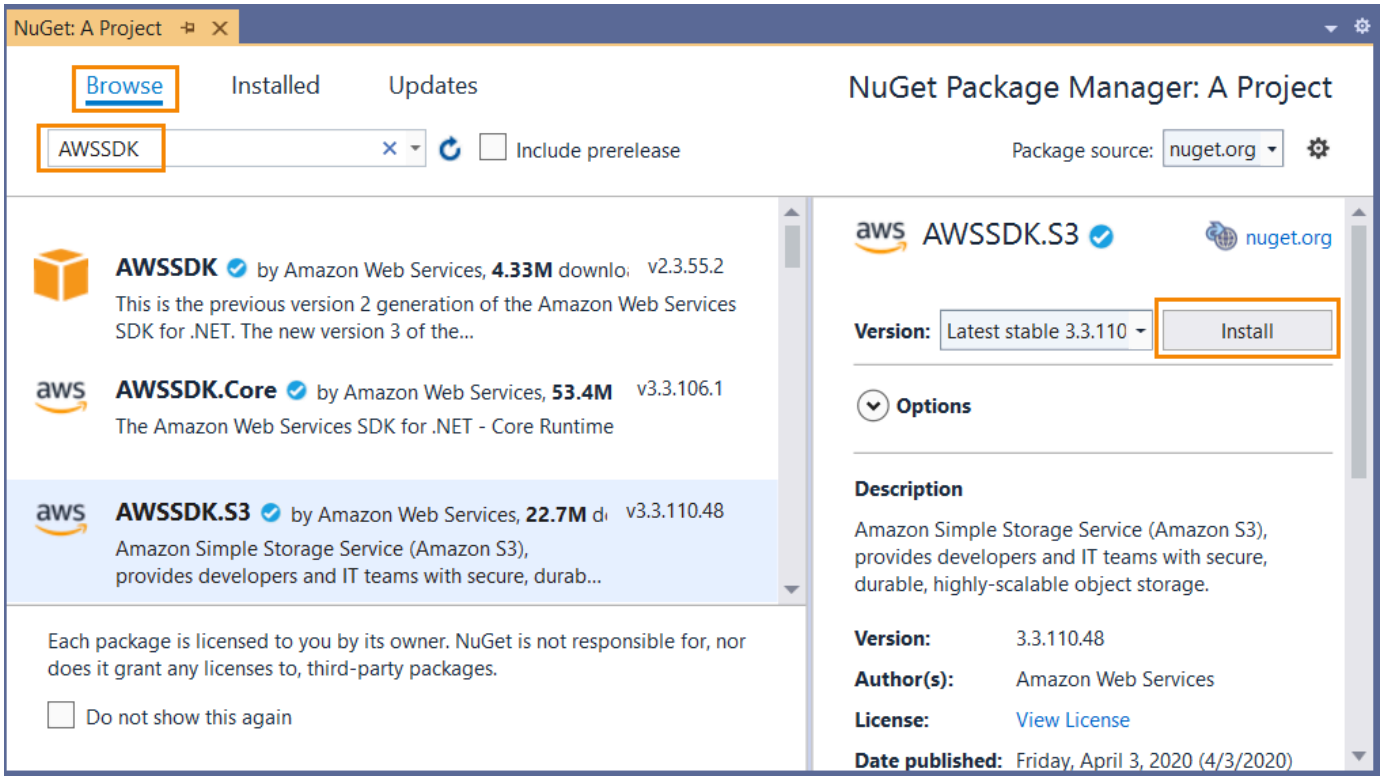
3. In your project's directory, run that .NET CLI command. NuGet also installs any dependencies, such as AWSSDK.Core.

> **Note**
>
> If you want only the latest version of a NuGet package, you can exclude version information from the command, as shown in the following example.
>
> **`dotnet add package AWSSDK.S3`**

Using NuGet from Visual Studio Solution Explorer

1. In **Solution Explorer**, right-click your project, and then choose **Manage NuGet Packages** from the context menu.
2. In the left pane of the **NuGet Package Manager**, choose **Browse**. You can then use the search box to search for the package you want to install. NuGet also installs any dependencies, such as AWSSDK.Core.

   The following figure shows installation of the **AWSSDK.S3** package.



Using NuGet from the Package Manager Console

In Visual Studio, choose **Tools**, **NuGet Package Manager**, **Package Manager Console**.

You can install the AWSSDK packages you want from the Package Manager Console by using the **`Install-Package`** command. For example, to install AWSSDK.S3, use the following command.

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

NuGet also installs any dependencies, such as AWSSDK.Core.

If you need to install an earlier version of a package, use the `-Version` option and specify the package version you want, as shown in the following example.

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

For more information about Package Manager Console commands, see the PowerShell reference in Microsoft's NuGet documentation.

Install AWSSDK assemblies without NuGet

This topic describes how you can use the AWSSDK assemblies that you obtained and stored locally (or on premises) as described in Obtaining AWSSDK assemblies. This is **not** the recommended method for handling SDK references, but is required in some environments.

> **Note**
>
> The recommended method for handling SDK references is to download and install just the NuGet packages that each project needs. That method is described in Install AWSSDK packages with NuGet.

To install AWSSDK assemblies

1. Create a folder in your project area for the required AWSSDK assemblies. As an example, you might call this folder `AwsAssemblies`.
2. If you haven't already done so, obtain the AWSSDK assemblies, which places the assemblies in some local download or installation folder. Copy the DLL files for the required assemblies from that download folder into your project (into the `AwsAssemblies` folder, in our example).

Be sure to also copy any dependencies. You can find information about dependencies on the GitHub website.

3. Make reference to the required assemblies as follows.

   Cross-platform development

   1. Open your project's `.csproj` file and add an `<ItemGroup>` element.
   2. In the `<ItemGroup>` element, add a `<Reference>` element with an `Include` attribute for each required assembly.

      For Amazon S3, for example, you would add the following lines to your project's `.csproj` file.

      On Linux and macOS:

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

      On Windows:

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

   3. Save you project's `.csproj` file.

   Windows with Visual Studio and .NET Core

   1. In Visual Studio, load your project and open **Project**, **Add Reference**.
   2. Choose the **Browse** button on the bottom of the dialog box. Navigate to your project's folder and the subfolder that you copied the required DLL files to (`AwsAssemblies`, for example).
   3. Select all the DLL files, choose **Add**, and choose **OK**.
   4. Save your project.

Setting the AWS Region for the AWS SDK for .NET

AWS Regions allow you to access AWS services that physically reside in a specific geographic region. This can be useful for redundancy and to keep your data and applications running close to where you and your users will access them.

To view the current list of all supported Regions and endpoints for each AWS service, see Service endpoints and quotas in the *AWS General Reference*. To view a list of existing Regional endpoints, see AWS service endpoints. To see detailed information about Regions, see Specify which AWS Regions your account can use.

You can create an AWS service client that goes to a particular Region. You can also configure your application with a Region that will be used for all AWS service clients. These two cases are explained next.

Create a service client with a particular Region

You can specify the Region for any of the AWS service clients in your application. Setting the Region in this way takes precedence over any global setting for that particular service client.

Existing Region

This example shows you how to instantiate an Amazon EC2 client in an existing Region. It uses defined RegionEndpoint fields.

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

New Region using RegionEndpoint class

This example shows you how to construct a new Region endpoint by using RegionEndpoint.GetBySystemName.

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

```
}
```

New Region using the service client configuration class

This example shows you how to use the `ServiceURL` property of the service client configuration class to specify the Region; in this case, using the AmazonEC2Config class.

This technique works even if the Region endpoint doesn't follow the regular Region endpoint pattern.

```
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
}
```

Specify a Region for all service clients

There are several ways you can specify a Region for all of the AWS service clients that your application creates. This Region is used for service clients that aren't created with a particular Region.

The AWS SDK for .NET looks for a Region value in the following order.

Profiles

Set in a profile that your application or the SDK has loaded. For more information, see Credential and profile resolution.

Environment variables

Set in the `AWS_REGION` environment variable.

On Linux or macOS:

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

On Windows:

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

> **Note**
>
> If you set this environment variable for the whole system (using `export` or `setx`), it affects all SDKs and toolkits, not just the AWS SDK for .NET.

AWSConfigs class

Set as an AWSConfigs.AWSRegion property.

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

Region resolution

If none of the methods described above are used to specify an AWS Region, the AWS SDK for .NET attempts to find a Region for the AWS service client to operate in.

Region resolution order

1. Application configuration files such as `app.config` and `web.config`.
2. Environment variables (`AWS_REGION` and `AWS_DEFAULT_REGION`).
3. A profile with the name specified by a value in `AWSConfigs.AWSProfileName`.
4. A profile with the name specified by the `AWS_PROFILE` environment variable.
5. The `[default]` profile.
6. Amazon EC2 instance metadata (if running on an EC2 instance).

If no Region is found, the SDK throws an exception stating that the AWS service client has no configured Region.

Special information about the China (Beijing) Region

To use services in the China (Beijing) Region, you must have an account and credentials that are specific to the China (Beijing) Region. Accounts and credentials for other AWS Regions won't work for the China (Beijing) Region. Likewise, accounts and credentials for the China (Beijing) Region won't work for other AWS Regions. For information about endpoints and protocols that are available in the China (Beijing) Region, see Beijing Region Endpoints.

Special information about new AWS services

New AWS services can be launched initially in a few Regions and then supported in other Regions. In these cases you don't need to install the latest SDK to access the new Regions for that service. You can specify newly added Regions on a per-client basis or globally, as shown earlier.

Credential and profile resolution

The AWS SDK for .NET searches for credentials in a certain order and uses the first available set for the current application.

Credential search order

1. Credentials that are explicitly set on the AWS service client, as described in Accessing credentials and profiles in an application.

   > **Note**
   >
   > That topic is in the Special considerations section because it isn't the preferred method for specifying credentials.

2. SessionAWSCredentials that are created from the `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, and `AWS_SESSION_TOKEN` environment variables, if all three variables have a value.
3. BasicAWSCredentials that are created from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables, if both variables have a value.
4. AssumeRoleWithWebIdentityCredentials that are created from the `AWS_WEB_IDENTITY_TOKEN_FILE` and `AWS_ROLE_ARN` environment variables, if both variables have a value.

5. A credentials profile with the name specified by a value in AWSConfigs.AWSProfileName.
6. A credentials profile with the name specified by the `AWS_PROFILE` environment variable.
7. The `[default]` credentials profile.
8. The container credential provider.
9. Amazon EC2 instance metadata.

If your application is running on an Amazon EC2 instance, such as in a production environment, use an IAM role as described in Granting access by using an IAM role. Otherwise, such as in prerelease testing, store your credentials in a file that uses the AWS credentials file format that your web application has access to on the server.

For additional information about environment variables and credentials profiles, see the following topics in the AWS SDKs and Tools Reference Guide: Environment variables, Environment variables list, and Shared config and credentials files.

Profile resolution

With two different storage mechanisms for credentials, it's important to understand how to configure the AWS SDK for .NET to use them. The AWSConfigs.AWSProfilesLocation property controls how the AWS SDK for .NET finds credential profiles.

AWSProfilesLocation	Profile resolution behavior
null (not set) or empty	Search the SDK Store if the platform supports it, and then search the shared AWS credentials file in the default location. If the profile isn't in either of those locations, search `~/.aws/config` (Linux or macOS) or `%USERPROFILE%\.aws\config` (Windows).
The path to a file in the AWS credentials file format	Search *only* the specified file for a profile with the specified name.

Using federated user account credentials

Applications that use the AWS SDK for .NET (AWSSDK.Core version 3.1.6.0 and later) can use federated user accounts through Active Directory Federation Services (AD FS) to access AWS services by using Security Assertion Markup Language (SAML).

Federated access support means users can authenticate using your Active Directory. Temporary credentials are granted to the user automatically. These temporary credentials, which are valid for one hour, are used when your application invokes AWS services. The SDK handles management of the temporary credentials. For domain-joined user accounts, if your application makes a call but the credentials have expired, the user is reauthenticated automatically and fresh credentials are granted. (For accounts that are not joined to a domain, the user is prompted to enter credentials before reauthentication.)

To use this support in your .NET application, you must first set up the role profile by using a PowerShell cmdlet. To learn how, see the AWS Tools for Windows PowerShell documentation.

After you set up the role profile, reference the profile in your application. There are a number of ways to do this, one of which is by using the AWSConfigs.AWSProfileName property in the same way you would with other credential profiles.

The *AWS Security Token Service* assembly (AWSSDK.SecurityToken) provides the SAML support to obtain AWS credentials. To use federated user account credentials, be sure this assembly is available to your application.

Specifying roles or temporary credentials

For applications that run on Amazon EC2 instances, the most secure way to manage credentials is to use IAM roles, as described in Granting access by using an IAM role.

For application scenarios in which the software executable is available to users outside your organization, we recommend that you design the software to use *temporary security credentials*. In addition to providing restricted access to AWS resources, these credentials have the benefit of expiring after a specified period of time. For more information about temporary security credentials, see the following:

- Temporary security credentials
- Amazon Cognito identity pools

Using proxy credentials

If your software communicates with AWS through a proxy, you can specify credentials for the proxy by using the `ProxyCredentials` property of the `Config` class of a service. The `Config` class of a service is typically part of the primary namespace for the service. Examples include the following: AmazonCloudDirectoryConfig in the Amazon.CloudDirectory namespace and AmazonGameLiftConfig in the Amazon.GameLift namespace.

For Amazon S3, for example, you could use code similar to the following, where `SecurelyStoredUserName` and `SecurelyStoredPassword` are the proxy user name and password specified in a NetworkCredential object.

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

> **Note**
>
> Earlier versions of the SDK used `ProxyUsername` and `ProxyPassword`, but these properties are deprecated.

Retries and timeouts

The AWS SDK for .NET enables you to configure the number of retries and the timeout values for HTTP requests to AWS services. If the default values for retries and timeouts are not appropriate for your application, you can adjust them for your specific requirements, but it is important to understand how doing so will affect the behavior of your application.

To determine which values to use for retries and timeouts, consider the following:

- How should the AWS SDK for .NET and your application respond when network connectivity degrades or an AWS service is unreachable? Do you want the call to fail fast, or is it appropriate for the call to keep retrying on your behalf?
- Is your application a user-facing application or website that must be responsive, or is it a background processing job that has more tolerance for increased latencies?
- Is the application deployed on a reliable network with low latency, or is it deployed at a remote location with unreliable connectivity?

Retries

Overview

The AWS SDK for .NET can retry requests that fail due to server-side throttling or dropped connections. There are two properties of service configuration classes that you can use to specify the retry behavior of a service client. Service configuration classes inherit these properties from the abstract Amazon.Runtime.ClientConfig class of the AWS SDK for .NET API Reference:

- `RetryMode` specifies one of three retry modes, which are defined in the Amazon.Runtime.RequestRetryMode enumeration.

  The default value for your application can be controlled by using the `AWS_RETRY_MODE` environment variable or the *retry_mode* setting in the shared AWS config file.

- `MaxErrorRetry` specifies the number of retries allowed at the service client level; the SDK retries the operation the specified number of times before failing and throwing an exception.

  The default value for your application can be controlled by using the `AWS_MAX_ATTEMPTS` environment variable or the *max_attempts* setting in the shared AWS config file.

Detailed descriptions for these properties can be found in the abstract Amazon.Runtime.ClientConfig class of the AWS SDK for .NET API Reference. Each value of `RetryMode` corresponds by default to a particular value of `MaxErrorRetry`, as shown in the following table.

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

Behavior

When your application starts

When your application starts, default values for `RetryMode` and `MaxErrorRetry` are configured by the SDK. These default values are used when you create a service client unless you specify other values.

- If the properties aren't set in your environment, the default for `RetryMode` is configured as *Legacy* and the default for `MaxErrorRetry` is configured with the corresponding value from the preceding table.
- If the retry mode has been set in your environment, that value is used as the default for `RetryMode`. The default for `MaxErrorRetry` is configured with the corresponding value from the preceding table unless the value for maximum errors has also been set in your environment (described next).
- If the value for maximum errors has been set in your environment, that value is used as the default for `MaxErrorRetry`. Amazon DynamoDB is the exception to this rule; the default DynamoDB value for `MaxErrorRetry` is always the value from the preceding table.

As your application runs

When you create a service client, you can use the default values for `RetryMode` and `MaxErrorRetry`, as described earlier, or you can specify other values. To specify other values, create and include a service configuration object such as AmazonDynamoDBConfig or AmazonSQSConfig when you create the service client.

These values can't be changed for a service client after it has been created.

Considerations

When a retry occurs, the latency of your request is increased. You should configure your retries based on your application limits for total request latency and error rates.

Timeouts

The AWS SDK for .NET enables you to configure request timeouts at the service client level and per method call. There are two mechanisms for configuring timeouts, which are covered in subsequent sections:

- If you are using asynchronous calls, you can use the `CancellationToken` parameter of the method.
- If you are using synchronous calls in .NET Framework, you can use the `Timeout` and `ReadWriteTimeout` properties of the abstract Amazon.Runtime.ClientConfig class.

Using the CancellationToken parameter for timeouts

The AWS SDK for .NET enables you to configure request timeouts on asynchronous calls by using the `CancellationToken` parameter. The following code snippet shows an example. The code throws `System.Threading.Tasks.TaskCanceledException` if the request isn't completed within 10 seconds.

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

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

Using the Timeout and ReadWriteTimeout properties for timeouts

> **Note**
>
> The `Timeout` property doesn't impact asynchronous calls. If you are using asynchronous calls see Using the `CancellationToken` parameter for timeouts instead.

The AWS SDK for .NET enables you to configure the request timeout and socket read/write timeout values at the service client level. These values are specified in the `Timeout` and the `ReadWriteTimeout` properties of the abstract Amazon.Runtime.ClientConfig class. These values are passed on as the `Timeout` and `ReadWriteTimeout` properties of the HttpWebRequest objects created by the AWS service client object. By default, the `Timeout` value is 100 seconds and the `ReadWriteTimeout` value is 300 seconds.

When your network has high latency, or conditions exist that cause an operation to be retried, using long timeout values and a high number of retries can cause some SDK operations to seem unresponsive.

> **Note**
>
> The version of the AWS SDK for .NET that targets the portable class library (PCL) uses the HttpClient class instead of the `HttpWebRequest` class, and supports the Timeout property only.

The following are the exceptions to the default timeout values. These values are overridden when you explicitly set the timeout values.

- `Timeout` and `ReadWriteTimeout` are set to the maximum values if the method being called uploads a stream, such as AmazonS3Client.PutObjectAsync(), AmazonS3Client.UploadPartAsync(), AmazonGlacierClient.UploadArchiveAsync(), and so on.
- The versions of the AWS SDK for .NET that target .NET Framework set `Timeout` and `ReadWriteTimeout` to the maximum values for all AmazonS3Client and AmazonGlacierClient objects.
- The versions of the AWS SDK for .NET that target the portable class library (PCL) and .NET Core set `Timeout` to the maximum value for all AmazonS3Client and AmazonGlacierClient objects.

The following example shows you how to specify *Standard* retry mode, a maximum of 3 retries, a timeout of 10 seconds, and a read/write timeout of 10 seconds (if applicable). The AmazonS3Client constructor is given an AmazonS3Config object.

```
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
  });
```

Observability

Observability is the extent to which a system's current state can be inferred from the data it emits. The data emitted is commonly referred to as telemetry.

The AWS SDK for .NET can provide two common telemetry signals, metrics and traces, as well as logging. You can wire up a TelemetryProvider to send telemetry data to an observability backend (such as AWS X-Ray or Amazon CloudWatch) and then act on it.

By default, telemetry signals are disabled in the SDK. This topic explains how to enable and configure telemetry output.

Additional resources

For more information about enabling and using observability, see the following resources:

- OpenTelemetry
- The blog post Enhancing Observability in the AWS SDK for .NET with OpenTelemetry.
- The blog post Announcing the general availability of AWS .NET OpenTelemetry libraries.
- Exporters for OpenTelemetry
- For examples of observability in the AWS Tools for PowerShell, see Observability in the Tools for PowerShell User Guide.

Configure a `TelemetryProvider`

You can configure a `TelemetryProvider` in your application globally for all service clients or for individual clients, as shown in the following examples. The the section called “Telemetry providers” section contains information about telemetry implementations, including information about implementations that are provided with the SDK.

Configure the default global telemetry provider

By default, every service client attempts to use the globally available telemetry provider. This way, you can set the provider once, and all clients will use it. This should be done only once, before you create any service clients.

The following code snippet shows you how to set the global telemetry provider. It then creates an Amazon S3 service client and tries to perform an operation that fails. The code adds

both tracing and metrics to the application. This code uses the following NuGet packages: `OpenTelemetry.Exporter.Console` and `OpenTelemetry.Instrumentation.AWS`.

> **Note**
>
> If you're using AWS IAM Identity Center for authentication, be sure to also add `AWSSDK.SSO` and `AWSSDK.SSOOIDC`.

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

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

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

var s3Client = new AmazonS3Client();

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

Console.Read();
```

Configure a telemetry provider for a specific service client

You can configure an individual service client with a specific telemetry provider (other than the global one). To do so, use the `TelemetryProvider` class of the Config object of a service client constructor. For example, see AmazonS3Config and look for the `TelemetryProvider` property. See the section called "Telemetry providers" for information about custom telemetry implementations.

Topics

- Metrics
- Telemetry providers

Metrics

The following table lists the telemetry metrics that the SDK emits. Configure a telemetry provider to make the metrics observable.

What metrics are emitted?

Metric name	Units	Type	Attributes	Description
client.call.duration	s	Histogra	rpc.service, rpc.method	Overall call duration (including retries and time to send or receive request and response body)
client.uptime	s	Histogra	rpc.service	The amount of time since a client was created
client.call.attempts	{attempt	Monotor Counter	rpc.service, rpc.method	The number of attempts for an individual operation
client.call.errors	{error}	Monotor Counter	rpc.service, rpc.method, exception.type	The number of errors for an operation
client.call.attempt_duration	s	Histogra	rpc.service, rpc.method	The time it takes to connect to the service, send the request, and get back HTTP status code and headers

Metric name	Units	Type	Attributes	Description
				(including time queued waiting to be sent)
client.call.resolve_endpoint_duration	s	Histogra	rpc.service, rpc.method	The time it takes to resolve an endpoint (endpoint resolver, not DNS) for the request
client.call.serialization_duration	s	Histogra	rpc.service, rpc.method	The time it takes to serialize a message body
client.call.deserialization_duration	s	Histogra	rpc.service, rpc.method	The time it takes to deserialize a message body
client.call.auth.signing_duration	s	Histogra	rpc.service, rpc.method	The time it takes to sign a request
client.call.auth.resolve_identity_duration	s	Histogra	rpc.service, rpc.method	The time it takes to acquire an identity (such as AWS credentials or a bearer token) from an Identity Provider
client.http.bytes_sent	By	Monotor Counter	server.address	The total number of bytes sent by the HTTP client
client.http.bytes_received	By	Monotor Counter	server.address	The total number of bytes received by the HTTP client

Following are the column descriptions:

- **Metric name**–The name of the emitted metric.
- **Units**–The unit of measure for the metric. Units are given in the UCUM case sensitive ("c/s") notation.
- **Type**–The type of instrument used to capture the metric.
- **Attributes**–The set of attributes (dimensions) emitted with the metric.

- **Description**–A description of what the metric is measuring.

Telemetry providers

The SDK provides an implementation of OpenTelemetry as a telemetry provider, which is described in the next section.

If you have specific telemetry requirements, already have a telemetry solution in mind, or need fine-grained control over how telemetry data is captured and processed, you can also implement your own telemetry provider.

Register your own implementation with the TelemetryProvider class. The following is a simple example of how to register your own `TracerProvider` and `MeterProvider`.

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

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

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

Topics

- Configure the OpenTelemetry-based telemetry provider

Configure the OpenTelemetry-based telemetry provider

The AWS SDK for .NET includes an implementation of an OpenTelemetry-based telemetry provider. For details about how to set this provider as the global telemetry provider, see Configure a

`TelemetryProvider`. To use this telemetry provider, you need the following resources in your project:

- The OpenTelemetry.Instrumentation.AWS NuGet package.
- A telemetry exporter such as OTLP or Console. For more information, see Exporters in the OpenTelemetry documentation.

The OpenTelemetry implementation included with the SDK can be configured to reduce the amount of tracing for HTTPS requests, credentials, and compression. To do so, set the `SuppressDownstreamInstrumentation` option to `true`, similar to the following:

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

For additional information about this provider, see the blog post Enhancing Observability in the AWS SDK for .NET with OpenTelemetry.

Additional information about users and roles

For doing .NET development on AWS or for running .NET applications on AWS, you need to have some combination of users, permission sets, and service roles that are appropriate for these tasks.

The specific users, permission sets, and service roles that you create, and the way in which you use them, will depend on the requirements of your applications. The following is some additional information about why they might be used and how to create them.

Users and permission sets

Although it's possible to use an IAM user account with long-term credentials to access AWS services, this is no longer a best practice and should be avoided. Even during development, it is a best practice to create users and permission sets in AWS IAM Identity Center and use temporary credentials provided by an identity source.

For development, you can use the user that you created or were given in Authenticating with AWS. If you have appropriate AWS Management Console permissions, you can also create different permission sets with least privilege for that user or create new users specifically for development

projects, providing permission sets with least privilege. The course of action you choose, if any, depends on your circumstances.

For more information about these users and permissions sets and how to create them, see Authentication and access in the *AWS SDKs and Tools Reference Guide* and Getting started in the *AWS IAM Identity Center User Guide*.

Service roles

You can set up an AWS service role to access AWS services on behalf of users. This type of access is appropriate if multiple people will be running your application remotely; for example, on an Amazon EC2 instance that you have created for this purpose.

The process for creating a service role varies depending on the situation, but is essentially the following.

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. Choose **Roles**, and then choose **Create role**.
3. Choose **AWS service**, find and select **EC2** (for example), and then choose the **EC2** use case (for example).
4. Choose **Next: Permissions**, and select the appropriate policies for the AWS services that your application will use.

   > **Warning**
   >
   > Do ***NOT*** choose the **AdministratorAccess** policy because that policy enables read and write permissions to almost everything in your account.

5. Choose **Next: Tags** and enter any tags you want.

   You can find information about tags in Control access using AWS resource tags in the IAM User Guide.
6. Choose **Next: Review** and provide a **Role name** and **Role description**. Then choose **Create role**.

You can find high-level information about IAM roles in Identities (users, groups, and roles) in the IAM User Guide. Find detailed information about roles in that guide's IAM roles topic.

Additional information about roles

- Use IAM roles for tasks for Amazon Elastic Container Service (Amazon ECS) tasks.
- Use IAM roles for applications that are running on Amazon EC2 instances.

Advanced configuration for your AWS SDK for .NET project

The topics in this section contain information about additional configuration tasks and methods that might be of interest to you.

Topics

- Using AWSSDK.Extensions.NETCore.Setup and the IConfiguration interface
- Configuring Other Application Parameters
- Configuration Files Reference for AWS SDK for .NET

Using AWSSDK.Extensions.NETCore.Setup and the IConfiguration interface

(This topic was formerly titled, "Configuring the AWS SDK for .NET with .NET Core")

One of the biggest changes in .NET Core is the removal of `ConfigurationManager` and the standard `app.config` and `web.config` files that were used with .NET Framework and ASP.NET applications.

Configuration in .NET Core is based on key-value pairs established by configuration providers. Configuration providers read configuration data into key-value pairs from a variety of configuration sources, including command-line arguments, directory files, environment variables, and settings files.

> **Note**
>
> For further information, see Configuration in ASP.NET Core.

To make it easy to use the AWS SDK for .NET with .NET Core, you can use the AWSSDK.Extensions.NETCore.Setup NuGet package. Like many .NET Core libraries, it adds

extension methods to the `IConfiguration` interface to make getting the AWS configuration seamless.

The source code for this package is on GitHub at https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup.

Using AWSSDK.Extensions.NETCore.Setup

Suppose that you create an ASP.NET Core Model-View-Controller (MVC) application, which can be accomplished with the **ASP.NET Core Web Application** template in Visual Studio or by running `dotnet new mvc ...` in the .NET Core CLI. When you create such an application, the constructor for `Startup.cs` handles configuration by reading in various input sources from configuration providers such as `appsettings.json`.

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

To use the `Configuration` object to get the *AWS* options, first add the `AWSSDK.Extensions.NETCore.Setup` NuGet package. Then, add your options to the configuration file as described next.

Notice that one of the files added to your project is `appsettings.Development.json`. This corresponds to an `EnvironmentName` set to **Development**. During development, you put your configuration in this file, which is only read during local testing. When you deploy an Amazon EC2 instance that has `EnvironmentName` set to **Production**, this file is ignored and the AWS SDK for .NET falls back to the IAM credentials and Region that are configured for the Amazon EC2 instance.

The following configuration settings show examples of the values you can add in the `appsettings.Development.json` file in your project to supply AWS settings.

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

To access a setting in a *CSHTML* file, use the `Configuration` directive.

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

<h1>Contact</h1>

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

To access the AWS options set in the file from code, call the `GetAWSOptions` extension method added to `IConfiguration`.

To construct a service client from these options, call `CreateServiceClient`. The following example shows how to create an Amazon S3 service client. (Be sure to add the AWSSDK.S3 NuGet package to your project.)

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

You can also create multiple service clients with incompatible settings by using multiple entries in the `appsettings.Development.json` file, as shown in the following examples where the configuration for `service1` includes the `us-west-2` Region and the configuration for `service2` includes the special endpoint *URL*.

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

You can then get the options for a specific service by using the entry in the JSON file. For example, to get the settings for `service1` use the following.

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

Allowed values in appsettings file

The following app configuration values can be set in the `appsettings.Development.json` file. The field names must use the casing shown. For details on these settings, see the `AWS.Runtime.ClientConfig` class.

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

ASP.NET Core dependency injection

The *AWSSDK.Extensions.NETCore.Setup* NuGet package also integrates with a new dependency injection system in ASP.NET Core. The `ConfigureServices` method in your application's `Startup` class is where the MVC services are added. If the application is using Entity Framework, this is also where that is initialized.

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

> **Note**
>
> Background on dependency injection in .NET Core is available on the .NET Core documentation site.

The `AWSSDK.Extensions.NETCore.Setup` NuGet package adds new extension methods to `IServiceCollection` that you can use to add AWS services to the dependency injection. The following code shows you how to add the AWS options that are read from `IConfiguration` to add Amazon S3 and DynamoDB to the list of services. (Be sure to add the AWSSDK.S3 and AWSSDK.DynamoDBv2 NuGet packages to your project.)

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

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

Now, if your MVC controllers use either `IAmazonS3` or `IAmazonDynamoDB` as parameters in their constructors, the dependency injection system passes in those services.

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

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

```
    ...

}
```

Configuring Other Application Parameters

> **Note**
>
> The information in this topic is specific to projects based on .NET Framework. The `App.config` and `Web.config` files are not present by default in projects based on .NET Core.

Open to view .NET Framework content

There are a number of application parameters that you can configure:

- `AWSLogging`
- `AWSLogMetrics`
- `AWSRegion`
- `AWSResponseLogging`
- `AWS.DynamoDBContext.TableNamePrefix`
- `AWS.S3.UseSignatureVersion4`
- `AWSEndpointDefinition`
- AWS Service-Generated Endpoints

These parameters can be configured in the application's `App.config` or `Web.config` file. Although you can also configure these with the AWS SDK for .NET API, we recommend you use the application's `.config` file. Both approaches are described here.

For more information about use of the `<aws>` element as described later in this topic, see Configuration Files Reference for AWS SDK for .NET.

AWSLogging

Configures how the SDK should log events, if at all. For example, the recommended approach is to use the `<logging>` element, which is a child element of the `<aws>` element:

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

Alternatively:

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

The possible values are:

`None`

Turn off event logging. This is the default.

`log4net`

Log using log4net.

`SystemDiagnostics`

Log using the `System.Diagnostics` class.

You can set multiple values for the `logTo` attribute, separated by commas. The following example sets both `log4net` and `System.Diagnostics` logging in the `.config` file:

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

Alternatively:

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

Alternatively, using the AWS SDK for .NET API, combine the values of the LoggingOptions enumeration and set the AWSConfigs.Logging property:

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

Changes to this setting take effect only for new AWS client instances.

AWSLogMetrics

Specifies whether or not the SDK should log performance metrics. To set the metrics logging configuration in the `.config` file, set the `logMetrics` attribute value in the `<logging>` element, which is a child element of the `<aws>` element:

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

Alternatively, set the `AWSLogMetrics` key in the `<appSettings>` section:

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

Alternatively, to set metrics logging with the AWS SDK for .NET API, set the AWSConfigs.LogMetrics property:

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

This setting configures the default `LogMetrics` property for all clients/configs. Changes to this setting take effect only for new AWS client instances.

AWSRegion

Configures the default AWS region for clients that have not explicitly specified a region. To set the region in the `.config` file, the recommended approach is to set the `region` attribute value in the `aws` element:

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

Alternatively, set the *AWSRegion* key in the `<appSettings>` section:

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

Alternatively, to set the region with the AWS SDK for .NET API, set the AWSConfigs.AWSRegion property:

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

For more information about creating an AWS client for a specific region, see AWS Region Selection. Changes to this setting take effect only for new AWS client instances.

AWSResponseLogging

Configures when the SDK should log service responses. The possible values are:

`Never`

Never log service responses. This is the default.

`Always`

Always log service responses.

`OnError`

Only log service responses when an error occurs.

To set the service logging configuration in the `.config` file, the recommended approach is to set the `logResponses` attribute value in the `<logging>` element, which is a child element of the `<aws>` element:

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

Alternatively, set the *AWSResponseLogging* key in the `<appSettings>` section:

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

Alternatively, to set service logging with the AWS SDK for .NET API, set the AWSConfigs.ResponseLogging property to one of the values of the ResponseLoggingOption enumeration:

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

Changes to this setting take effect immediately.

AWS.DynamoDBContext.TableNamePrefix

Configures the default `TableNamePrefix` the `DynamoDBContext` will use if not manually configured.

To set the table name prefix in the `.config` file, the recommended approach is to set the `tableNamePrefix` attribute value in the `<dynamoDBContext>` element, which is a child element of the `<dynamoDB>` element, which itself is a child element of the `<aws>` element:

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

Alternatively, set the `AWS.DynamoDBContext.TableNamePrefix` key in the `<appSettings>` section:

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

Alternatively, to set the table name prefix with the AWS SDK for .NET API, set the AWSConfigs.DynamoDBContextTableNamePrefix property:

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

Changes to this setting will take effect only in newly constructed instances of `DynamoDBContextConfig` and `DynamoDBContext`.

AWS.S3.UseSignatureVersion4

Configures whether or not the Amazon S3 client should use signature version 4 signing with requests.

To set signature version 4 signing for Amazon S3 in the `.config` file, the recommended approach is to set the `useSignatureVersion4` attribute of the `<s3>` element, which is a child element of the `<aws>` element:

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

Alternatively, set the `AWS.S3.UseSignatureVersion4` key to `true` in the `<appSettings>` section:

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

Alternatively, to set signature version 4 signing with the AWS SDK for .NET API, set the AWSConfigs.S3UseSignatureVersion4 property to `true`:

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

By default, this setting is `false`, but signature version 4 may be used by default in some cases or with some regions. When the setting is `true`, signature version 4 will be used for all requests. Changes to this setting take effect only for new Amazon S3 client instances.

AWSEndpointDefinition

Configures whether the SDK should use a custom configuration file that defines the regions and endpoints.

To set the endpoint definition file in the `.config` file, we recommend setting the `endpointDefinition` attribute value in the `<aws>` element.

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

Alternatively, you can set the *AWSEndpointDefinition* key in the `<appSettings>` section:

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

Alternatively, to set the endpoint definition file with the AWS SDK for .NET API, set the AWSConfigs.EndpointDefinition property:

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

If no file name is provided, then a custom configuration file will not be used. Changes to this setting take effect only for new AWS client instances. The `endpoints.json` file is available from https://github.com/aws/aws-sdk-net/blob/main/sdk/src/Core/endpoints.json.

AWS Service-Generated Endpoints

Some AWS services generate their own endpoints instead of consuming a region endpoint. Clients for these services consume a service Url that is specific to that service and your resources. Two

examples of these services are Amazon CloudSearch and AWS IoT. The following examples show how you can obtain the endpoints for those services.

Amazon CloudSearch Endpoints Example

The Amazon CloudSearch client is used for accessing the Amazon CloudSearch configuration service. You use the Amazon CloudSearch configuration service to create, configure, and manage search domains. To create a search domain, create a CreateDomainRequest object and provide the `DomainName` property. Create an AmazonCloudSearchClient object by using the request object. Call the CreateDomain method. The CreateDomainResponse object returned from the call contains a `DomainStatus` property that has both the `DocService` and `SearchService` endpoints. Create an AmazonCloudSearchDomainConfig object and use it to initialize `DocService` and `SearchService` instances of the AmazonCloudSearchDomainClient class.

```
// 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 Endpoints Example

To obtain the endpoint for AWS IoT, create an AmazonIoTClient object and call the DescribeEndPoint method. The returned DescribeEndPointResponse object contains the `EndpointAddress`. Create an AmazonIotDataConfig object, set the `ServiceURL` property, and use the object to instantiate the AmazonIotDataClient class.

```
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");
}
```

Configuration Files Reference for AWS SDK for .NET

> **Note**
>
> The information in this topic is specific to projects based on .NET Framework. The `App.config` and `Web.config` files are not present by default in projects based on .NET Core.

Open to view .NET Framework content

You can use a .NET project's `App.config` or `Web.config` file to specify AWS settings, such as AWS credentials, logging options, AWS service endpoints, and AWS regions, as well as some settings for AWS services, such as Amazon DynamoDB, Amazon EC2, and Amazon S3. The following information describes how to properly format an `App.config` or `Web.config` file to specify these types of settings.

> **Note**
>
> Although you can continue to use the `<appSettings>` element in an `App.config` or `Web.config` file to specify AWS settings, we recommend you use the `<configSections>` and `<aws>` elements as described later in this topic. For more information about the `<appSettings>` element, see the `<appSettings>` element examples in Configuring Your AWS SDK for .NET Application.

Note

Although you can continue to use the following `AWSConfigs` class properties in a code file to specify AWS settings, the following properties are deprecated and may not be supported in future releases:

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

In general, we recommend that instead of using `AWSConfigs` class properties in a code file to specify AWS settings, you should use the `<configSections>` and `<aws>` elements in an `App.config` or `Web.config` file to specify AWS settings, as described later in this topic. For more information about the preceding properties, see the `AWSConfigs` code examples in Configuring Your AWS SDK for .NET Application.

Topics

- Declaring an AWS Settings Section
- Allowed Elements
- Elements Reference

Declaring an AWS Settings Section

You specify AWS settings in an `App.config` or `Web.config` file from within the `<aws>` element. Before you can begin using the `<aws>` element, you must create a `<section>` element (which is a child element of the `<configSections>` element) and set its `name` attribute to `aws` and its `type` attribute to `Amazon.AWSSection, AWSSDK.Core`, as shown in the following example:

```
<?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>
```

The Visual Studio Editor does not provide automatic code completion (IntelliSense) for the <aws> element or its child elements.

To assist you in creating a correctly formatted version of the <aws> element, call the Amazon.AWSConfigs.GenerateConfigTemplate method. This outputs a canonical version of the <aws> element as a pretty-printed string, which you can adapt to your needs. The following sections describe the <aws> element's attributes and child elements.

Allowed Elements

The following is a list of the logical relationships among the allowed elements in an AWS settings section. You can generate the latest version of this list by calling the Amazon.AWSConfigs.GenerateConfigTemplate method, which outputs a canonical version of the <aws> element as a string you can adapt to your needs.

```
<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>
```

Elements Reference

The following is a list of the elements that are allowed in an AWS settings section. For each element, its allowed attributes and parent-child elements are listed.

Topics

- alias
- aws
- dynamoDB
- dynamoDBContext
- ec2
- logging
- map
- property
- proxy

- s3

alias

The <alias> element represents a single item in a collection of one or more from-table to to-table mappings that specifies a different table than one configured for a type. This element maps to an instance of the Amazon.Util.TableAlias class from the Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases property in the AWS SDK for .NET. Remapping is done before applying a table name prefix.

This element can include the following attributes:

fromTable

The from-table portion of the from-table to to-table mapping. This attribute maps to the Amazon.Util.TableAlias.FromTable property in the AWS SDK for .NET.

toTable

The to-table portion of the from-table to to-table mapping. This attribute maps to the Amazon.Util.TableAlias.ToTable property in the AWS SDK for .NET.

The parent of the <alias> element is the <tableAliases> element.

The <alias> element contains no child elements.

The following is an example of the <alias> element in use:

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

aws

The <aws> element represents the top-most element in an AWS settings section. This element can include the following attributes:

endpointDefinition

The absolute path to a custom configuration file that defines the AWS regions and endpoints to use. This attribute maps to the `Amazon.AWSConfigs.EndpointDefinition` property in the AWS SDK for .NET.

profileName

The profile name for stored AWS credentials that will be used to make service calls. This attribute maps to the `Amazon.AWSConfigs.AWSProfileName` property in the AWS SDK for .NET.

profilesLocation

The absolute path to the location of the credentials file shared with other AWS SDKs. By default, the credentials file is stored in the `.aws` directory in the current user's home directory. This attribute maps to the `Amazon.AWSConfigs.AWSProfilesLocation` property in the AWS SDK for .NET.

region

The default AWS region ID for clients that have not explicitly specified a region. This attribute maps to the `Amazon.AWSConfigs.AWSRegion` property in the AWS SDK for .NET.

The `<aws>` element has no parent element.

The `<aws>` element can include the following child elements:

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

The following is an example of the `<aws>` element in use:

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

```
  profilesLocation="C:\Configs">
  <!-- ... -->
</aws>
```

dynamoDB

The <dynamoDB> element represents a collection of settings for Amazon DynamoDB. This element can include the *conversionSchema* attribute, which represents the version to use for converting between .NET and DynamoDB objects. Allowed values include V1 and V2. This attribute maps to the `Amazon.DynamoDBv2.DynamoDBEntryConversion` class in the AWS SDK for .NET. For more information, see DynamoDB Series - Conversion Schemas.

The parent of the <dynamoDB> element is the <aws> element.

The <dynamoDB> element can include the <dynamoDBContext> child element.

The following is an example of the <dynamoDB> element in use:

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

dynamoDBContext

The <dynamoDBContext> element represents a collection of Amazon DynamoDB context-specific settings. This element can include the *tableNamePrefix* attribute, which represents the default table name prefix the DynamoDB context will use if it is not manually configured. This attribute maps to the `Amazon.Util.DynamoDBContextConfig.TableNamePrefix` property from the `Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix` property in the AWS SDK for .NET. For more information, see Enhancements to the DynamoDB SDK.

The parent of the <dynamoDBContext> element is the <dynamoDB> element.

The <dynamoDBContext> element can include the following child elements:

- <alias> (one or more instances)
- <map> (one or more instances)

The following is an example of the <dynamoDBContext> element in use:

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

ec2

The <ec2> element represents a collection of Amazon EC2 settings. This element can include the *useSignatureVersion4* attribute, which specifies whether signature version 4 signing will be used for all requests (true) or whether signature version 4 signing will not be used for all requests (false, the default). This attribute maps to the `Amazon.Util.EC2Config.UseSignatureVersion4` property from the `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4` property in the AWS SDK for .NET.

The parent of the <ec2> element is the element.

The <ec2> element contains no child elements.

The following is an example of the <ec2> element in use:

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

logging

The <logging> element represents a collection of settings for response logging and performance metrics logging. This element can include the following attributes:

`logMetrics`

Whether performance metrics will be logged for all clients and configurations (true); otherwise, false. This attribute maps to the `Amazon.Util.LoggingConfig.LogMetrics` property from the `Amazon.AWSConfigs.LoggingConfig.LogMetrics` property in the AWS SDK for .NET.

`logMetricsCustomFormatter`

The data type and assembly name of a custom formatter for logging metrics. This attribute maps to the `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter` property from the `Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter` property in the AWS SDK for .NET.

`logMetricsFormat`

The format in which the logging metrics are presented (maps to the `Amazon.Util.LoggingConfig.LogMetricsFormat` property from the `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` property in the AWS SDK for .NET).

Allowed values include:

`JSON`

Use JSON format.

`Standard`

Use the default format.

`logResponses`

When to log service responses (maps to the `Amazon.Util.LoggingConfig.LogResponses` property from the `Amazon.AWSConfigs.LoggingConfig.LogResponses` property in the AWS SDK for .NET).

Allowed values include:

`Always`

Always log service responses.

`Never`

Never log service responses.

`OnError`

Only log service responses when there are errors.

`logTo`

Where to log to (maps to the `LogTo` property from the `Amazon.AWSConfigs.LoggingConfig.LogTo` property in the AWS SDK for .NET).

Allowed values include one or more of:

`Log4Net`

Log to log4net.

None

Disable logging.

SystemDiagnostics

Log to System.Diagnostics.

The parent of the <logging> element is the <aws> element.

The <logging> element contains no child elements.

The following is an example of the <logging> element in use:

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

map

The <map> element represents a single item in a collection of type-to-table mappings from .NET types to DynamoDB tables (maps to an instance of the TypeMapping class from the Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings property in the AWS SDK for .NET). For more information, see Enhancements to the DynamoDB SDK.

This element can include the following attributes:

targetTable

The DynamoDB table to which the mapping applies. This attribute maps to the Amazon.Util.TypeMapping.TargetTable property in the AWS SDK for .NET.

type

The type and assembly name to which the mapping applies. This attribute maps to the Amazon.Util.TypeMapping.Type property in the AWS SDK for .NET.

The parent of the <map> element is the <dynamoDBContext> element.

The <map> element can include one or more instances of the <property> child element.

The following is an example of the <map> element in use:

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

property

The <property> element represents a DynamoDB property. (This element maps to an instance of the Amazon.Util.PropertyConfig class from the AddProperty method in the AWS SDK for .NET) For more information, see Enhancements to the DynamoDB SDK and DynamoDB Attributes.

This element can include the following attributes:

attribute

The name of an attribute for the property, such as the name of a range key. This attribute maps to the Amazon.Util.PropertyConfig.Attribute property in the AWS SDK for .NET.

converter

The type of converter that should be used for this property. This attribute maps to the Amazon.Util.PropertyConfig.Converter property in the AWS SDK for .NET.

ignore

Whether the associated property should be ignored (true); otherwise, false. This attribute maps to the Amazon.Util.PropertyConfig.Ignore property in the AWS SDK for .NET.

name

The name of the property. This attribute maps to the Amazon.Util.PropertyConfig.Name property in the AWS SDK for .NET.

version

Whether this property should store the item version number (true); otherwise, false. This attribute maps to the Amazon.Util.PropertyConfig.Version property in the AWS SDK for .NET.

The parent of the <property> element is the <map> element.

The <property> element contains no child elements.

The following is an example of the <property> element in use:

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

proxy

The <proxy> element represents settings for configuring a proxy for the AWS SDK for .NET to use. This element can include the following attributes:

host

The host name or IP address of the proxy server. This attributes maps to the Amazon.Util.ProxyConfig.Host property from the Amazon.AWSConfigs.ProxyConfig.Host property in the AWS SDK for .NET.

password

The password to authenticate with the proxy server. This attributes maps to the Amazon.Util.ProxyConfig.Password property from the Amazon.AWSConfigs.ProxyConfig.Password property in the AWS SDK for .NET.

port

The port number of the proxy. This attributes maps to the Amazon.Util.ProxyConfig.Port property from the Amazon.AWSConfigs.ProxyConfig.Port property in the AWS SDK for .NET.

username

The user name to authenticate with the proxy server. This attributes maps to the Amazon.Util.ProxyConfig.Username property from the Amazon.AWSConfigs.ProxyConfig.Username property in the AWS SDK for .NET.

The parent of the <proxy> element is the <aws> element.

The <proxy> element contains no child elements.

The following is an example of the <proxy> element in use:

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

s3

The <s3> element represents a collection of Amazon S3 settings. This element can include the *useSignatureVersion4* attribute, which specifies whether signature version 4 signing will be used for all requests (true) or whether signature version 4 signing will not be used for all requests (false, the default). This attribute maps to the `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` property in the AWS SDK for .NET.

The parent of the <s3> element is the <aws> element.

The <s3> element contains no child elements.

The following is an example of the <s3> element in use:

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

Using legacy credentials

The topics in this section provide information about using long-term or short-term credentials without using AWS IAM Identity Center.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

Note

The information in this topic is for circumstances where you need to obtain and manage short-term or long-term credentials manually. For additional information about short-

term and long-term credentials, see Other ways to authenticate in the *AWS SDKs and Tools Reference Guide*.
For best security practices, use AWS IAM Identity Center, as described in Authenticating with AWS.

Important warnings and guidance for credentials

Warnings for credentials

- ***Do NOT*** use your account's root credentials to access AWS resources. These credentials provide unrestricted account access and are difficult to revoke.
- ***Do NOT*** put literal access keys or credential information in your application files. If you do, you create a risk of accidentally exposing your credentials if, for example, you upload the project to a public repository.
- ***Do NOT*** include files that contain credentials in your project area.
- Be aware that any credentials stored in the shared AWS `credentials` file, are stored in plaintext.

Additional guidance for securely managing credentials

For a general discussion of how to securely manage AWS credentials, see AWS security credentials in the AWS General Reference and Security best practices and use cases in the IAM User Guide. In addition to those discussions, consider the following:

- Create additional users, such as users in IAM Identity Center, and use their credentials instead of using your AWS root user credentials. Credentials for other users can be revoked if necessary or are temporary by nature. In addition, you can apply a policy to each user for access to only certain resources and actions and thereby take a stance of least-privilege permissions.

- Use IAM roles for tasks for Amazon Elastic Container Service (Amazon ECS) tasks.

- Use IAM roles for applications that are running on Amazon EC2 instances.

- Use temporary credentials or environment variables for applications that are available to users outside your organization.

Topics

- Using the shared AWS credentials file
- Using the SDK Store (Windows only)

Using the shared AWS credentials file

(Be sure to review the important warnings and guidance for credentials.)

One way to provide credentials for your applications is to create profiles in the *shared AWS credentials file* and then store credentials in those profiles. This file can be used by the other AWS SDKs. It can also be used by the AWS CLI, the AWS Tools for Windows PowerShell, and the AWS toolkits for Visual Studio, JetBrains, and VS Code.

> **Warning**
>
> To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

> **Note**
>
> The information in this topic is for circumstances where you need to obtain and manage short-term or long-term credentials manually. For additional information about short-term and long-term credentials, see Other ways to authenticate in the *AWS SDKs and Tools Reference Guide*.
> For best security practices, use AWS IAM Identity Center, as described in Authenticating with AWS.

General information

By default, the shared AWS credentials file is located in the `.aws` directory within your home directory and is named `credentials`; that is, `~/.aws/credentials` (Linux or macOS) or `%USERPROFILE%\.aws\credentials` (Windows). For information about alternative locations, see Location of the shared files in the *AWS SDKs and Tools Reference Guide*. Also see Accessing credentials and profiles in an application.

The shared AWS credentials file is a plaintext file and follows a certain format. For information about the format of AWS credentials files, see Format of the credentials file in the *AWS SDKs and Tools Reference Guide*.

You can manage the profiles in the shared AWS credentials file in several ways.

- Use any text editor to create and update the shared AWS credentials file.
- Use the Amazon.Runtime.CredentialManagement namespace of the AWS SDK for .NET API, as shown later in this topic.
- Use commands and procedures for the AWS Tools for PowerShell and the AWS toolkits for Visual Studio, JetBrains, and VS Code.
- Use AWS CLI commands; for example, `aws configure set aws_access_key_id` and `aws configure set aws_secret_access_key`.

Examples of profile management

The following sections show examples of profiles in the shared AWS credentials file. Some of the examples show the result, which can be obtained through any of the credential-management methods described earlier. Other examples show how to use a particular method.

The default profile

The shared AWS credentials file will almost always have a profile named *default*. This is where the AWS SDK for .NET looks for credentials if no other profiles are defined.

The `[default]` profile typically looks something like the following.

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

Create a profile programmatically

This example shows you how to create a profile and save it to the shared AWS credentials file programmatically. It uses the following classes of the Amazon.Runtime.CredentialManagement namespace: CredentialProfileOptions, CredentialProfile, and SharedCredentialsFile.

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

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

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

Warning

Code such as this generally shouldn't be in your application. If you include it in your application, take appropriate precautions to ensure that plaintext keys can't possibly be seen in the code, over the network, or even in computer memory.

The following is the profile that's created by this example.

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

Update an existing profile programmatically

This example shows you how to programmatically update the profile that was created earlier. It uses the following classes of the Amazon.Runtime.CredentialManagement namespace: CredentialProfile and SharedCredentialsFile. It also uses the RegionEndpoint class of the Amazon namespace.

```
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);
    }
}
```

The following is the updated profile.

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

Note

You can also set the AWS Region in other locations and by using other methods. For more information, see Setting the AWS Region for the AWS SDK for .NET.

Using the SDK Store (Windows only)

(Be sure to review the important warnings and guidelines.)

On Windows, the *SDK Store* is another place to create profiles and store encrypted credentials for your AWS SDK for .NET application. It's located in `%USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json`. You can use the SDK Store during development as an alternative to the shared AWS credentials file.

Warning

To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

> **Note**
>
> The information in this topic is for circumstances where you need to obtain and manage short-term or long-term credentials manually. For additional information about short-term and long-term credentials, see Other ways to authenticate in the *AWS SDKs and Tools Reference Guide*.
> For best security practices, use AWS IAM Identity Center, as described in Authenticating with AWS.

General information

The SDK Store provides the following benefits:

- The credentials in the SDK Store are encrypted, and the SDK Store resides in the user's home directory. This limits the risk of accidentally exposing your credentials.
- The SDK Store also provides credentials to the AWS Tools for Windows PowerShell and the AWS Toolkit for Visual Studio.

SDK Store profiles are specific to a particular user on a particular host. You can't copy them to other hosts or other users. This means that you can't reuse SDK Store profiles that are on your development machine for other hosts or developer machines. It also means that you can't use SDK Store profiles in production applications.

You can manage the profiles in the SDK Store in the following ways:

- Use the graphical user interface (GUI) in the AWS Toolkit for Visual Studio.
- Use the Amazon.Runtime.CredentialManagement namespace of the AWS SDK for .NET API, as shown later in this topic.
- Use commands from the AWS Tools for Windows PowerShell; for example, `Set-AWSCredential` and `Remove-AWSCredentialProfile`.

Examples of profile management

The following examples show you how to programmatically create and update a profile in the SDK Store.

Create a profile programmatically

This example shows you how to create a profile and save it to the SDK Store programmatically. It uses the following classes of the Amazon.Runtime.CredentialManagement namespace: CredentialProfileOptions, CredentialProfile, and NetSDKCredentialsFile.

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

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

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

Warning

Code such as this generally shouldn't be in your application. If it's included in your application, take appropriate precautions to ensure that plaintext keys can't possibly be seen in the code, over the network, or even in computer memory.

The following is the profile that's created by this example.

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

Update an existing profile programmatically

This example shows you how to programmatically update the profile that was created earlier. It uses the following classes of the Amazon.Runtime.CredentialManagement namespace: CredentialProfile and NetSDKCredentialsFile. It also uses the RegionEndpoint class of the Amazon namespace.

```
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);
    }
}
```

The following is the updated profile.

```
"[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**
>
> You can also set the AWS Region in other locations and by using other methods. For more information, see Setting the AWS Region for the AWS SDK for .NET.

Using the AWS SDK for .NET

This section provides information about features of the AWS SDK for .NET that you might need to consider when creating your applications.

Before you begin, be sure you have set up your environment and configured your project.

For information about developing software for specific AWS services along with code examples, see Calling AWS services. For additional code examples, see SDK for .NET (v4) code examples.

Topics

- Programming asynchronously using the AWS SDK for .NET
- Using paginated results in the AWS SDK for .NET
- Support for HTTP 2 in the AWS SDK for .NET
- Additional tools

Programming asynchronously using the AWS SDK for .NET

The AWS SDK for .NET uses the *Task-based Asynchronous Pattern (TAP)* for its asynchronous implementation. To learn more about the TAP, see Task-based Asynchronous Pattern (TAP) on docs.microsoft.com.

This topic gives you an overview of how to use TAP in your calls to AWS service clients.

The asynchronous methods in the AWS SDK for .NET API are operations based on the `Task` class or the `Task<TResult>` class. See docs.microsoft.com for information about these classes: Task class, Task<TResult> class.

When these API methods are called in your code, they must be called within a function that is declared with the `async` keyword, as shown in the following example.

```
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;
}
```

As shown in the preceding code snippet, the preferred scope for the `async` declaration is the `Main` function. Setting this `async` scope ensures that all calls to AWS service clients are required to be asynchronous. If you can't declare `Main` to be asynchronous for some reason, you can use the `async` keyword on functions other than `Main` and then call the API methods from there, as shown in the following example.

```
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;
}
```

Notice the special `Task<>` syntax that's needed in `Main` when you use this pattern. In addition, you must use the **`Result`** member of the response to get the data.

You can see full examples of asynchronous calls to AWS service clients in the Creating a simple application section (Simple cross-platform app and Simple Windows-based app) and in Guided code examples.

Using paginated results in the AWS SDK for .NET

Some AWS services collect and store a large amount of data, which you can retrieve by using the API calls of the AWS SDK for .NET. If the amount of data you want to retrieve becomes too large for a single API call, you can break the results into more manageable pieces through the use of *pagination*.

To enable you to perform pagination, the request and response objects for many service clients in the SDK provide a *continuation token* (typically named `NextToken`). Some of these service clients also provide paginators.

Paginators enable you to avoid the overhead of the continuation token, which might involve loops, state variables, multiple API calls, and so on. When you use a paginator, you can retrieve data from an AWS service through a single line of code, a `foreach` loop's declaration. If multiple API calls are needed to retrieve the data, the paginator handles this for you.

Where do I find paginators?

Not all services provide paginators. One way to determine whether a service provides a paginator for a particular API is to look at the definition of a service client class in the AWS SDK for .NET API Reference.

For example, if you examine the definition for the AmazonCloudWatchLogsClient class, you see a `Paginators` property. This is the property that provides a paginator for Amazon CloudWatch Logs.

What do paginators give me?

Paginators contain properties that enable you to see full responses. They also typically contain one or more properties that enable to you access the most interesting portions of the responses, which we will call the *key results*.

For example, in the `AmazonCloudWatchLogsClient` mentioned earlier, the `Paginator` object contains a `Responses` property with the full DescribeLogGroupsResponse object from the API call. This `Responses` property contains, among other things, a collection of the log groups.

The Paginator object also contains one key result named `LogGroups`. This property holds just the log groups portion of the response. Having this key result enables you to reduce and simplify your code in many circumstances.

Synchronous vs. asynchronous pagination

Paginators provide both synchronous and asynchronous mechanisms for pagination. Synchronous pagination is available in .NET Framework 4.7.2 (or later) projects. Asynchronous pagination is available in .NET Core projects (.NET Core 3.1, .NET 5, and so on).

Because asynchronous operations and .NET Core are recommended, the example that comes next shows you asynchronous pagination. Information about how to perform the same tasks using synchronous pagination and .NET Framework 4.7.2 (or later) is shown after the example in Additional considerations for paginators.

Example

The following example shows you how to use the AWS SDK for .NET to display a list of log groups. For contrast, the example shows how to do this both with and without paginators. Before looking at the full code, shown later, consider the following snippets.

Getting CloudWatch log groups without paginators

```
      // 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));
```

Getting CloudWatch log groups by using paginators

```
      // 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);
```

```
    }
```

The results of these two snippets are exactly the same, so the advantage in using paginators can clearly be seen.

> **Note**
>
> Before you try to build and run the full code, be sure you have set up your environment and project.
> You might also need the Microsoft.Bcl.AsyncInterfaces NuGet package because asynchronous paginators use the `IAsyncEnumerable` interface.

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.CloudWatch

Programming elements:

- Namespace Amazon.CloudWatch

  Class AmazonCloudWatchLogsClient
- Namespace Amazon.CloudWatchLogs.Model

  Class DescribeLogGroupsRequest

  Class DescribeLogGroupsResponse

  Class LogGroup

Full code

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

```
}
```

Additional considerations for paginators

- **Paginators can't be used more than once**

  If you need the results of a particular AWS paginator in multiple locations in your code, you must not use a paginator object more than once. Instead, create a new paginator each time you need it. This concept is shown in the preceding example code in the `DisplayLogGroupsWithPaginators` method.

- **Synchronous pagination**

  Synchronous pagination is available for .NET Framework 4.7.2 (or later) projects.

  To see this, create a .NET Framework 4.7.2 (or later) project and copy the preceding code to it. Then simply remove the `await` keyword from the two `foreach` paginator calls, as shown in the following example.

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

  Build and run the project to see the same results you saw with asynchronous pagination.

Support for HTTP 2 in the AWS SDK for .NET

Some AWS services and operations require HTTP 2. For example, bi-directional streaming in Amazon Transcribe Streaming isn't possible over HTTP 1.1 and so requires HTTP 2 instead. Version 4 of the AWS SDK for .NET added support for HTTP 2 so that you can use these operations in your applications. For a bi-directional HTTP 2 operation, the behavior of the SDK when receiving streams is similar to that for HTTP 1.1. That is, when applications that use the SDK send events to the service, the request has a publisher that is assigned by the developer.

To see this behavior in action, consider the following example for Amazon Transcribe Streaming. It uses the Amazon.TranscribeStreaming and Amazon.TranscribeStreaming.Model namespaces.

In this example, the developer defines the `StartStreamTranscriptionRequest.AudioStreamPublisher` property with a callback function, which is a .NET `Func`. The SDK uses the `Func` defined for `AudioStreamPublisher` to pull events from the user's code to stream to the user. The SDK calls the `Func` until it returns null.

The code demonstrates how audio from a file can be streamed into Amazon Transcribe Streaming for processing.

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

CancellationTokenSource cancelSource = new CancellationTokenSource();

var client = new AmazonTranscribeStreamingClient(RegionEndpoint.USEast1);

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

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

    if (bytesRead == 0)
        return null;

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

    return audioEvent;
};

using var response = await client.StartStreamTranscriptionAsync(startRequest);
```

```
Console.WriteLine(response.HttpStatusCode);

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

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

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

_ = response.TranscriptResultStream.StartProcessingAsync();

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

Warning

Some bi-directional HTTP 2 operations, such as the `InvokeModelWithBidirectionalStreamAsync` method from Amazon Bedrock, the

Amazon.BedrockRuntime namespace, don't return a response from the initial invocation on the service client until some events have been published. This behavior could cause your application to be blocked. To avoid this situation, separate the application code that provides events to the publisher, and run it on a thread that is different from the thread that invokes operations on the service client.

Additional considerations

- AWS SDK for .NET support for HTTP 2 is available only in versions that target .NET 8 and above. It isn't available in versions that target .NET Framework.
- For more detailed information, see PR 3730 in the aws-sdk-net GitHub repository.

Additional tools

The following are some additional tools that you can use to ease the work of developing, deploying, and maintaining your .NET applications.

AWS Deploy Tool

After you've developed your cloud-native .NET Core application on a development machine, you can use the AWS Deploy Tool for the .NET CLI to more easily deploy your application to AWS.

For more information, see Deploy applications to AWS.

AWS Message Processing Framework for .NET

If you're using services such as Amazon SQS, Amazon SNS or Amazon EventBridge, you might be able to take advantage of the AWS Message Processing Framework for .NET. For more information, see AWS Message Processing Framework for .NET.

Integrations with .NET Aspire

You can take advantages of integrations with .NET Aspire to improve the inner dev loop. For more information, see Integrating AWS with .NET Aspire in the AWS SDK for .NET.

Advanced authentication and authorization with the AWS SDK for .NET

The topics in this section provide information about advanced techniques for authentication and authorization in your AWS SDK for .NET application.

Topics

- Single sign-on with the AWS SDK for .NET

Single sign-on with the AWS SDK for .NET

AWS IAM Identity Center is a cloud-based single sign-on (SSO) service that makes it easy to centrally manage SSO access to all of your AWS accounts and cloud applications. For full details, see the IAM Identity Center User Guide.

If you're unfamiliar with how an SDK interacts with IAM Identity Center, see the following information.

High-level pattern of interaction

At a high level, SDKs interact with IAM Identity Center in a manner similar to the following pattern:

1. IAM Identity Center is configured, typically through the IAM Identity Center console, and an SSO user is invited to participate.
2. The shared AWS `config` file on the user's computer is updated with SSO information.
3. The user signs in through IAM Identity Center and is given short-term credentials for the AWS Identity and Access Management (IAM) permissions that have been configured for them. This sign-in can be initiated through a non-SDK tool like the AWS CLI, or programmatically through a .NET application.
4. The user proceeds to do their work. When they run other applications that are using SSO, they don't need to sign in again to open the applications.

The rest of this topic provides reference information for setting up and using AWS IAM Identity Center. It provides supplemental and more advanced information than the basic SSO setup in Authenticating with AWS. If you're new to SSO on AWS, you might want to look at that topic first for fundamental information, and then at the following tutorials to see SSO in action:

- Tutorial: .NET application only
- Tutorial: AWS CLI and .NET application

This topic contains the following sections:

- Prerequisites
- Setting up an SSO profile
- Generating and using SSO tokens
- Additional resources
- Tutorials

Prerequisites

Before using IAM Identity Center, you must perform certain tasks, such as choosing an identity source and configuring the relevant AWS accounts and applications. For additional information, see the following:

- For general information about these tasks, see Getting started in the *IAM Identity Center User Guide*.
- For specific task examples, see the list of tutorials at the end of this topic. However, be sure to review the information in this topic before trying out the tutorials.

Setting up an SSO profile

After IAM Identity Center is configured in the relevant AWS account, a named profile for SSO must be added to the user's shared AWS `config` file. This profile is used to connect to the *AWS access portal*, which returns short-term credentials for the IAM permissions that have been configured for the user.

The shared `config` file is typically named `%USERPROFILE%\.aws\config` in Windows and `~/.aws/config` in Linux and macOS. You can use your preferred text editor to add a new profile for SSO. Alternatively, you can use the `aws configure sso` command. For more information about this command, see Configuring the AWS CLI to use IAM Identity Center in the *AWS Command Line Interface User Guide*.

The new profile is similar to the following:

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

The settings for the new profile are defined below. The first two settings define the AWS access portal. The other two settings are a pair that, taken together, define the permissions that have been configured for a user. All four settings are required.

`sso_start_url`

The URL that points to the organization's AWS access portal. To find this value, open the IAM Identity Center console, choose **Settings**, and find **portal URL**.

`sso_region`

The AWS Region that contains the access portal host. This is the Region that was selected as you enabled IAM Identity Center. It can be different from the Regions that you use for other tasks.

For a complete list of the AWS Regions and their codes, see Regional Endpoints in the *Amazon Web Services General Reference*.

`sso_account_id`

The ID of an AWS account that was added through the AWS Organizations service. To see the list of available accounts, go to the IAM Identity Center console and open the **AWS accounts** page. The account ID that you choose for this setting will correspond to the value that you plan to give to the `sso_role_name` setting, which is shown next.

`sso_role_name`

The name of an IAM Identity Center permission set. This permission set defines the permissions that a user is given through IAM Identity Center.

The following procedure is one way to find the value for this setting.

1. Go to the IAM Identity Center console and open the **AWS accounts** page.
2. Choose an account to display its details. The account you choose will be the one that contains the SSO user or group that you want to give SSO permissions to.

3. Look at the list of users and groups that are assigned to the account and find the user or group of interest. The permission set that you specify in the `sso_role_name` setting is one of the sets associated with this user or group.

When giving a value to this setting, use the name of the permission set, not the Amazon Resource Name (ARN).

Permission sets have IAM policies and custom-permissions policies attached to them. For more information, see Permission sets in the *IAM Identity Center User Guide*.

Generating and using SSO tokens

To use SSO, a user must first generate a temporary token and then use that token to access appropriate AWS applications and resources. For .NET applications, you can use the following methods to generate and use these temporary tokens:

- Create .NET applications that generate a token first, if necessary, and then use the token.
- Generate a token with the AWS CLI and then use the token in .NET applications.

These methods are described in the following sections and demonstrated in the tutorials.

> **Important**
>
> Your application must reference the following NuGet packages so that SSO resolution can work:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> Failure to reference these packages will result in a *runtime* exception.

.NET application only

This section shows you how to create a .NET application that generates a temporary SSO token, if necessary, and then uses that token. For a full tutorial of this process, see Tutorial for SSO using only .NET applications.

Generate and use an SSO token programmatically

In addition to using the AWS CLI, you can also generate an SSO token programmatically.

To do this, your application creates an `AWSCredentials` object for the SSO profile, which loads temporary credentials if any are available. Then, your application must cast the `AWSCredentials` object to an `SSOAWSCredentials` object and set some Options properties, including a callback method that is used to prompt the user for sign-in information, if necessary.

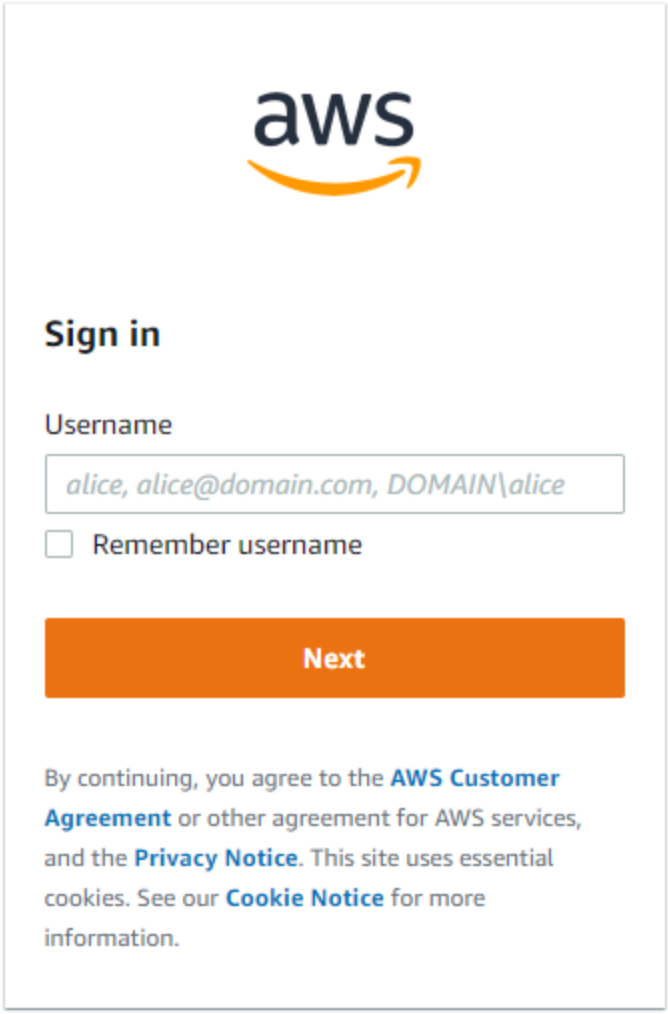
This method is shown in the following code snippet.

> **Important**
>
> Your application must reference the following NuGet packages so that SSO resolution can work:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> Failure to reference these packages will result in a *runtime* exception.

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

    var ssoCredentials = credentials as SSOAWSCredentials;

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

```
        Process.Start(new ProcessStartInfo
        {
            FileName = args.VerificationUriComplete,
            UseShellExecute = true
        });
    };
    ssoCredentials.Options.SupportsGettingNewToken = true;

    return ssoCredentials;
}
```

If an appropriate SSO token isn't available, the default browser window is launched and the appropriate sign-in page is opened. For example, if you're using IAM Identity Center as the **Identity source**, the user sees a sign-in page similar to the following:



> **Note**
>
> The text string that you provide for `SSOAWSCredentials.Options.ClientName` can't have spaces. If the string does have spaces, you'll get a *runtime* exception.

Tutorial for SSO using only .NET applications

AWS CLI and .NET application

This section shows you how to generate a temporary SSO token by using the AWS CLI, and how to use that token in an application. For a full tutorial of this process, see Tutorial for SSO using the AWS CLI and .NET applications.

Generate an SSO token by using the AWS CLI

In addition to generating a temporary SSO token programmatically, you use the AWS CLI to generate the token. The following information shows you how.

After the user creates an SSO-enabled profile as shown in a previous section, they run the `aws sso login` command from the AWS CLI. They must be sure to include the `--profile` parameter with the name of the SSO-enabled profile. This is shown in the following example:

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

If the user wants to generate a new temporary token after the current one expires, they can run the same command again.

Use the generated SSO token in a .NET application

The following information shows you how to use a temporary token that has already been generated.

> **Important**
>
> Your application must reference the following NuGet packages so that SSO resolution can work:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`

Failure to reference these packages will result in a *runtime* exception.

Your application creates an `AWSCredentials` object for the SSO profile, which loads the temporary credentials generated earlier by the AWS CLI. This is similar to the methods shown in Accessing credentials and profiles in an application and has the following form:

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

    return credentials;
}
```

The `AWSCredentials` object is then passed to the constructor for a service client. For example:

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

Note

Using `AWSCredentials` to load temporary credentials isn't necessary if your application has been built to use the `[default]` profile for SSO. In that case, the application can create AWS service clients without parameters, similar to "`var client = new AmazonS3Client();`".

Tutorial for SSO using the AWS CLI and .NET applications

Additional resources

For additional help, see the following resources.

- What is IAM Identity Center?
- Configuring the AWS CLI to use IAM Identity Center
- Using IAM Identity Center credentials in the AWS Toolkit for Visual Studio

Tutorials

Topics

- Tutorial for SSO using only .NET applications
- Tutorial for SSO using the AWS CLI and .NET applications

Tutorial for SSO using only .NET applications

This tutorial shows you how to enable SSO for a basic application and a test SSO user. It configures the application to generate a temporary SSO token programmatically instead of using the AWS CLI.

This tutorial shows you a small portion of SSO functionality in the AWS SDK for .NET. For full details about using IAM Identity Center with the AWS SDK for .NET, see the topic with background information. In that topic, see especially the high-level description for this scenario in the subsection called .NET application only.

> **Note**
>
> Several of the steps in this tutorial help you configure services like AWS Organizations and IAM Identity Center. If you've already performed that configuration, or if you're only interested in the code, you can skip to the section with the example code.

Prerequisites

- Configure your development environment if you haven't already done so. This is described in sections like Installing and configuring your toolchain for the AWS SDK for .NET and Getting started.
- Identify or create at least one AWS account that you can use to test SSO. For the purposes of this tutorial, this is called the *test AWS account* or simply *test account*.
- Identify an *SSO user* who can test SSO for you. This is a person who will be using SSO and the basic applications that you create. For this tutorial, that person might be you (the developer), or someone else. We also recommend a setup in which the SSO user is working on a computer that is not in your development environment. However, this isn't strictly necessary.
- The SSO user's computer must have a .NET framework installed that's compatible with the one you used to set up your development environment.

Set up AWS

This section shows you how to set up various AWS services for this tutorial.

To perform this setup, first sign in to the test AWS account as an administrator. Then, do the following:

Amazon S3

Go to the Amazon S3 console and add some innocuous buckets. Later in this tutorial, the SSO user will retrieve a list of these buckets.

AWS IAM

Go to the IAM console and add a few IAM users. If you give the IAM users permissions, limit the permissions to a few innocuous read-only permissions. Later in this tutorial, the SSO user will retrieve a list of these IAM users.

AWS Organizations

Go to the AWS Organizations console and enable Organizations. For more information, see Creating an organization in the AWS Organizations User Guide.

This action adds the test AWS account to the organization as the *management account*. If you have additional test accounts, you can invite them to join the organization, but doing so isn't necessary for this tutorial.

IAM Identity Center

Go to the IAM Identity Center console and enable SSO. Perform email verification if necessary. For more information, see Enable IAM Identity Center in the IAM Identity Center User Guide.

Then, perform the following configuration.

Configure IAM Identity Center

1. Go to the **Settings** page. Look for the **"access portal URL"** and record the value for later use in the `sso_start_url` setting.
2. In the banner of the AWS Management Console, look for the AWS Region that was set when you enabled SSO. This is the dropdown menu to the left of the AWS account ID. Record the Region code for later use in the `sso_region` setting. This code will be similar to `us-east-1`.
3. Create an SSO user as follows:

a. Go to the **Users** page.

b. Choose **Add user** and enter the user's **Username**, **Email address**, **First name**, and **Last name**. Then, choose **Next**.

c. Choose **Next** on the page for groups, then review the information and choose **Add user**.

4. Create a group as follows:

a. Go to the **Groups** page.

b. Choose **Create group** and enter the group's **Group name** and **Description**.

c. In the **Add users to group** section, select the test SSO user that you created earlier. Then, select **Create group.**

5. Create a permission set as follows:

a. Go to the **Permission sets** page and choose **Create permission set**.

b. Under **Permission set type**, select **Custom permission set** and choose **Next**.

c. Open **Inline policy** and enter the following policy:

JSON

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

d. For this tutorial, enter `SSOReadOnlyRole` as the **Permission set name**. Add a **Description** if you want and then choose **Next**.

e. Review the information and then choose **Create**.

f. Record the name of the permission set for later use in the `sso_role_name` setting.

6. Go to the **AWS accounts** page and choose the AWS account that you added to the organization earlier.
7. In the **Overview** section of that page, find the **Account ID** and record it for later use in the `sso_account_id` setting.
8. Choose the **Users and groups** tab and then choose **Assign users or groups**.
9. On the **Assign users and groups** page, choose the **Groups** tab, select the group that you created earlier, and choose **Next**.
10. Select the permission set that you created earlier and choose **Next**, then choose **Submit**. The configuration takes a few moments.

Create example applications

Create the following applications. They will be run on the SSO user's computer.

List Amazon S3 buckets

Include NuGet packages `AWSSDK.SSO` and `AWSSDK.SSOOIDC` in addition to `AWSSDK.S3` and `AWSSDK.SecurityToken`.

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


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

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

        // Class members.
```

```
        private static string profile = "my-sso-profile";

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

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

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

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

            var ssoCredentials = credentials as SSOAWSCredentials;

            ssoCredentials.Options.ClientName = "Example-SSO-App";
            ssoCredentials.Options.SsoVerificationCallback = args =>
            {
                // Launch a browser window that prompts the SSO user to complete an SSO
login.
                //  This method is only invoked if the session doesn't already have a
valid SSO token.
                // NOTE: Process.Start might not support launching a browser on macOS
or Linux. If not,
```

```
                //       use an appropriate mechanism on those systems instead.
                Process.Start(new ProcessStartInfo
                {
                    FileName = args.VerificationUriComplete,
                    UseShellExecute = true
                });
            };
            ssoCredentials.Options.SupportsGettingNewToken = true;

            return ssoCredentials;
        }

    }

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

List IAM users

Include NuGet packages `AWSSDK.SSO` and `AWSSDK.SSOOIDC` in addition to `AWSSDK.IdentityManagement` and `AWSSDK.SecurityToken`.

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


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

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

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

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

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

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

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

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

```
            var ssoCredentials = credentials as SSOAWSCredentials;

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

            return ssoCredentials;
        }

    }

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

In addition to displaying lists of Amazon S3 buckets and IAM users, these applications display the user identity ARN for the SSO-enabled profile, which is `my-sso-profile` in this tutorial.

These applications perform SSO sign-in tasks by providing a callback method in the Options property of an SSOAWSCredentials object.

Instruct SSO user

Ask the SSO user to check their email and accept the SSO invitation. They are prompted to set a password. The message might take a few minutes to arrive in the SSO user's inbox.

Give the SSO user the applications that you created earlier.

Then, have the SSO user do the following:

1. If the folder that contains the shared AWS `config` file doesn't exist, create it. If the folder does exist and has a subfolder called `.sso`, delete that subfolder.

   The location of this folder is typically `%USERPROFILE%\.aws` in Windows and `~/.aws` in Linux and macOS.

2. Create a shared AWS `config` file in that folder, if necessary, and add a profile to it as follows:

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

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

3. Run the Amazon S3 application.
4. In the resulting web sign-in page, sign in. Use the user name from the invitation message and the password that was created in response to the message.
5. When sign-in is complete, the application displays the list of S3 buckets.
6. Run the IAM application. The application displays the list of IAM users. This is true even though a second sign-in wasn't performed. The IAM application uses the temporary token that was created earlier.

Cleanup

If you don't want to keep the resources that you created during this tutorial, clean them up. These might be AWS resources or resources in your development environment such as files and folders.

Tutorial for SSO using the AWS CLI and .NET applications

This tutorial shows you how to enable SSO for a basic .NET application and a test SSO user. It uses the AWS CLI to generate a temporary SSO token instead of generating it programmatically.

This tutorial shows you a small portion of SSO functionality in the AWS SDK for .NET. For full details about using IAM Identity Center with the AWS SDK for .NET, see the topic with background information. In that topic, see especially the high-level description for this scenario in the subsection called AWS CLI and .NET application.

> **Note**
>
> Several of the steps in this tutorial help you configure services like AWS Organizations and IAM Identity Center. If you've already performed those configurations, or if you're only interested in the code, you can skip to the section with the example code.

Prerequisites

- Configure your development environment if you haven't already done so. This is described in sections like Installing and configuring your toolchain for the AWS SDK for .NET and Getting started.
- Identify or create at least one AWS account that you can use to test SSO. For the purposes of this tutorial, this is called the *test AWS account* or simply *test account*.
- Identify an *SSO user* who can test SSO for you. This is a person who will be using SSO and the basic applications that you create. For this tutorial, that person might be you (the developer), or someone else. We also recommend a setup in which the SSO user is working on a computer that is not in your development environment. However, this isn't strictly necessary.
- The SSO user's computer must have a .NET framework installed that's compatible with the one you used to set up your development environment.

- Be sure that the AWS CLI version 2 is installed on the SSO user's computer. You can check this by running `aws --version` in a command prompt or terminal.

Set up AWS

This section shows you how to set up various AWS services for this tutorial.

To perform this setup, first sign in to the test AWS account as an administrator. Then, do the following:

Amazon S3

Go to the Amazon S3 console and add some innocuous buckets. Later in this tutorial, the SSO user will retrieve a list of these buckets.

AWS IAM

Go to the IAM console and add a few IAM users. If you give the IAM users permissions, limit the permissions to a few innocuous read-only permissions. Later in this tutorial, the SSO user will retrieve a list of these IAM users.

AWS Organizations

Go to the AWS Organizations console and enable Organizations. For more information, see Creating an organization in the AWS Organizations User Guide.

This action adds the test AWS account to the organization as the *management account*. If you have additional test accounts, you can invite them to join the organization, but doing so isn't necessary for this tutorial.

IAM Identity Center

Go to the IAM Identity Center console and enable SSO. Perform email verification if necessary. For more information, see Enable IAM Identity Center in the IAM Identity Center User Guide.

Then, perform the following configuration.

Configure IAM Identity Center

1. Go to the **Settings** page. Look for the **"access portal URL"** and record the value for later use in the `sso_start_url` setting.
2. In the banner of the AWS Management Console, look for the AWS Region that was set when you enabled SSO. This is the dropdown menu to the left of the AWS account ID. Record the Region code for later use in the `sso_region` setting. This code will be similar to `us-east-1`.
3. Create an SSO user as follows:

    a. Go to the **Users** page.

    b. Choose **Add user** and enter the user's **Username**, **Email address**, **First name**, and **Last name**. Then, choose **Next**.

c. Choose **Next** on the page for groups, then review the information and choose **Add user**.

4. Create a group as follows:

   a. Go to the **Groups** page.

   b. Choose **Create group** and enter the group's **Group name** and **Description**.

   c. In the **Add users to group** section, select the test SSO user that you created earlier. Then, select **Create group.**

5. Create a permission set as follows:

   a. Go to the **Permission sets** page and choose **Create permission set**.

   b. Under **Permission set type**, select **Custom permission set** and choose **Next**.

   c. Open **Inline policy** and enter the following policy:

      JSON

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

   d. For this tutorial, enter `SSOReadOnlyRole` as the **Permission set name**. Add a **Description** if you want and then choose **Next**.

   e. Review the information and then choose **Create**.

   f. Record the name of the permission set for later use in the `sso_role_name` setting.

6. Go to the **AWS accounts** page and choose the AWS account that you added to the organization earlier.

7. In the **Overview** section of that page, find the **Account ID** and record it for later use in the `sso_account_id` setting.
8. Choose the **Users and groups** tab and then choose **Assign users or groups**.
9. On the **Assign users and groups** page, choose the **Groups** tab, select the group that you created earlier, and choose **Next**.
10. Select the permission set that you created earlier and choose **Next**, then choose **Submit**. The configuration takes a few moments.

Create example applications

Create the following applications. They will be run on the SSO user's computer.

List Amazon S3 buckets

Include NuGet packages `AWSSDK.SSO` and `AWSSDK.SSOOIDC` in addition to `AWSSDK.S3` and `AWSSDK.SecurityToken`.

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

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

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

        // Class members.
```

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

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

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

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

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

```
        }
    }
}
```

List IAM users

Include NuGet packages `AWSSDK.SSO` and `AWSSDK.SSOOIDC` in addition to `AWSSDK.IdentityManagement` and `AWSSDK.SecurityToken`.

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

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

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

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

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

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

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

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

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

In addition to displaying lists of Amazon S3 buckets and IAM users, these applications display the user identity ARN for the SSO-enabled profile, which is `my-sso-profile` in this tutorial.

Instruct SSO user

Ask the SSO user to check their email and accept the SSO invitation. They are prompted to set a password. The message might take a few minutes to arrive in the SSO user's inbox.

Give the SSO user the applications that you created earlier.

Then, have the SSO user do the following:

1. If the folder that contains the shared AWS `config` file doesn't exist, create it. If the folder does exist and has a subfolder called `.sso`, delete that subfolder.

   The location of this folder is typically `%USERPROFILE%\.aws` in Windows and `~/.aws` in Linux and macOS.

2. Create a shared AWS `config` file in that folder, if necessary, and add a profile to it as follows:

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

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

3. Run the Amazon S3 application. A runtime exception appears.

4. Run the following AWS CLI command:

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

5. In the resulting web sign-in page, sign in. Use the user name from the invitation message and the password that was created in response to the message.

6. Run the Amazon S3 application again. The application now displays the list of S3 buckets.

7. Run the IAM application. The application displays the list of IAM users. This is true even though a second sign-in wasn't performed. The IAM application uses the temporary token that was created earlier.

Cleanup

If you don't want to keep the resources that you created during this tutorial, clean them up. These might be AWS resources or resources in your development environment such as files and folders.

Deploy applications to AWS

After you've developed your cloud-native .NET Core application or service on a development machine, you'll want to deploy it to AWS. You can do this by using the AWS Management Console or certain services like CloudFormation or AWS Cloud Development Kit (AWS CDK). You can also use AWS tools that have been created for the purpose of deployment. By using these tools, you can do the following.

Deploy from the .NET CLI

You can use the following AWS tools for .NET CLI to deploy your applications to AWS:

- AWS Deploy Tool for .NET CLI - Supports deployments to AWS App Runner, Amazon Elastic Container Service (Amazon ECS) , and AWS Elastic Beanstalk.
- AWS Lambda Tools for .NET CLI - Supports deployments of AWS Lambda projects.

Deploy from the IDE toolkits

You can use AWS toolkits to deploy your applications directly from the IDE of your choice:

- **AWS Toolkit for Visual Studio**

  > **Note**
  >
  > The "Publish to AWS" feature in the toolkit exposes the same functionality as the AWS Deploy Tool for .NET CLI. To learn more, go to Publish to AWS in the *AWS Toolkit for Visual Studio User Guide*.

- **AWS Toolkit for JetBrains**

  See Work with AWS Serverless Applications and Work with AWS App Runner.

- **AWS Toolkit for VS Code**

  See Working with serverless applications and Using AWS App Runner.

- **AWS Toolkit for Azure DevOps**

Use cases

The following sections contain use case scenarios for certain types of applications, including information about how you would use the .NET CLI to deploy those applications.

- ASP.NET Core apps
- .NET Console apps
- Blazor WebAssembly apps
- AWS Lambda projects

ASP.NET Core apps

The AWS Deploy Tool for the .NET CLI helps you deploy your ASP.NET applications and guides you through a deployment process. It's an interactive tooling for the .NET CLI that helps deploy .NET applications with minimum AWS knowledge.

The Deploy Tool has the following capabilities:

- **Compute recommendations for your application** - Get the compute recommendations and learn which AWS compute is best suited for your application.
- **Dockerfile generation** - The tool generates a Dockerfile if needed, or uses an existing Dockerfile.
- **Auto packaging and deployment** – The tool builds the deployment artifacts, provisions the infrastructure by using a generated AWS CDK deployment project, and deploys your application to the chosen AWS compute.
- **Repeatable and shareable deployments** – You can generate and modify AWS CDK deployment projects to fit your specific use case. You can also version control your projects and share them with your team for repeatable deployments.
- **Help with learning AWS CDK for .NET** - The tool helps you gradually learn the underlying AWS tools that it is built on, such as the AWS CDK.

The AWS Deploy Tool supports deploying ASP.NET Core applications to the following AWS services:

- **Amazon ECS Service using AWS Fargate** - Supports deployments of web applications to Amazon Elastic Container Service (Amazon ECS) with compute power managed by an AWS Fargate serverless compute engine.

- **AWS App Runner** - Supports deployments to a fully managed service that makes it easy for developers to deploy containerized web applications and APIs at scale. No prior infrastructure experience is required.
- **AWS Elastic Beanstalk** - Supports deployments to a service that makes it easy for developers to deploy web applications and APIs to a fully managed environment at scale. No prior infrastructure experience is required.

To learn more, see the tool overview. To get started from there, navigate to **Documentation**, **Getting started**, and choose **How to install** for installation instructions.

.NET Console apps

The AWS Deploy Tool for the .NET CLI helps you deploy your .NET Console applications as a service or a scheduled task as a container image on Linux and guides you through a deployment process. If your application doesn't have a Dockerfile, the tool automatically generates it. Otherwise, an existing Dockerfile is used.

The Deploy Tool has the following capabilities:

- **Compute recommendations for your application** - Get the compute recommendations and learn which AWS compute is best suited for your application.
- **Dockerfile generation** - The tool generates a Dockerfile if needed, or uses an existing Dockerfile.
- **Auto packaging and deployment** – The tool builds the deployment artifacts, provisions the infrastructure by using a generated AWS CDK deployment project, and deploys your application to the chosen AWS compute.
- **Repeatable and shareable deployments** – You can generate and modify AWS CDK deployment projects to fit your specific use case. You can also version control your projects and share them with your team for repeatable deployments.
- **Help with learning AWS CDK for .NET** - The tool helps you gradually learn the underlying AWS tools that it is built on, such as the AWS CDK.

The AWS Deploy Tool supports deploying .NET Console applications to the following AWS services:

- **Amazon ECS Service using AWS Fargate** - Supports deployments of .NET applications as a service (for example, a background processor) to Amazon Elastic Container Service (Amazon ECS) with compute power managed by AWS Fargate serverless compute engine.

- **Amazon ECS Scheduled Task using AWS Fargate** - Supports deployments of .NET applications as a scheduled task (for example, end-of-day process) to Amazon ECS with compute power managed by AWS Fargate serverless compute engine.

To learn more, see the tool overview. To get started from there, navigate to **Documentation**, **Getting started**, and choose **How to install** for installation instructions.

Blazor WebAssembly apps

The AWS Deploy Tool for the .NET CLI helps you host your Blazor WebAssembly application in Amazon S3, using Amazon CloudFront for content network delivery. Your app is deployed to an S3 bucket for web hosting. The tool creates and configures an S3 bucket, and then uploads your Blazor application to the bucket.

The Deploy Tool has the following capabilities:

- **Auto packaging and deployment** – The tool builds the deployment artifacts, provisions the infrastructure by using a generated AWS CDK deployment project, and deploys your application to the chosen AWS compute.
- **Repeatable and shareable deployments** – You can generate and modify AWS CDK deployment projects to fit your specific use case. You can also version control your projects and share them with your team for repeatable deployments.
- **Help with learning AWS CDK for .NET** - The tool helps you gradually learn the underlying AWS tools that it is built on, such as the AWS CDK.

To learn more, see the tool overview. To get started from there, navigate to **Documentation**, **Getting started**, and choose **How to install** for installation instructions.

AWS Lambda projects

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. It runs your code on a high availability compute infrastructure and performs all of the administration of the compute resources. For more information about Lambda, see What is AWS Lambda? in the *AWS Lambda Developer Guide*.

You can deploy Lambda functions by using the .NET command line interface (CLI).

Topics

- Prerequisites
- Available Lambda commands
- Steps to deploy

Prerequisites

Before you start using the .NET CLI to deploy Lambda functions, you must meet the following prerequisites:

- Confirm that you have the .NET CLI installed. For example: `dotnet --version`. If needed, go to https://dotnet.microsoft.com/download to install it.
- Set up the .NET CLI to work with Lambda. For a description of how to do so, see .NET Core CLI in the *AWS Lambda Developer Guide*. In that procedure, the following is the deployment command:

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

  If you're not sure how to create an IAM role for this exercise, don't include the `--function-role role` part. The tool will help you create a new role.

Available Lambda commands

To list the Lambda commands that are available through the .NET CLI, open a command prompt or terminal and enter `dotnet lambda --help`. The command output will be similar to the following:

```
Amazon Lambda Tools for .NET applications
Project Home: https://github.com/aws/aws-extensions-for-dotnet-cli, https://github.com/
aws/aws-lambda-dotnet

Commands to deploy and manage AWS Lambda functions:

        deploy-function         Command to deploy the project to AWS Lambda
        ...
        (etc.)

To get help on individual commands execute:
```

```
        dotnet lambda help <command>
```

The output lists all the commands that are currently available.

Steps to deploy

The following instructions assume that you've created an AWS Lambda .NET project. For the purposes of this procedure, the project is named `DotNetCoreLambdaTest`.

1. Open a command prompt or terminal, and navigate to the folder containing your .NET Lambda project file.
2. Enter `dotnet lambda deploy-function`.
3. If prompted, enter the AWS Region (the Region to which your Lambda function will be deployed).
4. When prompted, enter the name of the function to deploy, for example, `DotNetCoreLambdaTest`. It can be the name of a function that already exists in your AWS account or one that hasn't been deployed there yet.
5. When prompted, select or create the IAM role that Lambda will assume when executing the function.

After successful completion, the message **New Lambda function created** is displayed.

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

If you deploy a function that already exists in your account, the deploy function asks only for the AWS Region (if necessary). In this case, the command output ends with `Updating code for existing function`.

After your Lambda function is deployed, it's ready to use. For more information, see Examples of How to Use AWS Lambda.

Lambda automatically monitors Lambda functions for you and reports metrics through Amazon CloudWatch. To monitor and troubleshoot your Lambda function, see Monitoring and troubleshooting Lambda applications.

Calling AWS services from the AWS SDK for .NET

The following sections contain examples, tutorials, tasks, and guides that show you how to use the AWS SDK for .NET to work with AWS services. These examples and tutorials rely on an API that the AWS SDK for .NET provides. To see what classes and methods are available in the API, see the AWS SDK for .NET API Reference.

If you're new to the AWS SDK for .NET, you might want to check out the Creating a simple application topic first. It gives you an introduction to the SDK.

You can find more code examples in the AWS Code Examples Repository and the awslabs repository on GitHub.

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

Topics

- Guided code examples for the AWS SDK for .NET
- Using AWS Lambda for compute service
- High-level libraries and frameworks for the AWS SDK for .NET
- Support for other AWS services and configuration

Guided code examples for the AWS SDK for .NET

The following sections contain code examples and provide guidance for the examples. They can help you learn how to use the AWS SDK for .NET to work with AWS services. For additional code examples, see Code examples.

> **Note**
>
> Code examples that are specific to V3 of the AWS SDK for .NET are also available. To find them, see Code examples with guidance and Code examples in the AWS SDK for .NET (V3) Developer Guide. However, be aware that if you use the V3-specific code examples with V4 of the SDK (the latest version), you might need to make adjustments according to the information in Migrating to version 4.

If you're new to the AWS SDK for .NET, you might want to check out the Creating a simple application topic first. It gives you an introduction to the SDK.

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

Topics

- Accessing CloudFormation with the AWS SDK for .NET
- Authenticating users with Amazon Cognito
- Using Amazon DynamoDB NoSQL databases
- Working with Amazon EC2
- Accessing AWS Identity and Access Management (IAM) with the AWS SDK for .NET
- Using Amazon Simple Storage Service Internet storage
- Sending Notifications From the Cloud Using Amazon Simple Notification Service
- Messaging using Amazon SQS

Accessing CloudFormation with the AWS SDK for .NET

The AWS SDK for .NET supports AWS CloudFormation, which creates and provisions AWS infrastructure deployments predictably and repeatedly.

APIs

The AWS SDK for .NET provides APIs for CloudFormation clients. The APIs enable you to work with CloudFormation features such as templates and stacks. This section contains a small number of examples that show you the patterns you can follow when working with these APIs. To view the full set of APIs, see the AWS SDK for .NET API Reference (and scroll to "Amazon.CloudFormation").

The AWS CloudFormation APIs are provided by the AWSSDK.CloudFormation package.

Prerequisites

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

Topics

Topics

- Listing AWS resources using AWS CloudFormation

Listing AWS resources using AWS CloudFormation

This example shows you how to use the AWS SDK for .NET to list the resources in CloudFormation stacks. The example uses the low-level API. The application takes no arguments, but simply gathers information for all stacks that are accessible to the user's credentials and then displays information about those stacks.

SDK references

NuGet packages:

- AWSSDK.CloudFormation

Programming elements:

- Namespace Amazon.CloudFormation

  Class AmazonCloudFormationClient
- Namespace Amazon.CloudFormation.Model

  Class ICloudFormationPaginatorFactory.DescribeStacks

  Class DescribeStackResourcesRequest

  Class DescribeStackResourcesResponse

  Class Stack

  Class StackResource

  Class Tag

```
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;

namespace CloudFormationActions;
```

```
public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
                    new DescribeStacksRequest());
            await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
            {
                // Basic information for each stack

 Console.WriteLine("\n-------------------------------------------------");
                Console.WriteLine($"\nStack: {stack.StackName}");
                Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                Console.WriteLine($"  Created: {stack.CreationTime}");

                // The tags of each stack (etc.)
                if (stack.Tags.Count > 0)
                {
                    Console.WriteLine("  Tags:");
                    foreach (Tag tag in stack.Tags)
                        Console.WriteLine($"    {tag.Key}, {tag.Value}");
```

```
                }

                // The resources of each stack
                DescribeStackResourcesResponse responseDescribeResources =
                    await _amazonCloudFormation.DescribeStackResourcesAsync(
                        new DescribeStackResourcesRequest
                        {
                            StackName = stack.StackName
                        });
                if (responseDescribeResources.StackResources.Count > 0)
                {
                    Console.WriteLine("  Resources:");
                    foreach (StackResource resource in responseDescribeResources
                                 .StackResources)
                        Console.WriteLine(
                            $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                }
            }

            Console.WriteLine("\n---------------------------------------------------");
            return true;
        }
        catch (AmazonCloudFormationException ex)
        {
            Console.WriteLine("Unable to get stack information:\n" + ex.Message);
            return false;
        }
        catch (AmazonServiceException ex)
        {
            if (ex.Message.Contains("Unable to get IAM security credentials"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are usnig SSO, be sure to install" +
                                  " the AWSSDK.SSO and AWSSDK.SSOOIDC packages.");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
```

```
        catch (ArgumentNullException ex)
        {
            if (ex.Message.Contains("Options property cannot be empty: ClientName"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are using SSO, have you logged in?");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
    }
```

Authenticating users with Amazon Cognito

> **Note**
>
> The information in this topic is specific to projects based on .NET Framework and the AWS SDK for .NET version 3.3 and earlier.

Using Amazon Cognito Identity, you can create unique identities for your users and authenticate them for secure access to your AWS resources such as Amazon S3 or Amazon DynamoDB. Amazon Cognito Identity supports public identity providers such as Amazon, Facebook, Twitter/Digits, Google, or any OpenID Connect-compatible provider as well as unauthenticated identities. Amazon Cognito also supports developer authenticated identities, which let you register and authenticate users using your own backend authentication process, while still using Amazon Cognito Sync to synchronize user data and access AWS resources.

For more information on Amazon Cognito, see the Amazon Cognito Developer Guide.

The following code examples show how to easily use Amazon Cognito Identity. The Credentials provider example shows how to create and authenticate user identities. The CognitoAuthentication extension library example shows how to use the CognitoAuthentication extension library to authenticate Amazon Cognito user pools.

Topics

- Amazon Cognito credentials provider
- Amazon CognitoAuthentication extension library examples

Amazon Cognito credentials provider

> **Note**
>
> The information in this topic is specific to projects based on .NET Framework and the AWS SDK for .NET version 3.3 and earlier.

`Amazon.CognitoIdentity.CognitoAWSCredentials`, found in the AWSSDK.CognitoIdentity NuGet package, is a credentials object that uses Amazon Cognito and the AWS Security Token Service (AWS STS) to retrieve credentials to make AWS calls.

The first step in setting up `CognitoAWSCredentials` is to create an "identity pool". (An identity pool is a store of user identity information that is specific to your account. The information is retrievable across client platforms, devices, and operating systems, so that if a user starts using the app on a phone and later switches to a tablet, the persisted app information is still available for that user. You can create a new identity pool from the Amazon Cognito console. If you are using the console, it will also provide you the other pieces of information you need:

- Your account number- A 12-digit number, such as 123456789012, that is unique to your account.
- The unauthenticated role ARN- A role that unauthenticated users will assume. For example, this role can provide read-only permissions to your data.
- The authenticated role ARN- A role that authenticated users will assume. This role can provide more extensive permissions to your data.

Set up CognitoAWSCredentials

The following code example shows how to set up `CognitoAWSCredentials`, which you can then use to make a call to Amazon S3 as an unauthenticated user. This enables you to make calls with just a minimum amount of data required to authenticate the user. User permissions are controlled by the role, so you can configure access as you need.

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials(
```

```
    accountId,        // Account number
    identityPoolId,   // Identity pool ID
    unAuthRoleArn,    // Role for unauthenticated users
    null,             // Role for authenticated users, not set
    region);
using (var s3Client = new AmazonS3Client(credentials))
{
    s3Client.ListBuckets();
}
```

Use AWS as an unauthenticated user

The following code example shows how you can start using AWS as an unauthenticated user, then authenticate through Facebook and update the credentials to use Facebook credentials. Using this approach, you can grant different capabilities to authenticated users via the authenticated role. For instance, you might have a phone application that permits users to view content anonymously, but allows them to post if they are logged on with one or more of the configured providers.

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials(
    accountId, identityPoolId,
    unAuthRoleArn,    // Role for unauthenticated users
    authRoleArn,      // Role for authenticated users
    region);
using (var s3Client = new AmazonS3Client(credentials))
{
    // Initial use will be unauthenticated
    s3Client.ListBuckets();

    // Authenticate user through Facebook
    string facebookToken = GetFacebookAuthToken();

    // Add Facebook login to credentials. This clears the current AWS credentials
    // and retrieves new AWS credentials using the authenticated role.
    credentials.AddLogin("graph.facebook.com", facebookAccessToken);

    // This call is performed with the authenticated role and credentials
    s3Client.ListBuckets();
}
```

The `CognitoAWSCredentials` object provides even more functionality if you use it with the `AmazonCognitoSyncClient` that is part of the AWS SDK for .NET. If you're using both `AmazonCognitoSyncClient` and `CognitoAWSCredentials`, you don't

have to specify the `IdentityPoolId` and `IdentityId` properties when making calls with the `AmazonCognitoSyncClient`. These properties are automatically filled in from `CognitoAWSCredentials`. The next code example illustrates this, as well as an event that notifies you whenever the `IdentityId` for `CognitoAWSCredentials` changes. The `IdentityId` can change in some cases, such as when changing from an unauthenticated user to an authenticated one.

```
CognitoAWSCredentials credentials = GetCognitoAWSCredentials();

// Log identity changes
credentials.IdentityChangedEvent += (sender, args) =>
{
    Console.WriteLine("Identity changed: [{0}] => [{1}]", args.OldIdentityId,
 args.NewIdentityId);
};

using (var syncClient = new AmazonCognitoSyncClient(credentials))
{
    var result = syncClient.ListRecords(new ListRecordsRequest
    {
        DatasetName = datasetName
        // No need to specify these properties
        //IdentityId = "...",
        //IdentityPoolId = "..."
    });
}
```

Amazon CognitoAuthentication extension library examples

> **Note**
>
> The information in this topic is specific to projects based on .NET Framework and the AWS SDK for .NET version 3.3 and earlier.

The CognitoAuthentication extension library, found in the Amazon.Extensions.CognitoAuthentication NuGet package, simplifies the authentication process of Amazon Cognito user pools for .NET Core and Xamarin developers. The library is built on top of the Amazon Cognito Identity provider API to create and send user authentication API calls.

Using the CognitoAuthentication extension library

Amazon Cognito has some built-in `AuthFlow` and `ChallengeName` values for a standard authentication flow to validate username and password through the Secure Remote Password (SRP). For more information about authentication flow, see Amazon Cognito User Pool Authentication Flow.

The following examples require these `using` statements:

```
// Required for all examples
using System;
using Amazon;
using Amazon.CognitoIdentity;
using Amazon.CognitoIdentityProvider;
using Amazon.Extensions.CognitoAuthentication;
using Amazon.Runtime;
// Required for the GetS3BucketsAsync example
using Amazon.S3;
using Amazon.S3.Model;
```

Use basic authentication

Create an AmazonCognitoIdentityProviderClient using AnonymousAWSCredentials, which do not require signed requests. You do not need to supply a region, the underlying code calls `FallbackRegionFactory.GetRegionEndpoint()` if a region is not provided. Create `CognitoUserPool` and `CognitoUser` objects. Call the `StartWithSrpAuthAsync` method with an `InitiateSrpAuthRequest` that contains the user password.

```
public static async void GetCredsAsync()
{
    AmazonCognitoIdentityProviderClient provider =
        new AmazonCognitoIdentityProviderClient(new
 Amazon.Runtime.AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);
    InitiateSrpAuthRequest authRequest = new InitiateSrpAuthRequest()
    {
        Password = "userPassword"
    };

    AuthFlowResponse authResponse = await
 user.StartWithSrpAuthAsync(authRequest).ConfigureAwait(false);
```

```
    accessToken = authResponse.AuthenticationResult.AccessToken;

}
```

Authenticate with challenges

Continuing the authentication flow with challenges, such as with NewPasswordRequired and Multi-Factor Authentication (MFA), is also simpler. The only requirements are the CognitoAuthentication objects, user's password for SRP, and the necessary information for the next challenge, which is acquired after prompting the user to enter it. The following code shows one way to check the challenge type and get the appropriate responses for MFA and NewPasswordRequired challenges during the authentication flow.

Do a basic authentication request as before, and `await` an `AuthFlowResponse`.
When the response is received loop through the returned `AuthenticationResult`
object. If the `ChallengeName` type is `NEW_PASSWORD_REQUIRED`, call the
`RespondToNewPasswordRequiredAsync` method.

```
public static async void GetCredsChallengesAsync()
{
    AmazonCognitoIdentityProviderClient provider =
        new AmazonCognitoIdentityProviderClient(new
 Amazon.Runtime.AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);
    InitiateSrpAuthRequest authRequest = new InitiateSrpAuthRequest(){
        Password = "userPassword"
    };

    AuthFlowResponse authResponse = await
 user.StartWithSrpAuthAsync(authRequest).ConfigureAwait(false);

    while (authResponse.AuthenticationResult == null)
    {
        if (authResponse.ChallengeName == ChallengeNameType.NEW_PASSWORD_REQUIRED)
        {
            Console.WriteLine("Enter your desired new password:");
            string newPassword = Console.ReadLine();

            authResponse = await user.RespondToNewPasswordRequiredAsync(new
 RespondToNewPasswordRequiredRequest()
            {
```

```
                SessionID = authResponse.SessionID,
                NewPassword = newPassword
            });
            accessToken = authResponse.AuthenticationResult.AccessToken;
        }
        else if (authResponse.ChallengeName == ChallengeNameType.SMS_MFA)
        {
            Console.WriteLine("Enter the MFA Code sent to your device:");
            string mfaCode = Console.ReadLine();

            AuthFlowResponse mfaResponse = await user.RespondToSmsMfaAuthAsync(new
 RespondToSmsMfaRequest()
            {
                SessionID = authResponse.SessionID,
                MfaCode = mfaCode

            }).ConfigureAwait(false);
            accessToken = authResponse.AuthenticationResult.AccessToken;
        }
        else
        {
            Console.WriteLine("Unrecognized authentication challenge.");
            accessToken = "";
            break;
        }
    }

    if (authResponse.AuthenticationResult != null)
    {
        Console.WriteLine("User successfully authenticated.");
    }
    else
    {
        Console.WriteLine("Error in authentication process.");
    }

}
```

Use AWS resources after authentication

Once a user is authenticated using the CognitoAuthentication library, the next step is to allow the user to access the appropriate AWS resources. To do this you must create an identity pool through the Amazon Cognito Federated Identities console. By specifying the Amazon Cognito user pool

you created as a provider, using its poolID and clientID, you can allow your Amazon Cognito user pool users to access AWS resources connected to your account. You can also specify different roles to enable both unauthenticated and authenticated users to access different resources. You can change these rules in the IAM console, where you can add or remove permissions in the **Action** field of the role's attached policy. Then, using the appropriate identity pool, user pool, and Amazon Cognito user information, you can make calls to different AWS resources. The following example shows a user authenticated with SRP accessing the different Amazon S3 buckets permitted by the associated identity pool's role

```
public async void GetS3BucketsAsync()
{
    var provider = new AmazonCognitoIdentityProviderClient(new
 AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);

    string password = "userPassword";

    AuthFlowResponse context = await user.StartWithSrpAuthAsync(new
 InitiateSrpAuthRequest()
    {
        Password = password
    }).ConfigureAwait(false);

    CognitoAWSCredentials credentials =
        user.GetCognitoAWSCredentials("identityPoolID", RegionEndpoint.<
 YourIdentityPoolRegion >);

    using (var client = new AmazonS3Client(credentials))
    {
        ListBucketsResponse response =
            await client.ListBucketsAsync(new
 ListBucketsRequest()).ConfigureAwait(false);

        foreach (S3Bucket bucket in response.Buckets)
        {
            Console.WriteLine(bucket.BucketName);
        }
    }
}
```

More authentication options

In addition to SRP, NewPasswordRequired, and MFA, the CognitoAuthentication extension library offers an easier authentication flow for:

- Custom - Initiate with a call to `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)`
- RefreshToken - Initiate with a call to `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP - Initiate with a call to `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP - Initiate with a call to `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

Call the appropriate method depending on the flow you want. Then continue prompting the user with challenges as they are presented in the `AuthFlowResponse` objects of each method call. Also call the appropriate response method, such as `RespondToSmsMfaAuthAsync` for MFA challenges and `RespondToCustomAuthAsync` for custom challenges.

Using Amazon DynamoDB NoSQL databases

> **Note**
>
> The programming models in these topics are present in both .NET Framework and .NET (Core), but the calling conventions differ, whether synchronous or asynchronous.

The AWS SDK for .NET supports Amazon DynamoDB, which is a fast NoSQL database service offered by AWS. The SDK provides three programming models for communicating with DynamoDB: the *low-level* model, the *document* model, and the *object persistence* model.

The following information introduces these models and their APIs, provides examples for how and when to use them, and gives you links to additional DynamoDB programming resources in the AWS SDK for .NET.

Low-Level Model

The low-level programming model wraps direct calls to the DynamoDB service. You access this model through the Amazon.DynamoDBv2 namespace.

Of the three models, the low-level model requires you to write the most code. For example, you must convert .NET data types to their equivalents in DynamoDB. However, this model gives you access to the most features.

The following examples show you how to use the low-level model to create a table, modify a table, and insert items into a table in DynamoDB.

Creating a Table

In the following example, you create a table by using the `CreateTable` method of the `AmazonDynamoDBClient` class. The `CreateTable` method uses an instance of the `CreateTableRequest` class that contains characteristics such as required item attribute names, primary key definition, and throughput capacity. The `CreateTable` method returns an instance of the `CreateTableResponse` class.

```
// 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);
}
```

Verifying That a Table is Ready to Modify

Before you can change or modify a table, the table has to be ready for modification. The following example shows how to use the low-level model to verify that a table in DynamoDB is ready. In this example, the target table to check is referenced through the `DescribeTable` method of the `AmazonDynamoDBClient` class. Every five seconds, the code checks the value of the table's `TableStatus` property. When the status is set to `ACTIVE`, the table is ready to be modified.

```
// 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);
```

Inserting an Item into a Table

In the following example, you use the low-level model to insert two items into a table in DynamoDB. Each item is inserted through the `PutItem` method of the `AmazonDynamoDBClient` class, using an instance of the `PutItemRequest` class. Each of the two instances of the `PutItemRequest` class takes the name of the table that the items will be inserted in, with a series of item attribute values.

```
// 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);
```

Document Model

The document programming model provides an easier way to work with data in DynamoDB. This model is specifically intended for accessing tables and items in tables. You access this model through the Amazon.DynamoDBv2.DocumentModel namespace.

Compared to the low-level programming model, the document model is easier to code against DynamoDB data. For example, you don't have to convert as many .NET data types to their equivalents in DynamoDB. However, this model doesn't provide access to as many features as the low-level programming model. For example, you can use this model to create, retrieve, update, and delete items in tables. However, to create the tables, you must use the low-level model. Compared to the object persistence model, this model requires you to write more code to store, load, and query .NET objects.

For more information about the DynamoDB document programming model, see .NET: Document model in the Amazon DynamoDB Developer Guide.

The following sections provide information about how to create a representation of the desired DynamoDB table, and examples about how to use the document model to insert items into tables and get items from tables.

Create a representation of the table

To perform data operations using the document model, you must first create an instance of the `Table` class that represents a specific table. There are two primary ways to do this.

LoadTable method

The first mechanism is to use one of the static `LoadTable` methods of the `Table` class, similar to the following example:

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

> **Note**
>
> While this mechanism works, under certain conditions, it can sometimes lead to additional latency or deadlocks due to cold-start and thread-pool behaviors. For more information about these behaviors, see the blog post Improved DynamoDB Initialization Patterns for the AWS SDK for .NET.

TableBuilder

An alternative mechanism, the `TableBuilder` class, was introduced in version 3.7.203 of the AWSSDK.DynamoDBv2 NuGet package. This mechanism can address the behaviors mentioned above by removing certain implicit method calls; specifically, the `DescribeTable` method. This mechanism is used in a manner similar to the following example:

```
var client = new AmazonDynamoDBClient();
var table = new TableBuilder(client, "Reply")
    .AddHashKey("Id", DynamoDBEntryType.String)
    .AddRangeKey("ReplyDateTime", DynamoDBEntryType.String)
    .AddGlobalSecondaryIndex("PostedBy-Message-index", "Author",
 DynamoDBEntryType.String, "Message", DynamoDBEntryType.String)
    .Build();
```

For more information about this alternative mechanism, see again the blog post Improved DynamoDB Initialization Patterns for the AWS SDK for .NET.

Inserting an item into a table

In the following example, a reply is inserted into the Reply table through the `PutItemAsync` method of the `Table` class. The `PutItemAsync` method takes an instance of the `Document` class; the `Document` class is simply a collection of initialized attributes.

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

// Create a representation of the "Reply" table
//  by using one of the mechanisms described previously.

// Then, add a reply to the table.
var newReply = new Document();
newReply["Id"] = Guid.NewGuid().ToString();
newReply["ReplyDateTime"] = DateTime.UtcNow;
newReply["PostedBy"] = "Author1";
newReply["Message"] = "Thank you!";

await table.PutItemAsync(newReply);
```

Getting an item from a table

In the following example, a reply is retrieved through the `GetItemAsync` method of the `Table` class. To determine the reply to get, the `GetItemAsync` method uses the hash-and-range primary key of the target reply.

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

// Create a representation of the "Reply" table
//  by using one of the mechanisms described previously.

// Then, get a reply from the table
//  where "guid" is the hash key and "datetime" is the range key.
var reply = await table.GetItemAsync(guid, datetime);
Console.WriteLine("Id = " + reply["Id"]);
Console.WriteLine("ReplyDateTime = " + reply["ReplyDateTime"]);
Console.WriteLine("PostedBy = " + reply["PostedBy"]);
```

```
Console.WriteLine("Message = " + reply["Message"]);
```

The preceding example implicitly converts the table values to strings for the `WriteLine` method. You can do explicit conversions by using the various "As[type]" methods of the `DynamoDBEntry` class. For example, you can explicitly convert the value for `Id` from a `Primitive` data type to a GUID through the `AsGuid()` method:

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

Object Persistence Model

The object persistence programming model is specifically designed for storing, loading, and querying .NET objects in DynamoDB. You access this model through the Amazon.DynamoDBv2.DataModel namespace.

Of the three models, the object persistence model is the easiest to code against whenever you are storing, loading, or querying DynamoDB data. For example, you work with DynamoDB data types directly. However, this model provides access only to operations that store, load, and query .NET objects in DynamoDB. For example, you can use this model to create, retrieve, update, and delete items in tables. However, you must first create your tables using the low-level model, and then use this model to map your .NET classes to the tables.

For more information about the DynamoDB object persistence programming model, see .NET: Object persistence model in the Amazon DynamoDB Developer Guide.

The following examples show you how to define a .NET class that represents a DynamoDB item, use an instance of the .NET class to insert an item into a DynamoDB table, and use an instance of the .NET class to get an item from the table.

Defining a .NET class that represents an item in a table

In the following example of a class definition, the `DynamoDBTable` attribute specifies the table name, while the `DynamoDBHashKey` and `DynamoDBRangeKey` attributes model the table's hash-and-range primary key. The `DynamoDBGlobalSecondaryIndexHashKey` attribute is defined so that a query for replies by a specific author can be constructed.

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

[DynamoDBTable("Reply")]
```

```
public class Reply
{
    [DynamoDBHashKey]
    public string Id { get; set; }

    [DynamoDBRangeKey(StoreAsEpoch = false)]
    public DateTime ReplyDateTime { get; set; }

    [DynamoDBGlobalSecondaryIndexHashKey("PostedBy-Message-Index",
        AttributeName ="PostedBy")]
    public string Author { get; set; }

    [DynamoDBGlobalSecondaryIndexRangeKey("PostedBy-Message-Index")]
    public string Message { get; set; }
}
```

Creating a context for the object persistence model

To use the object persistence programming model for DynamoDB, you must create a context, which provides a connection to DynamoDB and enables you to access tables, perform various operations, and run queries.

Basic context

The following example shows how to create the most basic context.

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

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

Context with DisableFetchingTableMetadata property

The following example shows how you might additionally set the `DisableFetchingTableMetadata` property of the `DynamoDBContextConfig` class to prevent implicit calls to the `DescribeTable` method.

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

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

```
{
    DisableFetchingTableMetadata = true
});
```

If the `DisableFetchingTableMetadata` property is set to `false` (the default), as shown in the first example, you can omit attributes that describe the key and index structure of table items from the `Reply` class. These attributes will instead be inferred through an implicit call to the `DescribeTable` method. If `DisableFetchingTableMetadata` is set to `true`, as shown in the second example, methods of the object persistence model such as `SaveAsync` and `QueryAsync` rely entirely on the attributes defined in the `Reply` class. In this case, a call to the `DescribeTable` method doesn't occur.

> **Note**
>
> Under certain conditions, calls to the `DescribeTable` method can sometimes lead to additional latency or deadlocks due to cold-start and thread-pool behaviors. For this reason, it is sometimes advantageous to avoid calls to that method.
> For more information about these behaviors, see the blog post Improved DynamoDB Initialization Patterns for the AWS SDK for .NET.

Using an instance of the .NET class to insert an item into a table

In this example, an item is inserted through the `SaveAsync` method of the `DynamoDBContext` class, which takes an initialized instance of the .NET class that represents the item.

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

// Create an appropriate context for the object persistence programming model,
//  examples of which have been described earlier.

// Create an object that represents the new item.
var reply = new Reply()
{
    Id = Guid.NewGuid().ToString(),
    ReplyDateTime = DateTime.UtcNow,
    Author = "Author1",
    Message = "Thank you!"
};
```

```
// Insert the item into the table.
await context.SaveAsync<Reply>(reply, new DynamoDBOperationConfig
{
    IndexName = "PostedBy-Message-index"
});
```

Using an instance of a .NET class to get items from a table

In this example, a query is created to find all the records of "Author1" by using the `QueryAsync` method of the `DynamoDBContext` class. Then, items are retrieved through the query's `GetNextSetAsync` method.

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

// Create an appropriate context for the object persistence programming model,
//  examples of which have been described earlier.

// Construct a query that finds all replies by a specific author.
var query = context.QueryAsync<Reply>("Author1", new DynamoDBOperationConfig
{
    IndexName = "PostedBy-Message-index"
});

// Display the result.
var set = await query.GetNextSetAsync();
foreach (var item in set)
{
    Console.WriteLine("Id = " + item.Id);
    Console.WriteLine("ReplyDateTime = " + item.ReplyDateTime);
    Console.WriteLine("PostedBy = " + item.Author);
    Console.WriteLine("Message = " + item.Message);
}
```

Additional information about the object persistence model

The examples and explanations shown above sometimes include a property of the `DynamoDBContext` class called `DisableFetchingTableMetadata`. This property, which was introduced in version 3.7.203 of the AWSSDK.DynamoDBv2 NuGet package, allows you to avoid certain conditions that might cause additional latency or deadlocks due to cold-start and thread-pool behaviors. For more information, see the blog post Improved DynamoDB Initialization Patterns for the AWS SDK for .NET.

The following is some additional information about this property.

- This property can be set globally in your `app.config` or `web.config` file if you're using .NET Framework.
- This property can be set globally by using the `AWSConfigsDynamoDB` class, as shown in the following example.

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

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

- In some cases, you can't add DynamoDB attributes to a .NET class; for example, if the class is defined in a dependency. In such cases, it's possible to still take advantage of the `DisableFetchingTableMetadata` property. To do so, use the `TableBuilder` class in addition to the `DisableFetchingTableMetadata` property. The `TableBuilder` class was also introduced in version 3.7.203 of the AWSSDK.DynamoDBv2 NuGet package.

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

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

var table = new TableBuilder(client, "Reply")
    .AddHashKey("Id", DynamoDBEntryType.String)
    .AddRangeKey("ReplyDateTime", DynamoDBEntryType.String)
    .AddGlobalSecondaryIndex("PostedBy-Message-index", "Author",
 DynamoDBEntryType.String,
        "Message", DynamoDBEntryType.String)
    .Build();

// This registers the "Reply" table we constructed via the builder.
context.RegisterTableDefinition(table);

// Now operations like this will work,
// even if the Reply class was not annotated with this index.
var query = context.QueryAsync<Reply>("Author1", new DynamoDBOperationConfig()
{
```

```
    IndexName = "PostedBy-Message-index"
});
```

More information

Using the AWS SDK for .NET to program DynamoDB, information and examples

- DynamoDB APIs
- DynamoDB Series Kickoff
- DynamoDB Series - Document Model
- DynamoDB Series - Conversion Schemas
- DynamoDB Series - Object Persistence Model
- DynamoDB Series - Expressions
- Using Expressions with Amazon DynamoDB and the AWS SDK for .NET
- JSON support in Amazon DynamoDB

Low-Level model, information and examples

- Working with Tables Using the AWS SDK for .NET Low-Level API
- Working with Items Using the AWS SDK for .NET Low-Level API
- Querying Tables Using the AWS SDK for .NET Low-Level API
- Scanning Tables Using the AWS SDK for .NET Low-Level API
- Working with Local Secondary Indexes Using the AWS SDK for .NET Low-Level API
- Working with Global Secondary Indexes Using the AWS SDK for .NET Low-Level API

Document model, information and examples

- DynamoDB Data Types
- DynamoDBEntry
- .NET: Document Model

Object persistence model, information and examples

- .NET: Object Persistence Model

Other useful information

- See Integrating AWS with .NET Aspire for information about developing with Amazon DynamoDB local through .NET Aspire.

Topics

- Using Expressions with Amazon DynamoDB and the AWS SDK for .NET
- JSON support in Amazon DynamoDB

Using Expressions with Amazon DynamoDB and the AWS SDK for .NET

> **Note**
>
> The information in this topic is specific to projects based on .NET Framework and the AWS SDK for .NET version 3.3 and earlier.

The following code examples demonstrate how to use the AWS SDK for .NET to program DynamoDB with expressions. *Expressions* denote the attributes you want to read from an item in a DynamoDB table. You also use expressions when writing an item, to indicate any conditions that must be met (also known as a *conditional update*) and how the attributes are to be updated. Some update examples are replacing the attribute with a new value, or adding new data to a list or a map. For more information, see Reading and Writing Items Using Expressions.

Topics

- Sample Data
- Get a Single Item by Using Expressions and the Item's Primary Key
- Get Multiple Items by Using Expressions and the Table's Primary Key
- Get Multiple Items by Using Expressions and Other Item Attributes
- Print an Item
- Create or Replace an Item by Using Expressions
- Update an Item by Using Expressions
- Delete an Item by Using Expressions
- More Info

Sample Data

The code examples in this topic rely on the following two example items in a DynamoDB table named `ProductCatalog`. These items describe information about product entries in a fictitious bicycle store catalog. These items are based on the example provided in Case Study: A ProductCatalog Item. The data type descriptors such as `BOOL`, `L`, `M`, `N`, `NS`, `S`, and `SS` correspond to those in the JSON Data Format.

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

```
    }
  }
}
```

Get a Single Item by Using Expressions and the Item's Primary Key

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` method and a set of expressions to get and then print the item that has an `Id` of `205`. Only the following attributes of the item are returned: `Id`, `Title`, `Description`, `Color`, `RelatedItems`, `Pictures`, and `ProductReviews`.

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

var client = new AmazonDynamoDBClient();
var request = new GetItemRequest
{
  TableName = "ProductCatalog",
  ProjectionExpression = "Id, Title, Description, Color, #ri, Pictures, #pr",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#ri", "RelatedItems" }
  },
  Key = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "205" } }
  },
};
var response = client.GetItem(request);

// PrintItem() is a custom function.
PrintItem(response.Item);
```

In the preceding example, the `ProjectionExpression` property specifies the attributes to be returned. The `ExpressionAttributeNames` property specifies the placeholder `#pr` to represent the `ProductReviews` attribute and the placeholder `#ri` to represent the `RelatedItems` attribute. The call to `PrintItem` refers to a custom function as described in Print an Item.

Get Multiple Items by Using Expressions and the Table's Primary Key

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.Query` method and a set of expressions to get and then print the item that has an `Id` of `301`, but only if the value of `Price` is greater than `150`. Only the following attributes of the item are returned: `Id`, `Title`, and all of the `ThreeStar` attributes in `ProductReviews`.

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

var client = new AmazonDynamoDBClient();
var request = new QueryRequest
{
  TableName = "ProductCatalog",
  KeyConditions = new Dictionary<string,Condition>
  {
    { "Id", new Condition()
      {
        ComparisonOperator = ComparisonOperator.EQ,
        AttributeValueList = new List<AttributeValue>
        {
          new AttributeValue { N = "301" }
        }
      }
    }
  },
  ProjectionExpression = "Id, Title, #pr.ThreeStar",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#p", "Price" }
  },
  ExpressionAttributeValues = new Dictionary<string,AttributeValue>
  {
    { ":val", new AttributeValue { N = "150" } }
  },
  FilterExpression = "#p > :val"
};
var response = client.Query(request);

foreach (var item in response.Items)
{
  // Write out the first page of an item's attribute keys and values.
```

```
  // PrintItem() is a custom function.
  PrintItem(item);
  Console.WriteLine("=====");
}
```

In the preceding example, the `ProjectionExpression` property specifies the attributes to be returned. The `ExpressionAttributeNames` property specifies the placeholder `#pr` to represent the `ProductReviews` attribute and the placeholder `#p` to represent the `Price` attribute. `#pr.ThreeStar` specifies to return only the `ThreeStar` attribute. The `ExpressionAttributeValues` property specifies the placeholder `:val` to represent the value `150`. The `FilterExpression` property specifies that `#p` (`Price`) must be greater than `:val` (`150`). The call to `PrintItem` refers to a custom function as described in Print an Item.

Get Multiple Items by Using Expressions and Other Item Attributes

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan` method and a set of expressions to get and then print all items that have a `ProductCategory` of `Bike`. Only the following attributes of the item are returned: `Id`, `Title`, and all of the attributes in `ProductReviews`.

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

var client = new AmazonDynamoDBClient();
var request = new ScanRequest
{
  TableName = "ProductCatalog",
  ProjectionExpression = "Id, Title, #pr",
  ExpressionAttributeValues = new Dictionary<string,AttributeValue>
  {
    { ":catg", new AttributeValue { S = "Bike" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#pc", "ProductCategory" }
  },
  FilterExpression = "#pc = :catg",
};
var response = client.Scan(request);

foreach (var item in response.Items)
```

```
{
  // Write out the first page/scan of an item's attribute keys and values.
  // PrintItem() is a custom function.
  PrintItem(item);
  Console.WriteLine("=====");
}
```

In the preceding example, the `ProjectionExpression` property specifies the attributes to be returned. The `ExpressionAttributeNames` property specifies the placeholder `#pr` to represent the `ProductReviews` attribute and the placeholder `#pc` to represent the `ProductCategory` attribute. The `ExpressionAttributeValues` property specifies the placeholder `:catg` to represent the value `Bike`. The `FilterExpression` property specifies that `#pc` (`ProductCategory`) must be equal to `:catg` (`Bike`). The call to `PrintItem` refers to a custom function as described in Print an Item.

Print an Item

The following example shows how to print an item's attributes and values. This example is used in the preceding examples that show how to Get a Single Item by Using Expressions and the Item's Primary Key, Get Multiple Items by Using Expressions and the Table's Primary Key, and Get Multiple Items by Using Expressions and Other Item Attributes.

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

In the preceding example, each attribute value has several data-type-specific properties that can be evaluated to determine the correct format to print the attribute. These properties include B, BOOL, BS, L, M, N, NS, NULL, S, and SS, which correspond to those in the JSON Data Format. For properties such as B, N, NULL, and S, if the corresponding property is not `null`, then the attribute is of the corresponding non-`null` data type. For properties such as BS, L, M, NS, and SS, if `Count` is greater than zero, then the attribute is of the corresponding non-zero-value data type. If all of the attribute's data-type-specific properties are either `null` or the `Count` equals zero, then the attribute corresponds to the BOOL data type.

Create or Replace an Item by Using Expressions

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem` method and a set of expressions to update the item that has a `Title` of `18-Bicycle 301`. If the item doesn't already exist, a new item is added.

```
// 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);
```

In the preceding example, the `ExpressionAttributeNames` property specifies the placeholder `#title` to represent the `Title` attribute. The `ExpressionAttributeValues` property specifies the placeholder `:product` to represent the value `18-Bicycle 301`. The `ConditionExpression` property specifies that `#title` (`Title`) must be equal to `:product` (`18-Bicycle 301`). The call to `CreateItemData` refers to the following custom function:

```
// 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;
}
```

In the preceding example, an example item with sample data is returned to the caller. A series of attributes and corresponding values are constructed, using data types such as BOOL, L, M, N, NS, S, and SS, which correspond to those in the JSON Data Format.

Update an Item by Using Expressions

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem` method and a set of expressions to change the `Title` to `18" Girl's Bike` for the item with `Id` of `301`.

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

var client = new AmazonDynamoDBClient();
var request = new UpdateItemRequest
{
  TableName = "ProductCatalog",
  Key = new Dictionary<string,AttributeValue>
  {
     { "Id", new AttributeValue { N = "301" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":newproduct", new AttributeValue { S = "18\" Girl's Bike" } }
  },
  UpdateExpression = "SET #title = :newproduct"
};
```

```
client.UpdateItem(request);
```

In the preceding example, the `ExpressionAttributeNames` property specifies the placeholder `#title` to represent the `Title` attribute. The `ExpressionAttributeValues` property specifies the placeholder `:newproduct` to represent the value `18" Girl's Bike`. The `UpdateExpression` property specifies to change `#title` (`Title`) to `:newproduct` (`18" Girl's Bike`).

Delete an Item by Using Expressions

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem` method and a set of expressions to delete the item with `Id` of `301`, but only if the item's `Title` is `18-Bicycle 301`.

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

var client = new AmazonDynamoDBClient();
var request = new DeleteItemRequest
{
  TableName = "ProductCatalog",
  Key = new Dictionary<string,AttributeValue>
  {
     { "Id", new AttributeValue { N = "301" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":product", new AttributeValue { S = "18-Bicycle 301" } }
  },
  ConditionExpression = "#title = :product"
};
client.DeleteItem(request);
```

In the preceding example, the `ExpressionAttributeNames` property specifies the placeholder `#title` to represent the `Title` attribute. The `ExpressionAttributeValues` property specifies the placeholder `:product` to represent the value `18-Bicycle 301`. The `ConditionExpression` property specifies that `#title` (`Title`) must equal `:product` (`18-Bicycle 301`).

More Info

For more information and code examples, see:

- DynamoDB Series - Expressions
- Accessing Item Attributes with Projection Expressions
- Using Placeholders for Attribute Names and Values
- Specifying Conditions with Condition Expressions
- Modifying Items and Attributes with Update Expressions
- Working with Items Using the AWS SDK for .NET Low-Level API
- Querying Tables Using the AWS SDK for .NET Low-Level API
- Scanning Tables Using the AWS SDK for .NET Low-Level API
- Working with Local Secondary Indexes Using the AWS SDK for .NET Low-Level API
- Working with Global Secondary Indexes Using the AWS SDK for .NET Low-Level API

JSON support in Amazon DynamoDB

> **Note**
>
> The information in this topic is specific to projects based on .NET Framework and the AWS SDK for .NET version 3.3 and earlier.

The AWS SDK for .NET supports JSON data when working with Amazon DynamoDB. This enables you to more easily get JSON-formatted data from, and insert JSON documents into, DynamoDB tables.

Topics

- Get Data from a DynamoDB Table in JSON Format
- Insert JSON Format Data into a DynamoDB Table
- DynamoDB Data Type Conversions to JSON
- More Info

Get Data from a DynamoDB Table in JSON Format

The following example shows how to get data from a DynamoDB table in JSON format:

```
// 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
//   }
```

In the preceding example, the `ToJson` method of the `Document` class converts an item from the table into a JSON-formatted string. The item is retrieved through the `GetItem` method of the `Table` class. To determine the item to get, in this example, the `GetItem` method uses the hash-and-range primary key of the target item. To determine the table to get the item from, the `LoadTable` method of the `Table` class uses an instance of the `AmazonDynamoDBClient` class and the name of the target table in DynamoDB.

Insert JSON Format Data into a DynamoDB Table

The following example shows how to use JSON format to insert an item into a DynamoDB table:

```
// 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);
```

In the preceding example, the `FromJson` method of the `Document` class converts a JSON-formatted string into an item. The item is inserted into the table through the `PutItem` method of the `Table` class, which uses the instance of the `Document` class that contains the item. To determine the table to insert the item into, the `LoadTable` method of the `Table` class is called, specifying an instance of the `AmazonDynamoDBClient` class and the name of the target table in DynamoDB.

DynamoDB Data Type Conversions to JSON

Whenever you call the `ToJson` method of the `Document` class, and then on the resulting JSON data you call the `FromJson` method to convert the JSON data back into an instance of a `Document` class, some DynamoDB data types will not be converted as expected. Specifically:

- DynamoDB sets (the `SS`, `NS`, and `BS` types) will be converted to JSON arrays.
- DynamoDB binary scalars and sets (the `B` and `BS` types) will be converted to base64-encoded JSON strings or lists of strings.

  In this scenario, you must call the `DecodeBase64Attributes` method of the `Document` class to replace the base64-encoded JSON data with the correct binary representation. The following example replaces a base64-encoded binary scalar item attribute in an instance of a `Document` class, named `Picture`, with the correct binary representation. This example also does the same for a base64-encoded binary set item attribute in the same instance of the `Document` class, named `RelatedPictures`:

  ```
  item.DecodeBase64Attributes("Picture", "RelatedPictures");
  ```

More Info

For more information and examples of programming JSON with DynamoDB with the AWS SDK for .NET, see:

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

Working with Amazon EC2

The AWS SDK for .NET supports Amazon EC2, which is a web service that provides resizable computing capacity. You use this computing capacity to build and host your software systems.

APIs

The AWS SDK for .NET provides APIs for Amazon EC2 clients. The APIs enable you to work with EC2 features such as security groups and key pairs. The APIs also enable you to control Amazon EC2 instances. This section contains a small number of examples that show you the patterns you can follow when working with these APIs. To view the full set of APIs, see the AWS SDK for .NET API Reference (and scroll to "Amazon.EC2").

The Amazon EC2 APIs are provided by the AWSSDK.EC2 NuGet package.

Prerequisites

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

About the examples

The examples in this section show you how to work with Amazon EC2 clients and manage Amazon EC2 instances.

The EC2 Spot Instance tutorial shows you how to request Amazon EC2 Spot Instances. Spot Instances enable you to access unused EC2 capacity for less than the On-Demand price.

Topics

- Working with security groups in Amazon EC2
- Working with Amazon EC2 key pairs
- Seeing your Amazon EC2 Regions and Availability Zones
- Working with Amazon EC2 instances
- Amazon EC2 Spot Instance tutorial

Working with security groups in Amazon EC2

In Amazon EC2, a *security group* acts as a virtual firewall that controls the network traffic for one or more EC2 instances. By default, EC2 associates your instances with a security group that allows

no inbound traffic. You can create a security group that allows your EC2 instances to accept certain traffic. For example, if you need to connect to an EC2 Windows instance, you must configure the security group to allow RDP traffic.

To read more about security groups see Amazon EC2 security groups in the Amazon EC2 User Guide.

> **Warning**
>
> EC2-Classic was retired on August 15, 2022. We recommend that you migrate from EC2-Classic to a VPC. For more information, see the blog post EC2-Classic Networking is Retiring – Here's How to Prepare.

For information about the APIs and prerequisites, see the parent section (Working with Amazon EC2).

Topics

- Enumerating security groups
- Creating security groups
- Updating security groups

Enumerating security groups

This example shows you how to use the AWS SDK for .NET to enumerate security groups. If you supply an Amazon Virtual Private Cloud ID, the application enumerates the security groups for that particular VPC. Otherwise, the application simply displays a list of all available security groups.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Enumerate security groups
- Complete code
- Additional considerations

Enumerate security groups

The following snippet enumerates your security groups. It enumerates all groups or the groups for a particular VPC if one is given.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.EC2

Programming elements:

- Namespace Amazon.EC2

  Class AmazonEC2Client
- Namespace Amazon.EC2.Model

  Class DescribeSecurityGroupsRequest

  Class DescribeSecurityGroupsResponse

  Class Filter

  Class SecurityGroup

The Code

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

Additional considerations

- Notice for the VPC case that the filter is constructed with the `Name` part of the name-value pair set to "vpc-id". This name comes from the description for the `Filters` property of the DescribeSecurityGroupsRequest class.

- To get the complete list of your security groups, you can also use DescribeSecurityGroupsAsync with no parameters.

- You can verify the results by checking the list of security groups in the Amazon EC2 console.

Creating security groups

This example shows you how to use the AWS SDK for .NET to create a security group. You can provide the ID of an existing VPC to create a security group for EC2 in a VPC. If you don't supply such an ID, the new security group will be for EC2-Classic if your AWS account supports this.

If you don't supply a VPC ID and your AWS account doesn't support EC2-Classic, the new security group will belong to the default VPC of your account.

Warning

EC2-Classic was retired on August 15, 2022. We recommend that you migrate from EC2-Classic to a VPC. For more information, see the blog post EC2-Classic Networking is Retiring – Here's How to Prepare.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Find existing security groups
- Create a security group
- Complete code

Find existing security groups

The following snippet searches for existing security groups with the given name in the given VPC.

The example at the end of this topic shows this snippet in use.

```
//
// 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;
```

```
    }
```

Create a security group

The following snippet creates a new security group if a group with that name doesn't exist in the given VPC. If no VPC is given and one or more groups with that name exist, the snippet simply returns the list of groups.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.EC2

Programming elements:

- Namespace Amazon.EC2

  Class AmazonEC2Client
- Namespace Amazon.EC2.Model

  Class CreateSecurityGroupRequest

  Class CreateSecurityGroupResponse

  Class DescribeSecurityGroupsRequest

  Class DescribeSecurityGroupsResponse

  Class Filter

  Class SecurityGroup

The code

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
```

```
using Amazon.EC2.Model;

namespace EC2CreateSecGroup
{
  // = = =
```

```
    }
  }


  //
  // 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);
    }
  }

}
```

Updating security groups

This example shows you how to use the AWS SDK for .NET to add a rule to a security group. In particular, the example adds a rule to allow inbound traffic on a given TCP port, which can be used, for example, for remote connections to an EC2 instance. The application takes the ID of an existing security group, an IP address (or address range) in CIDR format, and optionally a TCP port number. It then adds an inbound rule to the given security group.

> **Note**
>
> To use this example, you need an IP address (or address range) in CIDR format. See **Additional considerations** at this end of this topic for methods to obtain the IP address of your local computer.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Add an inbound rule
- Complete code
- Additional considerations

Add an inbound rule

The following snippet adds an inbound rule to a security group for a particular IP address (or range) and TCP port.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.EC2

Programming elements:

- Namespace Amazon.EC2

  Class AmazonEC2Client
- Namespace Amazon.EC2.Model

  Class AuthorizeSecurityGroupIngressRequest

  Class AuthorizeSecurityGroupIngressResponse

  Class IpPermission

  Class IpRange

The code

```
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);
    }
  }

}
```

Additional considerations

- If you don't supply a port number, the application defaults to port 3389. This is the port for Windows RDP, which enables you to connect to an EC2 instance running Windows. If you're launching an EC2 instance running Linux, you can use TCP port 22 (SSH) instead.

- Notice that the example sets `IpProtocol` to "tcp". The values for `IpProtocol` can be found in the description for the `IpProtocol` property of the IpPermission class.

- You might want the IP address of your local computer when you use this example. The following are some of the ways in which you can obtain the address.
  - If your local computer (from which you will connect to your EC2 instance) has a static public IP address, you can use a service to get that address. One such service is http://checkip.amazonaws.com/. To read more about authorizing inbound traffic, see Add rules to a security group and Security group rules for different use cases in the Amazon EC2 User Guide.
  - Another way to obtain the IP address of your local computer is to use the Amazon EC2 console.

Select one of your security groups, select the **Inbound rules** tab, and choose **Edit inbound rules**. In an inbound rule, open the drop-down menu in the **Source** column and choose **My IP** to see the IP address of your local computer in CIDR format. Be sure to **Cancel** the operation.

- You can verify the results of this example by examining the list of security groups in the Amazon EC2 console.

Working with Amazon EC2 key pairs

Amazon EC2 uses public–key cryptography to encrypt and decrypt login information. Public–key cryptography uses a public key to encrypt data, and then the recipient uses the private key to decrypt the data. The public and private keys are known as a key pair. If you want to log into an EC2 instance, you must specify a key pair when you launch it, and then provide the private key of the pair when you connect to it.

When you launch an EC2 instance, you can create a key pair for it or use one that you've already used when launching other instances. To read more about Amazon EC2 key pairs, see Working with Amazon EC2 key pairs in the Amazon EC2 User Guide.

For information about the APIs and prerequisites, see the parent section (Working with Amazon EC2).

Topics

- Creating and displaying key pairs
- Deleting key pairs

Creating and displaying key pairs

This example shows you how to use the AWS SDK for .NET to create a key pair. The application takes the name for the new key pair and the name of a PEM file (with a ".pem" extension). It creates the keypair, writes the private key to the PEM file, and then displays all available key pairs. If you provide no command-line arguments, the application simply displays all available key pairs.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Create the key pair
- Display available key pairs
- Complete code
- Additional considerations

Create the key pair

The following snippet creates a key pair and then stores the private key to the given PEM file.

The example at the end of this topic shows this snippet in use.

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

Display available key pairs

The following snippet displays a list of the available key pairs.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.EC2

Programming elements:

- Namespace Amazon.EC2

  Class AmazonEC2Client
- Namespace Amazon.EC2.Model

  Class CreateKeyPairRequest

  Class CreateKeyPairResponse

  Class DescribeKeyPairsResponse

  Class KeyPairInfo

The code

```
using System;
using System.Threading.Tasks;
using System.IO;
using Amazon.EC2;
using Amazon.EC2.Model;
using System.Collections.Generic;

namespace EC2CreateKeyPair
{
```

```
 // = = =
```

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

```
 // = = =
```

```
        }
      }

      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);
    }
  }

}
```

Additional considerations

- After you run the example, you can see the new key pair in the Amazon EC2 console.

- When you create a key pair, you must save the private key that is returned because you can't retrieve the private key later.

Deleting key pairs

This example shows you how to use the AWS SDK for .NET to delete a key pair. The application takes the name of a key pair. It deletes the key pair and then displays all available key pairs. If you provide no command-line arguments, the application simply displays all available key pairs.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Delete the key pair
- Display available key pairs
- Complete code

Delete the key pair

The following snippet deletes a key pair.

The example at the end of this topic shows this snippet in use.

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

Display available key pairs

The following snippet displays a list of the available key pairs.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.EC2

Programming elements:

- Namespace Amazon.EC2

  Class AmazonEC2Client
- Namespace Amazon.EC2.Model

  Class DeleteKeyPairRequest

  Class DescribeKeyPairsResponse

  Class KeyPairInfo

The code

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

```
}
```

Seeing your Amazon EC2 Regions and Availability Zones

Amazon EC2 is hosted in multiple locations worldwide. These locations are composed of Regions and Availability Zones. Each Region is a separate geographic area that has multiple, isolated locations known as Availability Zones.

To read more about Regions and Availability Zones, see Regions and Zones in the Amazon EC2 User Guide.

This example shows you how to use the AWS SDK for .NET to get details about the Regions and Availability Zones related to an EC2 client. The application displays lists of the Regions and Availability Zones available to an EC2 client.

SDK references

NuGet packages:

- AWSSDK.EC2

Programming elements:

- Namespace Amazon.EC2

  Class AmazonEC2Client
- Namespace Amazon.EC2.Model

  Class DescribeAvailabilityZonesResponse

  Class DescribeRegionsResponse

  Class AvailabilityZone

  Class Region

```
using System;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;
```

```
namespace EC2RegionsAndZones
{
  class Program
  {
    static async Task Main(string[] args)
    {
      Console.WriteLine(
        "Finding the Regions and Availability Zones available to an EC2 client...");

      // Create the EC2 client
      var ec2Client = new AmazonEC2Client();

      // Display the Regions and Availability Zones
      await DescribeRegions(ec2Client);
      await DescribeAvailabilityZones(ec2Client);
    }


    //
    // Method to display Regions
    private static async Task DescribeRegions(IAmazonEC2 ec2Client)
    {
      Console.WriteLine("\nRegions that are enabled for the EC2 client:");
      DescribeRegionsResponse response = await ec2Client.DescribeRegionsAsync();
      foreach (Region region in response.Regions)
        Console.WriteLine(region.RegionName);
    }


    //
    // Method to display Availability Zones
    private static async Task DescribeAvailabilityZones(IAmazonEC2 ec2Client)
    {
      Console.WriteLine("\nAvailability Zones for the EC2 client's region:");
      DescribeAvailabilityZonesResponse response =
        await ec2Client.DescribeAvailabilityZonesAsync();
      foreach (AvailabilityZone az in response.AvailabilityZones)
        Console.WriteLine(az.ZoneName);
    }
  }
}
```

Working with Amazon EC2 instances

You can use the AWS SDK for .NET to control Amazon EC2 instances with operations such as create, start, and terminate. The topics in this section provide some examples of how to do this. To read more about EC2 instances, see Amazon EC2 instances in the Amazon EC2 User Guide.

For information about the APIs and prerequisites, see the parent section (Working with Amazon EC2).

Topics

- Launching an Amazon EC2 instance
- Terminating an Amazon EC2 instance

Launching an Amazon EC2 instance

This example shows you how to use the AWS SDK for .NET to launch one or more identically configured Amazon EC2 instances from the same Amazon Machine Image (AMI). Using several inputs that you supply, the application launches an EC2 instance and then monitors the instance until it's out of the "Pending" state.

When your EC2 instance is running, you can connect to it remotely, as described in (optional) Connect to the instance.

> **Warning**
>
> EC2-Classic was retired on August 15, 2022. We recommend that you migrate from EC2-Classic to a VPC. For more information, see the blog post EC2-Classic Networking is Retiring – Here's How to Prepare.

The following sections provide snippets and other information for this example. The complete code for the example is shown after the snippets, and can be built and run as is.

Topics

- Gather what you need
- Launch an instance
- Monitor the instance

- Complete code
- Additional considerations
- (optional) Connect to the instance
- Clean up

Gather what you need

To launch an EC2 instance, you'll need several things.

- A VPC where the instance will be launched. If it'll be a Windows instance and you'll be connecting to it through RDP, the VPC will most likely need to have an internet gateway attached to it, as well as an entry for the internet gateway in the route table. For more information, see Internet gateways in the *Amazon VPC User Guide*.

- The ID of an existing subnet in the VPC where the instance will be launched. An easy way to find or create this is to sign in to the Amazon VPC console, but you can also obtain it programmatically by using the CreateSubnetAsync and DescribeSubnetsAsync methods.

  > **Note**
  >
  > If you don't supply this parameter, the new instance is launched in the default VPC for your account.

- The ID of an existing security group that belongs to the VPC where the instance will be launched. For more information, see Working with security groups in Amazon EC2.

- If you want to connect to the new instance, the security group mentioned earlier must have an appropriate inbound rule that allows SSH traffic on port 22 (Linux instance) or RDP traffic on port 3389 (Windows instance). For information about how to do this see Updating security groups, including the Additional considerations near the end of that topic.

- The Amazon Machine Image (AMI) that will be used to create the instance. For information about AMIs, see Amazon Machine Images (AMIs) in the Amazon EC2 User Guide. In particular, see Find an AMI and Shared AMIs.

- The name of an existing EC2 key pair, which is used to connect to the new instance. For more information, see Working with Amazon EC2 key pairs.

- The name of the PEM file that contains the private key of the EC2 key pair mentioned earlier. The PEM file is used when you connect remotely to the instance.

Launch an instance

The following snippet launches an EC2 instance.

The example near the end of this topic shows this snippet in use.

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

Monitor the instance

The following snippet monitors the instance until it's out of the "Pending" state.

The example near the end of this topic shows this snippet in use.

See the InstanceState class for the valid values of the `Instance.State.Code` property.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.EC2

Programming elements:

- Namespace Amazon.EC2

  Class AmazonEC2Client

  Class InstanceType
- Namespace Amazon.EC2.Model

  Class DescribeInstancesRequest

  Class DescribeInstancesResponse

  Class Instance

  Class InstanceNetworkInterfaceSpecification

  Class RunInstancesRequest

  Class RunInstancesResponse

The code

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

```
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2LaunchInstance
{
  // = = =
```

```
      // Launch the instances and wait for them to start running
      var instanceIds = await LaunchInstances(ec2Client, request);
      await CheckState(ec2Client, instanceIds);
    }


    //
    // Method to put together the properties needed to launch the instance.
    private static RunInstancesRequest GetRequestData(
      string groupID, string ami, string keyPairName, string subnetID)
    {
      // Common properties
      var groupIDs = new List<string>() { groupID };
      var request = new RunInstancesRequest()
      {
        // The first three of these would be additional command-line arguments or
similar.
        InstanceType = InstanceType.T1Micro,
        MinCount = 1,
        MaxCount = 1,
        ImageId = ami,
        KeyName = keyPairName
      };

      // Properties specifically for EC2 in a VPC.
      if(!string.IsNullOrEmpty(subnetID))
      {
        request.NetworkInterfaces =
          new List<InstanceNetworkInterfaceSpecification>() {
            new InstanceNetworkInterfaceSpecification() {
              DeviceIndex = 0,
              SubnetId = subnetID,
              Groups = groupIDs,
              AssociatePublicIpAddress = true
            }
          };
      }

      // Properties specifically for EC2-Classic
      else
      {
        request.SecurityGroupIds = groupIDs;
      }
```

```
      return request;
    }


    //
    // Method to launch the instances
    // Returns a list with the launched instance IDs
    private static async Task<List<string>> LaunchInstances(
      IAmazonEC2 ec2Client, RunInstancesRequest requestLaunch)
    {
      var instanceIds = new List<string>();
      RunInstancesResponse responseLaunch =
        await ec2Client.RunInstancesAsync(requestLaunch);

      Console.WriteLine("\nNew instances have been created.");
      foreach (Instance item in responseLaunch.Reservation.Instances)
      {
        instanceIds.Add(item.InstanceId);
        Console.WriteLine($"  New instance: {item.InstanceId}");
      }

      return instanceIds;
    }


    //
    // Method to wait until the instances are running (or at least not pending)
    private static async Task CheckState(IAmazonEC2 ec2Client, List<string>
instanceIds)
    {
      Console.WriteLine(
        "\nWaiting for the instances to start." +
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");

      int numberRunning;
      DescribeInstancesResponse responseDescribe;
      var requestDescribe = new DescribeInstancesRequest{
        InstanceIds = instanceIds};

      // Check every couple of seconds
      int wait = 2000;
      while(true)
      {
```

```
        // Get and check the status for each of the instances to see if it's past
pending.
        // Once all instances are past pending, break out.
        // (For this example, we are assuming that there is only one reservation.)
        Console.Write(".");
        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach(Instance i in responseDescribe.Reservations[0].Instances)
        {
          // Check the lower byte of State.Code property
          // Code == 0 is the pending state
          if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
          break;

        // Wait a bit and try again (unless the user wants to stop waiting)
        Thread.Sleep(wait);
        if(Console.KeyAvailable)
          break;
      }

      Console.WriteLine("\nNo more instances are pending.");
      foreach(Instance i in responseDescribe.Reservations[0].Instances)
      {
        Console.WriteLine($"For {i.InstanceId}:");
        Console.WriteLine($"  VPC ID: {i.VpcId}");
        Console.WriteLine($"  Instance state: {i.State.Name}");
        Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
        Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
        Console.WriteLine($"  Key pair name: {i.KeyName}");
      }
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2LaunchInstance -g <group-id> -a <ami-id> -k <keypair-name> [-s
<subnet-id>]" +
        "\n  -g, --group-id: The ID of the security group." +
        "\n  -a, --ami-id: The ID of an Amazon Machine Image." +
```

```
        "\n  -k, --keypair-name - The name of a key pair." +
        "\n  -s, --subnet-id: The ID of a subnet. Required only for EC2 in a VPC.");
    }
  }


  // = = =
```

```
        // If the first argument in this iteration doesn't start with a dash, it's a
 value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Additional considerations

- When checking the state of an EC2 instance, you can add a filter to the `Filter` property of the DescribeInstancesRequest object. Using this technique, you can limit the request to certain instances; for example, instances with a particular user-specified tag.

- For brevity, some properties were given typical values. Any or all of these properties can instead be determined programmatically or by user input.

- The values you can use for the `MinCount` and `MaxCount` properties of the RunInstancesRequest object are determined by the target Availability Zone and the maximum number of instances you're allowed for the instance type. For more information, see How many instances can I run in Amazon EC2 in the Amazon EC2 General FAQ.

- If you want to use a different instance type than this example, there are several instance types to choose from. For more information see Amazon EC2 instance types in the Amazon EC2 User Guide. Also see Instance Type Details and Instance Type Explorer.

- You can also attach an IAM role to an instance when you launch it. To do so, create an IamInstanceProfileSpecification object whose `Name` property is set to the name of an IAM role. Then add that object to the `IamInstanceProfile` property of the RunInstancesRequest object.

  > **Note**
  >
  > To launch an EC2 instance that has an IAM role attached, an IAM user's configuration must include certain permissions. For more information about the required permissions, see the Grant a user permission to pass an IAM role to an instance in the Amazon EC2 User Guide.

(optional) Connect to the instance

After an instance is running, you can connect to it remotely by using the appropriate remote client. For both Linux and Windows instances, you need the instance's public IP address or public DNS name. You also need the following.

For Linux instances

You can use an SSH client to connect to your Linux instance. Make sure that the security group you used when you launched the instance allows SSH traffic on port 22, as described in Updating security groups.

You also need the private portion of the key pair you used to launch the instance; that is, the PEM file.

For more information, see Connect to your Linux instance in the Amazon EC2 User Guide.

For Windows instances

You can use an RDP client to connect to your instance. Make sure that the security group you used when you launched the instance allows RDP traffic on port 3389, as described in Updating security groups.

You also need the Administrator password. You can obtain this by using the following example code, which requires the instance ID and the private portion of the key pair used to launch the instance; that is, the PEM file.

For more information, see Connect to your Windows instance in the Amazon EC2 User Guide.

> **Warning**
>
> This example code returns the plaintext Administrator password for your instance.

SDK references

NuGet packages:

- AWSSDK.EC2

Programming elements:

- Namespace Amazon.EC2

  Class AmazonEC2Client
- Namespace Amazon.EC2.Model

  Class GetPasswordDataRequest

Class GetPasswordDataResponse

The code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2GetWindowsPassword
{
  // = = =
```

```
      // 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);
    }
  }

}
```

Clean up

When you no longer need your EC2 instance, be sure to terminate it, as described in Terminating an Amazon EC2 instance.

Terminating an Amazon EC2 instance

When you no longer need one or more of your Amazon EC2 instances, you can terminate them.

This example shows you how to use the AWS SDK for .NET to terminate EC2 instances. It takes an instance ID as input.

SDK references

NuGet packages:

- AWSSDK.EC2

Programming elements:

- Namespace Amazon.EC2

  Class AmazonEC2Client
- Namespace Amazon.EC2.Model

  Class TerminateInstancesRequest

  Class 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);
      }
    }
  }
}
```

After you run the example, it's a good idea to sign in to the Amazon EC2 console to verify that the EC2 instance has been terminated.

Amazon EC2 Spot Instance tutorial

This tutorial shows you how to use the AWS SDK for .NET to manage Amazon EC2 Spot Instances.

Overview

Spot Instances enable you to request unused Amazon EC2 capacity for less than the On-Demand price. This can significantly lower your EC2 costs for applications that can be interrupted.

The following is a high-level summary of how Spot Instances are requested and used.

1. Create a Spot Instance request, specifying the maximum price you are willing to pay.
2. When the request is fulfilled, run the instance as you would any other Amazon EC2 instance.
3. Run the instance as long as you want and then terminate it, unless the *Spot Price* changes such that the instance is terminated for you.

4. Clean up the Spot Instance request when you no longer need it so that Spot Instances are no longer created.

This has been a very high level overview of Spot Instances. To gain a better understanding of Spot Instances, see Spot Instances in the Amazon EC2 User Guide.

About this tutorial

As you follow this tutorial, you use the AWS SDK for .NET to do the following:

- Create a Spot Instance request
- Determine when the Spot Instance request has been fulfilled
- Cancel the Spot Instance request
- Terminate associated instances

The following sections provide snippets and other information for this example. The complete code for the example is shown after the snippets, and can be built and run as is.

Topics

- Prerequisites
- Gather what you need
- Creating a Spot Instance request
- Determine the state of your Spot Instance request
- Clean up your Spot Instance requests
- Clean up your Spot Instances
- Complete code
- Additional considerations

Prerequisites

For information about the APIs and prerequisites, see the parent section (Working with Amazon EC2).

Gather what you need

To create a Spot Instance request, you'll need several things.

- The number of instances and their instance type. There are several instance types to choose from. For more information, see Amazon EC2 instance types in the Amazon EC2 User Guide. Also see Instance Type Details and Instance Type Explorer.

  The default number for this tutorial is 1.

- The Amazon Machine Image (AMI) that will be used to create the instance. For information about AMIs, see Amazon Machine Images (AMIs) in the Amazon EC2 User Guide. In particular, see Find an AMI and Shared AMIs.

- The maximum price that you're willing to pay per instance hour. You can see the prices for all instance types (for both On-Demand Instances and Spot Instances) on the Amazon EC2 pricing page. The default price for this tutorial is explained later.

- If you want to connect remotely to an instance, a security group with the appropriate configuration and resources. This is described in Working with security groups in Amazon EC2 and the information about gathering what you need and connecting to an instance in Launching an Amazon EC2 instance. For simplicity, this tutorial uses the security group named **default** that all newer AWS accounts have.

There are many ways to approach requesting Spot Instances. The following are common strategies:

- Make requests that are sure to cost less than on-demand pricing.
- Make requests based on the value of the resulting computation.
- Make requests so as to acquire computing capacity as quickly as possible.

The following explanations refer to the Spot Instance pricing history in the Amazon EC2 User Guide.

Reduce cost below On-Demand

You have a batch processing job that will take a number of hours or days to run. However, you are flexible with respect to when it starts and ends. You want to see if you can complete it for less than the cost of On-Demand Instances.

You examine the Spot Price history for instance types by using either the Amazon EC2 console or the Amazon EC2 API. After you've analyzed the price history for your desired instance type in a given Availability Zone, you have two alternative approaches for your request:

- Specify a request at the upper end of the range of Spot Prices, which are still below the On-Demand price, anticipating that your one-time Spot Instance request would most likely be fulfilled and run for enough consecutive compute time to complete the job.
- Specify a request at the lower end of the price range, and plan to combine many instances launched over time through a persistent request. The instances would run long enough, in aggregate, to complete the job at an even lower total cost.

Pay no more than the value of the result

You have a data processing job to run. You understand the value of the job's results well enough to know how much they're worth in terms of computing costs.

After you've analyzed the Spot Price history for your instance type, you choose a price at which the cost of the computing time is no more than the value of the job's results. You create a persistent request and allow it to run intermittently as the Spot Price fluctuates at or below your request.

Acquire computing capacity quickly

You have an unanticipated, short-term need for additional capacity that's not available through On-Demand Instances. After you've analyzed the Spot Price history for your instance type, you choose a price above the highest historical price to greatly improve the likelihood that your request will be fulfilled quickly and continue computing until it's complete.

After you have gathered what you need and chosen a strategy, you are ready to request a Spot Instance. For this tutorial the default maximum spot-instance price is set to be the same as the On-Demand price (which is $0.003 for this tutorial). Setting the price in this way maximizes the chances that the request will be fulfilled.

Creating a Spot Instance request

The following snippet shows you how to create a Spot Instance request with the elements you gathered earlier.

The example at the end of this topic shows this snippet in use.

```
    //
```

```
    // 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];
    }
```

The important value returned from this method is the Spot Instance request ID, which is contained in the `SpotInstanceRequestId` member of the returned SpotInstanceRequest object.

> **Note**
>
> You will be charged for any Spot Instances that are launched. To avoid unnecessary costs be sure to cancel any requests and terminate any instances.

Determine the state of your Spot Instance request

The following snippet shows you how to get information about your Spot Instance request. You can use that information to make certain decisions in your code, such as whether to continue waiting for a Spot Instance request to be fulfilled.

The example at the end of this topic shows this snippet in use.

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

The method returns information about the Spot Instance request such as the instance ID, it's state, and the status code. For more information about the status codes for Spot Instance requests, see Spot request status in the Amazon EC2 User Guide.

Clean up your Spot Instance requests

When you no longer need to request Spot Instances, it's important to cancel any outstanding requests to prevent those requests from being re-fulfilled. The following snippet shows you how to cancel a Spot Instance request.

The example at the end of this topic shows this snippet in use.

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

Clean up your Spot Instances

To avoid unnecessary costs, it's important that you terminate any instances that were started from Spot Instance requests; simply canceling Spot Instance requests will not terminate your instances, which means that you'll continue to be charged for them. The following snippet shows you how to terminate an instance after you obtain the instance identifier for an active Spot Instance.

The example at the end of this topic shows this snippet in use.

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

Complete code

The following code example calls the methods described earlier to create and cancel a Spot Instance request and terminate a Spot Instance.

SDK references

NuGet packages:

- AWSSDK.EC2

Programming elements:

- Namespace Amazon.EC2

  Class AmazonEC2Client

  Class InstanceType

- Namespace Amazon.EC2.Model

  Class CancelSpotInstanceRequestsRequest

  Class DescribeSpotInstanceRequestsRequest

  Class DescribeSpotInstanceRequestsResponse

  Class InstanceStateChange

  Class LaunchSpecification

  Class RequestSpotInstancesRequest

  Class RequestSpotInstancesResponse

  Class SpotInstanceRequest

  Class TerminateInstancesRequest

  Class TerminateInstancesResponse

The code

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

Additional considerations

- After you run the tutorial, it's a good idea to sign in to the Amazon EC2 console to verify that the Spot Instance request has been canceled and that the Spot Instance has been terminated.

Accessing AWS Identity and Access Management (IAM) with the AWS SDK for .NET

The AWS SDK for .NET supports AWS Identity and Access Management, which is a web service that enables AWS customers to manage users and user permissions in AWS.

An AWS Identity and Access Management (IAM) *user* is an entity that you create in AWS. The entity represents a person or application that interacts with AWS. For more information about IAM users, see IAM Users and IAM and STS Limits in the IAM User Guide.

You grant permissions to a user by creating an IAM *policy*. The policy contains a *policy document* that lists the actions that a user can perform and the resources those actions can affect. For more information about IAM policies, see Policies and Permissions in the *IAM User Guide*.

> **Warning**
>
> To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

APIs

The AWS SDK for .NET provides APIs for IAM clients. The APIs enable you to work with IAM features such as users, roles, and access keys.

This section contains a small number of examples that show you the patterns you can follow when working with these APIs. To view the full set of APIs, see the AWS SDK for .NET API Reference (and scroll to "Amazon.IdentityManagement").

This section also contains an example that shows you how to attach an IAM role to Amazon EC2 instances to make managing credentials easier.

The IAM APIs are provided by the AWSSDK.IdentityManagement NuGet package.

Prerequisites

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

Topics

Topics

- Creating IAM managed policies from JSON
- Display the policy document of an IAM managed policy
- Granting access by using an IAM role

Creating IAM managed policies from JSON

This example shows you how to use the AWS SDK for .NET to create an IAM managed policy from a given policy document in JSON. The application creates an IAM client object, reads the policy document from a file, and then creates the policy.

> **Note**
>
> For an example policy document in JSON, see the additional considerations at the end of this topic.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Create the policy
- Complete code
- Additional considerations

Create the policy

The following snippet creates an IAM managed policy with the given name and policy document.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.IdentityManagement

Programming elements:

- Namespace Amazon.IdentityManagement

  Class AmazonIdentityManagementServiceClient
- Namespace Amazon.IdentityManagement.Model

  Class CreatePolicyRequest

  Class CreatePolicyResponse

The code

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

namespace IamCreatePolicyFromJson
{
  // = = =
```

```
        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.");
    }
 }


 // = = =
```

```
        // 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);
    }
  }

}
```

Additional considerations

- The following is an example policy document that you can copy into a JSON file and use as input for this application:

  JSON

```
{
  "Version":"2012-10-17",
  "Id"  : "DotnetTutorialPolicy",
  "Statement" : [
    {
      "Sid" : "DotnetTutorialPolicyS3",
      "Effect" : "Allow",
      "Action" : [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource" : "*"
    },
    {
      "Sid" : "DotnetTutorialPolicyPolly",
      "Effect": "Allow",
      "Action": [
        "polly:DescribeVoices",
        "polly:SynthesizeSpeech"
      ],
      "Resource": "*"
    }
  ]
}
```

- You can verify that the policy was created by looking in the IAM console. In the **Filter policies** drop-down list, select **Customer managed**. Delete the policy when you no longer need it.

- For more information about policy creation, see Creating IAM policies and the IAM JSON policy reference in the IAM User Guide

Display the policy document of an IAM managed policy

This example shows you how to use the AWS SDK for .NET to display a policy document. The application creates an IAM client object, finds the default version of the given IAM managed policy, and then displays the policy document in JSON.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Find the default version
- Display the policy document
- Complete code

Find the default version

The following snippet finds the default version of the given IAM policy.

The example at the end of this topic shows this snippet in use.

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

Display the policy document

The following snippet displays the policy document in JSON of the given IAM policy.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.IdentityManagement

Programming elements:

- Namespace Amazon.IdentityManagement

  Class AmazonIdentityManagementServiceClient

- Namespace Amazon.IdentityManagement.Model

  Class GetPolicyVersionRequest

  Class GetPolicyVersionResponse

  Class ListPolicyVersionsRequest

  Class ListPolicyVersionsResponse

  Class PolicyVersion

The code

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

Granting access by using an IAM role

This tutorial shows you how to use the AWS SDK for .NET to enable IAM roles on Amazon EC2 instances.

Overview

All requests to AWS must be cryptographically signed by using credentials issued by AWS. Therefore, you need a strategy to manage credentials for applications that run on Amazon EC2 instances. You have to distribute, store, and rotate these credentials securely, but also keep them accessible to the applications.

With IAM roles, you can effectively manage these credentials. You create an IAM role and configure it with the permissions that an application requires, and then attach that role to an EC2 instance. To read more about the benefits of using IAM roles, see IAM roles for Amazon EC2 in the Amazon EC2 User Guide. Also see the information about IAM Roles in the IAM User Guide.

For an application that is built using the AWS SDK for .NET, when the application constructs a client object for an AWS service, the object searches for credentials from several potential sources. The order in which it searches is shown in Credential and profile resolution.

If the client object doesn't find credentials from any other source, it retrieves temporary credentials that have the same permissions as those that have been configured into the IAM role and are in the metadata of the EC2 instance. These credentials are used to make calls to AWS from the client object.

About this tutorial

As you follow this tutorial, you use the AWS SDK for .NET (and other tools) to launch an Amazon EC2 instance with an IAM role attached, and then see an application on the instance using the permissions of the IAM role.

Topics

- Create a sample Amazon S3 application
- Create an IAM role
- Launch an EC2 instance and attach the IAM role
- Connect to the EC2 instance
- Run the sample application on the EC2 instance
- Clean up

Create a sample Amazon S3 application

This sample application retrieves an object from Amazon S3. To run the application, you need the following:

- An Amazon S3 bucket that contains a text file.
- AWS credentials on your development machine that allow you to access to the bucket.

For information about creating an Amazon S3 bucket and uploading an object, see the Amazon Simple Storage Service User Guide. For information about AWS credentials, see Authenticating the AWS SDK for .NET with AWS.

Create a .NET Core project with the following code. Then test the application on your development machine.

> **Note**
>
> On your development machine, the .NET Core Runtime is installed, which enables you to run the application without publishing it. When you create an EC2 instance later in this tutorial, you can choose to install the .NET Core Runtime on the instance. This gives you a similar experience and a smaller file transfer.

However, you can also choose not to install the .NET Core Runtime on the instance. If you choose this course of action, you must publish the application so that all dependencies are included when you transfer it to the instance.

SDK references

NuGet packages:

- AWSSDK.S3

Programming elements:

- Namespace Amazon.S3

  Class AmazonS3Client
- Namespace Amazon.S3.Model

  Class GetObjectResponse

The code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

namespace S3GetTextItem
{
  // = = =
```

```
    {
      PrintHelp();
      return;
    }

    // Get the application arguments from the parsed list
    string bucket =
      CommandLine.GetArgument(parsedArgs, null, "-b", "--bucket-name");
    string item =
      CommandLine.GetArgument(parsedArgs, null, "-t", "--text-object");
    string outFile =
      CommandLine.GetArgument(parsedArgs, null, "-o", "--output-filename");
    if(   string.IsNullOrEmpty(bucket)
       || string.IsNullOrEmpty(item)
       || string.IsNullOrEmpty(outFile))
      CommandLine.ErrorExit(
        "\nOne or more of the required arguments is missing or incorrect." +
        "\nRun the command with no arguments to see help.");

    // Create the S3 client object and get the file object from the bucket.
    var response = await GetObject(new AmazonS3Client(), bucket, item);

    // Write the contents of the file object to the given output file.
    var reader = new StreamReader(response.ResponseStream);
    string contents = reader.ReadToEnd();
    using (var s = new FileStream(outFile, FileMode.Create))
    using (var writer = new StreamWriter(s))
      writer.WriteLine(contents);
  }


  //
  // Method to get an object from an S3 bucket.
  private static async Task<GetObjectResponse> GetObject(
    IAmazonS3 s3Client, string bucket, string item)
  {
      Console.WriteLine($"Retrieving {item} from bucket {bucket}.");
      return await s3Client.GetObjectAsync(bucket, item);
  }


  //
  // Command-line help
  private static void PrintHelp()
```

```
    {
      Console.WriteLine(
        "\nUsage: S3GetTextItem -b <bucket-name> -t <text-object> -o <output-filename>"
+
        "\n  -b, --bucket-name: The name of the S3 bucket." +
        "\n  -t, --text-object: The name of the text object in the bucket." +
        "\n  -o, --output-filename: The name of the file to write the text to.");
    }
  }


  // = = =
```

```
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }
```

```
}
```

If you want, you can temporarily remove the credentials you use on your development machine to see how the application responds. (But be sure to restore the credentials when you're finished.)

Create an IAM role

Create an IAM role that has the appropriate permissions to access Amazon S3.

1. Open the IAM console.
2. In the navigation pane, choose **Roles**, and then choose **Create Role**.
3. Select **AWS service**, find and choose **EC2**, and choose **Next: Permissions**.
4. Under **Attach permissions policies**, find and select **AmazonS3ReadOnlyAccess**. Review the policy if you want to, and then choose **Next: Tags**.
5. Add tags if you want and then choose **Next: Review**.
6. Type a name and description for the role, and then choose **Create role**. Remember this name because you'll need it when you launch your EC2 instance.

Launch an EC2 instance and attach the IAM role

Launch an EC2 instance with the IAM role you created previously. You can do so in the following ways.

- **Using the EC2 console**

  To launch an instance by using the EC2 console, see Launch an instance using the new launch instance wizard in the Amazon EC2 User Guide.

  As you look through the launch page, you should at least expand the **Advanced details** pane so that you can specify the IAM role you created earlier in **IAM instance profile**.

- **Using the AWS SDK for .NET**

  For information about this, see Launching an Amazon EC2 instance, including the Additional considerations near the end of that topic.

To launch an EC2 instance that has an IAM role attached, an IAM user's configuration must include certain permissions. For more information about the required permissions, see Grant a user permission to pass an IAM role to an instance in the Amazon EC2 User Guide.

Connect to the EC2 instance

Connect to the EC2 instance so that you can transfer the sample application to it and then run the application. You'll need the file that contains the private portion of the key pair you used to launch the instance; that is, the PEM file.

For information about connecting to an instance, see Connect to your Linux instance or Connect to your Windows instance in the Amazon EC2 User Guide. When you connect, do so in such a way that you can transfer files from your development machine to your instance.

If you're using Visual Studio on Windows, you can also connect to the instance by using the Toolkit for Visual Studio. For more information, see Connecting to an Amazon EC2 Instance in the AWS Toolkit for Visual Studio User Guide.

Run the sample application on the EC2 instance

1. Copy the application files from your local drive to your instance.

   Which files you transfer depends on how you built the application and whether your instance has the .NET Core Runtime installed. For information about how to transfer files to your instance, see Connect to your Linux instance (see the appropriate sub-section) or Transfer files to Windows instances in the Amazon EC2 User Guide.
2. Start the application and verify that it runs with the same results as on your development machine.
3. Verify that the application uses the credentials provided by the IAM role.

   a. Open the Amazon EC2 console.

   b. Select the instance and detach the IAM role through **Actions**, **Instance Settings**, **Attach/Replace IAM Role**.

   c. Run the application again and see that it returns an authorization error.

Clean up

When you are finished with this tutorial, and if you no longer want the EC2 instance you created, be sure to terminate the instance to avoid unwanted cost. You can do so in the Amazon EC2

console or programmatically, as described in Terminating an Amazon EC2 instance. If you want to, you can also delete other resources that you created for this tutorial. These might include an IAM role, an EC2 keypair and PEM file, a security group, etc.

Using Amazon Simple Storage Service Internet storage

The AWS SDK for .NET supports Amazon S3, which is storage for the Internet. It is designed to make web-scale computing easier for developers.

APIs

The AWS SDK for .NET provides APIs for Amazon S3 clients. The APIs enable you to work with Amazon S3 resources such as buckets and items. To view the full set of APIs for Amazon S3, see the following:

- AWS SDK for .NET API Reference (and scroll to "Amazon.S3").
- Amazon.Extensions.S3.Encryption documentation

The Amazon S3 APIs are provided by the following NuGet packages:

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

Prerequisites

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

Examples in this document

The following topics in this document show you how to use the AWS SDK for .NET to work with Amazon S3.

- Using KMS keys for S3 encryption

Examples in other documents

The following links to the Amazon S3 Developer Guide provide additional examples of how to use the AWS SDK for .NET to work with Amazon S3.

> **Note**
>
> Although these examples and additional programming considerations were created for Version 3 of the AWS SDK for .NET using .NET Framework, they are also viable for later versions of the AWS SDK for .NET using .NET Core. Small adjustments in the code are sometimes necessary.

Amazon S3 programming examples

- Managing ACLs
- Creating a Bucket
- Upload an Object
- Multipart Upload with the High-Level API (Amazon.S3.Transfer.TransferUtility)
- Multipart Upload with the Low-Level API
- Listing Objects
- Listing Keys
- Get an Object
- Copy an Object
- Copy an Object with the Multipart Upload API
- Deleting an Object
- Deleting Multiple Objects
- Restore an Object
- Configure a Bucket for Notifications
- Manage an Object's Lifecycle
- Generate a Pre-signed Object URL
- Managing Websites
- Enabling Cross-Origin Resource Sharing (CORS)

Additional programming considerations

- Using the AWS SDK for .NET for Amazon S3 Programming

- Making Requests Using IAM User Temporary Credentials
- Making Requests Using Federated User Temporary Credentials
- Specifying Server-Side Encryption
- Specifying Server-Side Encryption with Customer-Provided Encryption Keys

Using AWS KMS keys for Amazon S3 encryption in the AWS SDK for .NET

This example shows you how to use AWS Key Management Service keys to encrypt Amazon S3 objects. The application creates a customer master key (CMK) and uses it to create an AmazonS3EncryptionClientV2 object for client-side encryption. The application uses that client to create an encrypted object from a given text file in an existing Amazon S3 bucket. It then decrypts the object and displays its contents.

> **Warning**
>
> A similar class called `AmazonS3EncryptionClient` is deprecated and is less secure than the `AmazonS3EncryptionClientV2` class. To migrate existing code that uses `AmazonS3EncryptionClient`, see S3 Encryption Client Migration (V1 to V2).

Topics

- Create encryption materials
- Create and encrypt an Amazon S3 object
- Complete code
- Additional considerations

Create encryption materials

The following snippet creates an `EncryptionMaterials` object that contains a KMS key ID.

The example at the end of this topic shows this snippet in use.

```
      // 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 and encrypt an Amazon S3 object

The following snippet creates an `AmazonS3EncryptionClientV2` object that uses the encryption materials created earlier. It then uses the client to create and encrypt a new Amazon S3 object.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- Amazon.Extensions.S3.Encryption

Programming elements:

- Namespace Amazon.Extensions.S3.Encryption

  Class AmazonS3EncryptionClientV2

  Class AmazonS3CryptoConfigurationV2

  Class CryptoStorageMode

  Class EncryptionMaterialsV2
- Namespace Amazon.Extensions.S3.Encryption.Primitives

  Class KmsType
- Namespace Amazon.S3.Model

  Class GetObjectRequest

  Class GetObjectResponse

  Class PutObjectRequest
- Namespace Amazon.KeyManagementService

  Class AmazonKeyManagementServiceClient
- Namespace Amazon.KeyManagementService.Model

  Class CreateKeyRequest

  Class CreateKeyResponse

The code

```
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
{
  // = = =
```

```
      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);
    }
  }

}
```

Additional considerations

- You can check the results of this example. To do so, go to the Amazon S3 console and open the bucket you provided to the application. Then find the new object, download it, and open it in a text editor.

- The AmazonS3EncryptionClientV2 class implements the same interface as the standard `AmazonS3Client` class. This makes it easier to port your code to the `AmazonS3EncryptionClientV2` class so that encryption and decryption happen automatically and transparently in the client.

- One advantage of using an AWS KMS key as your master key is that you don't need to store and manage your own master keys; this is done by AWS. A second advantage is that the `AmazonS3EncryptionClientV2` class of the AWS SDK for .NET is interoperable with the `AmazonS3EncryptionClientV2` class of the AWS SDK for Java. This means you can encrypt with the AWS SDK for Java and decrypt with the AWS SDK for .NET, and vice versa.

  > **Note**
  >
  > The `AmazonS3EncryptionClientV2` class of the AWS SDK for .NET supports KMS master keys only when run in metadata mode. The instruction file mode of the `AmazonS3EncryptionClientV2` class of the AWS SDK for .NET is incompatible with the `AmazonS3EncryptionClientV2` class of the AWS SDK for Java.

- For more information about client-side encryption with the `AmazonS3EncryptionClientV2` class, and how envelope encryption works, see Client Side Data Encryption with AWS SDK for .NET and Amazon S3.

Sending Notifications From the Cloud Using Amazon Simple Notification Service

> **Note**
>
> The information in this topic is specific to projects based on .NET Framework and the AWS SDK for .NET version 3.3 and earlier.

The AWS SDK for .NET supports Amazon Simple Notification Service (Amazon SNS), which is a web service that enables applications, end users, and devices to instantly send notifications from the cloud. For more information, see Amazon SNS.

Listing Your Amazon SNS Topics

The following example shows how to list your Amazon SNS topics, the subscriptions to each topic, and the attributes for each topic. This example uses the default AmazonSimpleNotificationServiceClient.

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

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

do
{
  response = client.ListTopics(request);

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

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

    var ss = subs.Subscriptions;
```

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

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

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

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

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

    Console.WriteLine();
  }

  request.NextToken = response.NextToken;

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

Sending a Message to an Amazon SNS Topic

The following example shows how to send a message to an Amazon SNS topic. The example takes one argument, the ARN of the Amazon SNS topic.

```
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);
            }
        }
```

```
    }
}
```

See the complete example, including information on how to build and run the example from the command line, on GitHub.

Sending an SMS Message to a Phone Number

The following example shows how to send an SMS message to a telephone number. The example takes one argument, the telephone number, which must be in either of the two formats described in the comments.

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

See the complete example, including information on how to build and run the example from the command line, on GitHub.

Messaging using Amazon SQS

The AWS SDK for .NET supports Amazon Simple Queue Service (Amazon SQS), which is a message queuing service that handles messages or workflows between components in a system.

Amazon SQS queues provide a mechanism that enables you to send, store, and receive messages between software components such as microservices, distributed systems, and serverless applications. This enables you to decouple such components and frees you from the need to design and operate your own messaging system. For information about how queues and messages work in Amazon SQS, see Amazon SQS tutorials and Basic Amazon SQS architecture in the Amazon Simple Queue Service Developer Guide.

> **Important**
>
> Due to the distributed nature of queues, Amazon SQS can't guarantee that you'll receive messages in the precise order they're sent. If you need to preserve message order, use an Amazon SQS FIFO queue.

APIs

The AWS SDK for .NET provides APIs for Amazon SQS clients. The APIs enable you to work with Amazon SQS features such as queues and messages. This section contains a small number of examples that show you the patterns you can follow when working with these APIs. To view the full set of APIs, see the AWS SDK for .NET API Reference (and scroll to "Amazon.SQS").

The Amazon SQS APIs are provided by the AWSSDK.SQS NuGet package.

Prerequisites

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

Topics

Topics

- Creating Amazon SQS queues
- Updating Amazon SQS queues
- Deleting Amazon SQS queues
- Sending Amazon SQS messages
- Receiving Amazon SQS messages

Creating Amazon SQS queues

This example shows you how to use the AWS SDK for .NET to create an Amazon SQS queue. The application creates a dead-letter queue if you don't supply the ARN for one. It then creates a standard message queue, which includes a dead-letter queue (the one you supplied or the one that was created).

If you don't supply any command-line arguments, the application simply shows information about all existing queues.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Show existing queues
- Create the queue
- Get a queue's ARN
- Complete code
- Additional considerations

Show existing queues

The following snippet shows a list of the existing queues in the SQS client's region and the attributes of each queue.

The example at the end of this topic shows this snippet in use.

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

Create the queue

The following snippet creates a queue. The snippet includes the use of a dead-letter queue, but a dead-letter queue isn't necessarily required for your queues.

The example at the end of this topic shows this snippet in use.

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

Get a queue's ARN

The following snippet gets the ARN of the queue identified by the given queue URL.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.SQS

Programming elements:

- Namespace Amazon.SQS

  Class AmazonSQSClient

  Class QueueAttributeName
- Namespace Amazon.SQS.Model

  Class CreateQueueRequest

  Class CreateQueueResponse

  Class GetQueueAttributesResponse

  Class ListQueuesResponse

The code

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSCreateQueue
{
  // = = =
```

```
    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}.");
    }
 }


 // = = =
```

```
    //  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);
    }
  }

}
```

Additional considerations

- Your queue name must be composed of alphanumeric characters, hyphens, and underscores.
- Queue names and queue URLs are case-sensitive
- If you need the queue URL but only have the queue name, use one of the `AmazonSQSClient.GetQueueUrlAsync` methods.
- For information about the various queue attributes you can set, see CreateQueueRequest in the AWS SDK for .NET API Reference or SetQueueAttributes in the Amazon Simple Queue Service API Reference.
- This example specifies long polling for all messages on the queue that you create. This is done by using the `ReceiveMessageWaitTimeSeconds` attribute.

  You can also specify long polling during a call to the `ReceiveMessageAsync` methods of the AmazonSQSClient class. For more information, see Receiving Amazon SQS messages.

  For information about short polling versus long polling, see Short and long polling in the *Amazon Simple Queue Service Developer Guide*.

- A dead letter queue is one that other (source) queues can target for messages that aren't processed successfully. For more information, see Amazon SQS dead-letter queues in the Amazon Simple Queue Service Developer Guide.

- You can also see the list of queues and the results of this example in the Amazon SQS console.

Updating Amazon SQS queues

This example shows you how to use the AWS SDK for .NET to update an Amazon SQS queue. After some checks, the application updates the given attribute with the given value, and then shows all attributes for the queue.

If only the queue URL is included in the command-line arguments, the application simply shows all attributes for the queue.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Show queue attributes
- Validate attribute name
- Update queue attribute
- Complete code
- Additional considerations

Show queue attributes

The following snippet shows the attributes of the queue identified by the given queue URL.

The example at the end of this topic shows this snippet in use.

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

Validate attribute name

The following snippet validates the name of the attribute being updated.

The example at the end of this topic shows this snippet in use.

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

Update queue attribute

The following snippet updates an attribute of the queue identified by the given queue URL.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.SQS

Programming elements:

- Namespace Amazon.SQS

  Class AmazonSQSClient

  Class QueueAttributeName
- Namespace Amazon.SQS.Model

  Class GetQueueAttributesResponse

The code

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSUpdateQueue
{
  // = = =
```

```
        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.");
```

```
    }
  }


  // = = =
```

```
        {
          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);
    }
  }

}
```

Additional considerations

- To update the `RedrivePolicy` attribute, you must quote the entire value and escape the quotes for the key/value pairs, as appropriate for you operating system.

On Windows, for example, the value is constructed in a manner similar to the following:

```
"{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\",\"maxReceiveCount\":\"10\"}"
```

Deleting Amazon SQS queues

This example shows you how to use the AWS SDK for .NET to delete an Amazon SQS queue. The application deletes the queue, waits for up to a given amount of time for the queue to be gone, and then shows a list of the remaining queues.

If you don't supply any command-line arguments, the application simply shows a list of the existing queues.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Delete the queue
- Wait for the queue to be gone
- Show a list of existing queues
- Complete code
- Additional considerations

Delete the queue

The following snippet deletes the queue identified by the given queue URL.

The example at the end of this topic shows this snippet in use.

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

Wait for the queue to be gone

The following snippet waits for the deletion process to finish, which might take 60 seconds.

The example at the end of this topic shows this snippet in use.

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

Show a list of existing queues

The following snippet shows a list of the existing queues in the SQS client's region.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.SQS

Programming elements:

- Namespace Amazon.SQS

  Class AmazonSQSClient
- Namespace Amazon.SQS.Model

  Class ListQueuesResponse

The code

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

namespace SQSDeleteQueue
{
  // = = =
```

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

```
}
```

Additional considerations

- The `DeleteQueueAsync` API call doesn't check to see if the queue you're deleting is being used as a dead-letter queue. A more sophisticated procedure could check for this.

- You can also see the list of queues and the results of this example in the Amazon SQS console.

Sending Amazon SQS messages

This example shows you how to use the AWS SDK for .NET to send messages to an Amazon SQS queue, which you can create programmatically or by using the Amazon SQS console. The application sends a single message to the queue and then a batch of messages. The application then waits for user input, which can be additional messages to send to the queue or a request to exit the application.

This example and the next example about receiving messages can be used together to see message flow in Amazon SQS.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Send a message
- Send a batch of messages
- Delete all messages from the queue
- Complete code
- Additional considerations

Send a message

The following snippet sends a message to the queue identified by the given queue URL.

The example at the end of this topic shows this snippet in use.

```
    //
```

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

Send a batch of messages

The following snippet sends a batch of messages to the queue identified by the given queue URL.

The example at the end of this topic shows this snippet in use.

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

Delete all messages from the queue

The following snippet deletes all messages from the queue identified by the given queue URL. This is also known as *purging the queue*.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.SQS

Programming elements:

- Namespace Amazon.SQS

  Class AmazonSQSClient
- Namespace Amazon.SQS.Model

  Class PurgeQueueResponse

  Class SendMessageBatchResponse

  Class SendMessageResponse

  Class SendMessageBatchRequestEntry

  Class SendMessageBatchResultEntry

The code

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSSendMessages
{
```

```
  // = = =
```

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

Additional considerations

- For information about various limitations on messages, including the allowed characters, see Quotas related to messages in the Amazon Simple Queue Service Developer Guide.

- Messages stay in queues until they are deleted or the queue is purged. When a message has been received by an application, it won't be visible in the queue even though it still exists in the queue. For more information about visibility timeouts, see Amazon SQS visibility timeout.

- In additional to the message body, you can also add attributes to messages. For more information, see Message metadata.

Receiving Amazon SQS messages

This example shows you how to use the AWS SDK for .NET to receive messages from an Amazon SQS queue, which you can create programmatically or by using the Amazon SQS console. The

application reads a single message from the queue, processes the message (in this case, displays the message body on the console), and then deletes the message from the queue. The application repeats these steps until the user types a key on the keyboard.

This example and the previous example about sending messages can be used together to see message flow in Amazon SQS.

The following sections provide snippets of this example. The complete code for the example is shown after that, and can be built and run as is.

Topics

- Receive a message
- Delete a message
- Complete code
- Additional considerations

Receive a message

The following snippet receives a message from the queue identified by the given queue URL.

The example at the end of this topic shows this snippet in use.

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

Delete a message

The following snippet deletes a message from the queue identified by the given queue URL.

The example at the end of this topic shows this snippet in use.

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

Complete code

This section shows relevant references and the complete code for this example.

SDK references

NuGet packages:

- AWSSDK.SQS

Programming elements:

- Namespace Amazon.SQS

  Class AmazonSQSClient
- Namespace Amazon.SQS.Model

  Class ReceiveMessageRequest

  Class ReceiveMessageResponse

The code

```
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);
    }
  }
}
```

Additional considerations

- To specify long polling, this example uses the `WaitTimeSeconds` property for each call to the `ReceiveMessageAsync` method.

  You can also specify long polling for all messages on a queue by using the `ReceiveMessageWaitTimeSeconds` attribute when creating or updating the queue.

  For information about short polling versus long polling, see Short and long polling in the *Amazon Simple Queue Service Developer Guide*.

- During message processing, you can use the receipt handle to change the message visibility timeout. For information about how to do this, see the `ChangeMessageVisibilityAsync` methods of the AmazonSQSClient class.

- Calling the `DeleteMessageAsync` method unconditionally removes the message from the queue, regardless of the visibility timeout setting.

Using AWS Lambda for compute service

The AWS SDK for .NET supports AWS Lambda, which lets you run code without provisioning or managing servers. For more information, see the AWS Lambda product page and the AWS Lambda Developer Guide, particularly the section for Working with C#.

APIs

The AWS SDK for .NET provides APIs for AWS Lambda. The APIs enable you to work with Lambda features such as functions, triggers, and events. To view the full set of APIs, see Lambda in the AWS SDK for .NET API Reference.

The Lambda APIs are provided by NuGet packages.

Prerequisites

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

Additional information

See Integrating AWS with .NET Aspire for information about developing with AWS Lambda through .NET Aspire.

Topics

Topics

- Using annotations to write AWS Lambda functions

Using annotations to write AWS Lambda functions

When writing Lambda functions, you sometimes need to write a large amount of handler code and update AWS CloudFormation templates, among other tasks. Lambda Annotations is a framework to help ease these burdens for .NET 6 Lambda functions, thereby making the experience of writing Lambda feel more natural in C#.

As an example of the benefit of using the Lambda Annotations framework, consider the following snippets of code that add two numbers.

Without Lambda Annotations

```
public class Functions
{
    public APIGatewayProxyResponse LambdaMathPlus(APIGatewayProxyRequest request,
 ILambdaContext context)
    {
        if (!request.PathParameters.TryGetValue("x", out var xs))
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.BadRequest
            };
        }
        if (!request.PathParameters.TryGetValue("y", out var ys))
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.BadRequest
            };
        }

        var x = int.Parse(xs);
        var y = int.Parse(ys);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = (x + y).ToString(),
            Headers = new Dictionary<string, string> { { "Content-Type", "text/
plain" } }
        };
    }
```

```
}
```

With Lambda Annotations

```
public class Functions
{
    [LambdaFunction]
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

As is shown in the example, Lambda Annotations can remove the need for certain boiler plate code.

For details about how to use the framework as well as additional information, see the following resources:

- The GitHub README for documentation on the APIs and attributes of Lambda Annotations.
- The blog post for Lambda Annotations.
- The `Amazon.Lambda.Annotations` NuGet package.
- The Photo Asset Management project on GitHub. Specifically, see the PamApiAnnotations folder and references to Lambda Annotations in the project README.

> **Note**
>
> The preceding example is specific to V3 of the AWS SDK for .NET. If you use the example with V4 of the SDK (the latest version), you might need to make adjustments according to the information in Migrating to version 4.

High-level libraries and frameworks for the AWS SDK for .NET

The following sections contain information about high-level libraries and frameworks that aren't part of the core SDK functionality. These libraries and frameworks use the core SDK functionality to create features that ease certain tasks.

If you're new to the AWS SDK for .NET, you might want to check out the Creating a simple application topic first. It gives you an introduction to the SDK.

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

Topics

- AWS Message Processing Framework for .NET
- Integrating AWS with .NET Aspire in the AWS SDK for .NET

AWS Message Processing Framework for .NET

The AWS Message Processing Framework for .NET is an AWS-native framework that simplifies the development of .NET message processing applications that use AWS services such as Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS), and Amazon EventBridge. The framework reduces the amount of boiler-plate code developers need to write, allowing you to focus on your business logic when publishing and consuming messages. For details about how the framework can simplify your development, see the blog post Introducing the AWS Message Processing Framework for .NET (Preview). The first part in particular provides a demonstration that shows the difference between using low-level API calls and using the framework.

The Message Processing Framework supports the following activities and features:

- Sending messages to SQS and publishing events to SNS and EventBridge.
- Receiving and handling messages from SQS by using a long-running poller, which is typically used in background services. This includes managing the visibility timeout while a message is being handled to prevent other clients from processing it.
- Handling messages in AWS Lambda functions.
- FIFO (first-in-first-out) SQS queues and SNS topics.
- OpenTelemetry for logging.

For details about these activities and features see the **Features** section of the blog post and the topics listed below.

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

Additional resources

- The `AWS.Messaging` package on NuGet.org.
- The API reference.
- The README file in the GitHub repo at https://github.com/aws/aws-dotnet-messaging/
- .NET dependency injection from Microsoft.
- .NET Generic Host from Microsoft.

Topics

- Get started with the AWS Message Processing Framework for .NET
- Publish messages with the AWS Message Processing Framework for .NET
- Consume messages with the AWS Message Processing Framework for .NET
- Using FIFO with the AWS Message Processing Framework for .NET
- Logging and Open Telemetry for the AWS Message Processing Framework for .NET
- Customize the AWS Message Processing Framework for .NET
- Security for the AWS Message Processing Framework for .NET

Get started with the AWS Message Processing Framework for .NET

Before you begin, be sure you have set up your environment and configured your project. Also review the information in Using the SDK.

This topic provides information that will help you get started using the Message Processing Framework. In addition to prerequisite and configuration information, a tutorial is provided that shows you how to implement a common scenario.

Prerequisites and configuration

- The credentials you provide for your application must have appropriate permissions for the messaging service and operations that it uses. For more information, see the security topics for SQS, SNS, and EventBridge in their respective developer guides. Also see the portion of the README file on GitHub that discusses specific permissions.
- To use the AWS Message Processing Framework for .NET, you must add the `AWS.Messaging` NuGet package to your project. For example:

```
dotnet add package AWS.Messaging
```

- The framework integrates with .NET's dependency injection (DI) service container. You can configure the framework during your application's startup by calling `AddAWSMessageBus` to add it to the DI container.

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll publish messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");
});
```

Tutorial

This tutorial demonstrates how to use the AWS Message Processing Framework for .NET. It creates two applications: an ASP.NET Core Minimal API that sends messages to an Amazon SQS queue when it receives a request at an API endpoint, and a long-running console application that polls for these messages and handles them.

- The instructions in this tutorial favor the .NET CLI, but you can perform this tutorial by using either cross-platform tools such as the .NET CLI or Microsoft Visual Studio. For information about tools, see Installing and configuring your toolchain for the AWS SDK for .NET.
- This tutorial assumes that you're using your `[default]` profile for credentials. It also assumes that short-term credentials are available with appropriate permissions for sending and receiving Amazon SQS messages. For more information, see Authenticating the AWS SDK for .NET with AWS and the security topics for SQS.

> **Note**
>
> By running this tutorial, you might incur costs for SQS messaging.

Steps

- Create an SQS queue
- Create and run the publishing application
- Create and run the handling application
- Cleanup

Create an SQS queue

This tutorial requires an SQS queue to send messages to and receive messages from. A queue can be created by using one of the following commands for the AWS CLI or the AWS Tools for PowerShell. Take note of the queue URL that is returned so that you can specify it in the framework configuration that follows.

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

AWS Tools for PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

Create and run the publishing application

Use the following procedure to create and run the publishing application.

1. Open a command prompt or terminal. Find or create an operating system folder under which you can create a .NET project.
2. In that folder, run the following command to create the .NET project.

   ```
   dotnet new webapi --name Publisher
   ```

3. Navigate into the new project's folder. Add a dependency on the AWS Message Processing Framework for .NET.

   ```
   cd Publisher
   dotnet add package AWS.Messaging
   ```

> **Note**
>
> If you're using AWS IAM Identity Center for authentication, be sure to also add `AWSSDK.SSO` and `AWSSDK.SSOOIDC`.

4. Replace the code in `Program.cs` with the following code.

```
using AWS.Messaging;
using Microsoft.AspNetCore.Mvc;
using Publisher;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/
swashbuckle.
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();


// Configure the AWS Message Processing Framework for .NET.
builder.Services.AddAWSMessageBus(builder =>
{
    // Check for input SQS URL.
    // The SQS URL should be passed as a command line argument or set in the Debug
 launch profile.
    if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
    {
        // Register that you'll publish messages of type GreetingMessage:
        // 1. To a specified queue.
        // 2. Using the message identifier "greetingMessage", which will be used
        //    by handlers to route the message to the appropriate handler.
        builder.AddSQSPublisher<GreetingMessage>(args[0], "greetingMessage");
    }
    // You can map additional message types to queues or topics here as well.
});
var app = builder.Build();


// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
```

```
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

// Create an API Endpoint that receives GreetingMessage objects
// from the caller and then sends them as an SQS message.
app.MapPost("/greeting", async ([FromServices] IMessagePublisher publisher,
 Publisher.GreetingMessage message) =>
    {
        return await PostGreeting(message, publisher);
    })
.WithName("SendGreeting")
.WithOpenApi();

app.Run();

public partial class Program
{
    /// <summary>
    /// Endpoint for posting a greeting message.
    /// </summary>
    /// <param name="greetingMessage">The greeting message.</param>
    /// <param name="messagePublisher">The message publisher.</param>
    /// <returns>Async task result.</returns>
    public static async Task<IResult> PostGreeting(GreetingMessage greetingMessage,
        IMessagePublisher messagePublisher)
    {
        if (greetingMessage.SenderName == null || greetingMessage.Greeting == null)
        {
            return Results.BadRequest();
        }

        // Publish the message to the queue configured above.
        await messagePublisher.PublishAsync(greetingMessage);

        return Results.Ok();
    }
}

namespace Publisher
{
    /// <summary>
```

```
    /// This class represents the message contents.
    /// </summary>
    public class GreetingMessage
    {
        public string? SenderName { get; set; }
        public string? Greeting { get; set; }
    }
}
```

5. Run the following command. This should open a browser window with the Swagger UI, which allows you to explore and test your API.

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

6. Open the `/greeting` endpoint and choose **Try it out**.
7. Specify `senderName` and `greeting` values for the message, and choose **Execute**. This invokes your API, which sends the SQS message.

Create and run the handling application

Use the following procedure to create and run the handling application.

1. Open a command prompt or terminal. Find or create an operating system folder under which you can create a .NET project.
2. In that folder, run the following command to create the .NET project.

```
dotnet new console --name Handler
```

3. Navigate into the new project's folder. Add a dependency on the AWS Message Processing Framework for .NET. Also add the `Microsoft.Extensions.Hosting` package, which allows you to configure the framework through the .NET Generic Host.

```
cd Handler
dotnet add package AWS.Messaging
dotnet add package Microsoft.Extensions.Hosting
```

> **Note**
>
> If you're using AWS IAM Identity Center for authentication, be sure to also add `AWSSDK.SSO` and `AWSSDK.SSOOIDC`.

4. Replace the code in `Program.cs` with the following code.

```
using AWS.Messaging;
using Handler;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET.
    services.AddAWSMessageBus(builder =>
    {
        // Check for input SQS URL.
        // The SQS URL should be passed as a command line argument or set in the
 Debug launch profile.
        if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
        {
            // Register you'll poll the following queue.
            builder.AddSQSPoller(args[0]);

            // And that messages of type "greetingMessage" should be:
            // 1. Deserialized as GreetingMessage objects.
            // 2. Which are then passed to GreetingMessageHandler.
            builder.AddMessageHandler<GreetingMessageHandler,
 GreetingMessage>("greetingMessage");

        }
        // You can add additional message handlers here, using different message
 types.
    });
});

var host = builder.Build();
await host.RunAsync();
```

```
namespace Handler
{
    /// <summary>
    /// This class represents the message contents.
    /// </summary>
    public class GreetingMessage
    {
        public string? SenderName { get; set; }
        public string? Greeting { get; set; }
    }

    /// <summary>
    /// This handler is invoked each time you receive the message.
    /// </summary>
    public class GreetingMessageHandler : IMessageHandler<GreetingMessage>
    {
        public Task<MessageProcessStatus> HandleAsync(
            MessageEnvelope<GreetingMessage> messageEnvelope,
            CancellationToken token = default)
        {
            Console.WriteLine(
                $"Received message {messageEnvelope.Message.Greeting} from
 {messageEnvelope.Message.SenderName}");
            return Task.FromResult(MessageProcessStatus.Success());
        }
    }
}
```

5. Run the following command. This starts a long-running poller.

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

Shortly after startup the application will receive the message that was sent in the first part of this tutorial and log the following message:

```
Received message {greeting} from {senderName}
```

6. Press `Ctrl+C` to stop the poller.

Cleanup

Use one of the following commands for the AWS CLI or the AWS Tools for PowerShell to delete the queue.

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

AWS Tools for PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

Publish messages with the AWS Message Processing Framework for .NET

The AWS Message Processing Framework for .NET supports publishing one or more message types, processing one or more message types, or doing both in the same application.

The following code shows a configuration for an application that is publishing different message types to different AWS services.

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll send messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");

    // Register that you'll publish messages of type OrderInfo to an existing SNS topic
    builder.AddSNSPublisher<OrderInfo>("arn:aws:sns:us-west-2:012345678910:MyAppProd");

    // Register that you'll publish messages of type FoodItem to an existing
 EventBridge bus
    builder.AddEventBridgePublisher<FoodItem>("arn:aws:events:us-
west-2:012345678910:event-bus/default");
});
```

Once you have registered the framework during startup, inject the generic `IMessagePublisher` into your code. Call its `PublishAsync` method to publish any of the message types that were

configured above. The generic publisher will determine the destination to route the message to based on its type.

In the following example, an ASP.NET MVC controller receives both `ChatMessage` messages and `OrderInfo` events from users, and then publishes them to Amazon SQS and Amazon SNS respectively. Both message types can be published using the generic publisher that was configured above.

```
[ApiController]
[Route("[controller]")]
public class PublisherController : ControllerBase
{
    private readonly IMessagePublisher _messagePublisher;

    public PublisherController(IMessagePublisher messagePublisher)
    {
        _messagePublisher = messagePublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here.
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to
 the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }

        // Send the ChatMessage to SQS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }

    [HttpPost("order", Name = "Order")]
    public async Task<IActionResult> PublishOrder([FromBody] OrderInfo message)
    {
        if (message == null)
```

```
        {
            return BadRequest("An order was not submitted.");
        }

        // Publish the OrderInfo to SNS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }
}
```

In order to route a message to the appropriate handling logic, the framework uses metadata called the *message type identifier*. By default, this is the full name of the message's .NET type, including its assembly name. If you're both sending and handling messages, this mechanism works well if you share the definition of your message objects across projects. However, if the messages are redefined in different namespaces, or if you're exchanging messages with other frameworks or programming languages, then you might need to override the message type identifier.

```
var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET
    services.AddAWSMessageBus(builder =>
    {
        // Register that you'll publish messages of type GreetingMessage to an existing
 queue
        builder.AddSQSPublisher<GreetingMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd", "greetingMessage");
    });
});
```

Service-specific publishers

The example shown above uses the generic `IMessagePublisher`, which can publish to any supported AWS service based on the configured message type. The framework also provides service-specific publishers for Amazon SQS, Amazon SNS and Amazon EventBridge. These specific publishers expose options that only apply to that service, and can be injected using the types `ISQSPublisher`, `ISNSPublisher`, and `IEventBridgePublisher`.

For example, when sending messages to an SQS FIFO queue, you must set the appropriate message group ID. The following code shows the `ChatMessage` example again, but now using an `ISQSPublisher` to set SQS-specific options.

```
public class PublisherController : ControllerBase
{
    private readonly ISQSPublisher _sqsPublisher;

    public PublisherController(ISQSPublisher sqsPublisher)
    {
        _sqsPublisher = sqsPublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to
 the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }

        // Send the ChatMessage to SQS using the injected ISQSPublisher, with SQS-
specific options
        await _sqsPublisher.SendAsync(message, new SQSOptions
        {
            DelaySeconds = <delay-in-seconds>,
            MessageAttributes = <message-attributes>,
            MessageDeduplicationId = <message-deduplication-id>,
            MessageGroupId = <message-group-id>
        });

        return Ok();
    }
}
```

The same can be done for SNS and EventBridge, using `ISNSPublisher` and `IEventBridgePublisher` respectively.

```
await _snsPublisher.PublishAsync(message, new SNSOptions
{
    Subject = <subject>,
    MessageAttributes = <message-attributes>,
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

```
await _eventBridgePublisher.PublishAsync(message, new EventBridgeOptions
{
    DetailType = <detail-type>,
    Resources = <resources>,
    Source = <source>,
    Time = <time>,
    TraceHeader = <trace-header>
});
```

By default, messages of a given type are sent to the destination that is configured in advance. However, you can override the destination for a single message using the message-specific publishers. You can also override the underlying AWS SDK for .NET client that is used to publish the message, which can be useful in multi-tenant applications where you need to change roles or credentials, depending on the destination.

```
await _sqsPublisher.SendAsync(message, new SQSOptions
{
    OverrideClient = <override IAmazonSQS client>,
    QueueUrl = <override queue URL>
});
```

Consume messages with the AWS Message Processing Framework for .NET

The AWS Message Processing Framework for .NET allows you to consume messages that have been published by using the framework or one of the messaging services. The messages can be consumed in a variety of ways, some of which are described below.

Message Handlers

To consume messages, implement a message handler using the `IMessageHandler` interface for each message type you wish to process. The mapping between message types and message handlers is configured in the project startup.

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET
        services.AddAWSMessageBus(builder =>
        {
            // Register an SQS Queue that the framework will poll for messages.
            // NOTE: The URL given below is an example. Use the appropriate URL for
 your SQS Queue.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd");

            // Register all IMessageHandler implementations with the message type they
 should process.
            // Here messages that match our ChatMessage .NET type will be handled by
 our ChatMessageHandler
            builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
        });
    })
    .Build()
    .RunAsync();
```

The following code shows a sample message handler for a `ChatMessage` message.

```
public class ChatMessageHandler : IMessageHandler<ChatMessage>
{
    public Task<MessageProcessStatus> HandleAsync(MessageEnvelope<ChatMessage>
 messageEnvelope, CancellationToken token = default)
    {
        // Add business and validation logic here.
        if (messageEnvelope == null)
        {
            return Task.FromResult(MessageProcessStatus.Failed());
        }

        if (messageEnvelope.Message == null)
        {
            return Task.FromResult(MessageProcessStatus.Failed());
        }

        ChatMessage message = messageEnvelope.Message;

        Console.WriteLine($"Message Description: {message.MessageDescription}");
```

```
        // Return success so the framework will delete the message from the queue.
        return Task.FromResult(MessageProcessStatus.Success());
    }
}
```

The outer `MessageEnvelope` contains metadata used by the framework. Its `message` property is the message type (in this case `ChatMessage`).

You can return `MessageProcessStatus.Success()` to indicate that the message was processed successfully and the framework will delete the message from the Amazon SQS queue. When returning `MessageProcessStatus.Failed()`, the message will remain in the queue where it can be processed again or moved to a dead-letter queue, if configured.

Handling Messages in a Long-Running Process

You can call `AddSQSPoller` with an SQS queue URL to start a long-running `BackgroundService` that will continuously poll the queue and process messages.

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET
        services.AddAWSMessageBus(builder =>
        {
            // Register an SQS Queue that the framework will poll for messages.
            // NOTE: The URL given below is an example. Use the appropriate URL for
 your SQS Queue.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd", options =>
            {
                // The maximum number of messages from this queue that the framework
 will process concurrently on this client.
                options.MaxNumberOfConcurrentMessages = 10;

                // The duration each call to SQS will wait for new messages.
                options.WaitTimeSeconds = 20;
            });

            // Register all IMessageHandler implementations with the message type they
 should process.
            builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
```

```
        });
    })
    .Build()
    .RunAsync();
```

Configuring the SQS Message Poller

The SQS message poller can be configured by the `SQSMessagePollerOptions` when calling `AddSQSPoller`.

- `MaxNumberOfConcurrentMessages` - The maximum number of messages from the queue to process concurrently. The default value is 10.
- `WaitTimeSeconds` - The duration (in seconds) for which the `ReceiveMessage` SQS call waits for a message to arrive in the queue before returning. If a message is available, the call returns sooner than `WaitTimeSeconds`. The default value is 20.

Message Visibility Timeout Handling

SQS messages have a visibility timeout period. When one consumer begins handling a given message, it remains in the queue but is hidden from other consumers to avoid processing it more than once. If the message is not handled and deleted before becoming visible again, another consumer might attempt to handle the same message.

The framework will track and attempt to extend the visibility timeout for messages that it is currently handling. You can configure this behavior on the `SQSMessagePollerOptions` when calling `AddSQSPoller`.

- `VisibilityTimeout` - The duration in seconds that received messages are hidden from subsequent retrieve requests. The default value is 30.
- `VisibilityTimeoutExtensionThreshold` - When a message's visibility timeout is within this many seconds of expiring, the framework will extend the visibility timeout (by another `VisibilityTimeout` seconds). The default value is 5.
- `VisibilityTimeoutExtensionHeartbeatInterval` - How often in seconds that the framework will check for messages that are within `VisibilityTimeoutExtensionThreshold` seconds of expiring, and then extend their visibility timeout. The default value is 1.

In the following example, the framework will check every 1 second for messages that are still being handled. For those messages within 5 seconds of becoming visible again, the framework will automatically extend the visibility timeout of each message by another 30 seconds.

```
// NOTE: The URL given below is an example. Use the appropriate URL for your SQS Queue.
builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/MyAppProd",
 options =>
{
    options.VisibilityTimeout = 30;
    options.VisibilityTimeoutExtensionThreshold = 5;
    VisibilityTimeoutExtensionHeartbeatInterval = 1;
});
```

Handling messages in AWS Lambda functions

You can use the AWS Message Processing Framework for .NET with SQS's integration with Lambda. This is provided by the `AWS.Messaging.Lambda` package. Refer to its README to get started.

Using FIFO with the AWS Message Processing Framework for .NET

For use cases where message ordering and message deduplication are critical, the AWS Message Processing Framework for .NET supports first-in-first-out (FIFO) Amazon SQS queues and Amazon SNS topics.

Publishing

When publishing messages to a FIFO queue or topic, you must set the message group ID, which specifies the group that the message belongs to. Messages within a group are processed in order. You can set this on the SQS-specific and SNS-specific message publishers.

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
{
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

Subscribing

When handling messages from a FIFO queue, the framework handles messages within a given message group in the order in which they were received for each `ReceiveMessages` call. The

framework enters this mode of operation automatically when configured with a queue ending in `.fifo`.

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET.
        services.AddAWSMessageBus(builder =>
        {
            // Because this is a FIFO queue, the framework automatically handles these
 messages in order.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MPF.fifo");
            builder.AddMessageHandler<OrderMessageHandler, OrderMessage>();
        });
    })
    .Build()
    .RunAsync();
```

Logging and Open Telemetry for the AWS Message Processing Framework for .NET

The AWS Message Processing Framework for .NET is instrumented for OpenTelemetry to log traces for each message that is published or handled by the framework. This is provided by the `AWS.Messaging.Telemetry.OpenTelemetry` package. Refer to its README to get started.

> **Note**
>
> For security information related to logging, see Security for the AWS Message Processing Framework for .NET.

Customize the AWS Message Processing Framework for .NET

The AWS Message Processing Framework for .NET builds, sends, and handles messages in three different "layers":

1. At the outermost layer, the framework builds the AWS-native request or response specific to a service. With Amazon SQS for example, it builds `SendMessage` requests, and works with the `Message` objects that are defined by the service.

2. Inside the SQS request and response, the framework sets the `MessageBody` element (or `Message` for Amazon SNS or `Detail` for Amazon EventBridge) to a JSON-formatted CloudEvent. This contains metadata set by the framework that is accessible on the `MessageEnvelope` object when handling a message.
3. At the innermost layer, the `data` attribute inside the CloudEvent JSON object contains a JSON serialization of the .NET object that was sent or received as the message.

```
{
    "id":"b02f156b-0f02-48cf-ae54-4fbbe05cffba",
    "source":"/aws/messaging",
    "specversion":"1.0",
    "type":"Publisher.Models.ChatMessage",
    "time":"2023-11-21T16:36:02.8957126+00:00",
    "data":"<the ChatMessage object serialized as JSON>"
}
```

You can customize how the message envelope is configured and read:

- `"id"` uniquely identifies the message. By default it is set to a new GUID, but this can be overridden by implementing your own `IMessageIdGenerator` and injecting that into the DI container.
- `"type"` controls how the message is routed to handlers. By default this uses the full name of the .NET type that corresponds to the message. You can override this via the `messageTypeIdentifier` parameter when mapping the message type to the destination via `AddSQSPublisher`, `AddSNSPublisher`, or `AddEventBridgePublisher`.
- `"source"` indicates which system or server sent the message.
  - This will be the function name if publishing from AWS Lambda, the cluster name and task ARN if on Amazon ECS, the instance ID if on Amazon EC2, otherwise a fallback value of `/aws/messaging`.
  - You can override this via `AddMessageSource` or `AddMessageSourceSuffix` on the `MessageBusBuilder`.
- `"time"` set to the current DateTime in UTC. This can be overridden by implementing your own `IDateTimeHandler` and injecting that into the DI container.
- `"data"` contains a JSON representation of the .NET object that was sent or received as the message:

- `ConfigureSerializationOptions` on `MessageBusBuilder` allows you to configure the `System.Text.Json.JsonSerializerOptions` that will be used when serializing and deserializing the message.
- To inject additional attributes or transform the message envelope once the framework builds it, you can implement `ISerializationCallback` and register that via `AddSerializationCallback` on `MessageBusBuilder`.

Security for the AWS Message Processing Framework for .NET

The AWS Message Processing Framework for .NET relies on the AWS SDK for .NET for communicating with AWS. For more information about security in the AWS SDK for .NET, see Security for this AWS Product or Service.

For security purposes, the framework doesn't log data messages sent by the user.
If you want to enable this functionality for debugging purposes, you need to call `EnableDataMessageLogging()` in the Message Bus as follows:

```
builder.Services.AddAWSMessageBus(bus =>
{
    builder.EnableDataMessageLogging();
});
```

If you discover a potential security issue, refer to the security policy for reporting information.

Integrating AWS with .NET Aspire in the AWS SDK for .NET

.NET Aspire is a new way of building cloud-ready applications. In particular, it provides an orchestration for local environments in which to run, connect, and debug the components of distributed applications. To improve the inner dev loop for cloud ready applications, integrations with .NET Aspire have been created for connecting your .NET applications to AWS resources. These integrations are available through the Aspire.Hosting.AWS NuGet package.

The following .NET Aspire integrations are available:

- The ability to provision your AWS resources through CloudFormation. This integration is utilized within the .NET Aspire AppHost project.

  For more information, see the blog post Integrating AWS with .NET Aspire.

- Installing, configuring, and connecting the AWS SDK for .NET to Amazon DynamoDB local. This integration is utilized within the .NET Aspire AppHost project.

  For more information, see the blog post Integrating AWS with .NET Aspire.

- Enable a local development environment for AWS Lambda functions. This integration is utilized within the .NET Aspire AppHost project.

  For more information, see the blog posts Building and Debugging .NET Lambda applications with .NET Aspire (Part 1) and Building and Debugging .NET Lambda applications with .NET Aspire (Part 2).

  > **Note**
  >
  > This is prerelease documentation for a feature in preview release. It is subject to change.

  Because this feature is in preview, you will need to opt-in for preview features. For additional information about this preview feature and how to opt-in, see the development tracker issue on GitHub.

Additional information

For additional information and details about how you can use the integrations that are available in Aspire.Hosting.AWS, see the following resources.

- Blog post Integrating AWS with .NET Aspire.
- Blog posts Building and Debugging .NET Lambda applications with .NET Aspire (Part 1) and Building and Debugging .NET Lambda applications with .NET Aspire (Part 2).
- The integrations-on-dotnet-aspire-for-aws repo on GitHub.
- The detailed README for the Aspire.Hosting.AWS NuGet package.

Support for other AWS services and configuration

The AWS SDK for .NET supports AWS services in addition to those described in the preceding sections. For information about the APIs for all supported services, see the AWS SDK for .NET API Reference.

In addition to the namespaces for individual AWS services, the AWS SDK for .NET also provides the following APIs:

Area	Description	Resources
AWS Support	Programmatic access to AWS Support cases and Trusted Advisor features.	See Amazon.AWSSupport and Amazon.AWSSupport.Model.
General	Helper classes and enumerations.	See Amazon and Amazon.Util.

SDK for .NET (v4) code examples

The code examples in this topic show you how to use the AWS SDK for .NET (v4) with AWS.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Some services contain additional example categories that show how to leverage libraries or functions specific to the service.

Services

- Aurora examples using SDK for .NET (v4)
- Auto Scaling examples using SDK for .NET (v4)
- Amazon Bedrock examples using SDK for .NET (v4)
- Amazon Bedrock Runtime examples using SDK for .NET (v4)
- CloudFormation examples using SDK for .NET (v4)
- CloudWatch examples using SDK for .NET (v4)
- CloudWatch Logs examples using SDK for .NET (v4)
- Amazon Cognito Identity Provider examples using SDK for .NET (v4)
- AWS Control Tower examples using SDK for .NET (v4)
- DynamoDB examples using SDK for .NET (v4)
- Amazon EC2 examples using SDK for .NET (v4)
- Amazon ECS examples using SDK for .NET (v4)
- AWS IoT examples using SDK for .NET (v4)
- AWS IoT data examples using SDK for .NET (v4)
- Amazon Redshift examples using SDK for .NET (v4)
- Amazon S3 examples using SDK for .NET (v4)

Aurora examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with Aurora.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello Aurora

The following code examples show how to get started using Aurora.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
```

```
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here for
 example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        if (response.DBClusters == null)
        {
            Console.WriteLine($"\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\tCluster: database: {cluster.DatabaseName} identifier:
 {cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- For API details, see DescribeDBClusters in *AWS SDK for .NET API Reference*.

Topics

- Basics
- Actions

Basics

Learn the basics

The following code example shows how to:

- Create a custom Aurora DB cluster parameter group and set parameter values.
- Create a DB cluster that uses the parameter group.
- Create a DB instance that contains a database.
- Take a snapshot of the DB cluster, then clean up resources.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
/// </summary>
public class AuroraScenario
{
    /*
```

```
   Before running this .NET code example, set up your development environment,
including your credentials.

   This .NET example performs the following tasks:
   1.  Return a list of the available DB engine families for Aurora MySql using the
DescribeDBEngineVersionsAsync method.
   2.  Select an engine family and create a custom DB cluster parameter group using
the CreateDBClusterParameterGroupAsync method.
   3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
method.
   4.  Get some parameters in the group using the DescribeDBClusterParametersAsync
method.
   5.  Parse and display some parameters in the group.
   6.  Modify the auto_increment_offset and auto_increment_increment parameters
       using the ModifyDBClusterParameterGroupAsync method.
   7.  Get and display the updated parameters using the
DescribeDBClusterParametersAsync method with a source of "user".
   8.  Get a list of allowed engine versions using the
DescribeDBEngineVersionsAsync method.
   9.  Create an Aurora DB cluster that contains a MySql database and uses the
parameter group.
       using the CreateDBClusterAsync method.
   10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
method.
   11. Display and select from a list of instance classes available for the
selected engine and version
       using the paginated DescribeOrderableDBInstanceOptions method.
   12. Create a database instance in the cluster using the CreateDBInstanceAsync
method.
   13. Wait for the DB instance to be ready using the DescribeDBInstances method.
   14. Display the connection endpoint string for the new DB cluster.
   15. Create a snapshot of the DB cluster using the CreateDBClusterSnapshotAsync
method.
   16. Wait for DB snapshot to be ready using the DescribeDBClusterSnapshotsAsync
method.
   17. Delete the DB instance using the DeleteDBInstanceAsync method.
   18. Delete the DB cluster using the DeleteDBClusterAsync method.
   19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync methods.
   20. Delete the cluster parameter group using the
DeleteDBClusterParameterGroupAsync.
   */

   private static readonly string sepBar = new('-', 80);
   private static AuroraWrapper auroraWrapper = null!;
```

```
   private static ILogger logger = null!;
   private static readonly string engine = "aurora-mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonRDS>()
                   .AddTransient<AuroraWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder =>
       {
           builder.AddConsole();
       }).CreateLogger<AuroraScenario>();

       auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

       Console.WriteLine(sepBar);
       Console.WriteLine(
           "Welcome to the Amazon Aurora: get started with DB clusters example.");
       Console.WriteLine(sepBar);

       DBClusterParameterGroup parameterGroup = null!;
       DBCluster? newCluster = null;
       DBInstance? newInstance = null;

       try
       {
           var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

           parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);

           var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
```

```
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName,
parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
        }
        catch (Exception ex)
        {
```

```
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the Aurora DB parameter group family from a list of available
options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamilyAsync()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

        Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

        var parameterGroupFamilies =
            engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
        for (var i = 1; i <= parameterGroupFamilies.Count; i++)
        {
            var parameterGroupFamily = parameterGroupFamilies[i - 1];
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("2. Select an available DB parameter group family by
entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
    }

    /// <summary>
```

```
    /// Create and get information on a DB parameter group.
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the new
 DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBClusterParameterGroup>
 CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
 {dbParameterGroupFamily}:");

        var parameterGroup = await
 auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
 auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParameterG

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>> DescribeParametersInGroupAsync(string
 parameterGroupName, List<string>? parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
        Console.WriteLine(sepBar);

        var parameters =
```

```
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string parameterGroupName)
```

```
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
group.");

        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName, "user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
            i++;
        }
```

```
        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by entering
 a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);

        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
 clusterIdentifier);
        }
```

```
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
 auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
    public static async Task<OrderableDBInstanceOption> ChooseDBInstanceClass(string
 engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
```

```
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by entering
 a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
    /// <param name="instanceIdentifier">Instance identifier to use for the DB
 instance.</param>
    /// <returns>The new DB instance.</returns>
    public static async Task<DBInstance?> CreateNewInstance(
        string clusterIdentifier,
```

```
        string engineName,
        string engineVersion,
        string instanceClass,
        string instanceIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"12. Create a new DB instance with identifier
{instanceIdentifier}.");
        bool isInstanceReady = false;
        DBInstance newInstance;
        var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
        isInstanceReady = instances.FirstOrDefault(i =>
            i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

        if (isInstanceReady)
        {
            Console.WriteLine("Instance already created.");
            newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
        }
        else
        {
            newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
                clusterIdentifier,
                instanceIdentifier,
                engineName,
                engineVersion,
                instanceClass
            );

            Console.WriteLine("13. Waiting for DB instance to be ready...");
            while (!isInstanceReady)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                instances = await
auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
"available";
                newInstance = instances.First();
            }
        }
```

```
        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
 {cluster.DBClusterIdentifier}.");
        var snapshot = await auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;

        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
```

```
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }

        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
```

```
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
{parameterGroup.DBClusterParameterGroupName}.");
            await
auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGroup
            Console.WriteLine("Parameter group deleted.");
        }

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

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
```

```
                        ynResponse.Equals("y",
                            StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

Wrapper methods that are called by the scenario to manage Aurora actions.

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
```

```
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
 DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };
```

```
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string groupName,
List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
```

```
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
```

```
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
```

```
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
 instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
 dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }

    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
```

```
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
```

```
        // When creating the instance within a cluster, do not specify the name or
size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();
```

```
        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
```

```
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Actions

`CreateDBCluster`

The following code example shows how to use `CreateDBCluster`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Create a new cluster and database.
/// </summary>
/// <param name="dbName">The name of the new database.</param>
/// <param name="clusterIdentifier">The identifier of the cluster.</param>
/// <param name="parameterGroupName">The name of the parameter group.</param>
/// <param name="dbEngine">The engine to use for the new cluster.</param>
/// <param name="dbEngineVersion">The version of the engine to use.</param>
/// <param name="adminName">The admin username.</param>
/// <param name="adminPassword">The primary admin password.</param>
/// <returns>The cluster object.</returns>
public async Task<DBCluster> CreateDBClusterWithAdminAsync(
    string dbName,
    string clusterIdentifier,
    string parameterGroupName,
    string dbEngine,
    string dbEngineVersion,
    string adminName,
    string adminPassword)
{
    var request = new CreateDBClusterRequest
    {
        DatabaseName = dbName,
        DBClusterIdentifier = clusterIdentifier,
        DBClusterParameterGroupName = parameterGroupName,
        Engine = dbEngine,
        EngineVersion = dbEngineVersion,
        MasterUsername = adminName,
        MasterUserPassword = adminPassword,
    };

    var response = await _amazonRDS.CreateDBClusterAsync(request);
    return response.DBCluster;
}
```

- For API details, see CreateDBCluster in *AWS SDK for .NET API Reference*.

`CreateDBClusterParameterGroup`

The following code example shows how to use `CreateDBClusterParameterGroup`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }
```

- For API details, see CreateDBClusterParameterGroup in *AWS SDK for .NET API Reference*.

`CreateDBClusterSnapshot`

The following code example shows how to use `CreateDBClusterSnapshot`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }
```

- For API details, see CreateDBClusterSnapshot in *AWS SDK for .NET API Reference*.

`CreateDBInstance`

The following code example shows how to use `CreateDBInstance`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name or
size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }
```

- For API details, see CreateDBInstance in *AWS SDK for .NET API Reference*.

`DeleteDBCluster`

The following code example shows how to use `DeleteDBCluster`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }
```

- For API details, see DeleteDBCluster in *AWS SDK for .NET API Reference*.

`DeleteDBClusterParameterGroup`

The following code example shows how to use `DeleteDBClusterParameterGroup`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Delete a particular parameter group by name.
/// </summary>
/// <param name="groupName">The name of the parameter group.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteClusterParameterGroupByNameAsync(string groupName)
{
    var request = new DeleteDBClusterParameterGroupRequest
    {
        DBClusterParameterGroupName = groupName,
    };

    var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
    return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
}
```

- For API details, see DeleteDBClusterParameterGroup in *AWS SDK for .NET API Reference*.

`DeleteDBInstance`

The following code example shows how to use `DeleteDBInstance`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
```

```
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

- For API details, see DeleteDBInstance in *AWS SDK for .NET API Reference*.

`DescribeDBClusterParameterGroups`

The following code example shows how to use `DescribeDBClusterParameterGroups`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
DescribeCustomDBClusterParameterGroupAsync(string name)
    {
```

```
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

- For API details, see DescribeDBClusterParameterGroups in *AWS SDK for .NET API Reference*.

`DescribeDBClusterParameters`

The following code example shows how to use `DescribeDBClusterParameters`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };
```

```
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }
```

- For API details, see DescribeDBClusterParameters in *AWS SDK for .NET API Reference*.

`DescribeDBClusterSnapshots`

The following code example shows how to use `DescribeDBClusterSnapshots`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
```

```
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- For API details, see DescribeDBClusterSnapshots in *AWS SDK for .NET API Reference*.

`DescribeDBClusters`

The following code example shows how to use `DescribeDBClusters`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
 dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
```

```
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- For API details, see DescribeDBClusters in *AWS SDK for .NET API Reference*.

`DescribeDBEngineVersions`

The following code example shows how to use `DescribeDBEngineVersions`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
```

```
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- For API details, see DescribeDBEngineVersions in *AWS SDK for .NET API Reference*.

`DescribeDBInstances`

The following code example shows how to use `DescribeDBInstances`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
```

```
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

- For API details, see DescribeDBInstances in *AWS SDK for .NET API Reference*.

`DescribeOrderableDBInstanceOptions`

The following code example shows how to use `DescribeOrderableDBInstanceOptions`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
```

```
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }
```

- For API details, see DescribeOrderableDBInstanceOptions in *AWS SDK for .NET API Reference*.

`ModifyDBClusterParameterGroup`

The following code example shows how to use `ModifyDBClusterParameterGroup`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string groupName,
List<Parameter> parameters, int newValue = 0)
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");
```

```
                var choice = Console.ReadLine();
                int.TryParse(choice, out newValue);
            }

            p.ParameterValue = newValue.ToString();
        }
    }

    var request = new ModifyDBClusterParameterGroupRequest
    {
        Parameters = parameters,
        DBClusterParameterGroupName = groupName,
    };

    var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
    return result.DBClusterParameterGroupName;
}
```

- For API details, see ModifyDBClusterParameterGroup in *AWS SDK for .NET API Reference*.

Auto Scaling examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with Auto Scaling.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello Auto Scaling

The following code examples show how to get started using Auto Scaling.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace AutoScalingActions;

using Amazon.AutoScaling;

public class HelloAutoScaling
{
    /// <summary>
    /// Hello Amazon EC2 Auto Scaling. List EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="args"></param>
    /// <returns>Async Task.</returns>
    static async Task Main(string[] args)
    {
        var client = new AmazonAutoScalingClient();

        Console.WriteLine("Welcome to Amazon EC2 Auto Scaling.");
        Console.WriteLine("Let's get a description of your Auto Scaling groups.");

        var response = await client.DescribeAutoScalingGroupsAsync();

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

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

```
    }
}
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for .NET API Reference*.

Topics

- Basics
- Actions

Basics

Learn the basics

The following code example shows how to:

- Create an Amazon EC2 Auto Scaling group with a launch template and Availability Zones, and get information about running instances.
- Enable Amazon CloudWatch metrics collection.
- Update the group's desired capacity and wait for an instance to start.
- Terminate an instance in the group.
- List scaling activities that occur in response to user requests and capacity changes.
- Get statistics for CloudWatch metrics, then clean up resources.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
global using Amazon.AutoScaling;
global using Amazon.AutoScaling.Model;
global using Amazon.CloudWatch;
global using AutoScalingActions;
```

```
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;



using Amazon.EC2;
using Microsoft.Extensions.Configuration;
using Host = Microsoft.Extensions.Hosting.Host;

namespace AutoScalingBasics;

public class AutoScalingBasics
{

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EC2 Auto Scaling, Amazon
        // CloudWatch, and Amazon EC2.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonAutoScaling>()
                .AddAWSService<IAmazonCloudWatch>()
                .AddAWSService<IAmazonEC2>()
                .AddTransient<AutoScalingWrapper>()
                .AddTransient<CloudWatchWrapper>()
                .AddTransient<EC2Wrapper>()
                .AddTransient<UIWrapper>()
            )
            .Build();


        var autoScalingWrapper =
 host.Services.GetRequiredService<AutoScalingWrapper>();
        var cloudWatchWrapper =
 host.Services.GetRequiredService<CloudWatchWrapper>();
        var ec2Wrapper = host.Services.GetRequiredService<EC2Wrapper>();
```

```
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var imageId = configuration["ImageId"];
        var instanceType = configuration["InstanceType"];
        var launchTemplateName = configuration["LaunchTemplateName"];

        launchTemplateName += Guid.NewGuid().ToString();

        // The name of the Auto Scaling group.
        var groupName = configuration["GroupName"];

        uiWrapper.DisplayTitle("Auto Scaling Basics");
        uiWrapper.DisplayAutoScalingBasicsDescription();

        // Create the launch template and save the template Id to use when deleting
the
        // launch template at the end of the application.
        var launchTemplateId = await ec2Wrapper.CreateLaunchTemplateAsync(imageId!,
instanceType!, launchTemplateName);

        // Confirm that the template was created by asking for a description of it.
        await ec2Wrapper.DescribeLaunchTemplateAsync(launchTemplateName);

        uiWrapper.PressEnter();

        var availabilityZones = await ec2Wrapper.ListAvailabilityZonesAsync();

        Console.WriteLine($"Creating an Auto Scaling group named {groupName}.");
        await autoScalingWrapper.CreateAutoScalingGroupAsync(
            groupName!,
            launchTemplateName,
            availabilityZones[0].ZoneName);

        // Keep checking the details of the new group until its lifecycle state
        // is "InService".
        Console.WriteLine($"Waiting for the Auto Scaling group to be active.");
```

```
        List<AutoScalingInstanceDetails> instanceDetails;

        do
        {
            instanceDetails = await
autoScalingWrapper.DescribeAutoScalingInstancesAsync(groupName!);
        }
        while (instanceDetails.Count <= 0);

        Console.WriteLine($"Auto scaling group {groupName} successfully created.");
        Console.WriteLine($"{instanceDetails.Count} instances were created for the
group.");

        // Display the details of the Auto Scaling group.
        instanceDetails.ForEach(detail =>
        {
            Console.WriteLine($"Group name: {detail.AutoScalingGroupName}");
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Metrics collection");
        Console.WriteLine($"Enable metrics collection for {groupName}");
        await autoScalingWrapper.EnableMetricsCollectionAsync(groupName!);

        // Show the metrics that are collected for the group.

        // Update the maximum size of the group to three instances.
        Console.WriteLine("--- Update the Auto Scaling group to increase max size to
3 ---");
        int maxSize = 3;
        await autoScalingWrapper.UpdateAutoScalingGroupAsync(groupName!,
launchTemplateName, maxSize);

        Console.WriteLine("--- Describe all Auto Scaling groups to show the current
state of the group ---");
        var groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);

        uiWrapper.DisplayGroupDetails(groups!);

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Describe account limits");
```

```
        await autoScalingWrapper.DescribeAccountLimitsAsync();

        uiWrapper.WaitABit(60, "Waiting for the resources to be ready.");

        uiWrapper.DisplayTitle("Set desired capacity");
        int desiredCapacity = 2;
        await autoScalingWrapper.SetDesiredCapacityAsync(groupName!,
desiredCapacity);

        Console.WriteLine("Get the two instance Id values");

        // Empty the group before getting the details again.
        groups.Clear();
        groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);
        if (groups.Any())
        {
            foreach (AutoScalingGroup group in groups)
            {
                Console.WriteLine($"The group name is
{group.AutoScalingGroupName}");
                Console.WriteLine($"The group ARN is {group.AutoScalingGroupARN}");
                var instances = group.Instances;
                foreach (Amazon.AutoScaling.Model.Instance instance in instances)
                {
                    Console.WriteLine($"The instance id is {instance.InstanceId}");
                    Console.WriteLine($"The lifecycle state is
{instance.LifecycleState}");
                }
            }
        }

        uiWrapper.DisplayTitle("Scaling Activities");
        Console.WriteLine("Let's list the scaling activities that have occurred for
the group.");
        var activities = await
autoScalingWrapper.DescribeScalingActivitiesAsync(groupName!);
        if (activities.Any())
        {
            activities.ForEach(activity =>
            {
                Console.WriteLine($"The activity Id is {activity.ActivityId}");
                Console.WriteLine($"The activity details are {activity.Details}");
            });
```

```
        }

        // Display the Amazon CloudWatch metrics that have been collected.
        var metrics = await cloudWatchWrapper.GetCloudWatchMetricsAsync(groupName!);
        if (metrics.Any())
        {
            Console.WriteLine($"Metrics collected for {groupName}:");
            metrics.ForEach(metric =>
            {
                Console.Write($"Metric name: {metric.MetricName}\t");
                Console.WriteLine($"Namespace: {metric.Namespace}");
            });
        }

        var dataPoints = await
cloudWatchWrapper.GetMetricStatisticsAsync(groupName!);
        if (dataPoints.Any())
        {
            Console.WriteLine("Details for the metrics collected:");
            dataPoints.ForEach(detail => { Console.WriteLine(detail); });
        }

        // Disable metrics collection.
        Console.WriteLine("Disabling the collection of metrics for {groupName}.");
        var success = await
autoScalingWrapper.DisableMetricsCollectionAsync(groupName!);

        if (success)
        {
            Console.WriteLine($"Successfully stopped metrics collection for
{groupName}.");
        }
        else
        {
            Console.WriteLine($"Could not stop metrics collection for
{groupName}.");
        }

        // Terminate all instances in the group.
        uiWrapper.DisplayTitle("Terminating Auto Scaling instances");
        Console.WriteLine("Now terminating all instances in the Auto Scaling
group.");

        if (groups is not null)
```

```
        {
            groups.ForEach(group =>
            {
                // Only delete instances in the AutoScaling group we created.
                if (group.AutoScalingGroupName == groupName)
                {
                    group.Instances.ForEach(async instance =>
                    {
                        await
 autoScalingWrapper.TerminateInstanceInAutoScalingGroupAsync(instance.InstanceId);
                    });
                }
            });
        }

        // After all instances are terminated, delete the group.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Deleting the Auto Scaling group.");
        await autoScalingWrapper.DeleteAutoScalingGroupAsync(groupName!);

        // Delete the launch template.
        var deletedLaunchTemplateName = await
 ec2Wrapper.DeleteLaunchTemplateAsync(launchTemplateId);

        if (deletedLaunchTemplateName == launchTemplateName)
        {
            Console.WriteLine("Successfully deleted the launch template.");
        }

        Console.WriteLine("The demo is now concluded.");
    }
}


namespace AutoScalingBasics;

/// <summary>
/// A class to provide user interface methods for the EC2 AutoScaling Basics
/// scenario.
/// </summary>
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);
```

```
    /// <summary>
    /// Describe the steps in the EC2 AutoScaling Basics scenario.
    /// </summary>
    public void DisplayAutoScalingBasicsDescription()
    {
        Console.WriteLine("This code example performs the following operations:");
        Console.WriteLine(" 1. Creates an Amazon EC2 launch template.");
        Console.WriteLine(" 2. Creates an Auto Scaling group.");
        Console.WriteLine(" 3. Shows the details of the new Auto Scaling group");
        Console.WriteLine("    to show that only one instance was created.");
        Console.WriteLine(" 4. Enables metrics collection.");
        Console.WriteLine(" 5. Updates the Auto Scaling group to increase the");
        Console.WriteLine("    capacity to three.");
        Console.WriteLine(" 6. Describes Auto Scaling groups again to show the");
        Console.WriteLine("    current state of the group.");
        Console.WriteLine(" 7. Changes the desired capacity of the Auto Scaling");
        Console.WriteLine("    group to use an additional instance.");
        Console.WriteLine(" 8. Shows that there are now instances in the group.");
        Console.WriteLine(" 9. Lists the scaling activities that have occurred for
the group.");
        Console.WriteLine("10. Displays the Amazon CloudWatch metrics that have");
        Console.WriteLine("    been collected.");
        Console.WriteLine("11. Disables metrics collection.");
        Console.WriteLine("12. Terminates all instances in the Auto Scaling
group.");
        Console.WriteLine("13. Deletes the Auto Scaling group.");
        Console.WriteLine("14. Deletes the Amazon EC2 launch template.");
        PressEnter();
    }

    /// <summary>
    /// Display information about the Amazon Ec2 AutoScaling groups passed
    /// in the list of AutoScalingGroup objects.
    /// </summary>
    /// <param name="groups">A list of AutoScalingGroup objects.</param>
    public void DisplayGroupDetails(List<AutoScalingGroup> groups)
    {
        if (groups is null)
            return;

        groups.ForEach(group =>
        {
            Console.WriteLine($"Group name:\t{group.AutoScalingGroupName}");
            Console.WriteLine($"Group created:\t{group.CreatedTime}");
```

```
            Console.WriteLine($"Maximum number of instances:\t{group.MaxSize}");
            Console.WriteLine($"Desired number of instances:
\t{group.DesiredCapacity}");
        });
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }

    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
```

```
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

Define functions that are called by the scenario to manage launch templates and metrics. These functions wrap Auto Scaling, Amazon EC2, and CloudWatch actions.

```
namespace AutoScalingActions;

using Amazon.AutoScaling;
using Amazon.AutoScaling.Model;

/// <summary>
/// A class that includes methods to perform Amazon EC2 Auto Scaling
/// actions.
/// </summary>
public class AutoScalingWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;

    /// <summary>
    /// Constructor for the AutoScalingWrapper class.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected Amazon EC2 Auto Scaling
 client.</param>
    public AutoScalingWrapper(IAmazonAutoScaling amazonAutoScaling)
    {
        _amazonAutoScaling = amazonAutoScaling;
    }
```

```
    /// <summary>
    /// Create a new Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name to use for the new Auto Scaling
    /// group.</param>
    /// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
    /// launch template to use to create instances in the group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        string availabilityZone)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var zoneList = new List<string>
            {
                availabilityZone,
            };

        var request = new CreateAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            AvailabilityZones = zoneList,
            LaunchTemplate = templateSpecification,
            MaxSize = 6,
            MinSize = 1
        };
        try
        {
            var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
            Console.WriteLine($"{groupName} Auto Scaling Group created");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"{groupName} Auto Scaling Group already exists.");
            return true;
```

```
        }
    }



    /// <summary>
    /// Retrieve information about Amazon EC2 Auto Scaling quotas to the
    /// active AWS account.
    /// </summary>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DescribeAccountLimitsAsync()
    {
        var response = await _amazonAutoScaling.DescribeAccountLimitsAsync();
        Console.WriteLine("The maximum number of Auto Scaling groups is " +
 response.MaxNumberOfAutoScalingGroups);
        Console.WriteLine("The current number of Auto Scaling groups is " +
 response.NumberOfAutoScalingGroups);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }



    /// <summary>
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
```

```
        }
        return activities;
    }



    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
 null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

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

            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
```

```
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }



    /// <summary>
    /// Retrieve a list of information about Amazon EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling groups.</returns>
    public async Task<List<AutoScalingGroup>> DescribeAutoScalingGroupsAsync(
        string groupName)
    {
        var groups = new List<AutoScalingGroup>();
        var groupList = new List<string>
            {
                groupName,
            };

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

        var response = await
 _amazonAutoScaling.DescribeAutoScalingGroupsAsync(request);
        if (response.AutoScalingGroups != null)
        {
            groups = response.AutoScalingGroups;
        }

        return groups;
    }



    /// <summary>
    /// Delete an Auto Scaling group.
    /// </summary>
```

```
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAutoScalingGroupAsync(
        string groupName)
    {
        var deleteAutoScalingGroupRequest = new DeleteAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            ForceDelete = true,
        };

        var response = await
 _amazonAutoScaling.DeleteAutoScalingGroupAsync(deleteAutoScalingGroupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully deleted {groupName}");
            return true;
        }

        Console.WriteLine($"Couldn't delete {groupName}.");
        return false;
    }


    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
        };

        var response = await
 _amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

```
    /// <summary>
    /// Enable the collection of metric data for an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> EnableMetricsCollectionAsync(string groupName)
    {
        var listMetrics = new List<string>
            {
                "GroupMaxSize",
            };

        var collectionRequest = new EnableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
            Metrics = listMetrics,
            Granularity = "1Minute",
        };

        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {
            AutoScalingGroupName = groupName,
            DesiredCapacity = desiredCapacity,
        };
```

```
        var response = await
 _amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
        Console.WriteLine($"You have set the DesiredCapacity to
 {desiredCapacity}.");

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Terminate all instances in the Auto Scaling group in preparation for
    /// deleting the group.
    /// </summary>
    /// <param name="instanceId">The instance Id of the instance to terminate.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
        string instanceId)
    {
        var request = new TerminateInstanceInAutoScalingGroupRequest
        {
            InstanceId = instanceId,
            ShouldDecrementDesiredCapacity = false,
        };

        var response = await
 _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You have terminated the instance: {instanceId}");
            return true;
        }

        Console.WriteLine($"Could not terminate {instanceId}");
        return false;
    }


    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
```

```
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
 _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully updated the Auto Scaling group
 {groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }

}


namespace AutoScalingActions;

using Amazon.EC2;
using Amazon.EC2.Model;
```

```
public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEc2;

    /// <summary>
    /// Constructor for the EC2Wrapper class.
    /// </summary>
    /// <param name="amazonEc2">The injected Amazon EC2 client.</param>
    public EC2Wrapper(IAmazonEC2 amazonEc2)
    {
        _amazonEc2 = amazonEc2;
    }

    /// <summary>
    /// Create a new Amazon EC2 launch template.
    /// </summary>
    /// <param name="imageId">The image Id to use for instances launched
    /// using the Amazon EC2 launch template.</param>
    /// <param name="instanceType">The type of EC2 instances to create.</param>
    /// <param name="launchTemplateName">The name of the launch template.</param>
    /// <returns>Returns the TemplateID of the new launch template.</returns>
    public async Task<string> CreateLaunchTemplateAsync(
        string imageId,
        string instanceType,
        string launchTemplateName)
    {
        var request = new CreateLaunchTemplateRequest
        {
            LaunchTemplateData = new RequestLaunchTemplateData
            {
                ImageId = imageId,
                InstanceType = instanceType,
            },
            LaunchTemplateName = launchTemplateName,
        };

        var response = await _amazonEc2.CreateLaunchTemplateAsync(request);

        return response.LaunchTemplate.LaunchTemplateId;
    }

    /// <summary>
    /// Delete an Amazon EC2 launch template.
    /// </summary>
```

```
    /// <param name="launchTemplateId">The TemplateId of the launch template to
    /// delete.</param>
    /// <returns>The name of the EC2 launch template that was deleted.</returns>
    public async Task<string> DeleteLaunchTemplateAsync(string launchTemplateId)
    {
        var request = new DeleteLaunchTemplateRequest
        {
            LaunchTemplateId = launchTemplateId,
        };

        var response = await _amazonEc2.DeleteLaunchTemplateAsync(request);
        return response.LaunchTemplate.LaunchTemplateName;
    }

    /// <summary>
    /// Retrieve information about an EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DescribeLaunchTemplateAsync(string launchTemplateName)
    {
        var request = new DescribeLaunchTemplatesRequest
        {
            LaunchTemplateNames = new List<string> { launchTemplateName, },
        };

        var response = await _amazonEc2.DescribeLaunchTemplatesAsync(request);

        if (response.LaunchTemplates is not null)
        {
            response.LaunchTemplates.ForEach(template =>
            {
                Console.Write($"{template.LaunchTemplateName}\t");
                Console.WriteLine(template.LaunchTemplateId);
            });

            return true;
        }

        return false;
    }
```

```
    /// <summary>
    /// Retrieve the availability zones for the current region.
    /// </summary>
    /// <returns>A collection of availability zones.</returns>
    public async Task<List<AvailabilityZone>> ListAvailabilityZonesAsync()
    {
        var response = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());

        return response.AvailabilityZones;
    }
}


namespace AutoScalingActions;

using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// Contains methods to access Amazon CloudWatch metrics for the
/// Amazon EC2 Auto Scaling basics scenario.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;

    /// <summary>
    /// Constructor for the CloudWatchWrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch)
    {
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// Retrieve the metrics information collection for the Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A list of Metrics collected for the Auto Scaling group.</returns>
    public async Task<List<Metric>> GetCloudWatchMetricsAsync(string groupName)
    {
        var metrics = new List<Metric>();
```

```
        var filter = new DimensionFilter
        {
            Name = "AutoScalingGroupName",
            Value = $"{groupName}",
        };

        var request = new ListMetricsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = new List<DimensionFilter> { filter },
            Namespace = "AWS/AutoScaling",
        };

        var response = await _amazonCloudWatch.ListMetricsAsync(request);
        if (response.Metrics != null)
        {
            metrics = response.Metrics;
        }
        return metrics;
    }

    /// <summary>
    /// Retrieve the metric data collected for an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of data points.</returns>
    public async Task<List<Datapoint>> GetMetricStatisticsAsync(string groupName)
    {
        var dataPoints = new List<Datapoint>();
        var metricDimensions = new List<Dimension>
            {
                new Dimension
                {
                    Name = "AutoScalingGroupName",
                    Value = $"{groupName}",
                },
            };

        // The start time will be yesterday.
        var startTime = DateTime.UtcNow.AddDays(-1);

        var request = new GetMetricStatisticsRequest
        {
```

```
            MetricName = "AutoScalingGroupName",
            Dimensions = metricDimensions,
            Namespace = "AWS/AutoScaling",
            Period = 60, // 60 seconds.
            Statistics = new List<string>() { "Minimum" },
            StartTimeUtc = startTime,
            EndTimeUtc = DateTime.UtcNow,
        };

        var response = await _amazonCloudWatch.GetMetricStatisticsAsync(request);
        if (response.Datapoints != null)
        {
            dataPoints = response.Datapoints;
        }

        return dataPoints;
    }

}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Actions

`CreateAutoScalingGroup`

The following code example shows how to use `CreateAutoScalingGroup`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Create a new Amazon EC2 Auto Scaling group.
/// </summary>
/// <param name="groupName">The name to use for the new Auto Scaling
/// group.</param>
/// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
/// launch template to use to create instances in the group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateAutoScalingGroupAsync(
    string groupName,
    string launchTemplateName,
    string availabilityZone)
{
    var templateSpecification = new LaunchTemplateSpecification
    {
        LaunchTemplateName = launchTemplateName,
    };

    var zoneList = new List<string>
        {
            availabilityZone,
        };

    var request = new CreateAutoScalingGroupRequest
    {
        AutoScalingGroupName = groupName,
        AvailabilityZones = zoneList,
```

```
            LaunchTemplate = templateSpecification,
            MaxSize = 6,
            MinSize = 1
        };
        try
        {
            var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
            Console.WriteLine($"{groupName} Auto Scaling Group created");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"{groupName} Auto Scaling Group already exists.");
            return true;
        }
    }
```

- For API details, see CreateAutoScalingGroup in *AWS SDK for .NET API Reference*.

`DescribeAutoScalingGroups`

The following code example shows how to use `DescribeAutoScalingGroups`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
```

```
    public async Task<List<AutoScalingInstanceDetails>>
DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

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

            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }
```

- For API details, see DescribeAutoScalingGroups in *AWS SDK for .NET API Reference*.

`DescribeAutoScalingInstances`

The following code example shows how to use `DescribeAutoScalingInstances`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
 null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

            var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
            {
                MaxRecords = 10,
```

```
                InstanceIds = instanceIds,
            };

            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }
```

- For API details, see DescribeAutoScalingInstances in *AWS SDK for .NET API Reference*.

`DescribeScalingActivities`

The following code example shows how to use `DescribeScalingActivities`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
```

```
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
_amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }
```

- For API details, see DescribeScalingActivities in *AWS SDK for .NET API Reference*.

`DisableMetricsCollection`

The following code example shows how to use `DisableMetricsCollection`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
```

```
        {
            AutoScalingGroupName = groupName,
        };

        var response = await
_amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see DisableMetricsCollection in *AWS SDK for .NET API Reference*.

`EnableMetricsCollection`

The following code example shows how to use `EnableMetricsCollection`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Enable the collection of metric data for an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> EnableMetricsCollectionAsync(string groupName)
    {
        var listMetrics = new List<string>
            {
                "GroupMaxSize",
            };

        var collectionRequest = new EnableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
            Metrics = listMetrics,
            Granularity = "1Minute",
```

```
        };

        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see EnableMetricsCollection in *AWS SDK for .NET API Reference*.

`SetDesiredCapacity`

The following code example shows how to use `SetDesiredCapacity`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {
            AutoScalingGroupName = groupName,
            DesiredCapacity = desiredCapacity,
        };

        var response = await
_amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
```

```
        Console.WriteLine($"You have set the DesiredCapacity to
 {desiredCapacity}.");

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- For API details, see SetDesiredCapacity in *AWS SDK for .NET API Reference*.

`TerminateInstanceInAutoScalingGroup`

The following code example shows how to use `TerminateInstanceInAutoScalingGroup`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Terminate all instances in the Auto Scaling group in preparation for
    /// deleting the group.
    /// </summary>
    /// <param name="instanceId">The instance Id of the instance to terminate.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
        string instanceId)
    {
        var request = new TerminateInstanceInAutoScalingGroupRequest
        {
            InstanceId = instanceId,
            ShouldDecrementDesiredCapacity = false,
        };

        var response = await
 _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);
```

```
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You have terminated the instance: {instanceId}");
            return true;
        }

        Console.WriteLine($"Could not terminate {instanceId}");
        return false;
    }
```

- For API details, see TerminateInstanceInAutoScalingGroup in *AWS SDK for .NET API Reference*.

`UpdateAutoScalingGroup`

The following code example shows how to use `UpdateAutoScalingGroup`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
```

```
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
_amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully updated the Auto Scaling group
{groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }
```

- For API details, see UpdateAutoScalingGroup in *AWS SDK for .NET API Reference*.

Amazon Bedrock examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with Amazon Bedrock.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello Amazon Bedrock

The following code examples show how to get started using Amazon Bedrock.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using Amazon;
using Amazon.Bedrock;
using Amazon.Bedrock.Model;

namespace BedrockActions;

/// <summary>
/// This example shows how to list foundation models.
/// </summary>
internal class HelloBedrock
{
    /// <summary>
    /// Main method to call the ListFoundationModelsAsync method.
    /// </summary>
    /// <param name="args"> The command line arguments. </param>
    static async Task Main(string[] args)
    {
        // Specify a region endpoint where Amazon Bedrock is available. For a list
 of supported region see https://docs.aws.amazon.com/bedrock/latest/userguide/what-
is-bedrock.html#bedrock-regions
        AmazonBedrockClient bedrockClient = new(RegionEndpoint.USWest2);

        await ListFoundationModelsAsync(bedrockClient);

    }


    /// <summary>
    /// List foundation models.
    /// </summary>
```

```
   /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
   private static async Task ListFoundationModelsAsync(AmazonBedrockClient
bedrockClient)
   {
       Console.WriteLine("List foundation models with no filter.");

       try
       {
           var response = await bedrockClient.ListFoundationModelsAsync(new
ListFoundationModelsRequest()
           {
           });

           if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
           {
               foreach (var fm in response.ModelSummaries)
               {
                   WriteToConsole(fm);
               }
           }
           else
           {
               Console.WriteLine("Something wrong happened");
           }
       }
       catch (AmazonBedrockException e)
       {
           Console.WriteLine(e.Message);
       }
   }


   /// <summary>
   /// Write the foundation model summary to console.
   /// </summary>
   /// <param name="foundationModel"> The foundation model summary to write to
console. </param>
   private static void WriteToConsole(FoundationModelSummary foundationModel)
   {
       Console.WriteLine($"{foundationModel.ModelId}, Customization:
{string.Join(", ", foundationModel.CustomizationsSupported)}, Stream:
{foundationModel.ResponseStreamingSupported}, Input: {string.Join(",
", foundationModel.InputModalities)}, Output: {string.Join(", ",
foundationModel.OutputModalities)}");
```

```
    }
}
```

- For API details, see ListFoundationModels in *AWS SDK for .NET API Reference*.

Topics

- Actions

Actions

`ListFoundationModels`

The following code example shows how to use `ListFoundationModels`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

List the available Bedrock foundation models.

```
   /// <summary>
   /// List foundation models.
   /// </summary>
   /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
   private static async Task ListFoundationModelsAsync(AmazonBedrockClient
bedrockClient)
   {
       Console.WriteLine("List foundation models with no filter.");

       try
       {
           var response = await bedrockClient.ListFoundationModelsAsync(new
ListFoundationModelsRequest()
           {
```

```
            });

            if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                foreach (var fm in response.ModelSummaries)
                {
                    WriteToConsole(fm);
                }
            }
            else
            {
                Console.WriteLine("Something wrong happened");
            }
        }
        catch (AmazonBedrockException e)
        {
            Console.WriteLine(e.Message);
        }
    }
```

- For API details, see ListFoundationModels in *AWS SDK for .NET API Reference*.

Amazon Bedrock Runtime examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with Amazon Bedrock Runtime.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Anthropic Claude
- Cohere Command
- Meta Llama
- Mistral AI

Anthropic Claude

Converse

The following code example shows how to send a text message to Anthropic Claude, using Bedrock's Converse API.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a text message to Anthropic Claude, using Bedrock's Converse API.

```
// Use the Converse API to send a text message to Anthropic Claude.

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

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
```

```
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- For API details, see Converse in *AWS SDK for .NET API Reference*.

ConverseStream

The following code example shows how to send a text message to Anthropic Claude, using Bedrock's Converse API and process the response stream in real-time.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a text message to Anthropic Claude, using Bedrock's Converse API and process the response stream in real-time.

```
// Use the Converse API to send a text message to Anthropic Claude
// and print the response stream.

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

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
```

```
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- For API details, see ConverseStream in *AWS SDK for .NET API Reference*.

InvokeModel

The following code example shows how to send a text message to Anthropic Claude, using the Invoke Model API.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Use the Invoke Model API to send a text message.

```
// Use the native inference API to send a text message to Anthropic Claude.

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

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    anthropic_version = "bedrock-2023-05-31",
    max_tokens = 512,
    temperature = 0.5,
    messages = new[]
    {
        new { role = "user", content = userMessage }
    }
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
```

```
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["content"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- For API details, see InvokeModel in *AWS SDK for .NET API Reference*.

Cohere Command

Converse

The following code example shows how to send a text message to Cohere Command, using Bedrock's Converse API.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a text message to Cohere Command, using Bedrock's Converse API.

```
// Use the Converse API to send a text message to Cohere Command.

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

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
```

```
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- For API details, see Converse in *AWS SDK for .NET API Reference*.

ConverseStream

The following code example shows how to send a text message to Cohere Command, using Bedrock's Converse API and process the response stream in real-time.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a text message to Cohere Command, using Bedrock's Converse API and process the response stream in real-time.

```
// Use the Converse API to send a text message to Cohere Command
// and print the response stream.

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

```
// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
```

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

- For API details, see ConverseStream in *AWS SDK for .NET API Reference*.

Meta Llama

Converse

The following code example shows how to send a text message to Meta Llama, using Bedrock's Converse API.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a text message to Meta Llama, using Bedrock's Converse API.

```
// Use the Converse API to send a text message to Meta Llama.

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

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";
```

```
// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- For API details, see Converse in *AWS SDK for .NET API Reference*.

ConverseStream

The following code example shows how to send a text message to Meta Llama, using Bedrock's Converse API and process the response stream in real-time.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a text message to Meta Llama, using Bedrock's Converse API and process the response stream in real-time.

```
// Use the Converse API to send a text message to Meta Llama
// and print the response stream.

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

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
```

```
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- For API details, see ConverseStream in *AWS SDK for .NET API Reference*.

Mistral AI

Converse

The following code example shows how to send a text message to Mistral, using Bedrock's Converse API.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a text message to Mistral, using Bedrock's Converse API.

```
// Use the Converse API to send a text message to Mistral.

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

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
```

```
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- For API details, see Converse in *AWS SDK for .NET API Reference*.

ConverseStream

The following code example shows how to send a text message to Mistral, using Bedrock's Converse API and process the response stream in real-time.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Send a text message to Mistral, using Bedrock's Converse API and process the response stream in real-time.

```
// Use the Converse API to send a text message to Mistral
// and print the response stream.

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

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
```

```
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- For API details, see ConverseStream in *AWS SDK for .NET API Reference*.

CloudFormation examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with CloudFormation.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello CloudFormation

The following code example shows how to get started using CloudFormation.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;

namespace CloudFormationActions;

public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
```

```
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
                    new DescribeStacksRequest());
            if (paginatorForDescribeStacks.Stacks != null)
            {
                await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
                {
                    // Basic information for each stack
                    Console.WriteLine(
                        "\n-----------------------------------------------");
                    Console.WriteLine($"\nStack: {stack.StackName}");
                    Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                    Console.WriteLine($"  Created: {stack.CreationTime}");

                    // The tags of each stack (etc.)
                    if (stack.Tags != null && stack.Tags.Count > 0)
                    {
                        Console.WriteLine("  Tags:");
                        foreach (Tag tag in stack.Tags)
                            Console.WriteLine($"    {tag.Key}, {tag.Value}");
                    }

                    // The resources of each stack
                    DescribeStackResourcesResponse responseDescribeResources =
                        await _amazonCloudFormation.DescribeStackResourcesAsync(
                            new DescribeStackResourcesRequest
                            {
                                StackName = stack.StackName
                            });
                    if (responseDescribeResources.StackResources != null &&
responseDescribeResources.StackResources.Count > 0)
                    {
                        Console.WriteLine("  Resources:");
                        foreach (StackResource resource in responseDescribeResources
                                     .StackResources)
                            Console.WriteLine(
                                $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                    }
                }
            }
```

```
            Console.WriteLine("\n-----------------------------------------------");
            return true;
        }
        catch (AmazonCloudFormationException ex)
        {
            Console.WriteLine("Unable to get stack information:\n" + ex.Message);
            return false;
        }
        catch (AmazonServiceException ex)
        {
            if (ex.Message.Contains("Unable to get IAM security credentials"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are usnig SSO, be sure to install" +
                                  " the AWSSDK.SSO and AWSSDK.SSOOIDC packages.");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
        catch (ArgumentNullException ex)
        {
            if (ex.Message.Contains("Options property cannot be empty: ClientName"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are using SSO, have you logged in?");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
    }
```

- For API details, see [DescribeStackResources](#) in *AWS SDK for .NET API Reference*.

CloudWatch examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with CloudWatch.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello CloudWatch

The following code examples show how to get started using CloudWatch.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace CloudWatchActions;

public static class HelloCloudWatch
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon CloudWatch service.
```

```
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
ListMetricsRequest
        {
            Namespace = metricNamespace
        });
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
available in the {metricNamespace} namespace:");
        Console.WriteLine();
        if (response.Metrics != null)
        {
            foreach (var metric in response.Metrics.Take(5))
            {
                Console.WriteLine($"\tMetric: {metric.MetricName}");
                Console.WriteLine($"\tNamespace: {metric.Namespace}");
                Console.WriteLine(
                    $"\tDimensions: {string.Join(", ", metric.Dimensions.Select(m =>
$"{m.Name}:{m.Value}"))}");
                Console.WriteLine();
            }
        }
    }
}
```

- For API details, see ListMetrics in *AWS SDK for .NET API Reference*.

Topics

- Basics
- Actions

Basics

Learn the basics

The following code example shows how to:

- List CloudWatch namespaces and metrics.
- Get statistics for a metric and for estimated billing.
- Create and update a dashboard.
- Create and add data to a metric.
- Create and trigger an alarm, then view alarm history.
- Add an anomaly detector.
- Get a metric image, then clean up resources.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
public class CloudWatchScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
    https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

    This .NET example performs the following tasks:
        1. List and select a CloudWatch namespace.
        2. List and select a CloudWatch metric.
        3. Get statistics for a CloudWatch metric.
```

```
        4. Get estimated billing statistics for the last week.
        5. Create a new CloudWatch dashboard with two metrics.
        6. List current CloudWatch dashboards.
        7. Create a CloudWatch custom metric and add metric data.
        8. Add the custom metric to the dashboard.
        9. Create a CloudWatch alarm for the custom metric.
       10. Describe current CloudWatch alarms.
       11. Get recent data for the custom metric.
       12. Add data to the custom metric to trigger the alarm.
       13. Wait for an alarm state.
       14. Get history for the CloudWatch alarm.
       15. Add an anomaly detector.
       16. Describe current anomaly detectors.
       17. Get and display a metric image.
       18. Clean up resources.
   */

   private static ILogger logger = null!;
   private static CloudWatchWrapper _cloudWatchWrapper = null!;
   private static IConfiguration _configuration = null!;
   private static readonly List<string> _statTypes = new List<string>
{ "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
   private static SingleMetricAnomalyDetector? anomalyDetector = null!;

   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon service.
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
           .ConfigureServices((_, services) =>
           services.AddAWSService<IAmazonCloudWatch>()
           .AddTransient<CloudWatchWrapper>()
       )
       .Build();

       _configuration = new ConfigurationBuilder()
           .SetBasePath(Directory.GetCurrentDirectory())
           .AddJsonFile("settings.json") // Load settings from .json file.
           .AddJsonFile("settings.local.json",
               true) // Optionally, load local settings.
```

```
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CloudWatchScenario>();

        _cloudWatchWrapper = host.Services.GetRequiredService<CloudWatchWrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            var selectedNamespace = await SelectNamespace();
            var selectedMetric = await SelectMetric(selectedNamespace);
            await GetAndDisplayMetricStatistics(selectedNamespace, selectedMetric);
            await GetAndDisplayEstimatedBilling();
            await CreateDashboardWithMetrics();
            await ListDashboards();
            await CreateNewCustomMetric();
            await AddMetricToDashboard();
            await CreateMetricAlarm();
            await DescribeAlarms();
            await GetCustomMetricData();
            await AddMetricDataForAlarm();
            await CheckForMetricAlarm();
            await GetAlarmHistory();
            anomalyDetector = await AddAnomalyDetector();
            await DescribeAnomalyDetectors();
            await GetAndOpenMetricImage();
            await CleanupResources();
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
            await CleanupResources();
        }

    }

    /// <summary>
    /// Select a namespace.
    /// </summary>
    /// <returns>The selected namespace.</returns>
```

```
    private static async Task<string> SelectNamespace()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {namespaces[i]}");
        }

        var namespaceChoiceNumber = 0;
        while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
        {
            Console.WriteLine(
                "Select a namespace by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out namespaceChoiceNumber);
        }

        var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

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

        return selectedNamespace;
    }

    /// <summary>
    /// Select a metric from a namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <returns>The metric name.</returns>
    private static async Task<Metric> SelectMetric(string metricNamespace)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");

        var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

        for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
```

```
        {
            var dimensionsWithValues = namespaceMetrics[i].Dimensions
                .Where(d => !string.Equals("None", d.Value));
            Console.WriteLine($"\t{i + 1}. {namespaceMetrics[i].MetricName} " +
                              $"{string.Join(", :", dimensionsWithValues.Select(d =>
d.Value))}");
        }

        var metricChoiceNumber = 0;
        while (metricChoiceNumber < 1 || metricChoiceNumber >
namespaceMetrics.Count)
        {
            Console.WriteLine(
                "Select a metric by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out metricChoiceNumber);
        }

        var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

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

        return selectedMetric;
    }

    /// <summary>
    /// Get and display metric statistics for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayMetricStatistics(string metricNamespace,
Metric metric)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. Get CloudWatch metric statistics for the last day.");

        for (int i = 0; i < _statTypes.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {_statTypes[i]}");
        }

        var statisticChoiceNumber = 0;
```

```
        while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
        {
            Console.WriteLine(
                "Select a metric statistic by entering a number from the preceding
list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out statisticChoiceNumber);
        }

        var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
        var statisticsList = new List<string> { selectedStatistic };

        var metricStatistics = await
_cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
statisticsList, metric.Dimensions, 1, 60);

        if (!metricStatistics.Any())
        {
            Console.WriteLine($"No {selectedStatistic} statistics found for {metric}
in namespace {metricNamespace}.");
        }

        metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
        for (int i = 0; i < metricStatistics.Count && i < 10; i++)
        {
            var metricStat = metricStatistics[i];
            var statValue =
metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
            Console.WriteLine($"\t{i + 1}. Timestamp
{metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
        }

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

    /// <summary>
    /// Get and display estimated billing statistics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayEstimatedBilling()
    {
```

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Get CloudWatch estimated billing for the last
week.");

        var billingStatistics = await SetupBillingStatistics();

        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }

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

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CreateDashboardWithMetrics()
```

```
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
        var dashboardName = _configuration["dashboardName"];
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
        var newDashboardString = JsonSerializer.Serialize(
            newDashboard,
            new JsonSerializerOptions
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} was created.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List dashboards.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDashboards()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. List the CloudWatch dashboards in the current
account.");

        var dashboards = await _cloudWatchWrapper.ListDashboards();

        for (int i = 0; i < dashboards.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {dashboards[i].DashboardName}");
        }

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

```
    }

    /// <summary>
    /// Create and add data for a new custom metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateNewCustomMetric()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Create and add data for a new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var customData = await PutRandomMetricData(customMetricName,
customMetricNamespace);

        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");

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



    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
```

```
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Add the custom metric to the dashboard.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricToDashboard()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Add the new custom metric to the dashboard.");

        var dashboardName = _configuration["dashboardName"];

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var validationMessages = await SetupDashboard(customMetricNamespace,
customMetricName, dashboardName);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} updated with metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
```

```
    /// Set up a dashboard using a call to the wrapper class.
    /// </summary>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A list of validation messages.</returns>
    private static async Task<List<DashboardValidationMessage>> SetupDashboard(
        string customMetricNamespace, string customMetricName, string dashboardName)
    {
        // Get the dashboard model from configuration.
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

        // Add a new metric to the dashboard.
        newDashboard.Widgets.Add(new Widget
        {
            Height = 8,
            Width = 8,
            Y = 8,
            X = 0,
            Type = "metric",
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
```

```
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Create a CloudWatch alarm for the new metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var alarmName = _configuration["exampleAlarmName"];
        var accountId = _configuration["accountId"];
        var region = _configuration["region"];
        var emailTopic = _configuration["emailTopic"];
        var alarmActions = new List<string>();

        if (GetYesNoResponse(
                $"\tAdd an email action for topic {emailTopic} to alarm {alarmName}?
(y/n)"))
        {
            _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
alarmActions);
        }

        await _cloudWatchWrapper.PutMetricEmailAlarm(
            "Example metric alarm",
            alarmName,
            ComparisonOperator.GreaterThanOrEqualToThreshold,
            customMetricName,
            customMetricNamespace,
            100,
            alarmActions);

        Console.WriteLine($"\tAlarm {alarmName} added for metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
```

```
    }

    /// <summary>
    /// Describe Alarms.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAlarms()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Describe CloudWatch alarms in the current
account.");

        var alarms = await _cloudWatchWrapper.DescribeAlarms();
        alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

        for (int i = 0; i < alarms.Count && i < 10; i++)
        {
            var alarm = alarms[i];
            Console.WriteLine($"\t{i + 1}. {alarm.AlarmName}");
            Console.WriteLine($"\tState: {alarm.StateValue} for {alarm.MetricName}
{alarm.ComparisonOperator} {alarm.Threshold}");
        }

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

    /// <summary>
    /// Get the recent data for the metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetCustomMetricData()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. Get current data for new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var accountId = _configuration["accountId"];

        var query = new List<MetricDataQuery>
        {
            new MetricDataQuery
            {
                AccountId = accountId,
```

```
                Id = "m1",
                Label = "Custom Metric Data",
                MetricStat = new MetricStat
                {
                    Metric = new Metric
                    {
                        MetricName = customMetricName,
                        Namespace = customMetricNamespace,
                    },
                    Period = 1,
                    Stat = "Maximum"
                }
            }
        };

        var metricData = await _cloudWatchWrapper.GetMetricData(
            20,
            true,
            DateTime.UtcNow.AddMinutes(1),
            20,
            query);

        for (int i = 0; i < metricData.Count; i++)
        {
            if (metricData[i].Values != null)
            {
                for (int j = 0; j < metricData[i].Values.Count; j++)
                {
                    Console.WriteLine(
                        $"\tTimestamp {metricData[i].Timestamps[j]:G} Value:
{metricData[i].Values[j]}");
                }
            }
        }

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

    /// <summary>
    /// Add metric data to trigger an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricDataForAlarm()
    {
```

```
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"12. Add metric data to the custom metric to trigger an
alarm.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var nowUtc = DateTime.UtcNow;
        List<MetricDatum> customData = new List<MetricDatum>
        {
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-2)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-1)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc
            }
        };
        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");
        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);

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

    /// <summary>
    /// Check for a metric alarm using the DescribeAlarmsForMetric action.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckForMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"13. Checking for an alarm state.");
```

```
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var hasAlarm = false;
        var retries = 10;
        while (!hasAlarm && retries > 0)
        {
            var alarms = await
_cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
customMetricName);
            hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
            retries--;
            Thread.Sleep(20000);
        }

        Console.WriteLine(hasAlarm
            ? $"\tAlarm state found for {customMetricName}."
            : $"\tNo Alarm state found for {customMetricName} after 10 retries.");

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

    /// <summary>
    /// Get history for an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAlarmHistory()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"14. Get alarm history.");

        var exampleAlarmName = _configuration["exampleAlarmName"];

        var alarmHistory = await
_cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

        for (int i = 0; i < alarmHistory.Count; i++)
        {
            var history = alarmHistory[i];
            Console.WriteLine($"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
        }
        if (!alarmHistory.Any())
        {
```

```
            Console.WriteLine($"\tNo alarm history data found for
{exampleAlarmName}.");
        }

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

    /// <summary>
    /// Add an anomaly detector.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"15. Add an anomaly detector.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detector = new SingleMetricAnomalyDetector
        {
            MetricName = customMetricName,
            Namespace = customMetricNamespace,
            Stat = "Maximum"
        };
        await _cloudWatchWrapper.PutAnomalyDetector(detector);
        Console.WriteLine($"\tAdded anomaly detector for metric
{customMetricName}.");

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

    /// <summary>
    /// Describe anomaly detectors.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAnomalyDetectors()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"16. Describe anomaly detectors in the current
account.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
```

```
        var customMetricName = _configuration["customMetricName"];

        var detectors = await
_cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
customMetricName);

        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"\t{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state {detector.StateValue}");
        }

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

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"\tGetting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var memoryStream = await
_cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream, "MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"\tFile saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"\tPress enter to open the image.");
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;
```

```
        Process.Start(info);

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

    /// <summary>
    /// Clean up created resources.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"18. Clean up resources.");

        var dashboardName = _configuration["dashboardName"];
        if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
            Console.WriteLine($"\tDeleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"\tCleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
anomalyDetector != null)
        {
            Console.WriteLine($"\tCleaning up anomaly detector.");

            await _cloudWatchWrapper.DeleteAnomalyDetector(
                anomalyDetector);
        }

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

```
    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

Wrapper methods used by the scenario for CloudWatch actions.

```
/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
 ILogger<CloudWatchWrapper> logger)

    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
```

```
   /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
   /// <param name="filter">Optional dimension filter.</param>
   /// <param name="metricName">Optional metric name filter.</param>
   /// <returns>The list of metrics.</returns>
   public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
   {
       var results = new List<Metric>();
       var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
           new ListMetricsRequest
           {
               Namespace = metricNamespace,
               Dimensions = filter != null ? new List<DimensionFilter> { filter } :
null,
               MetricName = metricName
           });
       // Get the entire list using the paginator.
       await foreach (var metric in paginateMetrics.Metrics)
       {
           results.Add(metric);
       }

       return results;
   }

   /// <summary>
   /// Wrapper to get statistics for a specific CloudWatch metric.
   /// </summary>
   /// <param name="metricNamespace">The namespace of the metric.</param>
   /// <param name="metricName">The name of the metric.</param>
   /// <param name="statistics">The list of statistics to include.</param>
   /// <param name="dimensions">The list of dimensions to include.</param>
   /// <param name="days">The number of days in the past to include.</param>
   /// <param name="period">The period for the data.</param>
   /// <returns>A list of DataPoint objects for the statistics.</returns>
   public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
       string metricName, List<string> statistics, List<Dimension> dimensions, int
days, int period)
   {
       var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
           new GetMetricStatisticsRequest()
           {
               Namespace = metricNamespace,
```

```
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard was
 created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
 List<DashboardValidationMessage>();
    }


    /// <summary>
    /// Get information on a dashboard.
    /// </summary>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A JSON object with dashboard information.</returns>
    public async Task<string> GetDashboard(string dashboardName)
    {
```

```
        var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
            new GetDashboardRequest()
            {
                DashboardName = dashboardName
            });

        return dashboardResponse.DashboardBody;
    }


    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });
```

```
        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
```

```
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }

    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
 useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
```

```
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }

    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
        try
        {
            var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
```

```
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }

    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
```

```
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
```

```
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }

    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
```

```
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
```

```
   /// Describe anomaly detectors for a metric and namespace.
   /// </summary>
   /// <param name="metricNamespace">The namespace of the metric.</param>
   /// <param name="metricName">The metric of the anomaly detectors.</param>
   /// <returns>The list of detectors.</returns>
   public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
   {
       List<AnomalyDetector> detectors = new List<AnomalyDetector>();
       var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
           new DescribeAnomalyDetectorsRequest()
           {
               MetricName = metricName,
               Namespace = metricNamespace
           });

       await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
       {
           detectors.Add(data);
       }

       return detectors;
   }

   /// <summary>
   /// Delete a single metric anomaly detector.
   /// </summary>
   /// <param name="anomalyDetector">The anomaly detector to delete.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
   {
       var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
           new DeleteAnomalyDetectorRequest()
           {
               SingleMetricAnomalyDetector = anomalyDetector
           });

       return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
   }
```

```
    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
 _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors
  - GetMetricData
  - GetMetricStatistics
  - GetMetricWidgetImage
  - ListMetrics
  - PutAnomalyDetector
  - PutDashboard
  - PutMetricAlarm
  - PutMetricData

Actions

`DeleteAlarms`

The following code example shows how to use `DeleteAlarms`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Delete a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAlarms(List<string> alarmNames)
{
    var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
        new DeleteAlarmsRequest()
        {
            AlarmNames = alarmNames
        });

    return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- For API details, see DeleteAlarms in *AWS SDK for .NET API Reference*.

`DeleteAnomalyDetector`

The following code example shows how to use `DeleteAnomalyDetector`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteAnomalyDetector in *AWS SDK for .NET API Reference*.

`DeleteDashboards`

The following code example shows how to use `DeleteDashboards`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
_amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DeleteDashboards in *AWS SDK for .NET API Reference*.

`DescribeAlarmHistory`

The following code example shows how to use `DescribeAlarmHistory`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
    {
```

```
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }
```

- For API details, see DescribeAlarmHistory in *AWS SDK for .NET API Reference*.

`DescribeAlarms`

The following code example shows how to use `DescribeAlarms`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
```

```
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }
```

- For API details, see DescribeAlarms in *AWS SDK for .NET API Reference*.

`DescribeAlarmsForMetric`

The following code example shows how to use `DescribeAlarmsForMetric`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
```

```
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }
```

- For API details, see DescribeAlarmsForMetric in *AWS SDK for .NET API Reference*.

`DescribeAnomalyDetectors`

The following code example shows how to use `DescribeAnomalyDetectors`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
```

```
        {
            detectors.Add(data);
        }

        return detectors;
    }
```

- For API details, see DescribeAnomalyDetectors in *AWS SDK for .NET API Reference*.

`DisableAlarmActions`

The following code example shows how to use `DisableAlarmActions`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see DisableAlarmActions in *AWS SDK for .NET API Reference*.

`EnableAlarmActions`

The following code example shows how to use `EnableAlarmActions`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see EnableAlarmActions in *AWS SDK for .NET API Reference*.

`GetDashboard`

The following code example shows how to use `GetDashboard`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Get information on a dashboard.
/// </summary>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}
```

- For API details, see GetDashboard in *AWS SDK for .NET API Reference*.

`GetMetricData`

The following code example shows how to use `GetMetricData`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
 useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
```

```
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }
```

- For API details, see GetMetricData in *AWS SDK for .NET API Reference*.

`GetMetricStatistics`

The following code example shows how to use `GetMetricStatistics`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();
```

```
        return billingStatistics;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions, int
days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }
```

- For API details, see GetMetricStatistics in *AWS SDK for .NET API Reference*.

`GetMetricWidgetImage`

The following code example shows how to use `GetMetricWidgetImage`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
```

```
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }
```

- For API details, see GetMetricWidgetImage in *AWS SDK for .NET API Reference*.

`ListDashboards`

The following code example shows how to use `ListDashboards`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
```

```
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }
```

- For API details, see ListDashboards in *AWS SDK for .NET API Reference*.

`ListMetrics`

The following code example shows how to use `ListMetrics`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter> { filter } :
null,
                MetricName = metricName
```

```
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }
```

- For API details, see ListMetrics in *AWS SDK for .NET API Reference*.

`PutAnomalyDetector`

The following code example shows how to use `PutAnomalyDetector`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see PutAnomalyDetector in *AWS SDK for .NET API Reference*.

`PutDashboard`

The following code example shows how to use `PutDashboard`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Set up a dashboard using a call to the wrapper class.
/// </summary>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <param name="customMetricName">The metric name.</param>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A list of validation messages.</returns>
private static async Task<List<DashboardValidationMessage>> SetupDashboard(
    string customMetricNamespace, string customMetricName, string dashboardName)
{
    // Get the dashboard model from configuration.
    var newDashboard = new DashboardModel();
    _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

    // Add a new metric to the dashboard.
    newDashboard.Widgets.Add(new Widget
    {
        Height = 8,
        Width = 8,
        Y = 8,
        X = 0,
        Type = "metric",
        Properties = new Properties
        {
            Metrics = new List<List<object>>
```

```
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
 newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard was
 created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
```

```
            });

        return dashboardResponse.DashboardValidationMessages ?? new
List<DashboardValidationMessage>();
    }
```

- For API details, see PutDashboard in *AWS SDK for .NET API Reference*.

`PutMetricAlarm`

The following code example shows how to use `PutMetricAlarm`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
        try
        {
            var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
```

```
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
```

```
    }
```

- For API details, see PutMetricAlarm in *AWS SDK for .NET API Reference*.

`PutMetricData`

The following code example shows how to use `PutMetricData`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Add some metric data using a call to a wrapper class.
   /// </summary>
   /// <param name="customMetricName">The metric name.</param>
   /// <param name="customMetricNamespace">The metric namespace.</param>
   /// <returns></returns>
   private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
       string customMetricNamespace)
   {
       List<MetricDatum> customData = new List<MetricDatum>();
       Random rnd = new Random();

       // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
       var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
       for (int i = 0; i < 10; i++)
       {
           var metricValue = rnd.Next(0, 100);
           customData.Add(
               new MetricDatum
               {
                   MetricName = customMetricName,
```

```
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- For API details, see PutMetricData in *AWS SDK for .NET API Reference*.

CloudWatch Logs examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with CloudWatch Logs.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions
- Scenarios

Actions

`GetQueryResults`

The following code example shows how to use `GetQueryResults`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Gets the results of a CloudWatch Logs Insights query.
    /// </summary>
    /// <param name="queryId">The ID of the query.</param>
    /// <returns>The query results response.</returns>
    public async Task<GetQueryResultsResponse?> GetQueryResultsAsync(string queryId)
    {
        try
        {
            var request = new GetQueryResultsRequest
            {
                QueryId = queryId
            };

            var response = await
_amazonCloudWatchLogs.GetQueryResultsAsync(request);
            return response;
        }
        catch (ResourceNotFoundException ex)
```

```
        {
            _logger.LogError($"Query not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while getting query results:
 {ex.Message}");
            return null;
        }
    }
```

- For API details, see GetQueryResults in *AWS SDK for .NET API Reference*.

`StartQuery`

The following code example shows how to use `StartQuery`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Starts a CloudWatch Logs Insights query.
    /// </summary>
    /// <param name="logGroupName">The name of the log group to query.</param>
    /// <param name="queryString">The CloudWatch Logs Insights query string.</param>
    /// <param name="startTime">The start time for the query (seconds since
 epoch).</param>
    /// <param name="endTime">The end time for the query (seconds since epoch).</
param>
    /// <param name="limit">The maximum number of results to return.</param>
    /// <returns>The query ID if successful, null otherwise.</returns>
    public async Task<string?> StartQueryAsync(
        string logGroupName,
        string queryString,
```

```
        long startTime,
        long endTime,
        int limit = 10000)
    {
        try
        {
            var request = new StartQueryRequest
            {
                LogGroupName = logGroupName,
                QueryString = queryString,
                StartTime = startTime,
                EndTime = endTime,
                Limit = limit
            };

            var response = await _amazonCloudWatchLogs.StartQueryAsync(request);
            return response.QueryId;
        }
        catch (InvalidParameterException ex)
        {
            _logger.LogError($"Invalid parameter for query: {ex.Message}");
            return null;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Log group not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while starting query:
{ex.Message}");
            return null;
        }
    }
```

- For API details, see StartQuery in *AWS SDK for .NET API Reference*.

Scenarios

Run a large query

The following code example shows how to use CloudWatch Logs to query more than 10,000 records.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This is the main workflow that demonstrates the large query scenario.

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
using CloudWatchLogsActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace CloudWatchLogsScenario;

public class LargeQueryWorkflow
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.
    This .NET code example performs the following tasks for the CloudWatch Logs
 Large Query workflow:

    1. Prepare the Application:
       - Prompt the user to deploy CloudFormation stack and generate sample logs.
       - Deploy the CloudFormation template for resource creation.
       - Generate 50,000 sample log entries using CloudWatch Logs API.
```

```
       - Wait 5 minutes for logs to be fully ingested.

    2. Execute Large Query:
       - Perform recursive queries to retrieve all logs using binary search.
       - Display progress for each query executed.
       - Show total execution time and logs found.

    3. Clean up:
       - Prompt the user to delete the CloudFormation stack and all resources.
       - Destroy the CloudFormation stack and wait until removed.
    */

    public static ILogger<LargeQueryWorkflow> _logger = null!;
    public static CloudWatchLogsWrapper _wrapper = null!;
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    private static string _logGroupName = "/workflows/cloudwatch-logs/large-query";
    private static string _logStreamName = "stream1";
    private static long _queryStartDate;
    private static long _queryEndDate;

    public static bool _interactive = true;
    public static string _stackName = "CloudWatchLargeQueryStack";
    private static string _stackResourcePath = "../../../../../../../scenarios/
features/cloudwatch_logs_large_query/resources/stack.yaml";

    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter("Microsoft", LogLevel.Information))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatchLogs>()
                    .AddAWSService<IAmazonCloudFormation>()
                    .AddTransient<CloudWatchLogsWrapper>()
            )
            .Build();

        if (_interactive)
        {
            _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
                .CreateLogger<LargeQueryWorkflow>();
```

```
            _wrapper = host.Services.GetRequiredService<CloudWatchLogsWrapper>();
            _amazonCloudFormation =
host.Services.GetRequiredService<IAmazonCloudFormation>();
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
```

```
    /// Runs the scenario workflow. Used for testing.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
    /// Prepares the application by creating the necessary resources.
```

```
    /// </summary>
    /// <returns>True if the application was prepared successfully.</returns>
    public static async Task<bool> PrepareApplication()
    {
        Console.WriteLine("Preparing the application...");
        Console.WriteLine();

        try
        {
            var deployStack = !_interactive || GetYesNoResponse(
                "Would you like to deploy the CloudFormation stack and generate
sample logs? (y/n) ");

            if (deployStack)
            {
                if (_interactive)
                {
                    Console.Write(
                        $"Enter a path for the CloudFormation stack resource .yaml
file (or press Enter for default '{_stackResourcePath}'): ");
                    string? inputPath = Console.ReadLine();
                    if (!string.IsNullOrWhiteSpace(inputPath))
                    {
                        _stackResourcePath = inputPath;
                    }
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName);

                if (deploySuccess)
                {
                    Console.WriteLine();
                    Console.WriteLine("Generating 50,000 sample log entries...");
                    var generateSuccess = await GenerateSampleLogs();

                    if (generateSuccess)
                    {
                        Console.WriteLine();
                        Console.WriteLine("Sample logs created. Waiting 5 minutes
for logs to be fully ingested...");
                        await WaitWithCountdown(300);
```

```
                        Console.WriteLine("Application preparation complete.");
                        return true;
                    }
                }
            }
            else
            {
                _logGroupName = PromptUserForInput("Enter the log group name ",
_logGroupName);
                _logStreamName = PromptUserForInput("Enter the log stream name ",
_logStreamName);

                var startDateMs = PromptUserForLong("Enter the query start date
(milliseconds since epoch): ");
                var endDateMs = PromptUserForLong("Enter the query end date
(milliseconds since epoch): ");

                _queryStartDate = startDateMs / 1000;
                _queryEndDate = endDateMs / 1000;

                Console.WriteLine("Application preparation complete.");
                return true;
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while preparing the
application.");
        }

        Console.WriteLine("Application preparation failed.");
        return false;
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
```

```
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath)
            };

            var response = await _amazonCloudFormation.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    _logger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                _logger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            _logger.LogWarning($"CloudFormation stack '{stackName}' already exists.
Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName);
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
                }
            }
```

```
            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for CloudFormation stack creation to
complete.");
        return false;
    }

    /// <summary>
    /// Generates sample logs directly using CloudWatch Logs API.
    /// Creates 50,000 log entries spanning 5 minutes.
    /// </summary>
    /// <returns>True if logs were generated successfully.</returns>
    private static async Task<bool> GenerateSampleLogs()
    {
        const int totalEntries = 50000;
        const int entriesPerBatch = 10000;
        const int fiveMinutesMs = 5 * 60 * 1000;

        try
        {
            // Calculate timestamps
            var startTimeMs = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds();
            var timestampIncrement = fiveMinutesMs / totalEntries;

            Console.WriteLine($"Generating {totalEntries} log entries...");

            var entryCount = 0;
            var currentTimestamp = startTimeMs;
            var numBatches = totalEntries / entriesPerBatch;

            // Generate and upload logs in batches
            for (int batchNum = 0; batchNum < numBatches; batchNum++)
            {
                var logEvents = new List<InputLogEvent>();

                for (int i = 0; i < entriesPerBatch; i++)
                {
                    logEvents.Add(new InputLogEvent
                    {
```

```
                        Timestamp =
 DateTimeOffset.FromUnixTimeMilliseconds(currentTimestamp).UtcDateTime,
                        Message = $"Entry {entryCount}"
                    });

                    entryCount++;
                    currentTimestamp += timestampIncrement;
                }

                // Upload batch
                var success = await _wrapper.PutLogEventsAsync(_logGroupName,
 _logStreamName, logEvents);
                if (!success)
                {
                    _logger.LogError($"Failed to upload batch {batchNum + 1}/
{numBatches}");
                    return false;
                }

                Console.WriteLine($"Uploaded batch {batchNum + 1}/{numBatches}");
            }

            // Set query date range (convert milliseconds to seconds for query API)
            _queryStartDate = startTimeMs / 1000;
            _queryEndDate = (currentTimestamp - timestampIncrement) / 1000;

            Console.WriteLine($"Query start date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryStartDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Query end date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryEndDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Successfully uploaded {totalEntries} log entries");

            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while generating sample logs.");
            return false;
        }
    }

    /// <summary>
    /// Executes the large query workflow.
    /// </summary>
```

```
    public static async Task ExecuteLargeQuery()
    {
        Console.WriteLine("Starting recursive query to retrieve all logs...");
        Console.WriteLine();

        var queryLimit = PromptUserForInteger("Enter the query limit (max 10000) ",
10000);
        if (queryLimit > 10000) queryLimit = 10000;

        var queryString = "fields @timestamp, @message | sort @timestamp asc";

        var stopwatch = Stopwatch.StartNew();
        var allResults = await PerformLargeQuery(_logGroupName, queryString,
_queryStartDate, _queryEndDate, queryLimit);
        stopwatch.Stop();

        Console.WriteLine();
        Console.WriteLine($"Queries finished in {stopwatch.Elapsed.TotalSeconds:F3}
seconds.");
        Console.WriteLine($"Total logs found: {allResults.Count}");

        // Check for duplicates
        Console.WriteLine();
        Console.WriteLine("Checking for duplicate logs...");
        var duplicates = FindDuplicateLogs(allResults);
        if (duplicates.Count > 0)
        {
            Console.WriteLine($"WARNING: Found {duplicates.Count} duplicate log
entries!");
            Console.WriteLine("Duplicate entries (showing first 10):");
            foreach (var dup in duplicates.Take(10))
            {
                Console.WriteLine($"  [{dup.Timestamp}] {dup.Message} (appears
{dup.Count} times)");
            }

            var uniqueCount = allResults.Count - duplicates.Sum(d => d.Count - 1);
            Console.WriteLine($"Unique logs: {uniqueCount}");
        }
        else
        {
            Console.WriteLine("No duplicates found. All logs are unique.");
        }
        Console.WriteLine();
```

```
        var viewSample = !_interactive || GetYesNoResponse("Would you like to see a
sample of the logs? (y/n) ");
        if (viewSample)
        {
            Console.WriteLine();
            Console.WriteLine($"Sample logs (first 10 of {allResults.Count}):");
            for (int i = 0; i < Math.Min(10, allResults.Count); i++)
            {
                var timestamp = allResults[i].Find(f => f.Field ==
"@timestamp")?.Value ?? "N/A";
                var message = allResults[i].Find(f => f.Field ==
"@message")?.Value ?? "N/A";
                Console.WriteLine($"[{timestamp}] {message}");
            }
        }
    }

    /// <summary>
    /// Performs a large query using recursive binary search.
    /// </summary>
    private static async Task<List<List<ResultField>>> PerformLargeQuery(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit)
    {
        var queryId = await _wrapper.StartQueryAsync(logGroupName, queryString,
startTime, endTime, limit);
        if (queryId == null)
        {
            return new List<List<ResultField>>();
        }

        var results = await PollQueryResults(queryId);
        if (results == null || results.Count == 0)
        {
            return new List<List<ResultField>>();
        }

        var startDate =
DateTimeOffset.FromUnixTimeSeconds(startTime).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
```

```
        var endDate = DateTimeOffset.FromUnixTimeSeconds(endTime).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        Console.WriteLine($"Query date range: {startDate} ({startTime}s) to
 {endDate} ({endTime}s). Found {results.Count} logs.");

        if (results.Count < limit)
        {
            Console.WriteLine($"  -> Returning {results.Count} logs (less than limit
 of {limit})");
            return results;
        }

        Console.WriteLine($"  -> Hit limit of {limit}. Need to split and recurse.");

        // Get the timestamp of the last log (sorted to find the actual last one)
        var lastLogTimestamp = GetLastLogTimestamp(results);
        if (lastLogTimestamp == null)
        {
            Console.WriteLine($"  -> No timestamp found in results. Returning
 {results.Count} logs.");
            return results;
        }

        Console.WriteLine($"  -> Last log timestamp: {lastLogTimestamp}");

        // Parse the timestamp and add 1 millisecond to avoid querying the same log
 again
        var lastLogDate = DateTimeOffset.Parse(lastLogTimestamp + " +0000");
        Console.WriteLine($"  -> Last log as DateTimeOffset: {lastLogDate:yyyy-MM-
ddTHH:mm:ss.fffZ} ({lastLogDate.ToUnixTimeSeconds()}s)");

        var offsetLastLogDate = lastLogDate.AddMilliseconds(1);
        Console.WriteLine($"  -> Offset timestamp (last
 + 1ms): {offsetLastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ}
 ({offsetLastLogDate.ToUnixTimeSeconds()}s)");

        // Convert to seconds, but round UP to the next second to avoid overlapping
 with logs in the same second
        // This ensures we don't re-query logs that share the same second as the
 last log
        var offsetLastLogTime = offsetLastLogDate.ToUnixTimeSeconds();
        if (offsetLastLogDate.Millisecond > 0)
        {
            offsetLastLogTime++; // Move to the next full second
```

```
            Console.WriteLine($"  -> Adjusted to next full second:
 {offsetLastLogTime}s ({DateTimeOffset.FromUnixTimeSeconds(offsetLastLogTime):yyyy-
MM-ddTHH:mm:ss.fffZ})");
        }

        Console.WriteLine($"  -> Comparing: offsetLastLogTime={offsetLastLogTime}s
 vs endTime={endTime}s");
        Console.WriteLine($"  -> End time as date:
 {DateTimeOffset.FromUnixTimeSeconds(endTime):yyyy-MM-ddTHH:mm:ss.fffZ}");

        // Check if there's any time range left to query
        if (offsetLastLogTime >= endTime)
        {
            Console.WriteLine($"  -> No time range left to query. Offset time
 ({offsetLastLogTime}s) >= end time ({endTime}s)");
            return results;
        }

        // Split the remaining date range in half
        var (range1Start, range1End, range2Start, range2End) =
 SplitDateRange(offsetLastLogTime, endTime);

        var range1StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range1Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range1EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range1End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
        var range2StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range2Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range2EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range2End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");

        Console.WriteLine($"  -> Splitting remaining range:");
        Console.WriteLine($"     Range 1: {range1StartDate} ({range1Start}s) to
 {range1EndDate} ({range1End}s)");
        Console.WriteLine($"     Range 2: {range2StartDate} ({range2Start}s) to
 {range2EndDate} ({range2End}s)");

        // Query both halves recursively
        Console.WriteLine($"  -> Querying range 1...");
        var results1 = await PerformLargeQuery(logGroupName, queryString,
 range1Start, range1End, limit);
        Console.WriteLine($"  -> Range 1 returned {results1.Count} logs");
```

```
        Console.WriteLine($"  -> Querying range 2...");
        var results2 = await PerformLargeQuery(logGroupName, queryString,
range2Start, range2End, limit);
        Console.WriteLine($"  -> Range 2 returned {results2.Count} logs");

        // Combine all results
        var allResults = new List<List<ResultField>>(results);
        allResults.AddRange(results1);
        allResults.AddRange(results2);

        Console.WriteLine($"  -> Combined total: {allResults.Count} logs
({results.Count} + {results1.Count} + {results2.Count})");

        return allResults;
    }

    /// <summary>
    /// Gets the timestamp string of the most recent log from a list of logs.
    /// Sorts timestamps to find the actual last one.
    /// </summary>
    private static string? GetLastLogTimestamp(List<List<ResultField>> logs)
    {
        var timestamps = logs
            .Select(log => log.Find(f => f.Field == "@timestamp")?.Value)
            .Where(t => !string.IsNullOrEmpty(t))
            .OrderBy(t => t)
            .ToList();

        if (timestamps.Count == 0)
        {
            return null;
        }

        return timestamps[timestamps.Count - 1];
    }

    /// <summary>
    /// Splits a date range in half.
    /// Range 2 starts at midpoint + 1 second to avoid overlap.
    /// </summary>
    private static (long range1Start, long range1End, long range2Start, long
range2End) SplitDateRange(long startTime, long endTime)
    {
```

```
        var midpoint = startTime + (endTime - startTime) / 2;
        // Range 2 starts at midpoint + 1 to avoid querying the same second twice
        return (startTime, midpoint, midpoint + 1, endTime);
    }

    /// <summary>
    /// Polls for query results until complete.
    /// </summary>
    private static async Task<List<List<ResultField>>?> PollQueryResults(string
queryId)
    {
        int retryCount = 0;
        const int maxRetries = 60;
        const int retryDelay = 1000;

        while (retryCount < maxRetries)
        {
            var response = await _wrapper.GetQueryResultsAsync(queryId);
            if (response == null)
            {
                return null;
            }

            if (response.Status == QueryStatus.Complete)
            {
                return response.Results;
            }

            if (response.Status == QueryStatus.Failed ||
                response.Status == QueryStatus.Cancelled ||
                response.Status == QueryStatus.Timeout ||
                response.Status == QueryStatus.Unknown)
            {
                _logger.LogError($"Query failed with status: {response.Status}");
                return null;
            }

            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for query results.");
        return null;
    }
```

```
   /// <summary>
   /// Cleans up the resources created during the scenario.
   /// </summary>
   public static async Task<bool> Cleanup()
   {
       var cleanup = !_interactive || GetYesNoResponse(
           "Do you want to delete the CloudFormation stack and all resources? (y/n)
");

       if (cleanup)
       {
           try
           {
               var stackDeleteSuccess = await DeleteCloudFormationStack(_stackName,
false);
               return stackDeleteSuccess;
           }
           catch (Exception ex)
           {
               _logger.LogError(ex, "An error occurred while cleaning up the
resources.");
               return false;
           }
       }

       Console.WriteLine($"Resources will remain. Stack name: {_stackName}, Log
group: {_logGroupName}");
       _logger.LogInformation("CloudWatch Logs Large Query scenario is complete.");
       return true;
   }

   /// <summary>
   /// Deletes the CloudFormation stack and waits for confirmation.
   /// </summary>
   private static async Task<bool> DeleteCloudFormationStack(string stackName, bool
forceDelete)
   {
       var request = new DeleteStackRequest
       {
           StackName = stackName,
       };

       if (forceDelete)
```

```
        {
            request.DeletionMode = DeletionMode.FORCE_DELETE_STACK;
        }

        await _amazonCloudFormation.DeleteStackAsync(request);
        Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

        bool stackDeleted = await WaitForStackDeletion(stackName, forceDelete);

        if (stackDeleted)
        {
            Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
            return true;
        }
        else
        {
            _logger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
```

```
                var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }

                if (!forceDelete && describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_FAILED)
                {
                    return await DeleteCloudFormationStack(stackName, true);
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError($"Timed out waiting for CloudFormation stack '{stackName}'
to be deleted.");
        return false;
    }

    /// <summary>
    /// Waits with a countdown display.
    /// </summary>
    private static async Task WaitWithCountdown(int seconds)
    {
        for (int i = seconds; i > 0; i--)
        {
            Console.Write($"\rWaiting: {i} seconds remaining...  ");
            await Task.Delay(1000);
        }
        Console.WriteLine("\rWait complete.                      ");
```

```
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (_interactive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = "[a-zA-Z][-a-zA-Z0-9]*";
                if (!Regex.IsMatch(input, regex))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
{_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }

    /// <summary>
    /// Prompts the user for input with a default value.
    /// </summary>
    private static string PromptUserForInput(string prompt, string defaultValue)
    {
```

```
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            return string.IsNullOrWhiteSpace(input) ? defaultValue : input;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for an integer value.
    /// </summary>
    private static int PromptUserForInteger(string prompt, int defaultValue)
    {
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            if (string.IsNullOrWhiteSpace(input) || !int.TryParse(input, out var
result))
            {
                return defaultValue;
            }
            return result;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for a long value.
    /// </summary>
    private static long PromptUserForLong(string prompt)
    {
        if (_interactive)
        {
            Console.Write(prompt);
            string? input = Console.ReadLine();
            if (long.TryParse(input, out var result))
            {
                return result;
            }
        }
        return 0;
    }
```

```
    /// <summary>
    /// Finds duplicate log entries based on timestamp and message.
    /// </summary>
    private static List<(string Timestamp, string Message, int Count)>
 FindDuplicateLogs(List<List<ResultField>> logs)
    {
        var logSignatures = new Dictionary<string, int>();

        foreach (var log in logs)
        {
            var timestamp = log.Find(f => f.Field == "@timestamp")?.Value ?? "";
            var message = log.Find(f => f.Field == "@message")?.Value ?? "";
            var signature = $"{timestamp}|{message}";

            if (logSignatures.ContainsKey(signature))
            {
                logSignatures[signature]++;
            }
            else
            {
                logSignatures[signature] = 1;
            }
        }

        return logSignatures
            .Where(kvp => kvp.Value > 1)
            .Select(kvp =>
            {
                var parts = kvp.Key.Split('|');
                return (Timestamp: parts[0], Message: parts[1], Count: kvp.Value);
            })
            .OrderByDescending(x => x.Count)
            .ToList();
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - GetQueryResults
  - StartQuery

Amazon Cognito Identity Provider examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with Amazon Cognito Identity Provider.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

`ListUserPools`

The following code example shows how to use `ListUserPools`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List the Amazon Cognito user pools for an account.
    /// </summary>
    /// <returns>A list of UserPoolDescriptionType objects.</returns>
    public async Task<List<UserPoolDescriptionType>> ListUserPoolsAsync()
    {
        var userPools = new List<UserPoolDescriptionType>();

        var userPoolsPaginator = _cognitoService.Paginators.ListUserPools(new
ListUserPoolsRequest());
```

```
        await foreach (var response in userPoolsPaginator.Responses)
        {
            userPools.AddRange(response.UserPools);
        }

        return userPools;
    }
```

- For API details, see ListUserPools in *AWS SDK for .NET API Reference*.

AWS Control Tower examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with AWS Control Tower.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello AWS Control Tower

The following code examples show how to get started using AWS Control Tower.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using Amazon.ControlTower;
```

```
using Amazon.ControlTower.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;
using LogLevel = Microsoft.Extensions.Logging.LogLevel;

namespace ControlTowerActions;

/// <summary>
/// A class that introduces the AWS Control Tower by listing the
/// available baselines for the account.
/// </summary>
public class HelloControlTower
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Control Tower.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloControlTower>();

        var amazonClient = host.Services.GetRequiredService<IAmazonControlTower>();

        Console.Clear();
        Console.WriteLine("Hello, AWS Control Tower! Let's list available
 baselines:");
        Console.WriteLine();

        var baselines = new List<BaselineSummary>();
```

```
        try
        {
            var baselinesPaginator = amazonClient.Paginators.ListBaselines(new
 ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            Console.WriteLine($"{baselines.Count} baseline(s) retrieved.");
            foreach (var baseline in baselines)
            {
                Console.WriteLine($"\t{baseline.Name}");
            }
        }
        catch (Amazon.ControlTower.Model.AccessDeniedException)
        {
            Console.WriteLine("Access denied. Please ensure you have the necessary
 permissions.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
}
```

- For API details, see ListBaselines in *AWS SDK for .NET API Reference*.

Topics

- Basics
- Actions

Basics

Learn the basics

The following code example shows how to:

- List landing zones.
- List, enable, get, reset, and disable baselines.
- List, enable, get, and disable controls.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario demonstrating AWS Control Tower features.

```
using Amazon.ControlCatalog;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using Amazon.Organizations;
using Amazon.Organizations.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using ControlTowerActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace ControlTowerBasics;

/// <summary>
/// Scenario class for AWS Control Tower basics.
/// </summary>
public class ControlTowerBasics
{
    public static bool isInteractive = true;
    public static ILogger logger = null!;
    public static IAmazonOrganizations? orgClient = null;
    public static IAmazonSecurityTokenService? stsClient = null;
    public static ControlTowerWrapper? wrapper = null;
    private static string? ouArn;
    private static bool useLandingZone = false;
```

```
    /// <summary>
    /// Main entry point for the AWS Control Tower basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
                .AddAWSService<IAmazonControlCatalog>()
                .AddAWSService<IAmazonOrganizations>()
                .AddAWSService<IAmazonSecurityTokenService>()
                .AddTransient<ControlTowerWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<ControlTowerBasics>();

        wrapper = host.Services.GetRequiredService<ControlTowerWrapper>();
        orgClient = host.Services.GetRequiredService<IAmazonOrganizations>();
        stsClient = host.Services.GetRequiredService<IAmazonSecurityTokenService>();

        await RunScenario();
    }

    /// <summary>
    /// Runs the example scenario.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("\tWelcome to the AWS Control Tower with ControlCatalog
example scenario.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("This demo will walk you through working with AWS Control
Tower for landing zones,");
        Console.WriteLine("managing baselines, and working with controls.");

        try
        {
            var accountId = (await stsClient!.GetCallerIdentityAsync(new
GetCallerIdentityRequest())).Account;
```

```
            Console.WriteLine($"\nAccount ID: {accountId}");

            Console.WriteLine("\nSome demo operations require the use of a landing
 zone.");
            Console.WriteLine("You can use an existing landing zone or opt out of
 these operations in the demo.");
            Console.WriteLine("For instructions on how to set up a landing zone,");
            Console.WriteLine("see https://docs.aws.amazon.com/controltower/latest/
userguide/getting-started-from-console.html");

            // List available landing zones
            var landingZones = await wrapper!.ListLandingZonesAsync();
            if (landingZones.Count > 0)
            {
                Console.WriteLine("\nAvailable Landing Zones:");
                for (int i = 0; i < landingZones.Count; i++)
                {
                    Console.WriteLine($"{i + 1}. {landingZones[i].Arn}");
                }

                Console.Write($"\nDo you want to use the first landing zone in the
 list ({landingZones[0].Arn})? (y/n): ");
                if (GetUserConfirmation())
                {
                    useLandingZone = true;
                    Console.WriteLine($"Using landing zone: {landingZones[0].Arn}");
                    ouArn = await SetupOrganizationAsync();
                }
            }

            // Managing Baselines
            Console.WriteLine("\nManaging Baselines:");
            var baselines = await wrapper.ListBaselinesAsync();
            Console.WriteLine("\nListing available Baselines:");
            BaselineSummary? controlTowerBaseline = null;
            foreach (var baseline in baselines)
            {
                if (baseline.Name == "AWSControlTowerBaseline")
                    controlTowerBaseline = baseline;
                Console.WriteLine($"  - {baseline.Name}");
            }

            EnabledBaselineSummary? identityCenterBaseline = null;
            string? baselineArn = null;
```

```
            if (useLandingZone && ouArn != null)
            {
                Console.WriteLine("\nListing enabled baselines:");
                var enabledBaselines = await wrapper.ListEnabledBaselinesAsync();
                foreach (var baseline in enabledBaselines)
                {
                    if (baseline.BaselineIdentifier.Contains("baseline/
LN25R72TTG6IGPTQ"))
                        identityCenterBaseline = baseline;
                    Console.WriteLine($"  - {baseline.BaselineIdentifier}");
                }

                if (controlTowerBaseline != null)
                {
                    Console.Write("\nDo you want to enable the Control Tower
 Baseline? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine("\nEnabling Control Tower Baseline.");
                        var icBaselineArn = identityCenterBaseline?.Arn;
                        baselineArn = await wrapper.EnableBaselineAsync(ouArn,
                            controlTowerBaseline.Arn, "4.0", icBaselineArn ?? "");
                        var alreadyEnabled = false;
                        if (baselineArn != null)
                        {
                            Console.WriteLine($"Enabled baseline ARN:
 {baselineArn}");
                        }
                        else
                        {
                            // Find the enabled baseline
                            foreach (var enabled in enabledBaselines)
                            {
                                if (enabled.BaselineIdentifier ==
 controlTowerBaseline.Arn)
                                {
                                    baselineArn = enabled.Arn;
                                    break;
                                }
                            }

                            alreadyEnabled = true;
```

```
                            Console.WriteLine("No change, the selected baseline was
already enabled.");
                        }

                        if (baselineArn != null)
                        {
                            Console.Write("\nDo you want to reset the Control Tower
Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"\nResetting Control Tower
Baseline: {baselineArn}");
                                var operationId = await
wrapper.ResetEnabledBaselineAsync(baselineArn);
                                Console.WriteLine($"Reset baseline operation id:
{operationId}");
                            }

                            Console.Write("\nDo you want to disable the Control
Tower Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"Disabling baseline ARN:
{baselineArn}");
                                var operationId = await
wrapper.DisableBaselineAsync(baselineArn);
                                Console.WriteLine($"Disabled baseline operation id:
{operationId}");
                                if (alreadyEnabled)
                                {
                                    Console.WriteLine($"\nRe-enabling Control Tower
Baseline: {baselineArn}");
                                    // Re-enable the Control Tower baseline if it
was originally enabled.
                                    await wrapper.EnableBaselineAsync(ouArn,
                                        controlTowerBaseline.Arn, "4.0",
icBaselineArn ?? "");
                                }
                            }
                        }
                    }
                }
            }
```

```
            // Managing Controls
            Console.WriteLine("\nManaging Controls:");
            var controls = await wrapper.ListControlsAsync();
            Console.WriteLine("\nListing first 5 available Controls:");
            for (int i = 0; i < Math.Min(5, controls.Count); i++)
            {
                Console.WriteLine($"{i + 1}. {controls[i].Name} -
 {controls[i].Arn}");
            }

            if (useLandingZone && ouArn != null)
            {
                var enabledControls = await wrapper.ListEnabledControlsAsync(ouArn);
                Console.WriteLine("\nListing enabled controls:");
                for (int i = 0; i < enabledControls.Count; i++)
                {
                    Console.WriteLine($"{i + 1}.
 {enabledControls[i].ControlIdentifier}");
                }

                // Find first non-enabled control
                var enabledControlArns = enabledControls.Select(c =>
 c.Arn).ToHashSet();
                var controlArn = controls.FirstOrDefault(c => !
enabledControlArns.Contains(c.Arn))?.Arn;

                if (controlArn != null)
                {
                    Console.Write($"\nDo you want to enable the control
 {controlArn}? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine($"\nEnabling control: {controlArn}");
                        var operationId = await
 wrapper.EnableControlAsync(controlArn, ouArn);
                        if (operationId != null)
                        {
                            Console.WriteLine($"Enabled control with operation id:
 {operationId}");

                            Console.Write("\nDo you want to disable the control? (y/
n): ");
                            if (GetUserConfirmation())
                            {
```

```
                                Console.WriteLine("\nDisabling the control...");
                                var disableOpId = await
wrapper.DisableControlAsync(controlArn, ouArn);
                                Console.WriteLine($"Disable operation ID:
{disableOpId}");
                            }
                        }
                    }
                }
            }

            Console.WriteLine("\nThis concludes the example scenario.");
            Console.WriteLine("Thanks for watching!");
            Console.WriteLine(new string('-', 88));
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "An error occurred during the Control Tower
scenario.");
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }

    /// <summary>
    /// Sets up AWS Organizations and creates or finds a Sandbox OU.
    /// </summary>
    /// <returns>The ARN of the Sandbox organizational unit.</returns>
    private static async Task<string> SetupOrganizationAsync()
    {
        Console.WriteLine("\nChecking organization status...");

        try
        {
            var orgResponse = await orgClient!.DescribeOrganizationAsync(new
DescribeOrganizationRequest());
            var orgId = orgResponse.Organization.Id;
            Console.WriteLine($"Account is part of organization: {orgId}");
        }
        catch (AWSOrganizationsNotInUseException)
        {
            Console.WriteLine("No organization found. Creating a new
organization...");
            var createResponse = await orgClient!.CreateOrganizationAsync(new
CreateOrganizationRequest { FeatureSet = OrganizationFeatureSet.ALL });
```

```
            var orgId = createResponse.Organization.Id;
            Console.WriteLine($"Created new organization: {orgId}");
        }

        // Look for Sandbox OU
        var roots = await orgClient.ListRootsAsync(new ListRootsRequest());
        var rootId = roots.Roots[0].Id;

        Console.WriteLine("Checking for Sandbox OU...");
        var ous = await orgClient.ListOrganizationalUnitsForParentAsync(new
 ListOrganizationalUnitsForParentRequest { ParentId = rootId });
        var sandboxOu = ous.OrganizationalUnits.FirstOrDefault(ou => ou.Name ==
 "Sandbox");

        if (sandboxOu == null)
        {
            Console.WriteLine("Creating Sandbox OU...");
            var createOuResponse = await orgClient.CreateOrganizationalUnitAsync(new
 CreateOrganizationalUnitRequest { ParentId = rootId, Name = "Sandbox" });
            sandboxOu = createOuResponse.OrganizationalUnit;
            Console.WriteLine($"Created new Sandbox OU: {sandboxOu.Id}");
        }
        else
        {
            Console.WriteLine($"Found existing Sandbox OU: {sandboxOu.Id}");
        }

        return sandboxOu.Arn;
    }

    /// <summary>
    /// Gets user confirmation by waiting for input or returning true if not
 interactive.
    /// </summary>
    /// <returns>True if user enters 'y' or if isInteractive is false, otherwise
 false.</returns>
    private static bool GetUserConfirmation()
    {
        return Console.ReadLine()?.ToLower() == "y" || !isInteractive;
    }
}
```

Wrapper methods that are called by the scenario to manage Aurora actions.

```
using Amazon.ControlCatalog;
using Amazon.ControlCatalog.Model;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using ValidationException = Amazon.ControlTower.Model.ValidationException;

namespace ControlTowerActions;

/// <summary>
/// Methods to perform AWS Control Tower actions.
/// </summary>
public class ControlTowerWrapper
{
    private readonly IAmazonControlTower _controlTowerService;
    private readonly IAmazonControlCatalog _controlCatalogService;

    /// <summary>
    /// Constructor for the wrapper class containing AWS Control Tower actions.
    /// </summary>
    /// <param name="controlTowerService">The AWS Control Tower client object.</
param>
    /// <param name="controlCatalogService">The AWS Control Catalog client object.</
param>
    public ControlTowerWrapper(IAmazonControlTower controlTowerService,
 IAmazonControlCatalog controlCatalogService)
    {
        _controlTowerService = controlTowerService;
        _controlCatalogService = controlCatalogService;
    }

    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
        try
        {
            var landingZones = new List<LandingZoneSummary>();
```

```
            var landingZonesPaginator =
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            return baselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
```

```
        }
    }


    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
 _controlTowerService.Paginators.ListEnabledBaselines(new
 ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
 baseline if it is enabled.</param>
    /// <returns>The enabled baseline ARN or null if already enabled.</returns>
```

```
   public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
baselineIdentifier, string baselineVersion, string identityCenterBaseline)
   {
       try
       {
           var parameters = new List<EnabledBaselineParameter>
           {
               new EnabledBaselineParameter
               {
                   Key = "IdentityCenterEnabledBaselineArn",
                   Value = identityCenterBaseline
               }
           };

           var request = new EnableBaselineRequest
           {
               BaselineIdentifier = baselineIdentifier,
               BaselineVersion = baselineVersion,
               TargetIdentifier = targetIdentifier,
               Parameters = parameters
           };

           var response = await _controlTowerService.EnableBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return response.Arn;
       }
       catch (ValidationException ex) when (ex.Message.Contains("already enabled"))
       {
           Console.WriteLine("Baseline is already enabled for this target");
           return null;
```

```
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a baseline for a specific target and wait for the operation to
complete.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
    /// <returns>The operation ID or null if there was a conflict.</returns>
    public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new DisableBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await _controlTowerService.DisableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
```

```
        }
        catch (ConflictException ex)
        {
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Reset an enabled baseline for a specific target.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new ResetEnabledBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
```

```
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Target not found, unable to reset enabled
baseline.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Get the status of a baseline operation.
    /// </summary>
    /// <param name="operationId">The ID of the baseline operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetBaselineOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetBaselineOperationAsync(request);
            return response.BaselineOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
```

```
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get baseline operation status. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };

            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
 _controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
                enabledControls.AddRange(response.EnabledControls);
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
 (ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
 enabling controls.");
```

```
            return new List<EnabledControlSummary>();
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to enable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID or null if already enabled.</returns>
    public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
    {
        try
        {
            Console.WriteLine(controlArn);
            Console.WriteLine(targetIdentifier);

            var request = new EnableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.EnableControlAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
```

```
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to disable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
    {
        try
        {
            var request = new DisableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
```

```
            };

            var response = await _controlTowerService.DisableControlAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Control not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Get the status of a control operation.
    /// </summary>
    /// <param name="operationId">The ID of the control operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
    {
        try
        {
```

```
            var request = new GetControlOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetControlOperationAsync(request);
            return response.ControlOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all controls in the Control Tower control catalog.
    /// </summary>
    /// <returns>A list of control summaries.</returns>
    public async Task<List<ControlSummary>> ListControlsAsync()
    {
        try
        {
            var controls = new List<ControlSummary>();

            var controlsPaginator =
_controlCatalogService.Paginators.ListControls(new
Amazon.ControlCatalog.Model.ListControlsRequest());

            await foreach (var response in controlsPaginator.Responses)
            {
                controls.AddRange(response.Controls);
            }

            return controls;
        }
```

```
        catch (AmazonControlCatalogException ex)
        {
            Console.WriteLine($"Couldn't list controls. Here's why: {ex.ErrorCode}:
 {ex.Message}");
            throw;
        }
    }

}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - CreateLandingZone
  - DeleteLandingZone
  - DisableBaseline
  - DisableControl
  - EnableBaseline
  - EnableControl
  - GetControlOperation
  - GetLandingZoneOperation
  - ListBaselines
  - ListEnabledBaselines
  - ListEnabledControls
  - ListLandingZones
  - ResetEnabledBaseline

Actions

`DisableBaseline`

The following code example shows how to use `DisableBaseline`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Disable a baseline for a specific target and wait for the operation to
complete.
   /// </summary>
   /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
   /// <returns>The operation ID or null if there was a conflict.</returns>
   public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
   {
       try
       {
           var request = new DisableBaselineRequest
           {
               EnabledBaselineIdentifier = enabledBaselineIdentifier
           };

           var response = await _controlTowerService.DisableBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
```

```
        }
        catch (ConflictException ex)
        {
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- For API details, see DisableBaseline in *AWS SDK for .NET API Reference*.

`DisableControl`

The following code example shows how to use `DisableControl`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Disable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to disable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
    {
```

```
        try
        {
            var request = new DisableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.DisableControlAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Control not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- For API details, see DisableControl in *AWS SDK for .NET API Reference*.

`EnableBaseline`

The following code example shows how to use `EnableBaseline`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
 baseline if it is enabled.</param>
    /// <returns>The enabled baseline ARN or null if already enabled.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
 baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>
            {
                new EnabledBaselineParameter
                {
                    Key = "IdentityCenterEnabledBaselineArn",
                    Value = identityCenterBaseline
                }
            };

            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
            };
```

```
            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return response.Arn;
        }
        catch (ValidationException ex) when (ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Baseline is already enabled for this target");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- For API details, see EnableBaseline in *AWS SDK for .NET API Reference*.

`EnableControl`

The following code example shows how to use `EnableControl`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Enable a control for a specified target.
   /// </summary>
   /// <param name="controlArn">The ARN of the control to enable.</param>
   /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
   /// <returns>The operation ID or null if already enabled.</returns>
   public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           Console.WriteLine(controlArn);
           Console.WriteLine(targetIdentifier);

           var request = new EnableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.EnableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetControlOperationAsync(operationId);
               Console.WriteLine($"Control operation status: {status}");
               if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
               {
                   break;
               }
```

```
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

- For API details, see EnableControl in *AWS SDK for .NET API Reference*.

`GetBaselineOperation`

The following code example shows how to use `GetBaselineOperation`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Get the status of a baseline operation.
   /// </summary>
   /// <param name="operationId">The ID of the baseline operation.</param>
   /// <returns>The operation status.</returns>
   public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
   {
       try
       {
           var request = new GetBaselineOperationRequest
           {
               OperationIdentifier = operationId
           };

           var response = await
_controlTowerService.GetBaselineOperationAsync(request);
           return response.BaselineOperation.Status;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Operation not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't get baseline operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
   }
```

- For API details, see GetBaselineOperation in *AWS SDK for .NET API Reference*.

`GetControlOperation`

The following code example shows how to use `GetControlOperation`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Get the status of a control operation.
   /// </summary>
   /// <param name="operationId">The ID of the control operation.</param>
   /// <returns>The operation status.</returns>
   public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
   {
       try
       {
           var request = new GetControlOperationRequest
           {
               OperationIdentifier = operationId
           };

           var response = await
_controlTowerService.GetControlOperationAsync(request);
           return response.ControlOperation.Status;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Operation not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
   }
```

- For API details, see GetControlOperation in *AWS SDK for .NET API Reference*.

`ListBaselines`

The following code example shows how to use `ListBaselines`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            return baselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

- For API details, see ListBaselines in *AWS SDK for .NET API Reference*.

`ListEnabledBaselines`

The following code example shows how to use `ListEnabledBaselines`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
_controlTowerService.Paginators.ListEnabledBaselines(new
ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
```

```
        }
    }
```

- For API details, see ListEnabledBaselines in *AWS SDK for .NET API Reference*.

`ListEnabledControls`

The following code example shows how to use `ListEnabledControls`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };

            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
 _controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
```

```
                enabledControls.AddRange(response.EnabledControls);
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return new List<EnabledControlSummary>();
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- For API details, see ListEnabledControls in *AWS SDK for .NET API Reference*.

`ListLandingZones`

The following code example shows how to use `ListLandingZones`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
```

```
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- For API details, see ListLandingZones in *AWS SDK for .NET API Reference*.

`ResetEnabledBaseline`

The following code example shows how to use `ResetEnabledBaseline`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Reset an enabled baseline for a specific target.
    /// </summary>
```

```
   /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
   /// <returns>The operation ID.</returns>
   public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
   {
       try
       {
           var request = new ResetEnabledBaselineRequest
           {
               EnabledBaselineIdentifier = enabledBaselineIdentifier
           };

           var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Target not found, unable to reset enabled
baseline.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
```

```
    }
```

- For API details, see ResetEnabledBaseline in *AWS SDK for .NET API Reference*.

DynamoDB examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with DynamoDB.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello DynamoDB

The following code examples show how to get started using DynamoDB.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
```

```
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables = dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
```

```
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- For API details, see ListTables in *AWS SDK for .NET API Reference*.

Topics

- Basics
- Actions

Basics

Learn the basics

The following code example shows how to:

- Create a table that can hold movie data.
- Put, get, and update a single movie in the table.
- Write movie data to the table from a sample JSON file.
- Query for movies that were released in a given year.
- Scan for movies that were released in a range of years.
- Delete a movie from the table, then delete the table.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
{
    public static bool IsInteractive = true;

    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
```

```
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper = host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");

        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };

        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
```

```
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous" +
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie: {newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();

        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
```

```
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName, movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }

        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName, startYear,
endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
```

```
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is complete.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        if (!IsInteractive)
        {
            return;
        }

        Console.Clear();
        Console.WriteLine();
        Console.Write(new string(' ', 28));
        Console.WriteLine("DynamoDB Basics Example");
        Console.WriteLine(SepBar);
        Console.WriteLine("This demo application shows the basics of using DynamoDB
with the AWS SDK.");
        Console.WriteLine(SepBar);
        Console.WriteLine("The application does the following:");
        Console.WriteLine("\t1. Creates a table with partition: year and
sort:title.");
        Console.WriteLine("\t2. Adds a single movie to the table.");
        Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
        Console.WriteLine("\t4. Updates the rating and plot of the movie that was
just added.");
        Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
        Console.WriteLine("\t6. Deletes a movie.");
```

```
        Console.WriteLine("\t7. Uses QueryAsync to return all movies released in a
 given year.");
        Console.WriteLine("\t8. Uses ScanAsync to return all movies released within
 a range of years.");
        Console.WriteLine("\t9. Finally, it deletes the table that was just
 created.");
        WaitForEnter();
    }

    /// <summary>
    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        if (IsInteractive)
        {
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
}
```

Creates a table to contain movie data.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
```

```
            {
                new AttributeDefinition
                {
                    AttributeName = "title",
                    AttributeType = ScalarAttributeType.S,
                },
                new AttributeDefinition
                {
                    AttributeName = "year",
                    AttributeType = ScalarAttributeType.N,
                },
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement
                {
                    AttributeName = "year",
                    KeyType = KeyType.HASH,
                },
                new KeySchemaElement
                {
                    AttributeName = "title",
                    KeyType = KeyType.RANGE,
                },
            },
            BillingMode = BillingMode.PAY_PER_REQUEST,
        });

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

        var request = new DescribeTableRequest
        {
            TableName = response.TableDescription.TableName,
        };

        TableStatus status;

        int sleepDuration = 2000;

        do
        {
            Thread.Sleep(sleepDuration);
```

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

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

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table {tableName}.
 {ex.Message}");
            throw;
        }
    }
```

Adds a single movie to the table.

```

    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
```

```
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

Updates a single item in a table.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };
```

```
            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

Retrieves a single item from the movie table.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
```

```
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

Writes a batch of items to the movie table.

```
    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the DynamoDB table.
    /// </summary>
    /// <param name="movieFileName">The name of the JSON file.</param>
```

```
    /// <returns>A generic list of movie objects.</returns>
    public List<Movie> ImportMovies(string movieFileName)
    {
        var moviesList = new List<Movie>();
        if (!File.Exists(movieFileName))
        {
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
```

```
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

Deletes a single item from the table.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
```

```
            throw;
        }
    }
```

Queries the table for movies released in a particular year.

```
    /// <summary>
    /// Queries the table for movies released in a particular year and
    /// then displays the information for the movies returned.
    /// </summary>
    /// <param name="tableName">The name of the table to query.</param>
    /// <param name="year">The release year for which we want to
    /// view movies.</param>
    /// <returns>The number of movies that match the query.</returns>
    public async Task<int> QueryMoviesAsync(string tableName, int year)
    {
        try
        {
            var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
                .AddHashKey("year", DynamoDBEntryType.Numeric)
                .AddRangeKey("title", DynamoDBEntryType.String)
                .Build();

            var filter = new QueryFilter("year", QueryOperator.Equal, year);

            Console.WriteLine("\nFind movies released in: {year}:");

            var config = new QueryOperationConfig()
            {
                Limit = 10, // 10 items per page.
                Select = SelectValues.SpecificAttributes,
                AttributesToGet = new List<string>
                {
                    "title",
                    "year",
                },
                ConsistentRead = true,
                Filter = filter,
            };
```

```
            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

Scans the table for movies released in a range of years.

```
    /// <summary>
```

```
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
```

```
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

Deletes the movie table.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;
```

```
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
            return false;
        }
    }
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Actions

`BatchWriteItem`

The following code example shows how to use `BatchWriteItem`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Writes a batch of items to the movie table.

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    if (allMovies != null && allMovies.Any())
    {
```

```
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

- For API details, see BatchWriteItem in *AWS SDK for .NET API Reference*.

`CreateTable`

The following code example shows how to use `CreateTable`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
```

```
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

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

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };
```

```
            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

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

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

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table {tableName}.
{ex.Message}");
            throw;
        }
    }
```

- For API details, see CreateTable in *AWS SDK for .NET API Reference*.

`DeleteItem`

The following code example shows how to use `DeleteItem`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
```

```
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

- For API details, see DeleteItem in *AWS SDK for .NET API Reference*.

`DeleteTable`

The following code example shows how to use `DeleteTable`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
```

```
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
            return false;
        }
    }
```

- For API details, see DeleteTable in *AWS SDK for .NET API Reference*.

`GetItem`

The following code example shows how to use `GetItem`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Gets information about an existing movie from the table.
   /// </summary>
   /// <param name="newMovie">A Movie object containing information about
   /// the movie to retrieve.</param>
   /// <param name="tableName">The name of the table containing the movie.</param>
   /// <returns>A Dictionary object containing information about the item
   /// retrieved.</returns>
   public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
   {
       try
       {
           var key = new Dictionary<string, AttributeValue>
           {
               ["title"] = new AttributeValue { S = newMovie.Title },
               ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
           };

           var request = new GetItemRequest
           {
               Key = key,
               TableName = tableName,
           };

           var response = await _amazonDynamoDB.GetItemAsync(request);
           return response.Item;
       }
       catch (ResourceNotFoundException ex)
       {
           Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
           return new Dictionary<string, AttributeValue>();
       }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
 {ex.Message}");
            throw;
        }
    }
```

- For API details, see GetItem in *AWS SDK for .NET API Reference*.

`PutItem`

The following code example shows how to use `PutItem`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
```

```
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- For API details, see PutItem in *AWS SDK for .NET API Reference*.

`Query`

The following code example shows how to use `Query`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
            Filter = filter,
        };
```

```
            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- For API details, see Query in *AWS SDK for .NET API Reference*.

Scan

The following code example shows how to use Scan.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
            };
```

```
            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

- For API details, see Scan in *AWS SDK for .NET API Reference*.

`UpdateItem`

The following code example shows how to use `UpdateItem`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
```

```
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- For API details, see UpdateItem in *AWS SDK for .NET API Reference*.

Amazon EC2 examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with Amazon EC2.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello Amazon EC2

The following code examples show how to get started using Amazon EC2.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace EC2Actions;

public class HelloEc2
{
    /// <summary>
    /// HelloEc2 lists the existing security groups for the default users.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Elastic Compute Cloud (Amazon
 EC2).
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonEC2>()
                .AddTransient<EC2Wrapper>()
            )
```

```
            .Build();

        // Now the client is available for injection.
        var ec2Client = host.Services.GetRequiredService<IAmazonEC2>();

        try
        {
            // Retrieve information for up to 10 Amazon EC2 security groups.
            var request = new DescribeSecurityGroupsRequest { MaxResults = 10 };
            var securityGroups = new List<SecurityGroup>();

            var paginatorForSecurityGroups =
                ec2Client.Paginators.DescribeSecurityGroups(request);

            await foreach (var securityGroup in
 paginatorForSecurityGroups.SecurityGroups)
            {
                securityGroups.Add(securityGroup);
            }

            // Now print the security groups returned by the call to
            // DescribeSecurityGroupsAsync.
            Console.WriteLine("Welcome to the EC2 Hello Service example. " +
                              "\nLet's list your Security Groups:");
            securityGroups.ForEach(group =>
            {
                Console.WriteLine(
                    $"Security group: {group.GroupName} ID: {group.GroupId}");
            });
        }
        catch (AmazonEC2Exception ex)
        {
            Console.WriteLine($"An Amazon EC2 service error occurred while listing
 security groups. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing security groups.
 {ex.Message}");
        }
    }
}
```

- For API details, see DescribeSecurityGroups in *AWS SDK for .NET API Reference*.

Amazon ECS examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with Amazon ECS.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello Amazon ECS

The following code example shows how to get started using Amazon ECS.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
using Amazon.ECS;
using Amazon.ECS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace ECSActions;

/// <summary>
/// A class that introduces the Amazon ECS Client by listing the
/// cluster ARNs for the account.
/// </summary>
public class HelloECS
{
    static async System.Threading.Tasks.Task Main(string[] args)
```

```
   {
       // Use the AWS .NET Core Setup package to set up dependency injection for
the Amazon ECS client.
       // Use your AWS profile name, or leave it blank to use the default profile.
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonECS>()
           )
           .Build();

       var amazonECSClient = host.Services.GetRequiredService<IAmazonECS>();

       Console.WriteLine($"Hello Amazon ECS! Following are some cluster ARNS
available in the your account");
       Console.WriteLine();

       var clusters = new List<string>();

       var clustersPaginator = amazonECSClient.Paginators.ListClusters(new
ListClustersRequest());

       await foreach (var response in clustersPaginator.Responses)
       {
           clusters.AddRange(response.ClusterArns);
       }

       if (clusters.Count > 0)
       {
           clusters.ForEach(cluster =>
           {
               Console.WriteLine($"\tARN: {cluster}");
               Console.WriteLine($"Cluster Name: {cluster.Split("/").Last()}");
               Console.WriteLine();
           });
       }
       else
       {
           Console.WriteLine("No clusters were found.");
       }
```

```
    }
}
```

- For API details, see ListClusters in *AWS SDK for .NET API Reference*.

AWS IoT examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with AWS IoT.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello AWS IoT

The following code examples show how to get started using AWS IoT.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Hello AWS IoT example.
/// </summary>
public class HelloIoT
{
    /// <summary>
    /// Main method to run the Hello IoT example.
```

```
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var iotClient = new AmazonIoTClient();

        try
        {
            Console.WriteLine("Hello AWS IoT! Let's list your IoT Things:");
            Console.WriteLine(new string('-', 80));

            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await iotClient.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await iotClient.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            if (things is { Count: > 0 })
            {
                Console.WriteLine($"Found {things.Count} IoT Things:");
                foreach (var thing in things)
                {
```

```
                    Console.WriteLine($"- Thing Name: {thing.ThingName}");
                    Console.WriteLine($"  Thing ARN: {thing.ThingArn}");
                    Console.WriteLine($"  Thing Type: {thing.ThingTypeName ?? "No
 type specified"}");
                    Console.WriteLine($"  Version: {thing.Version}");

                    if (thing.Attributes?.Count > 0)
                    {
                        Console.WriteLine("  Attributes:");
                        foreach (var attr in thing.Attributes)
                        {
                            Console.WriteLine($"    {attr.Key}: {attr.Value}");
                        }
                    }
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No IoT Things found in your account.");
                Console.WriteLine("You can create IoT Things using the IoT Basics
 scenario example.");
            }

            Console.WriteLine("Hello IoT completed successfully.");
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            Console.WriteLine($"Request throttled, please try again later:
 {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list Things. Here's why: {ex.Message}");
        }
    }
}
```

- For API details, see listThings in *AWS SDK for .NET API Reference*.

Topics

- Basics
- Actions

Basics

Learn the basics

The following code example shows how to:

- Create an AWS IoT Thing.
- Generate a device certificate.
- Update an AWS IoT Thing with Attributes.
- Return a unique endpoint.
- List your AWS IoT certificates.
- Update an AWS IoT shadow.
- Write out state information.
- Creates a rule.
- List your rules.
- Search things using the Thing name.
- Delete an AWS IoT Thing.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario demonstrating AWS IoT features.

```
/// <summary>
/// Scenario class for AWS IoT basics.
/// </summary>
public class IoTBasics
{
```

```
    public static bool IsInteractive = true;
    public static IoTWrapper? Wrapper = null;
    public static IAmazonCloudFormation? CloudFormationClient = null;
    public static ILogger<IoTBasics> logger = null!;
    private static IoTWrapper _iotWrapper = null!;
    private static IAmazonCloudFormation _amazonCloudFormation = null!;
    private static ILogger<IoTBasics> _logger = null!;

    private static string _stackName = "IoTBasicsStack";
    private static string _stackResourcePath = "../../../../../../scenarios/basics/
iot/iot_usecase/resources/cfn_template.yaml";

    /// <summary>
    /// Main method for the IoT Basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIoT>(new AWSOptions() { Region =
 RegionEndpoint.USEast1 })
                    .AddAWSService<IAmazonCloudFormation>()
                        .AddTransient<IoTWrapper>()
                        .AddLogging(builder => builder.AddConsole())
                        .AddSingleton<IAmazonIotData>(sp =>
                        {
                            var iotService = sp.GetRequiredService<IAmazonIoT>();
                            var request = new DescribeEndpointRequest
                            {
                                EndpointType = "iot:Data-ATS"
                            };
                            var response =
 iotService.DescribeEndpointAsync(request).Result;
                            return new AmazonIotDataClient($"https://
{response.EndpointAddress}/");
                        })
            )
            .Build();

        logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<IoTBasics>();
```

```
        Wrapper = host.Services.GetRequiredService<IoTWrapper>();
        CloudFormationClient =
host.Services.GetRequiredService<IAmazonCloudFormation>();

        // Set the private fields for backwards compatibility
        _logger = logger;
        _iotWrapper = Wrapper;
        _amazonCloudFormation = CloudFormationClient;

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS IoT example scenario.");
        Console.WriteLine("This example program demonstrates various interactions
with the AWS Internet of Things (IoT) Core service.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        Console.WriteLine(new string('-', 80));

        try
        {
            await RunScenarioAsync();
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem running the scenario.");
            Console.WriteLine($"\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The AWS IoT scenario has successfully completed.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Run the IoT Basics scenario.
    /// </summary>
    /// <returns>A Task object.</returns>
    public static async Task RunScenarioAsync()
    {
        // Use static properties if available, otherwise use private fields
```

```
        var iotWrapper = Wrapper ?? _iotWrapper;
        var cloudFormationClient = CloudFormationClient ?? _amazonCloudFormation;
        var scenarioLogger = logger ?? _logger;

        await RunScenarioInternalAsync(iotWrapper, cloudFormationClient,
 scenarioLogger);
    }

    /// <summary>
    /// Internal method to run the IoT Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="iotWrapper">The IoT wrapper instance.</param>
    /// <param name="cloudFormationClient">The CloudFormation client instance.</
param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    private static async Task RunScenarioInternalAsync(IoTWrapper iotWrapper,
 IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        string thingName = $"iot-thing-{Guid.NewGuid():N}";
        string certificateArn = "";
        string certificateId = "";
        string ruleName = $"iotruledefault";
        string snsTopicArn = "";

        try
        {
            // Step 1: Create an AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("1. Create an AWS IoT Thing.");
            Console.WriteLine("An AWS IoT Thing represents a virtual entity in the
 AWS IoT service that can be associated with a physical device.");
            Console.WriteLine();

            if (IsInteractive)
            {
                Console.Write("Enter Thing name: ");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    thingName = userInput;
            }
            else
            {
                Console.WriteLine($"Using default Thing name: {thingName}");
```

```
            }

            var thingArn = await iotWrapper.CreateThingAsync(thingName);
            Console.WriteLine($"{thingName} was successfully created. The ARN value
is {thingArn}");
            Console.WriteLine(new string('-', 80));

            // Step 1.1: List AWS IoT Things
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("2. List AWS IoT Things.");
            Console.WriteLine("Now let's list the IoT Things to see the Thing we
just created.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var things = await iotWrapper.ListThingsAsync();
            Console.WriteLine($"Found {things.Count} IoT Things:");
            foreach (var thing in things.Take(10)) // Show first 10 things
            {
                Console.WriteLine($"Thing Name: {thing.ThingName}");
                Console.WriteLine($"Thing ARN: {thing.ThingArn}");
                if (thing.Attributes != null && thing.Attributes.Any())
                {
                    Console.WriteLine("Attributes:");
                    foreach (var attr in thing.Attributes)
                    {
                        Console.WriteLine($"  {attr.Key}: {attr.Value}");
                    }
                }
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 2: Generate a Device Certificate
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("3. Generate a device certificate.");
            Console.WriteLine("A device certificate performs a role in securing the
communication between devices (Things) and the AWS IoT platform.");
            Console.WriteLine();
```

```
            var createCert = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to create a certificate for {thingName}?
(y/n)");
                createCert = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Creating certificate for {thingName}...");
            }

            if (createCert?.ToLower() == "y")
            {
                var certificateResult = await
iotWrapper.CreateKeysAndCertificateAsync();
                if (certificateResult.HasValue)
                {
                    var (certArn, certPem, certId) = certificateResult.Value;
                    certificateArn = certArn;
                    certificateId = certId;

                    Console.WriteLine($"\nCertificate:");
                    // Show only first few lines of certificate for brevity
                    var lines = certPem.Split('\n');
                    for (int i = 0; i < Math.Min(lines.Length, 5); i++)
                    {
                        Console.WriteLine(lines[i]);
                    }
                    if (lines.Length > 5)
                    {
                        Console.WriteLine("...");
                    }

                    Console.WriteLine($"\nCertificate ARN:");
                    Console.WriteLine(certificateArn);

                    // Step 3: Attach the Certificate to the AWS IoT Thing
                    Console.WriteLine("Attach the certificate to the AWS IoT
Thing.");
                    var attachResult = await
iotWrapper.AttachThingPrincipalAsync(thingName, certificateArn);
                    if (attachResult)
```

```
                {
                    Console.WriteLine("Certificate attached to Thing
successfully.");
                }
                else
                {
                    Console.WriteLine("Failed to attach certificate to Thing.");
                }

                Console.WriteLine("Thing Details:");
                Console.WriteLine($"Thing Name: {thingName}");
                Console.WriteLine($"Thing ARN: {thingArn}");
            }
            else
            {
                Console.WriteLine("Failed to create certificate.");
            }
        }
        Console.WriteLine(new string('-', 80));

        // Step 4: Update an AWS IoT Thing with Attributes
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. Update an AWS IoT Thing with Attributes.");
        Console.WriteLine("IoT Thing attributes, represented as key-value pairs,
offer a pivotal advantage in facilitating efficient data");
        Console.WriteLine("management and retrieval within the AWS IoT
ecosystem.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }

        var attributes = new Dictionary<string, string>
        {
            { "Location", "Seattle" },
            { "DeviceType", "Sensor" },
            { "Firmware", "1.2.3" }
        };

        await iotWrapper.UpdateThingAsync(thingName, attributes);
        Console.WriteLine("Thing attributes updated successfully.");
        Console.WriteLine(new string('-', 80));
```

```
            // Step 5: Return a unique endpoint specific to the Amazon Web Services
account
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("5. Return a unique endpoint specific to the Amazon
Web Services account.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var endpoint = await iotWrapper.DescribeEndpointAsync();
            if (endpoint != null)
            {
                var subdomain = endpoint.Split('.')[0];
                Console.WriteLine($"Extracted subdomain: {subdomain}");
                Console.WriteLine($"Full Endpoint URL: https://{endpoint}");
            }
            else
            {
                Console.WriteLine("Failed to retrieve endpoint.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 6: List your AWS IoT certificates
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("6. List your AWS IoT certificates");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var certificates = await iotWrapper.ListCertificatesAsync();
            foreach (var cert in certificates.Take(5)) // Show first 5 certificates
            {
                Console.WriteLine($"Cert id: {cert.CertificateId}");
                Console.WriteLine($"Cert Arn: {cert.CertificateArn}");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));
```

```
            // Step 7: Create an IoT shadow
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("7. Update an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowPayload = JsonSerializer.Serialize(new
            {
                state = new
                {
                    desired = new
                    {
                        temperature = 25,
                        humidity = 50
                    }
                }
            });

            await iotWrapper.UpdateThingShadowAsync(thingName, shadowPayload);
            Console.WriteLine("Thing Shadow updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 8: Write out the state information, in JSON format
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("8. Write out the state information, in JSON
format.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowData = await iotWrapper.GetThingShadowAsync(thingName);
            Console.WriteLine($"Received Shadow Data: {shadowData}");
            Console.WriteLine(new string('-', 80));

            // Step 9: Set up resources (SNS topic and IAM role) and create a rule
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("9. Set up resources and create a rule");
```

```
            Console.WriteLine();

            // Deploy CloudFormation stack to create SNS topic and IAM role
            Console.WriteLine("Deploying CloudFormation stack to create SNS topic
and IAM role...");

            var deployStack = !IsInteractive || GetYesNoResponse("Would you like to
deploy the CloudFormation stack? (y/n) ");
            if (deployStack)
            {
                if (IsInteractive)
                {
                    Console.Write(
                        $"Enter stack resource file path (or press Enter for default
'{_stackResourcePath}'): ");
                    var userResourcePath = Console.ReadLine();
                    if (!string.IsNullOrEmpty(userResourcePath))
                        _stackResourcePath = userResourcePath;
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

                if (deploySuccess)
                {
                    // Get stack outputs
                    var stackOutputs = await GetStackOutputs(_stackName,
cloudFormationClient, scenarioLogger);
                    if (stackOutputs != null)
                    {
                        snsTopicArn = stackOutputs["SNSTopicArn"];
                        string roleArn = stackOutputs["RoleArn"];

                        Console.WriteLine($"Successfully deployed stack. SNS topic:
{snsTopicArn}");
                        Console.WriteLine($"Successfully deployed stack. IAM role:
{roleArn}");

                        if (IsInteractive)
                        {
                            Console.Write($"Enter Rule name (press Enter for default
'{ruleName}'): ");
```

```
                            var userRuleName = Console.ReadLine();
                            if (!string.IsNullOrEmpty(userRuleName))
                                ruleName = userRuleName;
                        }
                        else
                        {
                            Console.WriteLine($"Using default rule name:
{ruleName}");
                        }

                        // Now create the IoT rule with the CloudFormation outputs
                        var ruleResult = await
iotWrapper.CreateTopicRuleAsync(ruleName, snsTopicArn, roleArn);
                        if (ruleResult)
                        {
                            Console.WriteLine("IoT Rule created successfully.");
                        }
                        else
                        {
                            Console.WriteLine("Failed to create IoT rule.");
                        }
                    }
                    else
                    {
                        Console.WriteLine("Failed to get stack outputs. Skipping
rule creation.");
                    }
                }
                else
                {
                    Console.WriteLine("Failed to deploy CloudFormation stack.
Skipping rule creation.");
                }
            }
            else
            {
                Console.WriteLine("Skipping CloudFormation stack deployment and rule
creation.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 10: List your rules
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("10. List your rules.");
```

```
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var rules = await iotWrapper.ListTopicRulesAsync();
            Console.WriteLine("List of IoT Rules:");
            foreach (var rule in rules.Take(5)) // Show first 5 rules
            {
                Console.WriteLine($"Rule Name: {rule.RuleName}");
                Console.WriteLine($"Rule ARN: {rule.RuleArn}");
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 11: Search things using the Thing name
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("11. Search things using the Thing name.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var searchResults = await iotWrapper.SearchIndexAsync($"thingName:
{thingName}");
            if (searchResults.Any())
            {
                Console.WriteLine($"Thing id found using search is
 {searchResults.First().ThingId}");
            }
            else
            {
                Console.WriteLine($"No search results found for Thing:
 {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 12: Cleanup - Detach and delete certificate
            if (!string.IsNullOrEmpty(certificateArn))
            {
                Console.WriteLine(new string('-', 80));
```

```
                var deleteCert = "y";
                if (IsInteractive)
                {
                    Console.Write($"Do you want to detach and delete the certificate
for {thingName}? (y/n)");
                    deleteCert = Console.ReadLine();
                }
                else
                {
                    Console.WriteLine($"Detaching and deleting certificate for
{thingName}...");
                }

                if (deleteCert?.ToLower() == "y")
                {
                    Console.WriteLine("12. You selected to detach and delete the
certificate.");
                    if (IsInteractive)
                    {
                        Console.WriteLine("Press Enter to continue...");
                        Console.ReadLine();
                    }

                    await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
                    Console.WriteLine($"{certificateArn} was successfully removed
from {thingName}");

                    await iotWrapper.DeleteCertificateAsync(certificateId);
                    Console.WriteLine($"{certificateArn} was successfully
deleted.");
                }
                Console.WriteLine(new string('-', 80));
            }

            // Step 13: Delete the AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("13. Delete the AWS IoT Thing.");
            var deleteThing = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to delete the IoT Thing? (y/n)");
                deleteThing = Console.ReadLine();
            }
```

```
            else
            {
                Console.WriteLine($"Deleting IoT Thing {thingName}...");
            }

            if (deleteThing?.ToLower() == "y")
            {
                await iotWrapper.DeleteThingAsync(thingName);
                Console.WriteLine($"Deleted Thing {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 14: Clean up CloudFormation stack
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                Console.WriteLine(new string('-', 80));
                Console.WriteLine("14. Clean up CloudFormation stack.");
                Console.WriteLine("Deleting the CloudFormation stack and all
resources...");

                var cleanup = !IsInteractive || GetYesNoResponse("Do you want to
delete the CloudFormation stack and all resources? (y/n) ");
                if (cleanup)
                {
                    var ruleCleanupSuccess = await
iotWrapper.DeleteTopicRuleAsync(ruleName);

                    var stackCleanupSuccess = await
DeleteCloudFormationStack(_stackName, cloudFormationClient, scenarioLogger);
                    if (ruleCleanupSuccess && stackCleanupSuccess)
                    {
                        Console.WriteLine("Successfully cleaned up CloudFormation
stack and all resources.");
                    }
                    else
                    {
                        Console.WriteLine("Some cleanup operations failed. Check the
logs for details.");
                    }
                }
                else
                {
                    Console.WriteLine($"Resources will remain. Stack name:
{_stackName}");
```

```
                }
                Console.WriteLine(new string('-', 80));
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "Error occurred during scenario
 execution.");

            // Cleanup on error
            if (!string.IsNullOrEmpty(certificateArn) && !
string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DetachThingPrincipalAsync(thingName,
 certificateArn);
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
                }
            }

            if (!string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DeleteThingAsync(thingName);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during Thing
 cleanup.");
                }
            }

            // Clean up CloudFormation stack on error
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                try
                {
                    await _iotWrapper.DeleteTopicRuleAsync(ruleName);
```

```
                    await DeleteCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during CloudFormation
stack cleanup.");
                }
            }

            throw;
        }
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
                Capabilities = new List<string> { Capability.CAPABILITY_NAMED_IAM }
            };

            var response = await cloudFormationClient.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId,
cloudFormationClient, scenarioLogger);
```

```
                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    scenarioLogger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                scenarioLogger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            scenarioLogger.LogWarning($"CloudFormation stack '{stackName}' already
exists. Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName,
cloudFormationClient, scenarioLogger);
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was created successfully.</returns>
```

```
   private static async Task<bool> WaitForStackCompletion(string stackId,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
   {
       int retryCount = 0;
       const int maxRetries = 30;
       const int retryDelay = 10000;

       while (retryCount < maxRetries)
       {
           var describeStacksRequest = new DescribeStacksRequest
           {
               StackName = stackId
           };

           var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

           if (describeStacksResponse.Stacks.Count > 0)
           {
               if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
               {
                   return true;
               }
               if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                   describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
               {
                   return false;
               }
           }

           Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
           await Task.Delay(retryDelay);
           retryCount++;
       }

       scenarioLogger.LogError("Timed out waiting for CloudFormation stack creation
to complete.");
       return false;
   }
```

```
    /// <summary>
    /// Gets the outputs from the CloudFormation stack.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>A dictionary of stack outputs.</returns>
    private static async Task<Dictionary<string, string>?> GetStackOutputs(string
stackName, IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics>
scenarioLogger)
    {
        try
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            var response = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (response.Stacks.Count > 0)
            {
                var outputs = new Dictionary<string, string>();
                foreach (var output in response.Stacks[0].Outputs)
                {
                    outputs[output.OutputKey] = output.OutputValue;
                }
                return outputs;
            }

            return null;
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"Failed to get stack outputs for
{stackName}");
            return null;
        }
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
```

```
   /// <param name="stackName">The name of the CloudFormation stack.</param>
   /// <param name="cloudFormationClient">The CloudFormation client.</param>
   /// <param name="scenarioLogger">The logger.</param>
   /// <returns>True if the stack was deleted successfully.</returns>
   private static async Task<bool> DeleteCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
   {
       try
       {
           var request = new DeleteStackRequest
           {
               StackName = stackName
           };

           await cloudFormationClient.DeleteStackAsync(request);
           Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

           bool stackDeleted = await WaitForStackDeletion(stackName,
cloudFormationClient, scenarioLogger);

           if (stackDeleted)
           {
               Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
               return true;
           }
           else
           {
               scenarioLogger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
               return false;
           }
       }
       catch (Exception ex)
       {
           scenarioLogger.LogError(ex, $"An error occurred while deleting the
CloudFormation stack: {stackName}");
           return false;
       }
   }

   /// <summary>
   /// Waits for the stack to be deleted.
```

```
   /// </summary>
   /// <param name="stackName">The name of the CloudFormation stack.</param>
   /// <param name="cloudFormationClient">The CloudFormation client.</param>
   /// <param name="scenarioLogger">The logger.</param>
   /// <returns>True if the stack was deleted successfully.</returns>
   private static async Task<bool> WaitForStackDeletion(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
   {
       int retryCount = 0;
       const int maxRetries = 30;
       const int retryDelay = 10000;

       while (retryCount < maxRetries)
       {
           var describeStacksRequest = new DescribeStacksRequest
           {
               StackName = stackName
           };

           try
           {
               var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

               if (describeStacksResponse.Stacks.Count == 0 ||
                   describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
               {
                   return true;
               }
           }
           catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
           {
               return true;
           }

           Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
           await Task.Delay(retryDelay);
           retryCount++;
       }
```

```
        scenarioLogger.LogError($"Timed out waiting for CloudFormation stack
 '{stackName}' to be deleted.");
        return false;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
 StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (IsInteractive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
 for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = new System.Text.RegularExpressions.Regex("[a-zA-Z][-a-
zA-Z0-9]*");
                if (!regex.IsMatch(input))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
 {_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }
}
```

A wrapper class for AWS IoT SDK methods.

```
/// <summary>
/// Wrapper methods to use Amazon IoT Core with .NET.
/// </summary>
public class IoTWrapper
{
    private readonly IAmazonIoT _amazonIoT;
    private readonly IAmazonIotData _amazonIotData;
    private readonly ILogger<IoTWrapper> _logger;

    /// <summary>
    /// Constructor for the IoT wrapper.
    /// </summary>
    /// <param name="amazonIoT">The injected IoT client.</param>
    /// <param name="amazonIotData">The injected IoT Data client.</param>
    /// <param name="logger">The injected logger.</param>
    public IoTWrapper(IAmazonIoT amazonIoT, IAmazonIotData amazonIotData,
 ILogger<IoTWrapper> logger)
    {
        _amazonIoT = amazonIoT;
        _amazonIotData = amazonIotData;
        _logger = logger;
    }

    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
```

```
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };

            var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }
```

```
    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
```

```
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
 {response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
 {ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
```

```
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
            return new List<Certificate>();
        }
    }

    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };

            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
```

```
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        }
                    },
                    RuleDisabled = false
                }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
```

```
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteTopicRuleAsync(string ruleName)
    {
        try
        {
            var request = new DeleteTopicRuleRequest
            {
                RuleName = ruleName,
            };

            await _amazonIoT.DeleteTopicRuleAsync(request);
            _logger.LogInformation($"Deleted IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogWarning($"Rule {ruleName} not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists all IoT topic rules.
    /// </summary>
    /// <returns>List of topic rules, or empty list if listing failed.</returns>
    public async Task<List<TopicRuleListItem>> ListTopicRulesAsync()
    {
        try
        {
            var request = new ListTopicRulesRequest();
```

```
            var response = await _amazonIoT.ListTopicRulesAsync(request);

            _logger.LogInformation($"Retrieved {response.Rules.Count} IoT rules");
            return response.Rules;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list topic rules. Here's why:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
    }

    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
```

```
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });
```

```
            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if it's
ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await _amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
```

```
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
```

```
            _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
```

```
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
```

```
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
```

```
        }
    }

}
```

Actions

`AttachThingPrincipal`

The following code example shows how to use `AttachThingPrincipal`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Attaches a certificate to an IoT Thing.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="certificateArn">The ARN of the certificate to attach.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
       {
           var request = new AttachThingPrincipalRequest
           {
               ThingName = thingName,
               Principal = certificateArn
           };

           await _amazonIoT.AttachThingPrincipalAsync(request);
           _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
           return true;
```

```
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- For API details, see AttachThingPrincipal in *AWS SDK for .NET API Reference*.

`CreateKeysAndCertificate`

The following code example shows how to use `CreateKeysAndCertificate`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
```

```
                SetAsActive = true
            };

            var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- For API details, see CreateKeysAndCertificate in *AWS SDK for .NET API Reference*.

`CreateThing`

The following code example shows how to use `CreateThing`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
```

```
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- For API details, see CreateThing in *AWS SDK for .NET API Reference*.

`CreateTopicRule`

The following code example shows how to use `CreateTopicRule`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        },
                        RuleDisabled = false
                    }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- For API details, see CreateTopicRule in *AWS SDK for .NET API Reference*.

`DeleteCertificate`

The following code example shows how to use `DeleteCertificate`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
```

```
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- For API details, see DeleteCertificate in *AWS SDK for .NET API Reference*.

`DeleteThing`

The following code example shows how to use `DeleteThing`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
```

```
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }
```

- For API details, see DeleteThing in *AWS SDK for .NET API Reference*.

`DescribeEndpoint`

The following code example shows how to use `DescribeEndpoint`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
```

```
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- For API details, see DescribeEndpoint in *AWS SDK for .NET API Reference*.

`DetachThingPrincipal`

The following code example shows how to use `DetachThingPrincipal`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Detaches a certificate from an IoT Thing.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="certificateArn">The ARN of the certificate to detach.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
       {
           var request = new DetachThingPrincipalRequest
           {
               ThingName = thingName,
               Principal = certificateArn
           };

           await _amazonIoT.DetachThingPrincipalAsync(request);
           _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
           return false;
       }
       catch (Exception ex)
       {
           _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
           return false;
       }
```

```
    }
```

- For API details, see DetachThingPrincipal in *AWS SDK for .NET API Reference*.

`ListCertificates`

The following code example shows how to use `ListCertificates`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
            return new List<Certificate>();
        }
    }
```

- For API details, see ListCertificates in *AWS SDK for .NET API Reference*.

`ListThings`

The following code example shows how to use `ListThings`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
```

```
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }
```

- For API details, see ListThings in *AWS SDK for .NET API Reference*.

`SearchIndex`

The following code example shows how to use `SearchIndex`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Searches for IoT Things using the search index.
   /// </summary>
   /// <param name="queryString">The search query string.</param>
   /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
   public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
   {
       try
       {
           // First, try to perform the search
           var request = new SearchIndexRequest
           {
               QueryString = queryString
           };

           var response = await _amazonIoT.SearchIndexAsync(request);
           _logger.LogInformation($"Search found {response.Things.Count} Things");
           return response.Things;
       }
       catch (Amazon.IoT.Model.IndexNotReadyException ex)
       {
           _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
           return await SetupIndexAndRetrySearchAsync(queryString);
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
       {
           _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
           return await SetupIndexAndRetrySearchAsync(queryString);
       }
       catch (Amazon.IoT.Model.ThrottlingException ex)
```

```
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
```

```
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if it's
ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await _amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }
```

```
            _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }
```

- For API details, see SearchIndex in *AWS SDK for .NET API Reference*.

`UpdateThing`

The following code example shows how to use `UpdateThing`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
```

```
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- For API details, see UpdateThing in *AWS SDK for .NET API Reference*.

AWS IoT data examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with AWS IoT data.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

`GetThingShadow`

The following code example shows how to use `GetThingShadow`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
```

```
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- For API details, see GetThingShadow in *AWS SDK for .NET API Reference*.

`UpdateThingShadow`

The following code example shows how to use `UpdateThingShadow`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };

            await _amazonIotData.UpdateThingShadowAsync(request);
```

```
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- For API details, see UpdateThingShadow in *AWS SDK for .NET API Reference*.

Amazon Redshift examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with Amazon Redshift.

Basics are code examples that show you how to perform the essential operations within a service.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Get started

Hello Amazon Redshift

The following code examples show how to get started using Amazon Redshift.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Main method to run the Hello Amazon Redshift example.
    /// </summary>
    /// <param name="args">Command line arguments (not used).</param>
    public static async Task Main(string[] args)
    {
        var redshiftClient = new AmazonRedshiftClient();

        Console.WriteLine("Hello, Amazon Redshift! Let's list available clusters:");

        var clusters = new List<Cluster>();

        try
        {
            // Use pagination to retrieve all clusters.
            var clustersPaginator = redshiftClient.Paginators.DescribeClusters(new
DescribeClustersRequest());

            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");

            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }
        }
        catch (AmazonRedshiftException ex)
        {
```

```
            Console.WriteLine($"Couldn't list clusters. Here's why: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
```

- For API details, see DescribeClusters in *AWS SDK for .NET API Reference*.

Topics

- Basics
- Actions

Basics

Learn the basics

The following code example shows how to:

- Create a Redshift cluster.
- List databases in the cluster.
- Create a table named Movies.
- Populate the Movies table.
- Query the Movies table by year.
- Modify the Redshift cluster.
- Delete the Amazon Redshift cluster.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a Redshift wrapper class to manage operations.

```
/// <summary>
/// Wrapper class for Amazon Redshift operations.
/// </summary>
public class RedshiftWrapper
{
    private readonly IAmazonRedshift _redshiftClient;
    private readonly IAmazonRedshiftDataAPIService _redshiftDataClient;

    /// <summary>
    /// Constructor for RedshiftWrapper.
    /// </summary>
    /// <param name="redshiftClient">Amazon Redshift client.</param>
    /// <param name="redshiftDataClient">Amazon Redshift Data API client.</param>
    public RedshiftWrapper(IAmazonRedshift redshiftClient,
 IAmazonRedshiftDataAPIService redshiftDataClient)
    {
        _redshiftClient = redshiftClient;
        _redshiftDataClient = redshiftDataClient;
    }

    /// <summary>
    /// Create a new Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="databaseName">The name of the database.</param>
    /// <param name="masterUsername">The master username.</param>
    /// <param name="masterUserPassword">The master user password.</param>
    /// <param name="nodeType">The node type for the cluster.</param>
    /// <returns>The cluster that was created.</returns>
    public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
 databaseName,
        string masterUsername, string masterUserPassword, string nodeType =
 "ra3.large")
    {
        try
        {
            var request = new CreateClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DBName = databaseName,
                MasterUsername = masterUsername,
                MasterUserPassword = masterUserPassword,
```

```
                NodeType = nodeType,
                NumberOfNodes = 1,
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
    /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
    /// <returns>A list of clusters.</returns>
    public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
    {
        try
        {
            var clusters = new List<Cluster>();
            var request = new DescribeClustersRequest();
            if (!string.IsNullOrEmpty(clusterIdentifier))
            {
                request.ClusterIdentifier = clusterIdentifier;
            }

            var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
```

```
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
 {cluster.ClusterStatus})");
            }

            return clusters;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
 {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe clusters. Here's why:
 {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };
```

```
            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Delete an Amazon Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
```

```
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
```

```
            throw;
        }
    }

    /// <summary>
    /// Create a table in the Redshift database.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> CreateTableAsync(string clusterIdentifier, string
database, string dbUser)
    {
        try
        {
            var sqlStatement = @"
                CREATE TABLE Movies (
                    id INTEGER PRIMARY KEY,
                    title VARCHAR(250) NOT NULL,
                    year INTEGER NOT NULL
                )";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine("Table created: Movies");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create table. Here's why: {ex.Message}");
```

```
            throw;
        }
    }

    /// <summary>
    /// Insert a record into the Movies table using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="id">The movie ID.</param>
    /// <param name="title">The movie title.</param>
    /// <param name="year">The movie year.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> InsertMovieAsync(string clusterIdentifier, string
database, string dbUser,
        int id, string title, int year)
    {
        try
        {
            var sqlStatement = "INSERT INTO Movies (id, title, year) VALUES
(:id, :title, :year)";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "id", Value = id.ToString() },
                    new SqlParameter { Name = "title", Value = title },
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine($"Inserted: {title} ({year})");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
```

```
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't insert movie. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Query movies by year using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="year">The year to query.</param>
    /// <returns>A list of movie titles.</returns>
    public async Task<List<string>> QueryMoviesByYearAsync(string clusterIdentifier,
string database,
        string dbUser, int year)
    {
        try
        {
            var sqlStatement = "SELECT title FROM Movies WHERE year = :year";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            Console.WriteLine($"The identifier of the statement is {response.Id}");

            await WaitForStatementToCompleteAsync(response.Id);

            var results = await GetStatementResultAsync(response.Id);
```

```
            var movieTitles = new List<string>();

            foreach (var row in results)
            {
                if (row.Count > 0)
                {
                    var title = row[0].StringValue;
                    Console.WriteLine($"The Movie title field is {title}");
                    movieTitles.Add(title);
                }
            }

            return movieTitles;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't query movies. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>The statement description.</returns>
    public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
    {
        try
        {
            var request = new DescribeStatementRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.DescribeStatementAsync(request);
            return response;
```

```
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }
```

```
    /// <summary>
    /// Wait for a statement to complete execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    private async Task WaitForStatementToCompleteAsync(string statementId)
    {
        var status = StatusString.SUBMITTED;
        DescribeStatementResponse? response = null;

        while (status == StatusString.SUBMITTED || status == StatusString.PICKED ||
status == StatusString.STARTED)
        {
            await Task.Delay(1000); // Wait 1 second
            response = await DescribeStatementAsync(statementId);
            status = response.Status;
            Console.WriteLine($"...{status}");
        }

        if (status == StatusString.FINISHED)
        {
            Console.WriteLine("The statement is finished!");
        }
        else
        {
            var errorMessage = response?.Error ?? "Unknown error";
            Console.WriteLine($"The statement failed with status: {status}");
            Console.WriteLine($"Error message: {errorMessage}");
        }
    }

    /// <summary>
    /// Wait for a cluster to become available.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public async Task WaitForClusterAvailableAsync(string clusterIdentifier)
    {
        Console.WriteLine($"Wait until {clusterIdentifier} is available. This may
take a few minutes.");

        var startTime = DateTime.Now;
        var clusters = await DescribeClustersAsync(clusterIdentifier);
```

```
        while (clusters[0].ClusterStatus != "available")
        {
            var elapsed = DateTime.Now - startTime;
            Console.WriteLine($"Elapsed Time: {elapsed:mm\\:ss} - Waiting for
 cluster...");

            await Task.Delay(5000); // Wait 5 seconds
            clusters = await DescribeClustersAsync(clusterIdentifier);
        }

        var totalElapsed = DateTime.Now - startTime;
        Console.WriteLine($"Cluster is available! Total Elapsed Time:
 {totalElapsed:mm\\:ss}");
    }
}
```

Run an interactive scenario demonstrating Redshift basics.

```
/// <summary>
/// Amazon Redshift Getting Started Scenario.
/// </summary>
public class RedshiftBasics
{
    public static bool IsInteractive = true;
    public static RedshiftWrapper? Wrapper = null;
    public static ILogger logger = null!;
    private static readonly string _moviesFilePath = "../../../../../../resources/
sample_files/movies.json";

    /// <summary>
    /// Main method for the Amazon Redshift Getting Started scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRedshift>()
                    .AddAWSService<IAmazonRedshiftDataAPIService>()
                    .AddTransient<RedshiftWrapper>()
            )
```

```
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<RedshiftBasics>();

        Wrapper = host.Services.GetRequiredService<RedshiftWrapper>();

        await RunScenarioAsync();
    }

    /// <summary>
    /// Run the complete Amazon Redshift scenario.
    /// </summary>
    public static async Task RunScenarioAsync()
    {
        // Set all variables to default values
        string userName = "awsuser";
        string userPassword = "AwsUser1000";
        string clusterIdentifier = "redshift-cluster-movies";
        var databaseName = "dev";
        int recordCount = 50;
        int year = 2013;
        try
        {
            Console.WriteLine(

"================================================================================");
            Console.WriteLine("Welcome to the Amazon Redshift SDK Getting Started
scenario.");
            Console.WriteLine(
                "This .NET program demonstrates how to interact with Amazon Redshift
by using the AWS SDK for .NET.");
            Console.WriteLine("Let's get started...");
            Console.WriteLine(

"================================================================================");

            // Step 1: Get user credentials (if interactive)
            if (IsInteractive)
            {
                Console.WriteLine("Please enter a user name for the cluster (default
is awsuser):");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
```

```
                    userName = userInput;


 Console.WriteLine("=========================================================================
                Console.WriteLine("Please enter a user password for the cluster
 (default is AwsUser1000):");
                var passwordInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(passwordInput))
                    userPassword = passwordInput;


 Console.WriteLine("=========================================================================

                // Step 2: Get cluster identifier
                Console.WriteLine("Enter a cluster id value (default is redshift-
cluster-movies):");
                var clusterInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(clusterInput))
                    clusterIdentifier = clusterInput;
            }
            else
            {
                Console.WriteLine($"Using default values: userName={userName},
 clusterIdentifier={clusterIdentifier}");
            }

            // Step 3: Create Redshift cluster
            await Wrapper!.CreateClusterAsync(clusterIdentifier, databaseName,
 userName, userPassword);

 Console.WriteLine("=========================================================================

            // Step 4: Wait for cluster to become available

 Console.WriteLine("=========================================================================
            await Wrapper.WaitForClusterAvailableAsync(clusterIdentifier);

 Console.WriteLine("=========================================================================

            // Step 5: List databases

 Console.WriteLine("=========================================================================
            Console.WriteLine($" When you created {clusterIdentifier}, the dev
 database is created by default and used in this scenario.");
```

```
            Console.WriteLine(" To create a custom database, you need to have a
CREATEDB privilege.");
            Console.WriteLine(" For more information, see the documentation here:
https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_DATABASE.html.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

Console.WriteLine("=========================================================================



Console.WriteLine("=========================================================================
            Console.WriteLine($"List databases in {clusterIdentifier}");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ListDatabasesAsync(clusterIdentifier, userName,
databaseName);

Console.WriteLine("=========================================================================

            // Step 6: Create Movies table

Console.WriteLine("=========================================================================
            Console.WriteLine("Now you will create a table named Movies.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.CreateTableAsync(clusterIdentifier, databaseName,
userName);

Console.WriteLine("=========================================================================

            // Step 7: Populate the Movies table

Console.WriteLine("=========================================================================
            Console.WriteLine("Populate the Movies table using the Movies.json
file.");
```

```
            if (IsInteractive)
            {
                Console.WriteLine("Specify the number of records you would like to
add to the Movies Table.");
                Console.WriteLine("Please enter a value between 50 and 200.");
                Console.Write("Enter a value: ");

                var recordCountInput = Console.ReadLine();
                if (int.TryParse(recordCountInput, out var inputCount) && inputCount
is >= 50 and <= 200)
                {
                    recordCount = inputCount;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
{recordCount}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default record count: {recordCount}");
            }

            await PopulateMoviesTableAsync(clusterIdentifier, databaseName,
userName, recordCount);
            Console.WriteLine($"{recordCount} records were added to the Movies
table.");

Console.WriteLine("=============================================================================

            // Step 8 & 9: Query movies by year

Console.WriteLine("=============================================================================
            Console.WriteLine("Query the Movies table by year. Enter a value between
2012-2014.");

            if (IsInteractive)
            {
                Console.Write("Enter a year: ");
                var yearInput = Console.ReadLine();
                if (int.TryParse(yearInput, out var inputYear) && inputYear is >=
2012 and <= 2014)
```

```
                {
                    year = inputYear;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
 {year}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default year: {year}");
            }

            await Wrapper.QueryMoviesByYearAsync(clusterIdentifier, databaseName,
 userName, year);

 Console.WriteLine("=========================================================================

            // Step 10: Modify the cluster

 Console.WriteLine("=========================================================================
            Console.WriteLine("Now you will modify the Redshift cluster.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ModifyClusterAsync(clusterIdentifier, "wed:07:30-
wed:08:00");

 Console.WriteLine("=========================================================================

            // Step 11 & 12: Delete cluster confirmation

 Console.WriteLine("=========================================================================
            if (IsInteractive)
            {
                Console.WriteLine("Would you like to delete the Amazon Redshift
 cluster? (y/n)");
                var deleteResponse = Console.ReadLine();
                if (deleteResponse?.ToLower() == "y")
                {
```

```
                    await
Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
                }
            }
            else
            {
                Console.WriteLine("Deleting the Amazon Redshift cluster...");
                await Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            }

Console.WriteLine("=====================================================================



Console.WriteLine("=====================================================================
            Console.WriteLine("This concludes the Amazon Redshift SDK Getting
Started scenario.");

Console.WriteLine("=====================================================================
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during the scenario:
{ex.Message}");
            Console.WriteLine("Deleting the Amazon Redshift cluster...");
            await Wrapper!.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            throw;
        }
    }

    /// <summary>
    /// Populate the Movies table with data from the JSON file.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="recordCount">Number of records to insert.</param>
    private static async Task PopulateMoviesTableAsync(string clusterIdentifier,
string database, string dbUser, int recordCount)
    {
        if (!File.Exists(_moviesFilePath))
        {
            throw new FileNotFoundException($"Required movies data file not found
at: {_moviesFilePath}");
        }
```

```
        var jsonContent = await File.ReadAllTextAsync(_moviesFilePath);
        var options = new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        };
        var movies = JsonSerializer.Deserialize<List<Movie>>(jsonContent, options);

        if (movies == null || movies.Count == 0)
        {
            throw new InvalidOperationException("Failed to parse movies JSON file or
 file is empty.");
        }

        var insertCount = Math.Min(recordCount, movies.Count);

        for (int i = 0; i < insertCount; i++)
        {
            var movie = movies[i];
            await Wrapper!.InsertMovieAsync(clusterIdentifier, database, dbUser, i,
 movie.Title, movie.Year);
        }
    }

    /// <summary>
    /// Movie data model.
    /// </summary>
    private class Movie
    {
        public string Title { get; set; } = string.Empty;
        public int Year { get; set; }
    }
}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - CreateCluster
  - DescribeClusters
  - DescribeStatement
  - ExecuteStatement
  - GetStatementResult

- ListDatabasesPaginator
- ModifyCluster

Actions

`CreateCluster`

The following code example shows how to use `CreateCluster`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Create a new Amazon Redshift cluster.
   /// </summary>
   /// <param name="clusterIdentifier">The identifier for the cluster.</param>
   /// <param name="databaseName">The name of the database.</param>
   /// <param name="masterUsername">The master username.</param>
   /// <param name="masterUserPassword">The master user password.</param>
   /// <param name="nodeType">The node type for the cluster.</param>
   /// <returns>The cluster that was created.</returns>
   public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
databaseName,
       string masterUsername, string masterUserPassword, string nodeType =
"ra3.large")
   {
       try
       {
           var request = new CreateClusterRequest
           {
               ClusterIdentifier = clusterIdentifier,
               DBName = databaseName,
               MasterUsername = masterUsername,
               MasterUserPassword = masterUserPassword,
               NodeType = nodeType,
               NumberOfNodes = 1,
```

```
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }
```

- For API details, see CreateCluster in *AWS SDK for .NET API Reference*.

`DeleteCluster`

The following code example shows how to use `DeleteCluster`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Delete an Amazon Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
```

```
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- For API details, see DeleteCluster in *AWS SDK for .NET API Reference*.

`DescribeClusters`

The following code example shows how to use `DescribeClusters`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
```

```
   /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
   /// <returns>A list of clusters.</returns>
   public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
   {
       try
       {
           var clusters = new List<Cluster>();
           var request = new DescribeClustersRequest();
           if (!string.IsNullOrEmpty(clusterIdentifier))
           {
               request.ClusterIdentifier = clusterIdentifier;
           }

           var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
           await foreach (var response in clustersPaginator.Responses)
           {
               if (response.Clusters != null)
                   clusters.AddRange(response.Clusters);
           }

           Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
           foreach (var cluster in clusters)
           {
               Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
           }

           return clusters;
       }
       catch (ClusterNotFoundException ex)
       {
           Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
           throw;
       }
       catch (Exception ex)
       {
           Console.WriteLine($"Couldn't describe clusters. Here's why:
{ex.Message}");
           throw;
       }
```

```
    }
```

- For API details, see DescribeClusters in *AWS SDK for .NET API Reference*.

`DescribeStatement`

The following code example shows how to use `DescribeStatement`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   /// <summary>
   /// Describe a statement execution.
   /// </summary>
   /// <param name="statementId">The statement ID.</param>
   /// <returns>The statement description.</returns>
   public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
   {
       try
       {
           var request = new DescribeStatementRequest
           {
               Id = statementId
           };

           var response = await
_redshiftDataClient.DescribeStatementAsync(request);
           return response;
       }
       catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
       {
           Console.WriteLine($"Statement not found: {ex.Message}");
           throw;
       }
       catch (Exception ex)
```

```
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- For API details, see DescribeStatement in *AWS SDK for .NET API Reference*.

`GetStatementResult`

The following code example shows how to use `GetStatementResult`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- For API details, see GetStatementResult in *AWS SDK for .NET API Reference*.

`ListDatabases`

The following code example shows how to use `ListDatabases`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
```

```
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
            throw;
        }
    }
```

- For API details, see ListDatabases in *AWS SDK for .NET API Reference*.

`ModifyCluster`

The following code example shows how to use `ModifyCluster`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
```

```
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };

            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
 has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
 {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- For API details, see ModifyCluster in *AWS SDK for .NET API Reference*.

Amazon S3 examples using SDK for .NET (v4)

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for .NET (v4) with Amazon S3.

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions
- Scenarios

Actions

`CreatePresignedPost`

The following code example shows how to use `CreatePresignedPost`.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a presigned POST URL.

```
    /// <summary>
    /// Create a presigned POST URL with conditions.
    /// </summary>
    /// <param name="s3Client">The Amazon S3 client.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <param name="objectKey">The object key (path) where the uploaded file will
be stored.</param>
    /// <param name="expires">When the presigned URL expires.</param>
    /// <param name="fields">Dictionary of fields to add to the form.</param>
    /// <param name="conditions">List of conditions to apply.</param>
```

```
    /// <returns>A CreatePresignedPostResponse object with URL and form fields.</
returns>
    public async Task<CreatePresignedPostResponse> CreatePresignedPostAsync(
        IAmazonS3 s3Client,
        string bucketName,
        string objectKey,
        DateTime expires,
        Dictionary<string, string>? fields = null,
        List<S3PostCondition>? conditions = null)
    {
        var request = new CreatePresignedPostRequest
        {
            BucketName = bucketName,
            Key = objectKey,
            Expires = expires
        };

        // Add custom fields if provided
        if (fields != null)
        {
            foreach (var field in fields)
            {
                request.Fields.Add(field.Key, field.Value);
            }
        }

        // Add conditions if provided
        if (conditions != null)
        {
            foreach (var condition in conditions)
            {
                request.Conditions.Add(condition);
            }
        }

        return await s3Client.CreatePresignedPostAsync(request);
    }
```

- For API details, see CreatePresignedPost in *AWS SDK for .NET API Reference*.

Scenarios

Create a presigned URL

The following code example shows how to create a presigned URL for Amazon S3 and upload an object.

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create and use presigned POST URLs for direct browser uploads.

```
/// <summary>
/// Scenario demonstrating the complete workflow for presigned POST URLs:
/// 1. Create an S3 bucket
/// 2. Create a presigned POST URL
/// 3. Upload a file using the presigned POST URL
/// 4. Clean up resources
/// </summary>
public class CreatePresignedPostBasics
{
    public static ILogger<CreatePresignedPostBasics> _logger = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static UiMethods _uiMethods = null!;
    public static IHttpClientFactory _httpClientFactory = null!;
    public static bool _isInteractive = true;
    public static string? _bucketName;
    public static string? _objectKey;

    /// <summary>
    /// Set up the services and logging.
    /// </summary>
    /// <param name="host">The IHost instance.</param>
    public static void SetUpServices(IHost host)
    {
        var loggerFactory = LoggerFactory.Create(builder =>
        {
```

```
            builder.AddConsole();
        });
        _logger = new Logger<CreatePresignedPostBasics>(loggerFactory);

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();
        _httpClientFactory = host.Services.GetRequiredService<IHttpClientFactory>();
        _uiMethods = new UiMethods();
    }

    /// <summary>
    /// Perform the actions defined for the Amazon S3 Presigned POST scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        _isInteractive = !args.Contains("--non-interactive");

        // Set up dependency injection for Amazon S3
        using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
                    .AddTransient<S3Wrapper>()
                    .AddHttpClient()
            )
            .Build();

        SetUpServices(host);

        try
        {
            // Display overview
            _uiMethods.DisplayOverview();
            _uiMethods.PressEnter(_isInteractive);

            // Step 1: Create bucket
            await CreateBucketAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 2: Create presigned URL
            _uiMethods.DisplayTitle("Step 2: Create presigned POST URL");
            var response = await CreatePresignedPostAsync();
            _uiMethods.PressEnter(_isInteractive);
```

```
            // Step 3: Display URL and fields
            _uiMethods.DisplayTitle("Step 3: Presigned POST URL details");
            DisplayPresignedPostFields(response);
            _uiMethods.PressEnter(_isInteractive);

            // Step 4: Upload file
            _uiMethods.DisplayTitle("Step 4: Upload test file using presigned POST
URL");
            await UploadFileAsync(response);
            _uiMethods.PressEnter(_isInteractive);

            // Step 5: Verify file exists
            await VerifyFileExistsAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 6: Cleanup
            _uiMethods.DisplayTitle("Step 6: Clean up resources");
            await CleanupAsync();

            _uiMethods.DisplayTitle("S3 Presigned POST Scenario completed
successfully!");
            _uiMethods.PressEnter(_isInteractive);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error in scenario");
            Console.WriteLine($"Error: {ex.Message}");

            // Attempt cleanup if there was an error
            if (!string.IsNullOrEmpty(_bucketName))
            {
                _uiMethods.DisplayTitle("Cleaning up resources after error");
                await _s3Wrapper.DeleteBucketAsync(_bucketName);
                Console.WriteLine($"Cleaned up bucket: {_bucketName}");
            }
        }
    }

    /// <summary>
    /// Create an S3 bucket for the scenario.
    /// </summary>
    private static async Task CreateBucketAsync()
    {
```

```
        _uiMethods.DisplayTitle("Step 1: Create an S3 bucket");

        // Generate a default bucket name for the scenario
        var defaultBucketName = $"presigned-post-demo-
{DateTime.Now:yyyyMMddHHmmss}".ToLower();

        // Prompt user for bucket name or use default in non-interactive mode
        _bucketName = _uiMethods.GetUserInput(
            $"Enter S3 bucket name (or press Enter for '{defaultBucketName}'): ",
            defaultBucketName,
            _isInteractive);

        // Basic validation to ensure bucket name is not empty
        if (string.IsNullOrWhiteSpace(_bucketName))
        {
            _bucketName = defaultBucketName;
        }

        Console.WriteLine($"Creating bucket: {_bucketName}");

        await _s3Wrapper.CreateBucketAsync(_bucketName);

        Console.WriteLine($"Successfully created bucket: {_bucketName}");
    }


    /// <summary>
    /// Create a presigned POST URL.
    /// </summary>
    private static async Task<CreatePresignedPostResponse>
 CreatePresignedPostAsync()
    {
        _objectKey = "example-upload.txt";
        var expiration = DateTime.UtcNow.AddMinutes(10); // Short expiration for the
 demo

        Console.WriteLine($"Creating presigned POST URL for {_bucketName}/
{_objectKey}");
        Console.WriteLine($"Expiration: {expiration} UTC");

        var s3Client = _s3Wrapper.GetS3Client();

        var response = await _s3Wrapper.CreatePresignedPostAsync(
            s3Client, _bucketName!, _objectKey, expiration);
```

```
        Console.WriteLine("Successfully created presigned POST URL");
        return response;
    }

    /// <summary>
    /// Upload a file using the presigned POST URL.
    /// </summary>
    private static async Task UploadFileAsync(CreatePresignedPostResponse response)
    {

        // Create a temporary test file to upload
        string testFilePath = Path.GetTempFileName();
        string testContent = "This is a test file for the S3 presigned POST
scenario.";

        await File.WriteAllTextAsync(testFilePath, testContent);
        Console.WriteLine($"Created test file at: {testFilePath}");

        // Upload the file using the presigned POST URL
        Console.WriteLine("\nUploading file using the presigned POST URL...");
        var uploadResult = await UploadFileWithPresignedPostAsync(response,
testFilePath);

        // Display the upload result
        if (uploadResult.Success)
        {
            Console.WriteLine($"Upload successful! Status code:
{uploadResult.StatusCode}");
        }
        else
        {
            Console.WriteLine($"Upload failed with status code:
{uploadResult.StatusCode}");
            Console.WriteLine($"Error: {uploadResult.Response}");
            throw new Exception("File upload failed");
        }

        // Clean up the temporary file
        File.Delete(testFilePath);
        Console.WriteLine("Temporary file deleted");
    }

    /// <summary>
```

```
    /// Helper method to upload a file using a presigned POST URL.
    /// </summary>
    private static async Task<(bool Success, HttpStatusCode StatusCode, string
 Response)> UploadFileWithPresignedPostAsync(
        CreatePresignedPostResponse response,
        string filePath)
    {
        try
        {
            _logger.LogInformation("Uploading file {filePath} using presigned POST
 URL", filePath);

            using var httpClient = _httpClientFactory.CreateClient();
            using var formContent = new MultipartFormDataContent();

            // Add all the fields from the presigned POST response
            foreach (var field in response.Fields)
            {
                formContent.Add(new StringContent(field.Value), field.Key);
            }

            // Add the file content
            var fileStream = File.OpenRead(filePath);
            var fileName = Path.GetFileName(filePath);
            var fileContent = new StreamContent(fileStream);
            fileContent.Headers.ContentType = new MediaTypeHeaderValue("text/
plain");
            formContent.Add(fileContent, "file", fileName);

            // Send the POST request
            var httpResponse = await httpClient.PostAsync(response.Url,
 formContent);
            var responseContent = await httpResponse.Content.ReadAsStringAsync();

            // Log and return the result
            _logger.LogInformation("Upload completed with status code {statusCode}",
 httpResponse.StatusCode);

            return (httpResponse.IsSuccessStatusCode, httpResponse.StatusCode,
 responseContent);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error uploading file");
```

```
            return (false, HttpStatusCode.InternalServerError, ex.Message);
        }
    }

    /// <summary>
    /// Verify that the uploaded file exists in the S3 bucket.
    /// </summary>
    private static async Task VerifyFileExistsAsync()
    {
        _uiMethods.DisplayTitle("Step 5: Verify uploaded file exists");

        Console.WriteLine($"Checking if file exists at {_bucketName}/
{_objectKey}...");

        try
        {
            var metadata = await _s3Wrapper.GetObjectMetadataAsync(_bucketName!,
 _objectKey!);

            Console.WriteLine($"File verification successful! File exists in the
 bucket.");
            Console.WriteLine($"File size: {metadata.ContentLength} bytes");
            Console.WriteLine($"File type: {metadata.Headers.ContentType}");
            Console.WriteLine($"Last modified: {metadata.LastModified}");
        }
        catch (AmazonS3Exception ex) when (ex.StatusCode ==
 System.Net.HttpStatusCode.NotFound)
        {
            Console.WriteLine($"Error: File was not found in the bucket.");
            throw;
        }
    }

    private static void DisplayPresignedPostFields(CreatePresignedPostResponse
 response)
    {
        Console.WriteLine($"Presigned POST URL: {response.Url}");
        Console.WriteLine("Form fields to include:");

        foreach (var field in response.Fields)
        {
            Console.WriteLine($"  {field.Key}: {field.Value}");
        }
    }
```

```
    /// <summary>
    /// Clean up resources created by the scenario.
    /// </summary>
    private static async Task CleanupAsync()
    {
        if (!string.IsNullOrEmpty(_bucketName))
        {
            Console.WriteLine($"Deleting bucket {_bucketName} and its contents...");
            bool result = await _s3Wrapper.DeleteBucketAsync(_bucketName);

            if (result)
            {
                Console.WriteLine("Bucket deleted successfully");
            }
            else
            {
                Console.WriteLine("Failed to delete bucket - it may have been
 already deleted");
            }
        }
    }
}
```

Migrating your project for the AWS SDK for .NET

This section provides information about migration tasks for version 4 of the AWS SDK for .NET, and instructions about how to perform those tasks. For information about migrating to earlier versions (for example, migrating from V2 to V3, or from one V3 sub-version to another), see Migrate your project in the AWS SDK for .NET Developer Guide (version 3).

Also see What's new and Supported platforms.

Topics

- Migrating to version 4 of the AWS SDK for .NET
- Migrating from .NET Standard 1.3

Migrating to version 4 of the AWS SDK for .NET

The AWS SDK for .NET version 4 (V4) has a significant number of breaking changes from version 3 (V3) of the SDK. This topic describes the breaking changes in version 4 and possible work that you might need to do to migrate your environment or code from V3. For additional information about other noteworthy changes in the SDK, see the following resources:

- The development-tracker issue on GitHub: https://github.com/aws/aws-sdk-net/issues/3362.
- The blog post Preview 1 of AWS SDK for .NET V4.
- The blog post Preview 4 of AWS SDK for .NET V4.
- The blog post General Availability of AWS SDK for .NET V4.0.

.NET Framework

The .NET Framework 3.5 target has been removed from V4 of the AWS SDK for .NET. As a result, the SDK no longer supports .NET Framework 3.5. This version of the SDK is compiled against .NET Framework 4.7.2 and runs in the .NET 4.0 runtime. For more information, see Supported platforms.

Value types

Properties that use value types in classes that are used for making requests and responses have been changed to use nullable value types. Properties with the following types have been changed:

- `bool` has been changed to `bool?`
- `double` has been changed to `double?`
- `int` has been changed to `int?`
- `float` has been changed to `float?`
- `long` has been changed to `long?`
- `Datetime` has been changed to `Datetime?`

For additional information about this change, see the blog post Preview 1 of AWS SDK for .NET V4.

Collections

Properties that use collections in classes that are used for making requests and responses now default to `null`. As a result, your code needs to verify that a collection isn't null before trying to use it. For example:

```
var sqsClient = new AmazonSQSClient();
var listResponse = await sqsClient.ListQueuesAsync(new ListQueuesRequest());
if (listResponse.QueueUrls != null)
{
    foreach (string qUrl in listResponse.QueueUrls)
    {
        // Perform operations on each queue such as displaying all the attributes.
    }
}
```

The V3 behavior of initializing collections can be restored by setting `Amazon.AWSConfigs.InitializeCollections` to `true`. This property also exists in V3 for users who want to try this behavior change before upgrading to V4.

For additional information about this change, see the blog post Preview 1 of AWS SDK for .NET V4.

AWS Security Token Service (STS)

- **The regional endpoint**

  When using credential providers that rely on AWS STS, the calls always use the regional endpoint. This differs from V3 of the SDK, which used the `us-east-1` Region by default when running in the public partition regardless of the configured Region.

- **The `StsRegionalEndpointsValue` enum**

  The `StsRegionalEndpointsValue` enum was removed from the Amazon.Runtime namespace. Any code using that enum should be removed.

- **The `STSAssumeRoleAWSCredentials` class**

  The deprecated STS assume role credential provider, `STSAssumeRoleAWSCredentials`, has been removed from the Amazon.SecurityToken namespace. Use AssumeRoleAWSCredentials from Amazon.Runtime instead.

Changes related to `ClientConfig`

The Amazon.Runtime.ClientConfig class is the base class of service client configuration classes like AmazonS3Config. The following changes were made to this base class.

- **Default retry mode**

  The `RetryMode` property defaults to `Standard` instead of `Legacy`. As a result, the `Legacy` value has been removed from the Amazon.Runtime.RequestRetryMode enum.

- **Default configuration mode**

  The `DefaultConfigurationMode` property defaults to `Standard` instead of `Legacy`. As a result, the `Legacy` value has been removed from the Amazon.Runtime.DefaultConfigurationMode enum.

- **The `ReadWriteTimeout` property**

  The obsolete `ReadWriteTimeout` property was removed from all targets except .NET Framework 4.7.2.

The AWSSDK.Extensions.NETCore.Setup NuGet package

The AWSSDK.Extensions.NETCore.Setup NuGet package has been updated to alleviate issues that were present in V3 of the SDK as well as to make the package safe for Native AOT. These changes are summarized below. For detailed information see PR 3353 in the aws-sdk-net repository on GitHub.

- **The `DefaultClientConfig` class**

The `DefaultClientConfig` class is no longer inherited from the service client configuration base class Amazon.Runtime.ClientConfig. The relevant properties from `ClientConfig` have been replicated on `DefaultClientConfig` using nullable value types. This change allows us to detect when a value has been set on `DefaultClientConfig` when copying the values to the configuration being created for the service client.

One particular result of this change is that `DefaultClientConfig.HttpClientFactory` is no longer available in V4. Use `AWSConfigs.HttpClientFactory` instead. For additional information, see GitHub issue 3790.

Another result of this change is that the generic overloads of the `IConfiguration.GetAWSOptions` extension method that accepted a service configuration object have been removed. The non-generic overload should be used instead, and the SDK will automatically handle populating service-specific settings. For additional information, see GitHub issue 3866.

- **Native AOT**

  A new mechanism for creating service clients that uses C# 11 static interface methods has been added to the package. This change eliminates the need to do Assembly type loads to create instances of service clients, including string manipulation of the service interface name to compute the service client type, which is incompatible with Native AOT. This change is available only for .NET 8 and later; older versions still use the original mechanism.

For additional information in this guide about this package, see AWSSDK.Extensions.NETCore.Setup and IConfiguration. The source code for this package is on GitHub at https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup.

`CookieSigner` and `UrlSigner`

The `CookieSigner` and `UrlSigner` extensions for Amazon CloudFront have been moved to a separate extension package called AWSSDK.Extensions.CloudFront.Signers. This change is to support OpenSSL 3 and take a dependency on BouncyCastle.Cryptography.

The source code for this package is on GitHub at https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.CloudFront.Signers.

DateTime versus UTC DateTime

Some V3 classes have a DateTime property that's marked as "deprecated" or "obsolete", as well as an alternative UTC DateTime property. In these classes, the obsolete DateTime property has been removed, and the name of the UTC DateTime property has been changed to the original name of the DateTime property.

The following are some examples of classes for which this change has been implemented.

- DescribeSpotPriceHistoryRequest:
  - The obsolete `StartTime` property has been removed, and the name of the `StartTimeUtc` property has been changed to "StartTime".
  - The obsolete `EndTime` property has been removed, and the name of the `EndTimeUtc` property has been changed to "EndTime".
- CreateFleetRequest
  - The obsolete `ValidFrom` property has been removed, and the name of the `ValidFromUtc` property has been changed to "ValidFrom".
  - The obsolete `ValidUntil` property has been removed, and the name of the `ValidUntilUtc` property has been changed to "ValidUntil".

This change might lead to offset times if an application is using the original, obsolete DateTime property. A compile time error will occur for code that uses the UTC DateTime property.

DateTime parsing

The DateTimeUnmarshaller class has been updated. This class had been parsing and returning DateTime strings as local time. In some cases, these values were being converted back to UTC due to a prior update, but not always. Now, DateTime strings that are unmarshalled are assumed to be UTC and will be specified and unmarshalled as UTC. This update includes the following behavior changes.

Certain timestamp properties that are based on the DateTime class were being parsed into local times. These included response unmarshallers for timestamps and list timestamps for formats `TimestampFormat.ISO8601` and `TimestampFormat.RFC822`. DateTime parsing has been updated to return UTC times instead.

ConvertFromUnixEpochSeconds and ConvertFromUnixEpochMilliseconds

The ConvertFromUnixEpochSeconds and ConvertFromUnixEpochMilliseconds methods, which convert Unix epoch seconds to a DateTime structure, were returning the Unix Epoch time as a local time instead of a UTC time. These methods now return UTC time.

Logging

The way in which you enable logging in the SDK has been updated for V4. Logging to the console and system diagnostics works the same as V3; that is, by setting the `LoggingConfig.LogTo` property of the AWSConfigs class to either `LoggingOptions.Console` or `LoggingOptions.SystemDiagnostics`. The LoggingOptions option for `log4net` has been removed along with the SDK's internal logic for using reflection to attach to an in-memory instance of `log4net`.

To include the SDK's logging into a logging framework, a separate adaptor package is used to connect the SDK with the logging framework. Use the AWSSDK.Extensions.Logging.Log4NetAdaptor package for `log4net` and the AWSSDK.Extensions.Logging.ILoggerAdaptor package for `Microsoft.Extensions.Logging`. The following code examples show you how to configure logging in these two cases.

Example configuration for log4net

Add the `AWSSDK.Extensions.Logging.Log4NetAdaptor` NuGet package and call the static `ConfigureAWSSDKLogging` method from `Log4NetAWSExtensions`.

```
using Amazon.DynamoDBv2;
using Amazon.Extensions.Logging.Log4NetAdaptor;
using log4net;

[assembly: log4net.Config.XmlConfigurator(ConfigFile = "log4net.config")]

Log4NetAWSExtensions.ConfigureAWSSDKLogging();
var logger = LogManager.GetLogger(typeof(Program));
```

Example configuration for Microsoft.Extensions.Logging

Add the `AWSSDK.Extensions.Logging.ILoggerAdaptor` NuGet package and call the `ConfigureAWSSDKLogging` extension method from the `ILoggerFactory` interface.

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.Services.GetRequiredService<ILoggerFactory>()
    .ConfigureAWSSDKLogging();
```

Support for HTTP 2

Support for HTTP 2 has been added to enable bi-directional streaming. For more information see Support for HTTP 2.

Single sign-on

The default value of the `SupportsGettingNewToken` property of the SSOAWSCredentialsOptions class has been changed from `true` to `false`. If you have applications that use the SSOAWSCredentials class to obtain SSO credentials, you might need to set the `Options.SupportsGettingNewToken` property to `true`. For examples of this configuration, see the code examples in Tutorial for SSO using only .NET applications. For additional information, see PR 3737 in the aws-sdk-net GitHub repository.

Changes specific to DynamoDB

The following changes are specific to Amazon DynamoDB. Many of them are breaking changes.

For additional information about changes to DynamoDB in V4 of the AWS SDK for .NET, see the blog post Preview 4 of AWS SDK for .NET V4.

Open to view items

V4 changes in the SDK for DynamoDB address some issues around testability, but primarily center around the high-level libraries:

- The .NET Document Model, named DocumentModel in the code.
- The .NET Object Persistence Model, named DataModel in the code.

For detailed information about these programming modes, see DynamoDB in this guide.

Document Model: Updated exception for mocked `IAmazonDynamoDB` interface

In the document model prior to V4 of the SDK, if a Table was initialized with a mocked IAmazonDynamoDB interface, it would return `NullReferenceException`. V4 of the SDK returns `InvalidOperationException` instead. Async `Table` methods should work with a mocked client but you might still see exceptions when calling synchronous methods from `.NET/Core/Standard`.

For more information about this change, see PR 3388 on GitHub.

Document Model: `FromJson` and `ToJson` methods

The `FromJson` and `ToJson` methods of the Document class now use `System.Text.Json` instead of LitJson for serialization, and LitJson has been removed from V4 of the SDK. A benefit of using `System.Text.Json` is that this parser supports using the .NET `Decimal` type, which supports higher precision for numeric floating point properties.

Object Persistence Model: The `DynamoDBOperationConfig` class

In the object persistence model, the following changes have been made to the shared DynamoDBOperationConfig class:

- The class has been separated into new operation-specific classes such as SaveConfig, LoadConfig, and QueryConfig. Methods that take `DynamoDBOperationConfig` have been marked as obsolete and are subject to removal in the future.

  For more information about this change, see PR 3421 on GitHub.
- The `MetadataCachingMode` and `DisableFetchingTableMetadata` properties have been removed from the class. These properties were not included in the new operation-specific classes mentioned earlier. The removed properties are table-level settings that should be specified on the global `Context` property of the AWSConfigsDynamoDB class or on the DynamoDBContextConfig class.

  For more information about this change, see PR 3422 on GitHub.
- The class no longer inherits from the DynamoDBContextConfig class. This prevents you from passing a `DynamoDBOperationConfig` object in to the constructor for DynamoDBContext, where some properties on the operation-specific config (such as `OverrideTableName`) don't apply.

  For more information about this change, see PR 3422 on GitHub.

Object Persistence Model: Polymorphism

The DynamoDBPolymorphicTypeAttribute class was added to the object persistence model. This class enables support for serialization and deserialization of polymorphic types. For more information, see PR 3643 on GitHub.

Document Model and Object Persistence Model: Mockable operations

New operation-specific interfaces have been added that allow customers to mock DynamoDB operations. The factory methods on the IDynamoDBContext interface have been updated to return the new interfaces.

For more information about this change, see PR 3450 on GitHub.

- Object Persistence Model
  - Mock `BatchGet` operations via the `IBatchGet` and `IMultiTableBatchGet` interfaces.
  - Mock `BatchWrite` operations via the `IBatchWrite` and `IMultiTableBatchWrite` interfaces.
  - Mock `TransactGet` operations via the `ITransactGet` and `IMultiTableTransactGet` interfaces.
  - Mock `TransactWrite` operations via the `ITransactWrite` and `IMultiTableTransactWrite` interfaces.
  - Mock `Scan` and `Query` operations via the `IAsyncSearch` interface.
- Document Model
  - Mock `Table` operations via the `ITable` interface.
  - Mock `Scan` and `Query` operations via the `ISearch` interface.
  - Mock `TransactWrite` operations via the `IDocumentTransactWrite` and `IMultiTableDocumentTransactWrite` interfaces.
  - Mock `TransactGet` operations via the `IDocumentTransactGet` and `IMultiTableDocumentTransactGet` interfaces.
  - Mock `BatchWrite` operations via the `IDocumentBatchWrite` and `IMultiTableDocumentBatchWrite` interfaces.
  - Mock `BatchGet` operations via the `IDocumentBatchGet` and `IMultiTableDocumentBatchGet` interfaces.

Document Model and Object Persistence Model: Support for Native AOT

A limitation of Native AOT is support for nested .NET types. In some cases, these nested types might go unnoticed by the trimming component of the .NET compiler. In this case, you might receive an exception such as: "`System.InvalidOperationException: Type <type> is unsupported, it cannot be instantiated.`"

You can work around this limitation by adding the `DynamicDependency` somewhere in the code path that informs the trimmer about the dependency on the sub-type. The constructor of the top-level .NET type being saved is a likely place. The following code example shows you how to use the `DynamicDependency` attribute:

```
[DynamoDBTable("TestTable")]
class TypeWithNestedTypeProperty
{
    [DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(SubType))]
    public TypeWithNestedTypeProperty()
    {

    }

    [DynamoDBHashKey]
    public string Id { get; set; }
    public string Name { get; set; }

    public SubType SubType { get; set; }
}

class SubType
{
    public string SubName { get; set; }
}
```

DynamoDBStreams

DynamoDBStreams has been removed from the AWSSDK.DynamoDB NuGet package and is available in its own package, AWSSDK.DynamoDBStreams, and has its own namespace, `Amazon.DynamoDBStreams`.

Allow removal of the `TableNamePrefix` value

You can now remove the value of the `TableNamePrefix` property in the DynamoDBContextConfig class on the individual operation level. For more information about this change, see PR 3476 on GitHub.

`RetrieveDateTimeInUtc` property

For the DynamoDBContextConfig class, the default value of the `RetrieveDateTimeInUtc` property has been changed to `true`.

`DynamoDBContextTableNamePrefix` property

Removed the `DynamoDBContextTableNamePrefix` property from AWSConfigsDynamoDB class. Users should call `AWSConfigsDynamoDB.Context.TableNamePrefix` instead of

Changes specific to EC2

The following changes are specific to Amazon EC2. Most or all of them are breaking changes.

Open to view items

`GetDecryptedPassword`

The `GetDecryptedPassword` extension for Amazon EC2 has been moved to a separate extension package called AWSSDK.Extensions.EC2.DecryptPassword. This change is to support OpenSSL 3 and take a dependency on BouncyCastle.Cryptography.

The source code for this package is on GitHub at https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.EC2.DecryptPassword.

Support for Amazon EC2 IMDSv1

Support for Instance Metadata Service Version 1 (IMDSv1) has been removed. V4 of the SDK always uses Instance Metadata Service Version 2 (IMDSv2) when fetching credentials and other metadata from the IMDS. For more information about the IMDS, see Use the IMDS in the Amazon EC2 User Guide.

Programming elements that were changed or removed

- The entire `Amazon.EC2.Import` namespace and code have been removed.

- The entire `Amazon.EC2.Util` namespace and code have been removed, which includes the AMI utilities that were used to look up EC2 AMIs for Windows.
- The obsolete `IpRanges` property has been removed from the IpPermission class. Use the `Ipv4Ranges` or `Ipv6Ranges` properties instead.
- The following obsolete fields have been removed from the EC2InstanceMetadata class: `EC2_METADATA_SVC`, `EC2_METADATA_ROOT`, `EC2_USERDATA_ROOT`, `EC2_DYNAMICDATA_ROOT`, and `EC2_APITOKEN_URL`.

Changes specific to S3

The following changes are specific to Amazon S3. Most or all of them are breaking changes.

Open to view items

AWS Region us-east-1

Amazon S3 service clients configured for the `us-east-1` Region can no longer access buckets in other Regions. Buckets must be accessed with S3 service clients configured for the Region the bucket is in.

For additional information about this change, see the blog post Preview 4 of AWS SDK for .NET V4.

S3 encryption client

The Amazon S3 encryption client, which is defined in the `Amazon.S3.Encryption` namespace, has been removed from the AWSSDK.S3 package. This client has been moved to its own package called Amazon.Extensions.S3.Encryption, and the documentation for it is at https://aws.github.io/amazon-s3-encryption-client-dotnet/api/Amazon.Extensions.S3.Encryption.html. For information about migration, see the section called "S3 Encryption Client Migration (V1 to V2)". For more information about S3 encryption, see Supported encryption algorithms in the Amazon S3 Encryption Client Developer Guide.

S3 tagging directive for `CopyObject`

The `TaggingDirective` property has been exposed as a public property of the CopyObjectRequest class, which is used by `AmazonS3Client.CopyObject` methods. This property corresponds to the Amazon S3 `x-amz-tagging-directive` parameter, as defined in the CopyObject action.

The tagging directive is no longer automatically set to COPY. If a developer doesn't specify a tagging directive, the S3 backend automatically assumes it is COPY, but if a developer explicitly sets the property to null, then the value isn't set at all.

The `UseArnRegion` property for S3 configuration

The `UseArnRegion` property of the Amazon.S3.AmazonS3Config class has been updated such that the `AWS_S3_USE_ARN_REGION` environment variable takes precedence over the `s3_use_arn_region` setting in the shared AWS `config` file. For more information about these variables and settings, see Settings reference in the AWS SDKs and Tools Reference Guide.

Leading slashes for the `CopyObject` and `CopyPart` methods

Leading slashes will no longer be trimmed for the Amazon S3 `CopyObject` and `CopyPart` methods. The `DisableTrimmingLeadingSlash` property has been removed from the CopyObjectRequest and CopyPartRequest classes.

The `DoesS3BucketExist...` methods

The obsolete `DoesS3BucketExist` and `DoesS3BucketExistAsync` methods have been removed from the AmazonS3Util class, which implements the ICoreAmazonS3 interface. These methods were removed because they always use HTTP. Use DoesS3BucketExistV2 and DoesS3BucketExistV2Async instead.

SDK always uses SigV4

Version 4 of the AWS SDK for .NET always uses AWS Signature Version 4 (SigV4) for signing requests. This change results in the following related changes:

- The `UseSignatureVersion4` property of the AWSConfigsS3 class has been removed.
- The `SignatureVersion` property of the Amazon.Runtime.ClientConfig class has been removed. This property was used only by Amazon S3 for backward compatibility.
- The `RegionEndpoint.Endpoint` class has been removed. This includes the `SignatureVersionOverride` property, which was used to override signature versions for Amazon S3. Use the service-specific `client.DetermineServiceOperationEndPoint()` method instead.
- Updated methods AmazonS3Util.PostUpload and S3PostUploadSignedPolicy.GetSignedPolicy to use SigV4. As a consequence, the `S3PostUploadSignedPolicy.GetSignedPolicyV4`

method was removed because `GetSignedPolicy` now performs the same function. In addition, `GetSignedPolicy` has been given a third parameter for Region endpoint.

The GetACL and PutACL methods

The `GetACL` and `PutACL` methods of the AmazonS3Client class have been marked as obsolete. To access the functionality of these methods, use the following new methods instead: `GetBucketACL`, `PutBucketACL`, `GetObjectACL`, and `PutObjectACL`.

Obsolete programming elements removed

A number of programming elements of the Amazon S3 implementation were removed from V4 of the SDK, including enumeration values, types, methods, namespaces, etc. These are listed below, if not already covered previously, along with potential steps that you can take to accommodate their removal.

- The `DisableMD5Stream` property has been removed from the TransferUtilityUploadRequest class. Use the `DisableDefaultChecksumValidation` property instead.

  In addition, the `CalculateContentMD5Header` property has been removed from the `TransferUtilityUploadRequest` class. This property is no longer needed because the SDK computes a checksum by default.
- The `ServerSideEncryptionMethod` and `ServerSideEncryptionKeyManagementServiceKeyId` properties have been removed from the CopyPartRequest class. Use the properties with the same names in the InitiateMultipartUploadRequest class instead, which is used in some of the `InitiateMultipartUpload...` methods of the AmazonS3Client class.
- The `Expires` property has been removed from the GetObjectResponse class. Use the `ExpiresString` property instead. The string might not be in a valid timestamp format, so your code should use the `TryParse` method when converting to a `DateTime`.
- Obsolete AWS Region identifiers have been removed from the S3Region enumeration.
- The `Prefix` property has been removed from the LifecycleRule class. Use the `Filter` property instead.

  In addition, the `NoncurrentVersionTransition` and `Transition` properties have been removed from the `LifecycleRule` class. Use the `NoncurrentVersionTransitions` and `Transitions` collections instead.

- The `Event` property has been removed from the TopicConfiguration class. Use the `Events` collection instead.
- CalculateContentMD5Header property. This property no longer needed to be set because the SDK will compute a checksum by default.
- The `Bucket` property has been removed from the SelectObjectContentRequest class. Use the `BucketName` property instead.
- The `NumberOfUploadThreads` property has been removed from the TransferUtilityConfig class. Use `ConcurrentServiceRequests` property instead.

Programming elements that were removed

A number of programming elements were removed from V4 of the SDK, including enumeration values, types, methods, namespaces, etc. These are listed below, if not already covered previously, along with potential steps that you can take to accommodate their removal.

Open to view items

The `Amazon.Auth.AccessControlPolicy.ActionIdentifiers` namespace

The `Amazon.Auth.AccessControlPolicy.ActionIdentifiers` namespace has been removed. This includes IAM action identifiers, which were defined in the `IdentityandAccessManagementActionIdentifiers` class. Code that uses these action identifiers should be changed to use string values of the action name.

For additional information, see the section called “Creating managed policies from JSON” and Overview of JSON policies in the IAM User Guide.

The `ClientConfig` class

The Amazon.Runtime.ClientConfig class is the base class of service client configuration classes like AmazonS3Config. The following programming elements have been removed from this class.

- The `DetermineServiceURL` and `DetermineDnsSuffix` methods have been removed. Use the `DetermineServiceOperationEndpoint` method of the service client instead; for example, AmazonS3Client.DetermineServiceOperationEndpoint.
- The `ReadEntireResponse` property has been removed. Use one of the following instead:
  - The `LogResponses` property of the AWSConfigs.LoggingConfig class.

- The `LogResponse` property of the client configuration; for example, AmazonS3Config.

The `Amazon.Runtime` namespace

The Amazon.Runtime namespace was updated as follows:

- The obsolete `ECSTaskCredentials` class has been removed from the namespace. Use the GenericContainerCredentials provider instead, which also supports Amazon EKS Pod Identities.
- The obsolete `StoredProfileAWSCredentials` and `StoredProfileCredentials` classes have been removed from the namespace. Use the NetSDKCredentialsFile or the SharedCredentialsFile class of the Amazon.Runtime.CredentialManagement namespace instead.
- The obsolete `HasCachedAccessTokenAvailable` method of the SSOAWSCredentials class has been removed from the namespace.
- The obsolete `EnvironmentAWSCredentials` class has been removed from the namespace. Use the AppConfigAWSCredentials class instead.
- The obsolete `StoredProfileFederatedCredentials` class has been removed from the namespace. Use the FederatedAWSCredentials class instead.
- The following obsolete classes have been removed from the namespace: `EnvironmentVariableAWSEndpointDiscoveryEnabled`, `ProfileAWSEndpointDiscoveryEnabled`, and `FallbackEndpointDiscoveryEnabledFactory`.
- The obsolete `UseSigV4` property has been removed from the AmazonWebServiceRequest class. Use the `SignatureVersion` property instead.
- The `ProfileIniFile` class in the `Amazon.Runtime.Internal.Util` namespace has an overloaded method called `TryGetSection`. The versions of the method that don't support the `out` parameter for `nestedProperties` have been removed from the class.
- The obsolete `EventBridgeSigner` class in the `Amazon.Runtime.Internal.Auth` namespace has been removed.
- The obsolete `Parameters` dictionary property has been removed from the WebServiceRequestEventArgs class. Use the `ParameteCollection` property instead.

BouncyCastle

The source copy of BouncyCastle has been removed from V4 of the SDK.

The `StoredProfileSAMLCredentials` class

The obsolete `StoredProfileSAMLCredentials` class in the Amazon.SecurityToken.SAML namespace has been removed. Use the FederatedAWSCredentials class in the Amazon.Runtime namespace instead.

The `AWSSDKUtils` class

The following methods have been removed from the AWSSDKUtils class: `ResolveResourcePath`, `ProtectEncodedSlashUrlEncode`, and `ConvertToUnixEpochMilliSeconds`.

The `ProfileManager` class

The obsolete `ProfileManager` class has been removed from the Amazon.Util namespace. Use the NetSDKCredentialsFile or SharedCredentialsFile class from the Amazon.Runtime.CredentialManagement namespace instead.

The `AWSConfigs` class

The following obsolete properties have been removed from the AWSConfigs class: `Logging`, `ResponseLogging`, and `LogMetrics`. Use the `LoggingConfig` property instead.

The `ConditionFactory` class

The method with the following signature has been removed from the ConditionFactory class: `NewCondition(ConditionFactory.DateComparisonType, DateTime)`. Use the NewConditionUtc method instead.

Amazon CloudFront utilities

The obsolete `Amazon.CloudFront.Util` namespace and `AmazonCloudFrontUtil` class have been removed.

AWS IoT

In the ListPrincipalThingsResponse class, a legacy customization for a `NextToken` override has been removed in favor of pagination.

AWS Lambda

The following `Invoke...` methods of the AmazonLambdaClient class have been removed because the names were confusing.

- The V3 method with the following signature has been removed: `InvokeAsyncResponse InvokeAsync(InvokeAsyncRequest)`. This is a synchronous method in V3 of the SDK. Use `InvokeResponse Invoke(InvokeRequest)` (for synchronous processing) or `Task InvokeAsync(InvokeRequest, CancellationToken)` (for asynchronous processing) instead.
- The V3 method with the following signature has been removed: `Task InvokeAsyncAsync(InvokeAsyncRequest, CancellationToken)`. This is an asynchronous method in V3 of the SDK. Use `Task InvokeAsync(InvokeRequest, CancellationToken)` instead.

Amazon SageMaker Runtime

Obsolete constructors for the PayloadPart class have been removed.

Migrating from .NET Standard 1.3

On June 27 2019 Microsoft ended support for .NET Core 1.0 and .NET Core 1.1 versions. Following this announcement, AWS ended support for .NET Standard 1.3 on the AWS SDK for .NET on December 31, 2020.

AWS continued to provide service updates and security fixes on the AWS SDK for .NET targeting .NET Standard 1.3 until October 1, 2020. After that date, the .NET Standard 1.3 target went into Maintenance mode, which meant that no new updates were released; AWS applied critical bug fixes and security patches only.

On December 31, 2020, support for .NET Standard 1.3 on the AWS SDK for .NET came to its end of life. After that date no bug fixes or security patches were applied. Artifacts built with that target remain available for download on NuGet.

What you need to do

- If you're running applications using .NET Framework, you're not affected.
- If you're running applications using .NET Core 2.0 or higher, you're not affected.
- If you're running applications using .NET Core 1.0 or .NET Core 1.1, migrate your applications to a newer version of .NET Core by following Microsoft migration instructions. We recommend a minimum of .NET Core 3.1.

- If you're running business critical applications that cannot be upgraded at this time, you can continue using your current version of AWS SDK for .NET.

If you have questions or concerns, contact AWS Support.

Security for this AWS Product or Service

Cloud security at Amazon Web Services (AWS) is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations. Security is a shared responsibility between AWS and you. The Shared Responsibility Model describes this as Security of the Cloud and Security in the Cloud.

Security of the Cloud – AWS is responsible for protecting the infrastructure that runs all of the services offered in the AWS Cloud and providing you with services that you can use securely. Our security responsibility is the highest priority at AWS, and the effectiveness of our security is regularly tested and verified by third-party auditors as part of the AWS Compliance Programs.

Security in the Cloud – Your responsibility is determined by the AWS service you are using, and other factors including the sensitivity of your data, your organization's requirements, and applicable laws and regulations.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Topics

- Data protection in this AWS Product or Service
- Identity and Access Management
- Compliance Validation for this AWS Product or Service
- Resilience for this AWS Product or Service
- Infrastructure Security for this AWS Product or Service
- Enforcing a minimum TLS version in the AWS SDK for .NET
- S3 Encryption Client Migration (V1 to V2)
- S3 Encryption Client Migration (V2 to V4)

Data protection in this AWS Product or Service

The AWS shared responsibility model applies to data protection in this AWS product or service. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on

this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ. For information about data protection in Europe, see the AWS Shared Responsibility Model and GDPR blog post on the *AWS Security Blog*.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with this AWS product or service or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Identity and Access Management

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use AWS resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How AWS services work with IAM
- Troubleshooting AWS identity and access

Audience

How you use AWS Identity and Access Management (IAM) differs, depending on the work that you do in AWS.

Service user – If you use AWS services to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more AWS features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in AWS, see Troubleshooting AWS identity and access or the user guide of the AWS service you are using.

Service administrator – If you're in charge of AWS resources at your company, you probably have full access to AWS. It's your job to determine which AWS features and resources your service users should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with AWS, see the user guide of the AWS service you are using.

IAM administrator – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to AWS. To view example AWS identity-based policies that you can use in IAM, see the user guide of the AWS service you are using.

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook

credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Access control lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, AWS WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see Access control list (ACL) overview in the *Amazon Simple Storage Service Developer Guide*.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How AWS services work with IAM

To get a high-level view of how AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

To learn how to use a specific AWS service with IAM, see the security section of the relevant service's User Guide.

Troubleshooting AWS identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with AWS and IAM.

Topics

- I am not authorized to perform an action in AWS
- I am not authorized to perform iam:PassRole
- I want to allow people outside of my AWS account to access my AWS resources

I am not authorized to perform an action in AWS

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `awes:`*`GetWidget`* permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `awes:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to AWS.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in AWS. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my AWS resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether AWS supports these features, see How AWS services work with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Compliance Validation for this AWS Product or Service

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Resilience for this AWS Product or Service

The AWS global infrastructure is built around AWS Regions and Availability Zones.

AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking.

With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS Global Infrastructure.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Infrastructure Security for this AWS Product or Service

This AWS product or service uses managed services, and therefore is protected by the AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access this AWS Product or Service through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the AWS Security Token Service (AWS STS) to generate temporary security credentials to sign requests.

This AWS product or service follows the shared responsibility model through the specific Amazon Web Services (AWS) services it supports. For AWS service security information, see the AWS service security documentation page and AWS services that are in scope of AWS compliance efforts by compliance program.

Enforcing a minimum TLS version in the AWS SDK for .NET

To increase security when communicating with AWS services, you should configure the AWS SDK for .NET to use TLS 1.2 or later.

The AWS SDK for .NET uses the underlying .NET runtime to determine which security protocol to use. By default, current versions of .NET use the latest configured protocol that the operating system supports. Your application can override this SDK behavior, but it's *not recommended* to do so.

.NET Core

By default, .NET Core uses the latest configured protocol that the operating system supports. The AWS SDK for .NET doesn't provide a mechanism to override this.

If you're using a .NET Core version earlier than 2.1, we *strongly* recommend you upgrade your .NET Core version.

See the following for information specific to each operating system.

Windows

Modern distributions of Windows have TLS 1.2 support enabled by default. If you're running on Windows 7 SP1 or Windows Server 2008 R2 SP1, you need to ensure that TLS 1.2 support is enabled in the registry, as described at https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12. If you're running an earlier distribution, you must upgrade your operating system. For information about TLS 1.3 support in Windows, check the latest Microsoft documentation for the minimum required client or server versions.

macOS

If you're running .NET Core 2.1 or later, TLS 1.2 is enabled by default. TLS 1.2 is supported by OS X Mavericks v10.9 or later. .NET Core version 2.1 and later require newer versions of macOS, as described at https://learn.microsoft.com/en-us/dotnet/core/install/windows?tabs=net80&pivots=os-macos.

If you're using .NET Core 1.0, .NET Core uses OpenSSL on macOS, a dependency that must be installed separately. OpenSSL added support for TLS 1.2 in version 1.0.1, and added support for TLS 1.3 in version 1.1.1.

Linux

.NET Core on Linux requires OpenSSL, which comes bundled with many Linux distributions. But it can also be installed separately. OpenSSL added support for TLS 1.2 in version 1.0.1, and added support for TLS 1.3 in version 1.1.1. If you're using a modern version of .NET Core (2.1 or later) and have installed a package manager, it's likely that a more modern version of OpenSSL was installed for you.

To be sure, you can run **`openssl version`** in a terminal and verify that the version is later than 1.0.1.

.NET Framework

If you're running a modern version of .NET Framework (4.7 or later) and a modern version of Windows (at least Windows 8 for clients, Windows Server 2012 or later for servers), TLS 1.2 is enabled and used by default.

If you're using a .NET Framework runtime that doesn't use the operating system settings (.NET Framework 3.5 through 4.5.2), the AWS SDK for .NET will attempt to add support for TLS 1.1 and TLS 1.2 to the supported protocols. If you're using .NET Framework 3.5, this will be successful only if the appropriate hot patch is installed, as follows:

- Windows 10 version 1511 and Windows Server 2016 – KB3156421
- Windows 8.1 and Windows Server 2012 R2 – KB3154520
- Windows Server 2012 – KB3154519
- Windows 7 SP1 and Server 2008 R2 SP1 – KB3154518

> **Warning**
>
> On August 15th, 2024, the AWS SDK for .NET ended support for .NET Framework 3.5 and changed the minimum .NET Framework version to 4.7.2. For more information, see the blog post Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK for .NET.

If your application is running on a newer .NET Framework on Windows 7 SP1 or Windows Server 2008 R2 SP1, you need to ensure that TLS 1.2 support is enabled in the registry, as described at https://learn.microsoft.com/en-us/windows-server/security/tls/tls-registry-settings#tls-12. Newer versions of Windows have it enabled by default.

For detailed best practices for using TLS with .NET Framework, see the Microsoft article at https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls.

AWS Tools for PowerShell

AWS Tools for PowerShell use the AWS SDK for .NET for all calls to AWS services. The behavior of your environment depends on the version of Windows PowerShell you're running, as follows.

Windows PowerShell 2.0 through 5.x

Windows PowerShell 2.0 through 5.x run on .NET Framework. You can verify which .NET runtime (2.0 or 4.0) is being used by PowerShell by using the following command.

```
$PSVersionTable.CLRVersion
```

- When using .NET Runtime 2.0, follow the instructions provided earlier regarding the AWS SDK for .NET and .NET Framework 3.5.

> **Warning**
>
> On August 15th, 2024, the AWS SDK for .NET ended support for .NET Framework 3.5 and changed the minimum .NET Framework version to 4.7.2. For more information, see the blog post Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK for .NET.

- When using .NET Runtime 4.0, follow the instructions provided earlier regarding the AWS SDK for .NET and .NET Framework 4+.

Windows PowerShell 6.0

Windows PowerShell 6.0 and newer run on .NET Core. You can verify which version of .NET Core is being used by running the following command.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning
 $true).FrameworkName
```

Follow the instructions provided earlier regarding the AWS SDK for .NET and the relevant version of .NET Core.

Xamarin

For Xamarin, see the directions at https://learn.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/transport-layer-security. In summary:

For Android

- Requires Android 5.0 or later.
- **Project Properties**, **Android Options**: HttpClient implementation must be set to **Android** and the SSL/TLS implementation set to **Native TLS 1.2+**.

For iOS

- Requires iOS 7 or later.
- **Project Properties**, **iOS Build**: HttpClient implementation must be set to **NSUrlSession**.

For macOS

- Requires macOS 10.9 or later.
- **Project Options**, **Build**, **Mac Build**: HttpClient implementation must be set to **NSUrlSession**.

Unity

You must use Unity 2018.2 or later, and use the .NET 4.x Equivalent scripting runtime. You can set this in **Project Settings**, **Configuration**, **Player**, as described at https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html. The .NET 4.x Equivalent scripting runtime enables TLS 1.2 support to all Unity platforms running Mono or IL2CPP.

Browser (for Blazor WebAssembly)

WebAssembly runs in the browser instead of on the server, and uses the browser for handling HTTP traffic. Therefore, TLS support is determined by browser support.

Blazor WebAssembly, in preview for ASP.NET Core 3.1, is supported only in browsers that support WebAssembly, as described at https://learn.microsoft.com/en-us/aspnet/core/blazor/supported-platforms. All mainstream browsers supported TLS 1.2 before supporting WebAssembly. If this is the case for your browser, then if your app runs, it can communicate over TLS 1.2.

See your browser's documentation for more information and verification.

S3 Encryption Client Migration (V1 to V2)

> **Note**
>
> If you are using V2 and want to migrate to V4, see S3 Encryption Client Migration (V2 to V4).

This topic shows how to migrate your applications from Version 1 (V1) of the Amazon Simple Storage Service (S3) encryption client to Version 2 (V2), and ensure application availability throughout the migration process.

Objects that are encrypted with the V2 client can't be decrypted with the V1 client. In order to ease migration to the new client without having to re-encrypt all objects at once, a "V1-transitional" client has been provided. This client can *decrypt* both V1- and V2-encrypted objects, but *encrypts*

objects only in V1-compatible format. The V2 client can *decrypt* both V1- and V2-encrypted objects (when enabled for V1 objects), but *encrypts* objects only in V2-compatible format.

Migration Overview

This migration happens in three phases. These phases are introduced here and described in detail later. Each phase must be completed for *all* clients that use shared objects before the next phase is started.

1. **Update existing clients to V1-transitional clients to read new formats.** First, update your applications to take a dependency on the V1-transitional client instead of the V1 client. The V1-transitional client enables your existing code to decrypt objects written by the new V2 clients and objects written in V1-compatible format.

   > **Note**
   >
   > The V1-transitional client is provided for migration purposes only. Proceed to upgrading to the V2 client after moving to the V1-transitional client.

2. **Migrate V1-transitional clients to V2 clients to write new formats.** Next, replace all V1-transitional clients in your applications with V2 clients, and set the security profile to `V2AndLegacy`. Setting this security profile on V2 clients enables those clients to decrypt objects that were encrypted in V1-compatible format.
3. **Update V2 clients to no longer read V1 formats.** Finally, after all clients have been migrated to V2 and all objects have been encrypted or re-encrypted in V2-compatible format, set the V2 security profile to `V2` instead of `V2AndLegacy`. This prevents the decryption of objects that are in V1-compatible format.

Update Existing Clients to V1-transitional Clients to Read New Formats

The V2 encryption client uses encryption algorithms that older versions of the client don't support. The first step in the migration is to update your V1 decryption clients so that they can read the new format.

The V1-transitional client enables your applications to decrypt both V1- and V2-encrypted objects. This client is a part of the Amazon.Extensions.S3.Encryption NuGet package. Perform the following steps on each of your applications to use the V1-transitional client.

1. Take a new dependency on the Amazon.Extensions.S3.Encryption package. If your project depends directly on the **AWSSDK.S3** or **AWSSDK.KeyManagementService** packages, you must either update those dependencies or remove them so that their updated versions will be pulled in with this new package.
2. Change the appropriate `using` statement from `Amazon.S3.Encryption` to `Amazon.Extensions.S3.Encryption`, as follows:

```
// using Amazon.S3.Encryption;
  using Amazon.Extensions.S3.Encryption;
```

3. Rebuild and redeploy your application.

The V1-transitional client is fully API-compatible with the V1 client, so no other code changes are required.

Migrate V1-transitional Clients to V2 Clients to Write New Formats

The V2 client is a part of the Amazon.Extensions.S3.Encryption NuGet package. It enables your applications to decrypt both V1- and V2-encrypted objects (if configured to do so), but encrypts objects only in V2-compatible format.

After updating your existing clients to read the new encryption format, you can proceed to safely update your applications to the V2 encryption and decryption clients. Perform the following steps on each of your applications to use the V2 client:

1. Change `EncryptionMaterials` to `EncryptionMaterialsV2`.
    a. When using KMS:
        i. Provide a KMS key ID.
        ii. Declare the encryption method that you are using; that is, `KmsType.KmsContext`.
        iii. Provide an encryption context to KMS to associate with this data key. You can send an empty dictionary (Amazon encryption context will still be merged in), but providing additional context is encouraged.
    b. When using user-provided key wrap methods (symmetric or asymmetric encryption):
        i. Provide an AES or an RSA instance that contains the encryption materials.

   ii. Declare which encryption algorithm to use; that is, `SymmetricAlgorithmType.AesGcm` or `AsymmetricAlgorithmType.RsaOaepSha1`.

2. Change `AmazonS3CryptoConfiguration` to `AmazonS3CryptoConfigurationV2` with the `SecurityProfile` property set to `SecurityProfile.V2AndLegacy`.
3. Change `AmazonS3EncryptionClient` to `AmazonS3EncryptionClientV2`. This client takes the newly converted `AmazonS3CryptoConfigurationV2` and `EncryptionMaterialsV2` objects from the previous steps.

Example: KMS to KMS+Context

Pre-migration

```
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);
```

Post-migration

```
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);
```

Example: Symmetric Algorithm (AES-CBC to AES-GCM Key Wrap)

`StorageMode` can be either `ObjectMetadata` or `InstructionFile`.

Pre-migration

```
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);
```

Post-migration

```
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

When decrypting with AES-GCM, read the entire object to the end before you start using the decrypted data. This is to verify that the object hasn't been modified since it was encrypted.

Example: Asymmetric Algorithm (RSA to RSA-OAEP-SHA1 Key Wrap)

`StorageMode` can be either `ObjectMetadata` or `InstructionFile`.

Pre-migration

```
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);
```

Post-migration

```
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);
```

Update V2 Clients to No Longer Read V1 Formats

Eventually, all objects will have been encrypted or re-encrypted using a V2 client. *After this conversion is complete*, you can disable V1 compatibility in the V2 clients by setting the `SecurityProfile` property to `SecurityProfile.V2`, as shown in the following snippet.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

S3 Encryption Client Migration (V2 to V4)

> **Note**
>
> If you are using V1 and want to migrate to V4, you must first migrate to V2. See S3 Encryption Client Migration (V1 to V2).

This topic shows how to migrate your applications from Version 2 (V2) of the Amazon Simple Storage Service (S3) encryption client to Version 4 (V4), and ensure application availability throughout the migration process. V4 uses AES-GCM with key commitment for content encryption and introduces Commitment Policies to enhance security against key substitution attacks.

The V4 client is available in the Amazon.Extensions.S3.Encryption NuGet package.

> **Important**
>
> **Breaking Changes:** V4 clients configured with `RequireEncryptRequireDecrypt` policy cannot decrypt objects encrypted with V1 or V2 clients. Only the latest V2 clients can decrypt V4-encrypted objects with key commitment. Before moving to the `RequireEncryptRequireDecrypt` policy, you must re-encrypt all your existing data using V4 clients with key commitment enabled.

Understanding V4 Concepts

V4 uses AES-GCM with key commitment for content encryption and introduces key security concepts that enhance the protection of your encrypted data:

Commitment Policy

Commitment Policy controls how the encryption client handles key commitment during encryption and decryption operations. V4 supports three commitment policies:

`ForbidEncryptAllowDecrypt`

Encryption: Without commitment

Decryption: Allows non-committing objects

Security: Does not enforce commitment and may allow tampering

Compatibility: All V2 and V4 implementations can read objects encrypted with this policy

`RequireEncryptAllowDecrypt`

Encryption: With key commitment

Decryption: Allows both committing and non-committing objects

Security: New objects are protected against key substitution attacks, old objects still readable

Compatibility: Only V4 supports this policy

`RequireEncryptRequireDecrypt`(Default for V4)

Encryption: With key commitment

Decryption: Only committing objects

Security: Full commitment enforcement for maximum security

Compatibility: Only V4 supports this policy

AES GCM with Key Commitment

V4 uses AES-GCM with key commitment for content encryption, which provides enhanced security:

- *Tampering Protection:* It protects against key substitution attacks by cryptographically binding the key to the encrypted data.
- *Version Compatibility:* Objects encrypted with key commitment can only be decrypted by V4 clients and newer versions.

> **Warning**
>
> Before enabling key commitment encryption in production, ensure that all applications that need to decrypt your objects have been upgraded to V4 or newer as V2 client is being deprecated.

Update Existing Clients to Read V4 Formats

The V4 encryption client uses encryption algorithms that older versions of the client don't support. The first step in the migration is to update your V2 clients so that they can read the new V4 format.

Update NuGet Package Dependencies

Update your applications to use the latest version of the Amazon.Extensions.S3.Encryption NuGet package that includes V4 support. Perform the following steps on each of your applications:

1. Update to the latest Amazon.Extensions.S3.Encryption package. If your project depends directly on the **AWSSDK.S3** or **AWSSDK.KeyManagementService** packages, you must either update those dependencies or remove them so that their updated versions will be pulled in with this new package.
2. Ensure your `using` statements reference the correct namespace:

   ```
   using Amazon.Extensions.S3.Encryption;
   using Amazon.Extensions.S3.Encryption.Primitives;
   ```

3. Rebuild and redeploy your application.

Your existing V2 clients will continue to work with the updated package and will be able to decrypt objects encrypted by V4 clients (depending on the commitment policy used).

Build and Deploy Applications

After updating your NuGet package dependencies:

1. Build your application to ensure all dependencies are resolved correctly.
2. Test your application in a development environment to verify that existing functionality continues to work.
3. Deploy the updated application to your production environment.

This update enables your existing V2 clients to decrypt objects that will be encrypted by V4 clients, ensuring compatibility during the migration process.

Migrate to V4 Clients

After updating your existing clients to read the new encryption format, you can proceed to safely update your applications to use V4 encryption and decryption clients. The V4 client provides enhanced security through key commitment while maintaining compatibility with existing encrypted objects.

4-Step Migration Process

The migration from V2 to V4 follows a structured 4-step process to ensure compatibility and security. Each step represents a specific configuration that should be deployed across all your applications before proceeding to the next step.

1. **Step 0: V2 Client (Starting Point)** - Your existing V2 implementation
2. **Step 1: V4 with V2 Compatibility** - Migrate to V4 client while maintaining V2-compatible encryption behavior
3. **Step 2: V4 with Key Commitment Writes** - Start encrypting with key commitment while allowing decryption of legacy objects
4. **Step 3: V4 with Full Enforcement** - Require key commitment for both encryption and decryption

Step 0: V2 Client (Starting Point)

This represents your existing V2 client configuration. This step demonstrates the starting state before migration.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 0: V2 Client - Starting configuration
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV2(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

#pragma warning disable 0618
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
#pragma warning enable 0618
```

```
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Step 1: V4 with V2 Compatibility

Migrate to the V4 client while maintaining identical behavior to V2. This step uses `ForbidEncryptAllowDecrypt` policy to encrypt without commitment and allow decryption of all objects.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 1: V4 Client with V2 compatibility
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.ForbidEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcm);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Behavior: Encrypts without commitment, can decrypt both committing and non-committing objects. Identical to V2 behavior.

Step 2: V4 with Key Commitment Writes

Start encrypting with key commitment while maintaining backward compatibility for decryption. This step uses `RequireEncryptAllowDecrypt` policy.

> **⚠ Warning**
>
> Before deploying Step 2, ensure all readers have been updated to Step 1 or later to handle key commitment encryption.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 2: V4 Client with key commitment writes
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Behavior: Encrypts with commitment, can decrypt both committing and non-committing objects. New objects are protected against key substitution attacks.

Step 3: V4 with Full Enforcement

Require key commitment for both encryption and decryption. This step uses `RequireEncryptRequireDecrypt` policy for maximum security.

> **Warning**
>
> Before deploying Step 3, ensure all objects in your system have been re-encrypted with key commitment (Step 2). This step will fail to decrypt objects encrypted without commitment.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 3: V4 Client with full key commitment enforcement
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptRequireDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Behavior: Encrypts with commitment, only decrypts objects encrypted with commitment. Maximum security against key substitution attacks.

Additional Configuration Examples

This section provides additional examples for configuring V4 clients with different options during migration.

Enabling Legacy Support

To enable V4 clients to read objects encrypted by V1 and V2 clients, configure the client with a commitment policy that allows legacy decryption:

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

// Configure V4 client to read V1/V2 objects
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4AndLegacy,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};

// This configuration allows:
// - Encryption: With commitment (secure)
// - Decryption: Both V2 (non-committing) and V4 (committing) objects
```

Use this configuration during migration when you need to decrypt objects encrypted by older clients while ensuring new objects are encrypted with enhanced security.

Configuring Storage Method

V4 supports two storage methods for encryption metadata. Choose the method that best fits your use case:

Object Metadata (Default)

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
// Encryption metadata is stored in S3 object metadata
```

Instruction File

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.InstructionFile
};
// Encryption metadata is stored in a separate S3 object (instruction file)
```

Use `InstructionFile` when you need to preserve object metadata for other purposes or when working with objects that have metadata size limitations.

Configuring Commitment Policy

Choose the appropriate commitment policy based on your security requirements and migration phase:

Migration Phase (V2 Compatibility)

```
// For migration: encrypt without commitment, allow all decryption
var migrationConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.ForbidEncryptAllowDecrypt);
```

Transition Phase (Recommended)

```
// For transition: encrypt with commitment, allow legacy decryption
var transitionConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt);
```

Full Security Phase

```
// For maximum security: require commitment for both encryption and decryption
var secureConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptRequireDecrypt);
```

Start with `ForbidEncryptAllowDecrypt` during initial migration, move to `RequireEncryptAllowDecrypt` for the transition phase, and finally use `RequireEncryptRequireDecrypt` when all clients have been upgraded and all objects have been re-encrypted with commitment.

Special considerations for the AWS SDK for .NET

This section contains considerations for special cases where the normal configurations or procedures aren't appropriate or sufficient.

Topics

- Obtaining assemblies for the AWS SDK for .NET
- Accessing credentials and profiles in an application
- Special considerations for Unity support
- Special considerations for Xamarin support

Obtaining assemblies for the AWS SDK for .NET

This topic describes how you can obtain the AWSSDK assemblies and store them locally (or on premises) for use in your projects. This is **not** the recommended method for handling SDK references, but is required in some environments.

> **Note**
>
> The recommended method for handling SDK references is to download and install just the NuGet packages that each project needs. That method is described in Install AWSSDK packages with NuGet.

If you can't or aren't allowed to download and install NuGet packages on a per-project basis, the following options are available to you.

Download and extract ZIP files

(Remember that this isn't the recommended method for handling references to the AWS SDK for .NET.)

1. Download one of the following ZIP files:

   - aws-sdk-net8.0.zip - Assemblies that support .NET 8 and later.
   - aws-sdk-netcoreapp3.1.zip - Assemblies that support .NET Core 3.1 and later.

- aws-sdk-netstandard2.0.zip - Assemblies that support .NET Standard 2.0 and 2.1.
- aws-sdk-net472.zip - Assemblies that support .NET Framework 4.5 and later.

2. Extract the assemblies to some "download" folder on your file system; it doesn't matter where. Make note of this folder.
3. When you set up your project, you get the required assemblies from this folder, as described in Install AWSSDK assemblies without NuGet.

Accessing credentials and profiles in an application

The preferred method for using credentials is to allow the AWS SDK for .NET to find and retrieve them for you, as described in Credential and profile resolution.

However, you can also configure your application to actively retrieve profiles and credentials, and then explicitly use those credentials when creating an AWS service client.

To actively retrieve profiles and credentials, use classes from the Amazon.Runtime.CredentialManagement namespace.

- To find a profile in a file that uses the AWS credentials file format (either the shared AWS credentials file in its default location or a custom credentials file), use the SharedCredentialsFile class. Files in this format are sometimes simply called *credentials files* in this text for brevity.
- To find a profile in the SDK Store, use the NetSDKCredentialsFile class.
- To search in both a credentials file and the SDK Store, depending on the configuration of a class property, use the CredentialProfileStoreChain class.

  You can use this class to find profiles. You can also use this class to request AWS credentials directly instead of using the `AWSCredentialsFactory` class (described next).
- To retrieve or create various types of credentials from a profile, use the AWSCredentialsFactory class.

The following sections provide examples for these classes.

Examples for class CredentialProfileStoreChain

You can get credentials or profiles from the CredentialProfileStoreChain class by using the TryGetAWSCredentials or TryGetProfile methods. The `ProfilesLocation` property of the class determines the behavior of the methods, as follows:

- If `ProfilesLocation` is null or empty, search the SDK Store if the platform supports it, and then search the shared AWS credentials file in the default location.
- If the `ProfilesLocation` property contains a value, search the credentials file specified in the property.

Get credentials from the SDK Store or the shared AWS credentials file

This example shows you how to get credentials by using the `CredentialProfileStoreChain` class and then use the credentials to create an AmazonS3Client object. The credentials can come from the SDK Store or from the shared AWS credentials file at the default location.

This example also uses the Amazon.Runtime.AWSCredentials class.

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

Get a profile from the SDK Store or the shared AWS credentials file

This example shows you how to get a profile by using the CredentialProfileStoreChain class. The credentials can come from the SDK Store or from the shared AWS credentials file at the default location.

This example also uses the CredentialProfile class.

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Get credentials from a custom credentials file

This example shows you how to get credentials by using the CredentialProfileStoreChain class. The credentials come from a file that uses the AWS credentials file format but is at an alternate location.

This example also uses the Amazon.Runtime.AWSCredentials class.

```
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
}
```

Examples for classes SharedCredentialsFile and AWSCredentialsFactory

Create an AmazonS3Client by using the SharedCredentialsFile class

This examples shows you how to find a profile in the shared AWS credentials file, create AWS credentials from the profile, and then use the credentials to create an AmazonS3Client object. The example uses the SharedCredentialsFile class.

This example also uses the CredentialProfile class and the Amazon.Runtime.AWSCredentials class.

```
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**
>
> The NetSDKCredentialsFile class can be used in exactly the same way, except you would instantiate a new NetSDKCredentialsFile object instead of a SharedCredentialsFile object.

Special considerations for Unity support

When using the AWS SDK for .NET and .NET Standard 2.0 for your Unity application, your application must reference the AWS SDK for .NET assemblies (DLL files) directly rather than using NuGet. Given this requirement, the following are important actions you will need to perform.

- You need to obtain the AWS SDK for .NET assemblies and apply them to your project. For information about how to do this, see Download and extract ZIP files in the topic Obtaining AWSSDK assemblies.

- You need to include the following DLLs in your Unity project alongside the DLLs for **AWSSDK.Core** and the other AWS services you're using. Starting with version 3.5.109 of the AWS SDK for .NET, the .NET Standard ZIP file contains these additional DLLs.
  - Microsoft.Bcl.AsyncInterfaces.dll
  - System.Runtime.CompilerServices.Unsafe.dll
  - System.Threading.Tasks.Extensions.dll

- If you're using IL2CPP to build your Unity project, you must add a `link.xml` file to your Asset folder to prevent code stripping. The `link.xml` file must list all of the AWSSDK assemblies you are using, and each must include the `preserve="all"` attribute. The following snippet shows an example of this file.

```
<linker>
    <assembly fullname="AWSSDK.Core" preserve="all"/>
    <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
    <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

> **Note**
>
> To read interesting background information related to this requirement, see the article at https://aws.amazon.com/blogs/developer/referencing-the-aws-sdk-for-net-standard-2-0-from-unity-xamarin-or-uwp/.

Special considerations for Xamarin support

Xamarin projects (new and existing) must target .NET Standard 2.0. See .NET Standard 2.0 Support in Xamarin.Forms and .NET implementation support.

Also see the information about Portable Class Library and Xamarin.

API reference for the AWS SDK for .NET

The AWS SDK for .NET provides an API that you can use to access AWS services. To see what classes and methods are available in the API, see the AWS SDK for .NET API Reference.

In addition to the general reference given above, each of the examples under the Guided code examples section contains references to the specific methods and classes that are used in that example.

About API reference versions

The API reference described earlier is for version 3.0 and later of the AWS SDK for .NET.

For information about migrating from older versions of the SDK, see Migrating your project

To find deprecated content for earlier versions of the SDK API reference, see the following item(s):

- AWS SDK for .NET API Reference V1 (deprecated)
- AWS SDK for .NET API Reference V2 (deprecated)

To view a deprecated AWS SDK for .NET API reference, you will need to extract it and configure a web browser. The instructions shown next are examples of how to do this. They are based on using the Google Chrome web browser on a Windows system. Adapt them to your specific web browser and operating system.

View the deprecated API Reference V1

1. Download the ZIP file for the deprecated AWS SDK for .NET API Reference V1. By default, the name of the ZIP file is `sdkfornet-api-ref_v1_deprecated.zip`. That name will be used throughout these instructions.
2. Place the ZIP file in a folder of your choice. For these instructions, the folder name is assumed to be `C:\work\temp\api-refs\V1`.
3. Right-click on the ZIP file and choose **Extract All**. Accept the default location, which is `C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1` for these instructions.
4. Create a shortcut for Google Chrome in the `C:\work\temp\api-refs\V1` folder. Be careful not to move the original application.

5. In the properties of the new shortcut, set the following fields:

   - **Target**: "*<path to the Google Chrome application>*" --disable-web-security --user-data-dir=C:\work\temp\api-refs\V1\data "C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1\Index.html"

     For the --user-data-dir argument, use a folder name that works for your environment. The folder doesn't have to exist.
   - **Start in**: C:\work\temp\api-refs\V1
6. Give the shortcut an appropriate name.
7. Open the shortcut to view the old API reference.

View the deprecated API Reference V2

1. Download the ZIP file for the deprecated AWS SDK for .NET API Reference V2. By default, the name of the ZIP file is sdkfornet-api-ref_v2_deprecated.zip. That name will be used throughout these instructions.
2. Place the ZIP file in a folder of your choice. For these instructions, the folder name is assumed to be C:\work\temp\api-refs\V2.
3. Right-click on the ZIP file and choose **Extract All**. Accept the default location, which is C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2 for these instructions.
4. Create a shortcut for Google Chrome in the C:\work\temp\api-refs\V2 folder. Be careful not to move the original application.
5. In the properties of the new shortcut, set the following fields:

   - **Target**: "*<path to the Google Chrome application>*" --disable-web-security --user-data-dir=C:\work\temp\api-refs\V2\data "C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2\Index.html"

     For the --user-data-dir argument, use a folder name that works for your environment. The folder doesn't have to exist.
   - **Start in**: C:\work\temp\api-refs\V2
6. Give the shortcut an appropriate name.
7. Open the shortcut to view the old API reference.

Document history

The following table describes the important changes since the last release of the *AWS SDK for .NET Developer Guide*. For notification about updates to this documentation, you can subscribe to an RSS feed.

Change	Description	Date
What's new	Added a note about version 2.0 of the AWS Deploy Tool for .NET CLI.	October 17, 2025
AWS OpsWorks	AWS OpsWorks has reached End of Life. It has been removed from the documentation.	September 16, 2025
What's new	End-of-support has been announced for V3 of the AWS SDK for .NET.	August 21, 2025
What's new	General availability for the Distributed Cache Provider	July 2, 2025
Message Processing Framework	The AWS Message Processing Framework for .NET has reached general availability!	May 5, 2025
What's new	Version 4 of the AWS SDK for .NET is generally available! For information about migrating your applications to V4, see Migrating to version 4.	April 28, 2025
Migrating to version 4 of the AWS SDK for .NET	Included preview information about migraging applications built with the AWS SDK for .NET to version 4.	March 3, 2025

Integrating AWS with .NET Aspire in the AWS SDK for .NET	Included information about how AWS has been Integrated with .NET Aspire in the AWS SDK for .NET.	February 15, 2025
Observability	Announced the GA release for observability	February 10, 2025
What's new	Added information about new default behavior for integrity protection.	January 15, 2025
S3 Encryption Client V4 Migration	Added information about migrating from V2 to V4.	December 9, 2024
What's new	Added information about the fourth preview release of the AWS SDK for .NET version 4.	November 15, 2024