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

**AWS SDK for .NET Developer Guide** ......................................................... 1

What's in the SDK ................................................................................................. 1

How to Use This Guide .......................................................................................... 1

Supported Services and Revision History .............................................................. 2

Getting Started with the AWS SDK for .NET ...................................................... 3

Create an AWS Account and Credentials ............................................................. 3

  Signing Up for an AWS Account ......................................................................... 3
  Creating an IAM User ......................................................................................... 3

Install the .NET Development Environment ....................................................... 4

  Required ............................................................................................................ 4

Install AWSSDK Assemblies ................................................................................. 4

  Installing the AWS SDK for .NET ..................................................................... 5
  Installing AWSSDK Assemblies with NuGet ....................................................... 5

Start a New Project ............................................................................................... 6

Platforms Supported by the AWS SDK for .NET .................................................. 6

  .NET Framework 4.5 ....................................................................................... 6
  .NET Framework 3.5 ....................................................................................... 7
  .NET Core ....................................................................................................... 7

Platform Class Library .......................................................................................... 7

Unity Support ....................................................................................................... 7

More Info ............................................................................................................. 7

Programming with the AWS SDK for .NET ......................................................... 8

Configuring Your AWS SDK for .NET Application ............................................. 8

  Configuring the AWS SDK for .NET with .NET Core ...................................... 9
  Configuring AWS Credentials .......................................................................... 11
  AWS Region Selection ...................................................................................... 19
  Configuring Other Application Parameters .................................................... 21
  Configuration Files Reference for AWS SDK for .NET .................................... 26

Amazon Web Services Asynchronous APIs for .NET ........................................... 34

  Asynchronous API for .NET Framework 4.5, Windows Store, and Windows Phone 8 34
  Asynchronous API for .NET Framework 3.5 ...................................................... 34

Retries and Timeouts .......................................................................................... 40

  Retries ............................................................................................................. 41
  Timeouts ........................................................................................................ 41
  Example .......................................................................................................... 41

Migrating Your Code to Version 3 of the AWS SDK for .NET ............................. 42

  About the AWS SDK for .NET Versions .......................................................... 42
  Architecture Redesign for the SDK ................................................................. 42
  Breaking Changes ............................................................................................ 42

Code Examples .................................................................................................... 44

Listing AWS Resources using AWS CloudFormation ........................................... 44

Creating and Managing Encryption Keys Using Using AWS Key Management Service .................................................................................................................. 45

  AWS Key Management Service Customer Master Key (CMK) Examples .......... 46
  AWS Key Management Service Data Key Examples ....................................... 52
  AWS Key Management Service Alias Examples .......................................... 56
  AWS Key Management Service Grant Examples ......................................... 59
  AWS Key Management Service Policy Examples ......................................... 63
  AWS Key Management Service Key Material Examples .............................. 66
  AWS Key Management Service Tag Examples .............................................. 69

Using AWS KMS Keys with the AmazonS3EncryptionClient in the AWS SDK for .NET ................................................................. 72

Using Amazon DynamoDB NoSQL Databases .................................................... 73

  Low-Level Model ......................................................................................... 74
  Document Model ......................................................................................... 76
  Object Persistence Model ............................................................................. 77

AWS Region Selection .......................................................................................... 19

Configuring AWS Credentials ........................................................................... 11

Configuring the AWS SDK for .NET with .NET Core ........................................ 9

Configuring Other Application Parameters .................................................... 21

Configuration Files Reference for AWS SDK for .NET .................................... 26

Asynchronous API for .NET Framework 4.5, Windows Store, and Windows Phone 8 34

Asynchronous API for .NET Framework 3.5 ...................................................... 34

Retries ............................................................................................................. 41

Timeouts ........................................................................................................ 41

Example .......................................................................................................... 41

About the AWS SDK for .NET Versions .......................................................... 42

Architecture Redesign for the SDK ................................................................. 42

Breaking Changes ............................................................................................ 42

Listing AWS Resources using AWS CloudFormation ........................................... 44

Creating and Managing Encryption Keys Using Using AWS Key Management Service .................................................................................................................. 45

AWS Key Management Service Customer Master Key (CMK) Examples .......... 46

AWS Key Management Service Data Key Examples ....................................... 52

AWS Key Management Service Alias Examples .......................................... 56

AWS Key Management Service Grant Examples ......................................... 59

AWS Key Management Service Policy Examples ......................................... 63

AWS Key Management Service Key Material Examples .............................. 66

AWS Key Management Service Tag Examples .............................................. 69

Using AWS KMS Keys with the AmazonS3EncryptionClient in the AWS SDK for .NET ................................................................. 72

Using Amazon DynamoDB NoSQL Databases .................................................... 73

Low-Level Model ......................................................................................... 74

Document Model ......................................................................................... 76

Object Persistence Model ............................................................................. 77
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>More Info</td>
<td>79</td>
</tr>
<tr>
<td>Using Expressions with Amazon DynamoDB and the AWS SDK for .NET</td>
<td>79</td>
</tr>
<tr>
<td>JSON Support in Amazon DynamoDB with the AWS SDK for .NET</td>
<td>89</td>
</tr>
<tr>
<td>Managing ASP.NET Session State with Amazon DynamoDB</td>
<td>91</td>
</tr>
<tr>
<td>Elastic Beanstalk Examples</td>
<td>93</td>
</tr>
<tr>
<td>Elastic Beanstalk Environment Management Examples</td>
<td>94</td>
</tr>
<tr>
<td>Elastic Beanstalk Application Management Examples</td>
<td>101</td>
</tr>
<tr>
<td>Elastic Beanstalk Configuration Management Examples</td>
<td>104</td>
</tr>
<tr>
<td>Deploying Applications Using Amazon EC2</td>
<td>107</td>
</tr>
<tr>
<td>Amazon EC2 Instances Examples</td>
<td>108</td>
</tr>
<tr>
<td>Amazon EC2 Spot Instances Examples</td>
<td>125</td>
</tr>
<tr>
<td>Storing Archival Data Using Amazon Glacier</td>
<td>129</td>
</tr>
<tr>
<td>Programming Models</td>
<td>130</td>
</tr>
<tr>
<td>Managing Users With AWS Identity and Access Management (IAM)</td>
<td>132</td>
</tr>
<tr>
<td>Managing IAM Account Aliases</td>
<td>133</td>
</tr>
<tr>
<td>Managing IAM Users</td>
<td>135</td>
</tr>
<tr>
<td>Managing IAM Access Keys</td>
<td>137</td>
</tr>
<tr>
<td>Working with IAM Policies</td>
<td>141</td>
</tr>
<tr>
<td>Working with IAM Server Certificates</td>
<td>145</td>
</tr>
<tr>
<td>List IAM Account Information</td>
<td>147</td>
</tr>
<tr>
<td>Granting Access Using an IAM Role</td>
<td>148</td>
</tr>
<tr>
<td>Running Serverless Web Applications Using AWS Lambda</td>
<td>152</td>
</tr>
<tr>
<td>AWS Lambda Function Examples</td>
<td>153</td>
</tr>
<tr>
<td>AWS Lambda Authentication Examples</td>
<td>158</td>
</tr>
<tr>
<td>AWS Lambda Alias Examples</td>
<td>160</td>
</tr>
<tr>
<td>AWS Lambda Versioning Examples</td>
<td>163</td>
</tr>
<tr>
<td>Detecting faces with Amazon Rekognition</td>
<td>164</td>
</tr>
<tr>
<td>Managing Domain Name System (DNS) Resources Using Amazon Route 53</td>
<td>166</td>
</tr>
<tr>
<td>Using Amazon Simple Storage Service Internet Storage</td>
<td>170</td>
</tr>
<tr>
<td>Sending and Receiving Notifications From the Cloud Using Amazon Simple Notification Service</td>
<td>171</td>
</tr>
<tr>
<td>Adding a Permission Using the AWS SDK for .NET</td>
<td>172</td>
</tr>
<tr>
<td>Confirming a Subscription Using the AWS SDK for .NET</td>
<td>172</td>
</tr>
<tr>
<td>Deleting a Topic Using the AWS SDK for .NET</td>
<td>173</td>
</tr>
<tr>
<td>Listing Subscriptions Using the AWS SDK for .NET</td>
<td>173</td>
</tr>
<tr>
<td>Listing Subscriptions by Topic Using the AWS SDK for .NET</td>
<td>174</td>
</tr>
<tr>
<td>Listing Topics Using the AWS SDK for .NET</td>
<td>175</td>
</tr>
<tr>
<td>Sending an SMS Message Using the AWS SDK for .NET</td>
<td>175</td>
</tr>
<tr>
<td>Creating a Subscription and Publishing a Message Using the AWS SDK for .NET</td>
<td>176</td>
</tr>
<tr>
<td>Deleting a Subscription Using the AWS SDK for .NET</td>
<td>177</td>
</tr>
<tr>
<td>Messaging Using Amazon SQS</td>
<td>178</td>
</tr>
<tr>
<td>Creating an Amazon SQS Client</td>
<td>178</td>
</tr>
<tr>
<td>Creating an Amazon SQS Queue</td>
<td>179</td>
</tr>
<tr>
<td>Constructing Amazon SQS Queue URLs</td>
<td>179</td>
</tr>
<tr>
<td>Sending an Amazon SQS Message</td>
<td>180</td>
</tr>
<tr>
<td>Sending an Amazon SQS Message Batch</td>
<td>180</td>
</tr>
<tr>
<td>Receiving a Message from an Amazon SQS Queue</td>
<td>182</td>
</tr>
<tr>
<td>Deleting a Message from an Amazon SQS Queue</td>
<td>182</td>
</tr>
<tr>
<td>Enabling Long Polling in Amazon SQS</td>
<td>183</td>
</tr>
<tr>
<td>Using Amazon SQS Queues</td>
<td>184</td>
</tr>
<tr>
<td>Using Amazon SQS Dead Letter Queues</td>
<td>185</td>
</tr>
<tr>
<td>Monitoring Your AWS Resources Using Amazon CloudWatch</td>
<td>186</td>
</tr>
<tr>
<td>Describing, Creating, and Deleting Alarms in Amazon CloudWatch</td>
<td>187</td>
</tr>
<tr>
<td>Using Alarms in Amazon CloudWatch</td>
<td>189</td>
</tr>
<tr>
<td>Getting Metrics from Amazon CloudWatch</td>
<td>190</td>
</tr>
<tr>
<td>Sending Events to Amazon CloudWatch Events</td>
<td>192</td>
</tr>
<tr>
<td>Using Subscription Filters in Amazon CloudWatch Logs</td>
<td>195</td>
</tr>
<tr>
<td>Programming AWS OpsWorks to Work with Stacks and Applications</td>
<td>197</td>
</tr>
</tbody>
</table>
AWS SDK for .NET Developer Guide

The AWS SDK for .NET makes it easier for Windows developers to build .NET applications that tap into the cost-effective, scalable, and reliable AWS services such as Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2). The SDK supports development on any platform that supports the .NET Framework 3.5 or later, and you can develop applications with the SDK using Visual Studio 2010 or later.

The AWS SDK for .NET targets .NET Standard 1.3. You can use it with .NET Core 1.x or .NET Core 2.0. Use the async and await keywords to perform and manage asynchronous operations for all AWS products without blocking.

Topics
- What's in the SDK (p. 1)
- How to Use This Guide (p. 1)
- Supported Services and Revision History (p. 2)

What's in the SDK

The AWS SDK for .NET includes the following:

- The current version of the AWS SDK for .NET
- Sample code that demonstrates how to use the AWS SDK for .NET with several AWS services

To simplify installation, AWS provides the AWS Tools for Windows, which is a Windows installation package that includes:

- The AWS SDK for .NET
- The AWS Toolkit for Visual Studio (see the Toolkit for Visual Studio User Guide)

As an alternative to installing the AWS Tools for Windows, you can use NuGet to download individual AWSSDK service assemblies for a specific application project. With NuGet, you can download assemblies for each service you want to consume. This gives you more flexibility to manage assembly versions. For more information, see Installing AWSSDK Assemblies with NuGet (p. 5).

Note
We recommend using Visual Studio Professional 2013 or later to implement your applications.

How to Use This Guide

The AWS SDK for .NET Developer Guide describes how to implement applications for AWS using the AWS SDK for .NET, and includes the following:

Getting Started with the AWS SDK for .NET (p. 3)

How to install and configure the AWS SDK for .NET. If you have not used the AWS SDK for .NET before or are having trouble with its configuration, you should start here.
Programming with the AWS SDK for .NET (p. 8)

The basics of how to implement applications with the AWS SDK for .NET that applies to all AWS services. This section also includes information about how to migrate code to the latest version of the AWS SDK for .NET, and describes the differences between the last version and this one.

Code Examples (p. 44)

A set of tutorials, walkthroughs, and examples showing how to use the AWS SDK for .NET to create applications for particular AWS services.

Additional Resources (p. 199)

More resources outside of this guide that provide valuable information about AWS and the AWS SDK for .NET.

A related document, AWS SDK for .NET API Reference, provides a detailed description of each namespace and class.

Supported Services and Revision History

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.

To see what changed in a given release, see the SDK change log.
Getting Started with the AWS SDK for .NET

To get started with the AWS SDK for .NET, complete the following tasks:

Topics
- Create an AWS Account and Credentials (p. 3)
- Install the .NET Development Environment (p. 4)
- Install AWSSDK Assemblies (p. 4)
- Start a New Project (p. 6)
- Platforms Supported by the AWS SDK for .NET (p. 6)

Create an AWS Account and Credentials

To use the AWS SDK for .NET to access AWS, you need an AWS account and AWS credentials. To increase the security of your AWS account, we recommend that you use an IAM user to provide access credentials instead of using your root account credentials.

Note
For an overview of IAM users and why they are important for the security of your account, see Overview of Identity Management: Users in the IAM User Guide.

Signing Up for an AWS Account

To sign up for an AWS account
2. Follow the on-screen instructions. Part of the sign-up procedure involves receiving a phone call and entering a PIN using your phone keypad.

Next, create an IAM user and download (or copy) its secret access key. To use the AWS SDK for .NET, you must have a set of valid AWS credentials, which consist of an access key and a secret key. These keys are used to sign programmatic web service requests and enable AWS to verify that the request comes from an authorized source. You can obtain a set of account credentials when you create your account. However, we recommend that you do not use these credentials with AWS SDK for .NET. Instead, create one or more IAM users, and use those credentials. For applications that run on Amazon EC2 instances, you can use IAM roles to provide temporary credentials.

Creating an IAM User

To create an IAM user
1. Go to the IAM console (you may need to sign in to AWS first).
2. Click Users in the sidebar to view your IAM users.
3. If you don't have any IAM users set up, click Create New Users to create one.
4. Select the IAM user in the list that you'll use to access AWS.
5. Open the **Security Credentials** tab, and click **Create Access Key**.

   **Note**
   You can have a maximum of two active access keys for any given IAM user. If your IAM user has two access keys already, then you'll need to delete one of them before creating a new key.

6. In the resulting dialog box, choose **Download Credentials** to download the credential file to your computer, or click **Show User Security Credentials** to view the IAM user's access key ID and secret access key (which you can copy and paste).

   **Important**
   There is no way to obtain the secret access key once you close the dialog. You can, however, delete its associated access key ID and create a new one.

Next, you set your credentials in the AWS shared credentials file or in the environment.

The preferred approach for handling credentials is to create a profile for each set of credentials in the SDK Store. You can create and manage profiles with the AWS Toolkit for Visual Studio, PowerShell cmdlets, or programmatically with the AWS SDK for .NET. These credentials are encrypted and stored separately from any project. You then reference the profile by name in your application, and the credentials are inserted at build time. This approach ensures that your credentials are not unintentionally exposed with your project on a public site. For more information, see Setting Up the AWS Toolkit for Visual Studio and Configuring AWS Credentials (p. 11).

For more information about managing your credentials, see Best Practices for Managing AWS Access Keys.

To view your current account activity and manage your account at any time, go to http://aws.amazon.com and choose **My Account/Console**.

---

**Install the .NET Development Environment**

To install the AWS SDK for .NET, you must have the following installed.

**Required**

- Microsoft .NET Framework 3.5 or later
- Microsoft Visual Studio 2010 or later

   **Note**
   We recommend using Visual Studio Professional 2010 or later to implement your applications.

The AWS SDK for .NET is installed with the AWS Toolkit for Visual Studio, a plugin that provides a user interface for managing your AWS resources from Visual Studio, and also includes the AWS Tools for Windows PowerShell. To install the AWS SDK for .NET and AWS Toolkit for Visual Studio, see Setting Up the AWS Toolkit for Visual Studio.

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

**Install AWSSDK Assemblies**

You can install the AWSSDK assemblies by installing the AWS SDK for .NET or by installing the AWS assemblies with NuGet.
Installing the AWS SDK for .NET

The following procedure describes how to install the AWS SDK for .NET, which contains the AWS SDK for .NET, the AWS Toolkit for Visual Studio, and the Tools for Windows PowerShell.

Note
The AWS SDK for .NET is also available on GitHub.

To install the AWS SDK for .NET

1. Go to AWS SDK for .NET.
2. In the Downloads section, choose Download MSI Installer to download the installer.
3. To start installation, run the downloaded installer and follow the on-screen instructions.

Note
By default, the AWS SDK for .NET is installed in the Program Files directory, which requires administrator privileges. To install the AWS SDK for .NET as a non-administrator, choose a different installation directory.

4. (Optional) You can use NuGet to install individual AWSSDK service assemblies and extensions for the AWS SDK for .NET, which include a session state provider and a trace listener. For more information, see Installing AWSSDK Assemblies with NuGet (p. 5).

Installing AWSSDK Assemblies with NuGet

NuGet is a package management system for the .NET platform. With NuGet, you can add the AWSSDK assemblies, as well as the TraceListener and SessionProvider extensions, to your application.

NuGet always has the most recent versions of the AWSSDK assemblies, and also enables you to install previous versions. NuGet is aware of dependencies between assemblies and installs all required assemblies automatically. Assemblies installed with NuGet are stored with your solution instead of in a central location, such as in the Program Files directory. 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.

NuGet is installed automatically with Visual Studio 2010 or later. If you are using an earlier version of Visual Studio, you can install NuGet from the Visual Studio Gallery on MSDN.

You can use NuGet from Solution Explorer or from the Package Manager Console.

NuGet AWSSDK Packages

The NuGet website provides a page for every package available through NuGet. The page for each package includes a sample command line for installing the package using the Package Manager Console. Each page also includes a list of the previous versions of the package that are available through NuGet. For a list of AWSSDK packages available from NuGet, see AWSSDK Packages.

Using NuGet from Solution Explorer

To use NuGet from 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 Manage NuGet Packages dialog box, choose Online. You can then use the search box in the top-right corner to search for the package you want to install.
The following figure shows the **AWSSDK - Core Runtime** assembly package. You can see NuGet is aware that this package has a dependency on the **AWSSDK.Core** assembly package; NuGet automatically installs the **AWSSDK.Core** package, if it is not already installed.

**Using NuGet from the Package Manager Console**

**To use NuGet from the Package Manager Console in Visual Studio**

- **Visual Studio 2010**
  
  From the **Tools** menu, choose **Library Package Manager**, and then click **Package Manager Console**.

- **Visual Studio 2012 and later**
  
  From the **Tools** menu, choose **Nuget Package Manager**, and then click **Package Manager Console**.

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

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

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

To install an earlier version of a package, use the `-Version` option and specify the package version you want. For example, to install version 3.1.0.0 of the AWS SDK for .NET assembly, use the following command line.

```
PM> Install-Package AWSSDK.Core -Version 3.1.0.0
```

For more information about Package Manager Console commands, see [Package Manager Console Commands (v1.3)](#).

**Start a New Project**

The Toolkit for Visual Studio includes C# project templates for a variety of AWS services. The best way to get started developing an application targeting the AWS services is to use one of the examples in the Toolkit for Visual Studio based on the templates. For a list of the available examples see [Working with AWS Services in the Toolkit for Visual Studio](#).

**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 Framework 4.5**

This version of the AWS SDK for .NET is compiled against .NET Framework 4.5 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

This version of the AWS SDK for .NET is compiled against .NET Framework 3.5, and runs either the .NET 2.0 or .NET 4.0 runtime. AWS service clients support synchronous and asynchronous calling patterns and use the older Begin and End pattern.

**Note**
The AWS SDK for .NET is not Federal Information Processing Standard (FIPS) compliant when used by applications built against version 2.0 of the CLR. For details on how you can substitute a FIPS compliant implementation in that environment, refer to CryptoConfig on the Microsoft blog and the CLR Security team's HMACSHA256 class (HMACSHA256Cng) in Security.Cryptography.dll.

.NET Core

The AWS SDK for .NET supports applications written for .NET Core. 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 Amazon S3’s TransferUtility which will only support asynchronous calls in the .NET Core environment. For details, see Configuring the AWS SDK for .NET with .NET Core (p. 9).

Portable Class Library

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 AWS Mobile SDK for .NET and Xamarin for more details. AWS service clients only support asynchronous calling patterns.

Unity Support

The AWS SDK for .NET supports generating Assemblies for Unity. More information can be found in the Unity README.

More Info

- Migrating Your Code to Version 3 of the AWS SDK for .NET (p. 42)
Programming with the AWS SDK for .NET

This section provides general information about developing software with the AWS SDK for .NET.

For information about developing software for specific AWS services, see Code Examples (p. 44).

Topics
- Configuring Your AWS SDK for .NET Application (p. 8)
- Amazon Web Services Asynchronous APIs for .NET (p. 34)
- Retries and Timeouts (p. 40)
- Migrating Your Code to Version 3 of the AWS SDK for .NET (p. 42)

Configuring Your AWS SDK for .NET Application

You can configure your AWS SDK for .NET application to specify AWS credentials, logging options, endpoints, or signature version 4 support with Amazon S3.

The recommended way to configure an application is to use the <aws> element in the project's App.config or Web.config file. The following example specifies the AWSRegion (p. 22) and AWSLogging (p. 21) parameters.

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

Another way to configure an application is to edit the <appSettings> element in the project's App.config or Web.config file. The following example specifies the AWSRegion (p. 22) and AWSLogging (p. 21) parameters.

```xml
<configuration>
  <appSettings>
    <add key="AWSRegion" value="us-west-2"/>
    <add key="AWSLogging" value="log4net"/>
  </appSettings>
</configuration>
```

These settings take effect only after the application has been rebuilt.

Although you can configure an AWS SDK for .NET application programmatically by setting property values in the AWSConfigs class, we recommend you use the aws element instead. The following example specifies the AWSRegion (p. 22) and AWSLogging (p. 21) parameters:

```csharp
AWSConfigs.AWSRegion = "us-west-2";
```
AWS SDK for .NET Developer Guide
Configuring the AWS SDK for .NET with .NET Core

AWSConfigs.Logging = LoggingOptions.Log4Net;

Programmatically defined parameters override any values that were specified in an App.config or Web.config file. Some programmatically defined parameter values take effect immediately; others take effect only after you create a new client object. For more information, see Configuring AWS Credentials (p. 11).

**Topics**

- Configuring the AWS SDK for .NET with .NET Core (p. 9)
- Configuring AWS Credentials (p. 11)
- AWS Region Selection (p. 19)
- Configuring Other Application Parameters (p. 21)
- Configuration Files Reference for AWS SDK for .NET (p. 26)

## 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 ubiquitously with .NET Framework and ASP.NET applications. For traditional .NET applications, the AWS SDK for .NET uses this configuration system to set things like AWS credentials and region so that you don't have to do this in code.

The configuration system in .NET Core allows any type of input source from any location. Also, the configuration object isn't a global singleton like the ConfigurationManager in standard .NET applications, so the AWS SDK for .NET doesn't have access to read settings from it.

**Note**

For background on the .NET Core configuration system, read the Configuration topic in the .NET Core documentation.

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.

### Using AWSSDK.Extensions.NETCore.Setup

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.

There are three different configuration files, one for each potential environment: Development, Staging, and Production. The value is stored in the ASPNETCORE_ENVIRONMENT variable. You can find this setting in the Properties | Debug tab of Visual Studio.

When ASPNETCORE_ENVIRONMENT is set to Development, configuration options are read from the appsettings.Development.json file. It is read-only during local testing. When you deploy an Amazon EC2 instance that has ASPNETCORE_ENVIRONMENT set to Production, this file is ignored and the AWS SDK for .NET falls back to the IAM credentials and region configured for the Amazon EC2 instance.

For more details, see working with multiple environments on the .NET Core documentation site.

The configuration below shows an example of the values you can add in the appsettings.Development.json file in your project to supply AWS settings.

```json
{
    "AWS": {
```
To access the AWS options set in the file from code, call the `GetAWSOptions` extension method added on `IConfiguration`. To construct a service client from these options, call `CreateServiceClient`. The following example code shows how to create an Amazon S3 service client.

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

**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 in the list below. For details on these settings, refer to 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 `Startup` is where the MVC services are added. If the application is using Entity Framework, this is also where that is initialized.

```csharp
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 how to add the AWS options that are read from IConfiguration to add Amazon S3 and DynamoDB to our list of services.

```csharp
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.

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

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

### Configuring AWS Credentials

You must manage your AWS credentials securely and avoid practices that can unintentionally expose your credentials to the public. In this topic, we describe how you configure your application's AWS credentials so that they remain secure.

- Don't use your account's root credentials to access your AWS resources. These credentials provide unrestricted account access and are difficult to revoke.
- Don't put literal access keys in your application, including the project's App.config or Web.config file. If you do, you create a risk of accidentally exposing your credentials if, for example, you upload the project to a public repository.

**Note**

We assume you have created an AWS account and have access to your credentials. If you haven't yet, see Create an AWS Account and Credentials (p. 3).

The following are general guidelines for securely managing credentials:

- Create IAM users and use their IAM user credentials instead of using your AWS root user. IAM user credentials are easier to revoke if they're compromised. You can apply a policy to each IAM user that restricts the user to a specific set of resources and actions.
- During application development, the preferred approach for managing credentials is to put a profile for each set of IAM user credentials in the SDK Store. You can also use a plaintext credentials file to store profiles that contain credentials. Then you can reference a specific profile programmatically instead of storing the credentials in your project files. To limit the risk of unintentionally exposing credentials, you should store the SDK Store or credentials file separately from your project files.
• 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.

The following topics describe how to manage credentials for an AWS SDK for .NET application. For a discussion of how to securely manage AWS credentials, see Best Practices for Managing AWS Access Keys.

Using the SDK Store

During development of your AWS SDK for .NET application, add a profile to the SDK Store for each set of credentials you want to use in your application. This prevents the accidental exposure of your AWS credentials. The SDK Store is located in the C:\Users\<username>\AppData\Local\AWSToolkit folder in the RegisteredAccounts.json file. The SDK Store provides the following benefits:

• The SDK Store can contain multiple profiles from any number of accounts.
• 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.
• You reference the profile by name in your application and the associated credentials are referenced at run time. Your source files never contain the credentials.
• If you include a profile named default, the AWS SDK for .NET uses that profile. This is also true if you don't provide another profile name, or if the specified name isn't found.
• The SDK Store also provides credentials to AWS Tools for Windows PowerShell and the AWS Toolkit for Visual Studio.

Note
SDK Store profiles are specific to a particular user on a particular host. They cannot be copied to other hosts or other users. For this reason, you cannot use SDK Store profiles in production applications. For more information, see Credential and Profile Resolution (p. 17).

You can manage the profiles in the SDK Store in several ways.

• Use the graphical user interface (GUI) in the AWS Toolkit for Visual Studio to manage profiles. For more information about adding credentials to the SDK Store by using the GUI, see Specifying Credentials in the AWS Toolkit for Visual Studio.
• You can manage your profiles from the command line by using the Set-AWSCredentials cmdlet in AWS Tools for Windows PowerShell. For more information, see Using AWS Credentials.
• You can create and manage your profiles programmatically by using the Amazon.Runtime.CredentialManagement.CredentialProfile class.

The following examples show how to create a basic profile and SAML profile and add them to the SDK Store by using the RegisterProfile method.

Create a Profile and Save it to the .NET Credentials File

Create an Amazon.Runtime.CredentialManagement.CredentialProfileOptions object and set its AccessKey and SecretKey properties. Create an Amazon.Runtime.CredentialManagement.CredentialProfile object. Provide the name of the profile and the CredentialProfileOptions object you created. Optionally, set the Region property for the profile. Instantiate a NetSDKCredentialsFile object and call the RegisterProfile method to register the profile.

```csharp
var options = new CredentialProfileOptions
```
AWS SDK for .NET Developer Guide
Configuring AWS Credentials

```csharp
{
    AccessKey = "access_key",
    SecretKey = "secret_key"
};

var profile = new Amazon.Runtime.CredentialManagement.CredentialProfile("basic_profile",
    options);
profile.Region = RegionEndpoint.USWest1;
var netSDKFile = new NetSDKCredentialsFile();
netSDKFile.RegisterProfile(profile);
```

The `RegisterProfile` method is used to register a new profile. Your application typically calls this method only once for each profile.

### Create a SAML Endpoint and an Associated Profile and Save it to the .NET Credentials File

Create an `Amazon.Runtime.CredentialManagement.SAMLEndpoint` object. Provide the name and endpoint URI parameters. Create an `Amazon.Runtime.CredentialManagement.SAMLEndpointManager` object. Call the `RegisterEndpoint` method to register the endpoint. Create an `Amazon.Runtime.CredentialManagement.CredentialProfileOptions` object and set its `EndpointName` and `RoleArn` properties. Create an `Amazon.Runtime.CredentialManagement.CredentialProfile` object and provide the name of the profile and the `CredentialProfileOptions` object you created. Optionally, set the Region property for the profile. Instantiate a `NetSDKCredentialsFile` object and call the `RegisterProfile` method to register the profile.

```csharp
var endpoint = new SAMLEndpoint("endpoint1", new Uri("https://some_saml_endpoint"),
    SAMLAuthenticationType.Kerberos);
var endpointManager = new SAMLEndpointManager();
endpointManager.RegisterEndpoint(endpoint);
options = new CredentialProfileOptions
    {
        EndpointName = "endpoint1",
        RoleArn = "arn:aws:iam::999999999999:role/some-role"
    };
profile = new CredentialProfile("federated_profile", options);
netSDKFile = new NetSDKCredentialsFile();
netSDKFile.RegisterProfile(profile);
```

### Using a Credentials File

You can also store profiles in a shared credentials file. This file can be used by the other AWS SDKs, the AWS CLI and AWS Tools for Windows PowerShell. To reduce the risk of accidentally exposing credentials, store the credentials file separately from any project files, usually in the user's home folder. Be aware that the profiles in credentials files are stored in plaintext.

Use can manage the profiles in the shared credentials file in two ways:

- You can use a text editor. The file is named `credentials`, and the default location is under your user's home folder. For example, if your user name is `awsuser`, the credentials file would be `C:\users\awsuser\.aws\credentials`.

  The following is an example of a profile in the credentials file.

```ini
[{profile_name}]
aws_access_key_id = {accessKey}
aws_secret_access_key = {secretKey}
```

For more information, see

**Note**

If you include a profile named default, the AWS SDK for .NET uses that profile by default if it can't find the specified profile. You can store the credentials file that contains the profiles in a location you choose, such as C:\aws_service_credentials\credentials. You then explicitly specify the file path in the AWSProfilesLocation attribute in your project's App.config or Web.config file. For more information, see Specifying a Profile (p. 17).

- You can programmatically manage the credentials file by using the classes in the Amazon.Runtime.CredentialManagement namespace.

Create a Profile and Save it to the Shared Credentials File

Create an Amazon.Runtime.CredentialManagement.CredentialProfileOptions object and set its AccessKey and SecretKey properties. Create an Amazon.Runtime.CredentialManagement.CredentialProfile object. Provide the name of the profile and the CredentialProfileOptions you created. Optionally, set the Region property for the profile. Instantiate an Amazon.Runtime.CredentialManagement.SharedCredentialsFile object and call the RegisterProfile method to register the profile.

```csharp
options = new CredentialProfileOptions
{
    AccessKey = "access_key",
    SecretKey = "secret_key"
};
profile = new CredentialProfile("shared_profile", options);
profile.Region = RegionEndpoint.USWest1;
var sharedFile = new SharedCredentialsFile();
sharedFile.RegisterProfile(profile);
```

The RegisterProfile method is used to register a new profile. Your application will normally call this method only once for each profile.

Create a Source Profile and an Associated Assume Role Profile and Save It to the Credentials File

Create an Amazon.Runtime.CredentialManagement.CredentialProfileOptions object for the source profile and set its AccessKey and SecretKey properties. Create an Amazon.Runtime.CredentialManagement.CredentialProfile object. Provide the name of the profile and the CredentialProfileOptions you created. Instantiate an Amazon.Runtime.CredentialManagement.SharedCredentialsFile object and call the RegisterProfile method to register the profile. Create another Amazon.Runtime.CredentialManagement.CredentialProfileOptions object for the assumed role profile and set the SourceProfile and RoleArn properties for the profile. Create an Amazon.Runtime.CredentialManagement.CredentialProfile object for the assumed role. Provide the name of the profile and the CredentialProfileOptions you created.

```csharp
// Create the source profile and save it to the shared credentials file
var sourceProfileOptions = new CredentialProfileOptions
{
    AccessKey = "access_key",
    SecretKey = "secret_key"
};
var sourceProfile = new CredentialProfile("source_profile", sourceProfileOptions);
sharedFile = new SharedCredentialsFile();
```
sharedFile.RegisterProfile(sourceProfile);

// Create the assume role profile and save it to the shared credentials file
var assumeRoleProfileOptions = new CredentialProfileOptions
{
    SourceProfile = "source_profile",
    RoleArn = "arn:aws:iam::999999999999:role/some-role"
};
var assumeRoleProfile = new CredentialProfile("assume_role_profile",
assumeRoleProfileOptions);
sharedFile.RegisterProfile(assumeRoleProfile);

Update an Existing Profile in the Shared Credentials File

Create an Amazon.Runtime.CredentialManagement.SharedCredentialsFile object. Set the Region, AccessKey and SecretKey properties for the profile. Call the TryGetProfile method. If the profile exists, use an Amazon.Runtime.CredentialManagement.SharedCredentialsFile instance to call the RegisterProfile method to register the updated profile.

sharedFile = new SharedCredentialsFile();
CredentialProfile basicProfile;
if (sharedFile.TryGetProfile("basicProfile", out basicProfile))
{
    basicProfile.Region = RegionEndpoint.USEast1;
    basicProfile.Options.AccessKey = "different_access_key";
    basicProfile.Options.SecretKey = "different_secret_key";
    sharedFile.RegisterProfile(basicProfile);
}

Accessing Credentials and Profiles in an Application

You can easily locate credentials and profiles in the .NET credentials file or in the shared credentials file by using the Amazon.Runtime.CredentialManagement.CredentialProfileStoreChain class. This is the way the .NET SDK looks for credentials and profiles. The CredentialProfileStoreChain class automatically checks in both credentials files.

You can get credentials or profiles by using the TryGetAWSCredentials or TryGetProfile methods. The ProfilesLocation property determines the behavior of the CredentialsProfileChain, as follows:

1. If ProfilesLocation is non-null and non-empty, search the shared credentials file at the disk path in the ProfilesLocation property.
2. If ProfilesLocation is null or empty and the platform supports the .NET credentials file, search the .NET credentials file. If the profile is not found, search the shared credentials file in the default location.
3. If ProfilesLocation is null or empty and the platform doesn't support the .NET credentials file, search the shared credentials file in the default location.

Get Credentials from the SDK Credentials File or the Shared Credentials File in the Default Location.

Create a CredentialProfileStoreChain object and an Amazon.Runtime.AWSCredentials object. Call the TryGetAWSCredentials method. Provide the profile name and the AWSCredentials object in which to return the credentials.
if (chain.TryGetAWSCredentials("basic_profile", out awsCredentials))
{
    // use awsCredentials
}

Get a Profile from the SDK Credentials File or the Shared Credentials File in the Default Location

Create a CredentialProfileStoreChain object and an Amazon.Runtime.CredentialManagement.CredentialProfile object. Call the TryGetProfile method and provide the profile name and CredentialProfile object in which to return the credentials.

```csharp
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Get AWSCredentials from a File in the Shared Credentials File Format at a File Location

Create a CredentialProfileStoreChain object and provide the path to the credentials file. Create an AWSCredentials object. Call the TryGetAWSCredentials method. Provide the profile name and the AWSCredentials object in which to return the credentials.

```csharp
var chain = new CredentialProfileStoreChain("c:\Users\sdkuser\customCredentialsFile.ini");
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("basic_profile", out awsCredentials))
{
    // Use awsCredentials
}
```

How to Create an AmazonS3Client Using the SharedCredentialsFile Class

You can create an AmazonS3Client object that uses the credentials for a specific profile by using the Amazon.Runtime.CredentialManagement.SharedCredentialsFile class. The AWS SDK for .NET loads the credentials contained in the profile automatically. You might do this if you want to use a specific profile for a given client that is different from the profile you specify in App.Config.

```csharp
CredentialProfile basicProfile;
AWSCredentials awsCredentials;
var sharedFile = new SharedCredentialsFile();
if (sharedFile.TryGetProfile("basic_profile", out basicProfile) &&
    AWSCredentialsFactory.TryGetAWSCredentials(basicProfile, sharedFile, out
    awsCredentials))
{
    using (var client = new AmazonS3Client(awsCredentials, basicProfile.Region))
    {
        var response = client.ListBuckets();
    }
}
```

If you want to use the default profile, and have the AWS SDK for .NET automatically use your default credentials to create the client object use the following code.

```csharp
using (var client = new AmazonS3Client(RegionEndpoint.US-West2))
```
Credential and Profile Resolution

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

1. The client configuration, or what is explicitly set on the AWS service client.
2. BasicAWSCredentials that are created from the AWSAccessKey and AWSSecretKey AppConfig values, if they're available.
3. A credentials profile with the name specified by a value in AWSConfigs.AWSProfileName (set explicitly or in AppConfig).
4. The default credentials profile.
5. SessionAWSCredentials that are created from the AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and AWS_SESSION_TOKEN environment variables, if they're all non-empty.
6. BasicAWSCredentials that are created from the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment variables, if they're both non-empty.
7. IAM Roles for Tasks for Amazon EC2 Container Service (Amazon ECS) tasks.
8. EC2 instance metadata.

SDK Store profiles are specific to a particular user on a particular host. You can't copy them to other hosts or other users. For this reason, you can't reuse SDK Store profiles that are on your development machine on other hosts or developer machines. If your application is running on an Amazon EC2 instance, use an IAM role as described in Using IAM Roles for EC2 Instances with the AWS SDK for .NET (p. 148). Otherwise, store your credentials in a credentials file that your web application has access to on the server.

Profile Resolution

With two different credentials file types, it's important to understand how to configure the AWS SDK for .NET and AWS Tools for Windows PowerShell to use them. The AWSConfigs.AWSProfilesLocation (set explicitly or in AppConfig) controls how the AWS SDK for .NET finds credential profiles. The -ProfileLocation command line argument controls how AWS Tools for Windows PowerShell finds a profile. Here's how the configuration works in both cases.

<table>
<thead>
<tr>
<th>Profile Location Value</th>
<th>Profile Resolution Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>null (not set) or empty</td>
<td>First search the .NET credentials file for a profile with the specified name. If the profile isn't there, search %HOME%.aws\credentials. If the profile isn't there, search %HOME%.aws\config.</td>
</tr>
<tr>
<td>The path to a file in the shared credentials file format</td>
<td>Search only the specified file for a profile with the specified name.</td>
</tr>
</tbody>
</table>

Specifying a Profile

Profiles are the preferred way to use credentials in an AWS SDK for .NET application. You don't have to specify where the profile is stored. You only reference the profile by name. The AWS SDK for .NET retrieves the corresponding credentials, as described in the previous section.
The preferred way to specify a profile is to define an `AWSProfileName` value in the `appSettings` section of your application’s `App.config` or `Web.config` file. The associated credentials are incorporated into the application during the build process.

The following example specifies a profile named `development`.

```xml
<configuration>
  <appSettings>
    <add key="AWSProfileName" value="development"/>
  </appSettings>
</configuration>
```

This example assumes the profile exists in the SDK Store or in a credentials file in the default location. If your profiles are stored in a credentials file in another location, specify the location by adding a `AWSProfilesLocation` attribute value in the `<appSettings>` element. The following example specifies `C:\aws_service_credentials\credentials` as the credentials file.

```xml
<configuration>
  <appSettings>
    <add key="AWSProfileName" value="development"/>
    <add key="AWSProfilesLocation" value="C:\aws_service_credentials\credentials"/>
  </appSettings>
</configuration>
```

The deprecated alternative way to specify a profile is shown below for completeness, but we do not recommend it.

```xml
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws profileName="development" profilesLocation="C:\aws_service_credentials\credentials"/>
</configuration>
```

Using Federated User Account Credentials

Applications that use the AWS SDK for .NET (`.NET SDK.Core` version 3.1.6.0 and later) can use federated user accounts through Active Directory Federation Services (AD FS) to access AWS web 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 web 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 non-domain-joined accounts, 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 setup the role profile, reference the profile in your application's app.config/web.config file with the AWSProfileName key in the same way you would with other credential profiles.

The SDK Security Token Service assembly (AWSSDK.SecurityToken.dll), which is loaded at runtime, provides the SAML support to obtain AWS credentials. Be sure this assembly is available to your application at run time.

**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 *Using IAM Roles for EC2 Instances with the AWS SDK for .NET* (p. 148).

For application scenarios in which the software executable is available to users outside your organization, we recommend 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:

- Using Security Tokens to Grant Temporary Access to Your AWS Resources
- Authenticating Users of AWS Mobile Applications with a Token Vending Machine.

Although the title of the second article refers specifically to mobile applications, the article contains information that is useful for any AWS application deployed outside of your organization.

**Using Proxy Credentials**

If your software communicates with AWS through a proxy, you can specify credentials for the proxy by using the ProxyCredentials property on the AmazonS3Config class for the service. For example, for Amazon S3 you could use code similar to the following, where {my-username} and {my-password} are the proxy user name and password specified in a NetworkCredential object.

```csharp
AmazonS3Config config = new AmazonS3Config();
config.ProxyCredentials = new NetworkCredential("my-username", "my-password");
```

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

**AWS Region Selection**

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. You can specify a region when creating the AWS service client by using the RegionEndpoint class.

Here is an example that instantiates an Amazon EC2 client in a specific region.

```csharp
AmazonEC2Client ec2Client = new AmazonEC2Client(RegionEndpoint.USEast1);
```

Regions are isolated from each other. For example, you can't access US East (N. Virginia) resources when using the EU (Ireland) region. If your code needs access to multiple AWS Regions, we recommend you create a separate client for each 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...
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. You can specify newly added regions on a per-client basis or globally.

**Per-Client**

Construct the new region endpoint by using `GetBySystemName`:

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

You can also use the `ServiceURL` property of the service client configuration class to specify the region. This technique works even if the region endpoint does not follow the regular region endpoint pattern.

```csharp
var ec2ClientConfig = new AmazonEC2Config
{
    // Specify the endpoint explicitly
    ServiceURL = "https://ec2.us-west-new.amazonaws.com"
};
using (var ec2Client = new AmazonEC2Client(newRegion))
{
    // Make a request to EC2 using ec2Client
}
```

**Globally**

You can set the region globally in three ways.

You can set the `AWSConfigs.AWSRegion` property,

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

You can set the `AWSRegion` key in the `appSettings` section of the `app.config` file.

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

You can set the `region` attribute in the `aws` section as described in `AWSRegion`.

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

To view the current list of all supported regions and endpoints for each AWS service, see [Regions and Endpoints](#) in the *Amazon Web Services General Reference*. 
Configuring Other Application Parameters

In addition to configuring credentials (p. 11), you can configure a number of other application parameters:

- AWSLogging (p. 21)
- AWSLogMetrics (p. 22)
- AWSRegion (p. 22)
- AWSResponseLogging (p. 22)
- AWS.DynamoDBContext.TableNamePrefix (p. 23)
- AWS.S3.UseSignatureVersion4 (p. 24)
- AWSEndpointDefinition (p. 24)
- AWS Service-Generated Endpoints (p. 24)

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 (p. 26).

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:

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

Alternatively:

```xml
<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:

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

Alternatively:
Configuring Other Application Parameters

```xml
<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:

```csharp
AWSSDKLib.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:

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

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

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

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

```csharp
AWSSDKLib.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:

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

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

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

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

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

For more information about creating an AWS client for a specific region, see `AWS Region Selection (p. 19)`. 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:

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

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

```xml
<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:

```csharp
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.

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

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

```xml
<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:

```csharp
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 endpoint.json file is available from https://github.com/aws/aws-sdk-net/blob/master/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.

```csharp
// Create domain and retrieve DocService and SearchService endpoints
Directory.SearchStatus 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\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
    };
    Console.WriteLine("SearchService query " + searchReq.Query + " returned " + domainSearchService.
```
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.

```csharp
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 IoT client");
}
```

Configuration Files Reference for AWS SDK for .NET

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 (p. 8).

**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 (p. 8).

**Topics**
- Declaring an AWS Settings Section (p. 27)
- Allowed Elements (p. 27)
- Elements Reference (p. 28)

### 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
<?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.

```xml
<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
```
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 (p. 28)
- aws (p. 29)
- dynamoDB (p. 30)
- dynamoDBContext (p. 30)
- ec2 (p. 30)
- logging (p. 31)
- map (p. 32)
- property (p. 32)
- proxy (p. 33)
- s3 (p. 34)

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:

```xml
<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:

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

```xml
<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:

```xml
<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:

```xml
<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:

```xml
<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:

```xml
<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:

```xml
<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 attribute 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 attribute 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 attribute 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 attribute 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:

```xml
<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:

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

Amazon Web Services Asynchronous APIs for .NET

Asynchronous API for .NET Framework 4.5, Windows Store, and Windows Phone 8

The AWS SDK for .NET uses the new task-based asynchronous pattern for .NET Framework version 4.5, Windows Store, and Windows Phone 8. You can use the `async` and `await` keywords to perform and manage asynchronous operations for all AWS products without blocking.

To learn more about the task-based asynchronous pattern, see Task-based Asynchronous Pattern (TAP) on MSDN.

Asynchronous API for .NET Framework 3.5

The AWS SDK for .NET supports asynchronous (async) versions of most of the method calls exposed by the .NET client classes. The async methods enable you to call AWS services without having your code block on the response from the service. For example, you can make a request to write data to Amazon S3 or DynamoDB and then have your code continue to do other work while AWS processes the requests.

Syntax of Async Request Methods

There are two phases to making an asynchronous request to an AWS service. The first is to call the `Begin` method for the request. This method initiates the asynchronous operation. The corresponding `End` method retrieves the response from the service and also provides an opportunity to handle exceptions that might have occurred during the operation.
Note
Calling the End method is not required. Assuming no errors are encountered, the asynchronous operation will complete whether or not you call End.

Begin Method Syntax
In addition to taking a request object parameter, such as PutItemRequest, the async Begin methods take two additional parameters: a callback function and a state object. Instead of returning a service response object, the Begin methods return a result of type IAsyncResult. For the definition of this type, go to the MSDN documentation.

Synchronous Method

```csharp
PutItemResponse PutItem(
    PutItemRequest putItemRequest
)
```

Asynchronous Method

```csharp
IAsyncResult BeginPutItem(
    GetSessionTokenRequest getSessionTokenRequest, {AsyncCallback callback}, {Object state}
)
```

AsyncCallback Callback

The callback function is called when the asynchronous operation is complete. When the function is called, it receives a single parameter of type IAsyncResult. The callback function has the following signature.

```csharp
void Callback(IAsyncResult asyncResult)
```

Object State

The third parameter, state, is a user-defined object that is made available to the callback function as the AsyncState property of the IAsyncResult parameter, that is, asyncResult.AsyncState.

Calling Patterns

- Passing a callback function and a state object.
- Passing a callback function, but passing null for the state object.
- Passing null for both the callback function and the state object.

This topic provides an example of each of these patterns.

Using IAsyncResult.AsyncWaitHandle

In some circumstances, the code that calls the Begin method might need to enable another method that it calls to wait on the completion of the asynchronous operation. In these situations, it can pass the method the WaitHandle returned by the IAsyncResult.AsyncWaitHandle property of the IAsyncResult return value. The method can then wait for the asynchronous operation to complete by calling WaitOne on this WaitHandle.

Examples

All of the following examples assume the following initialization code.

```csharp
public static void TestPutObjectAsync()
```
// Create a client AmazonS3Client
client = new AmazonS3Client();

PutObjectResponse response;
IAsyncResult asyncResult;

// // Create a PutObject request // // You will need to use your own bucket name below in order // to run this sample code. // PutObjectRequest request = new PutObjectRequest
{ BucketName = "{PUT YOUR OWN EXISTING BUCKET NAME HERE}", Key = "Item", ContentBody = "This is sample content..." }; // // // additional example code // }

No Callback Specified

The following example code calls BeginPutObject, performs some work, and then calls EndPutObject to retrieve the service response. The call to EndPutObject is enclosed in a try block to catch any exceptions that might have been thrown during the operation.

asyncResult = client.BeginPutObject(request, null, null);
while ( ! asyncResult.IsCompleted ) {
    // // Do some work here // //
    try {
        response = client.EndPutObject(asyncResult);
    } catch (AmazonS3Exception s3Exception) {
        // // Code to process exception // //
    }
}

Simple Callback

This example assumes the following callback function has been defined.

public static void SimpleCallback(IAsyncResult asyncResult)
{
    Console.WriteLine("Finished PutObject operation with simple callback");
}

The following line of code calls BeginPutObject and specifies the above callback function. When the PutObject operation is complete, the callback function is called. The call to BeginPutObject specifies null for the state parameter because the simple callback function does not access the AsyncState property of the asyncResult parameter. Neither the calling code or the callback function call EndPutObject. Therefore, the service response is effectively discarded and any exceptions that occur during the operation are ignored.
asyncResult = client.BeginPutObject(request, SimpleCallback, null);

**Callback with Client**

This example assumes the following callback function has been defined.

```csharp
public static void CallbackWithClient(IAsyncResult asyncResult)
{
try {
    AmazonS3Client s3Client = (AmazonS3Client) asyncResult.AsyncState;
    PutObjectResponse response = s3Client.EndPutObject(asyncResult);
    Console.WriteLine("Finished PutObject operation with client callback");
}
catch (AmazonS3Exception s3Exception) {
    //
    // Code to process exception
    //
}
}
```

The following line of code calls `BeginPutObject` and specifies the preceding callback function. When the `PutObject` operation is complete, the callback function is called. In this example, the call to `BeginPutObject` specifies the Amazon S3 client object for the `state` parameter. The callback function uses the client to call the `EndPutObject` method to retrieve the server response. Because any exceptions that occurred during the operation will be received when the callback calls `EndPutObject`, this call is placed within a `try` block.

asyncResult = client.BeginPutObject(request, CallbackWithClient, client);

**Callback with State Object**

This example assumes the following class and callback function have been defined.

```csharp
class ClientState
{
    AmazonS3Client client;
    DateTime startTime;

    public AmazonS3Client Client
    {
        get { return client; }
        set { client = value; }
    }

    public DateTime Start
    {
        get { return startTime; }
        set { startTime = value; }
    }
}
```

```csharp
public static void CallbackWithState(IAsyncResult asyncResult)
{
try {
    ClientState state = asyncResult.AsyncState as ClientState;
    AmazonS3Client s3Client = (AmazonS3Client)state.Client;
    PutObjectResponse response = state.Client.EndPutObject(asyncResult);
    Console.WriteLine("Finished PutObject. Elapsed time: {0}");
```
(DateTime.Now - state.Start).ToString();
}
catch (AmazonS3Exception s3Exception) {
    //
    // Code to process exception
    //
}
}

The following line of code calls BeginPutObject and specifies the above callback function. When the PutObject operation is complete, the callback function is called. In this example, the call to BeginPutObject specifies, for the state parameter, an instance of the ClientState class defined previously. This class embeds the Amazon S3 client as well as the time at which BeginPutObject is called. The callback function uses the Amazon S3 client object to call the EndPutObject method to retrieve the server response. The callback also extracts the start time for the operation and uses it to print the time it took for the asynchronous operation to complete.

As in the previous examples, because exceptions that occur during the operation are received when EndPutObject is called, this call is placed within a try block.

asyncResult = client.BeginPutObject(
    request, CallbackWithState, new ClientState { Client = client, Start = DateTime.Now } );

Complete Sample

The following code sample demonstrates the patterns you can use when calling the asynchronous request methods.

```csharp
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Text;
using System.Threading;
using Amazon;
using Amazon.Runtime;
using Amazon.S3;
using Amazon.S3.Model;

namespace async_aws_net
{
    class ClientState
    {
        AmazonS3Client client;
        DateTime startTime;

        public AmazonS3Client Client
        {
            get { return client; } 
            set { client = value; } 
        }

        public DateTime Start
        {
            get { return startTime; } 
            set { startTime = value; } 
        }
    }

    class Program
    {
```
```csharp
public static void Main(string[] args)
{
    TestPutObjectAsync();
}

public static void SimpleCallback(IAsyncResult asyncResult)
{
    Console.WriteLine("Finished PutObject operation with simple callback");
    Console.Write("\n\n");
}

public static void CallbackWithClient(IAsyncResult asyncResult)
{
    try {
        AmazonS3Client s3Client = (AmazonS3Client) asyncResult.AsyncState;
        PutObjectResponse response = s3Client.EndPutObject(asyncResult);
        Console.WriteLine("Finished PutObject operation with client callback");
        Console.WriteLine("Service Response:");
        Console.WriteLine("-----------------" + response);
        Console.Write("\n\n");
    } catch (AmazonS3Exception s3Exception) {
        // Code to process exception
        //
    }
}

public static void CallbackWithState(IAsyncResult asyncResult)
{
    try {
        ClientState state = asyncResult.AsyncState as ClientState;
        AmazonS3Client s3Client = (AmazonS3Client) state.Client;
        PutObjectResponse response = state.Client.EndPutObject(asyncResult);
        Console.WriteLine("Finished PutObject operation with state callback that started at {0}", (DateTime.Now - state.Start).ToString() + state.Start);
        Console.WriteLine("Service Response:");
        Console.WriteLine("-----------------" + response);
        Console.Write("\n\n");
    } catch (AmazonS3Exception s3Exception) {
        // Code to process exception
        //
    }
}

public static void TestPutObjectAsync()
{
    // Create a client
    AmazonS3Client client = new AmazonS3Client();
    PutObjectResponse response;
    IAsyncResult asyncResult;
    // Create a PutObject request
    // You will need to change the BucketName below in order to run this sample code.
    PutObjectRequest request = new PutObjectRequest
```
{  
  BucketName = "PUT-YOUR-OWN-EXISTING-BUCKET-NAME-HERE",  
  Key = "Item",  
  ContentBody = "This is sample content..."
};

response = client.PutObject(request);
Console.WriteLine("Finished PutObject operation for {0}.", request.Key);
Console.WriteLine("Service Response:"');
Console.WriteLine("-----------------"');
Console.WriteLine("{(0)\n", response);
Console.Write("\n\n");

request.Key = "Item1";
asyncResult = client.BeginPutObject(request, null, null);
while ( ! asyncResult.IsCompleted ) {

  // Do some work here

}

try {
  response = client.EndPutObject(asyncResult);
}

catch (AmazonS3Exception s3Exception) {

}

Console.WriteLine("Finished Async PutObject operation for {0}.", request.Key );
Console.WriteLine("Service Response:"');
Console.WriteLine("-----------------"');
Console.WriteLine(response);
Console.Write("\n\n");

request.Key = "Item2";
asyncResult = client.BeginPutObject(request, SimpleCallback, null);

request.Key = "Item3";
asyncResult = client.BeginPutObject(request, CallbackWithClient, client);

request.Key = "Item4";
asyncResult = client.BeginPutObject(request, CallbackWithState,  
  new ClientState { Client = client, Start = DateTime.Now } );

Thread.Sleep( TimeSpan.FromSeconds(5) );
}
}

See Also

- Getting Started with the AWS SDK for .NET (p. 3)
- Programming with the AWS SDK for .NET (p. 8)

Retries and Timeouts

The AWS SDK for .NET allows 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

The AWS SDK for .NET will retry requests that fail due to server-side throttling or dropped connections. You can use the `MaxErrorRetry` property of the `ClientConfig` class to specify the number of retries at the service client level. The AWS SDK for .NET will retry the operation the specified number of times before failing and throwing an exception. By default, the `MaxErrorRetry` property is set to 4, except for the `AmazonDynamoDBConfig` class, which defaults to 10 retries. When a retry occurs, it increases the latency of your request. You should configure your retries based on your application limits for total request latency and error rates.

## Timeouts

The AWS SDK for .NET allows 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 `ClientConfig` class, respectively. 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.PutObject()`, `AmazonS3Client.UploadPart()`, `AmazonGlacierClient.UploadArchive()`, and so on.
- The version of the AWS SDK for .NET that targets the .NET Framework 4.5 sets `Timeout` and `ReadWriteTimeout` to the maximum values for all `AmazonS3Client` and `AmazonGlacierClient` objects.
- The version of the AWS SDK for .NET that targets the portable class library (PCL) sets `Timeout` to the maximum value for all `AmazonS3Client` and `AmazonGlacierClient` objects.

## Example

The following example shows how to specify a maximum of 2 retries, a timeout of 10 seconds, and a read/write timeout of 10 seconds for an `AmazonS3Client` object.

```csharp
var client = new AmazonS3Client(
    new AmazonS3Config
```
Migrating Your Code to Version 3 of the AWS SDK for .NET

This topic describes changes in version 3 of the AWS SDK for .NET and how to migrate your code to this version of the SDK.

About the AWS SDK for .NET Versions

The AWS SDK for .NET, originally released in November 2009, was designed for .NET Framework 2.0. Since that release, .NET has improved with .NET Framework 4.0 and .NET Framework 4.5, and added new target platforms: WinRT and Windows Phone.

AWS SDK for .NET version 2 was updated to take advantage of the new features of the .NET platform and to target WinRT and Windows Phone.

AWS SDK for .NET version 3 has been updated to make the assemblies modular.

Architecture Redesign for the SDK

The entire version 3 of the AWS SDK for .NET is redesigned to be modular. Each service is now implemented in its own assembly, instead of in one global assembly. You no longer have to add the entire AWS SDK for .NET to your application. You can now add assemblies only for the AWS services your application uses.

Breaking Changes

The following sections describe changes to version 3 of the AWS SDK for .NET.

AWSClientFactory Removed

The `Amazon.AWSClientFactory` class was removed. Now, to create a service client, use the constructor of the service client. For example, to create an `AmazonEC2Client`:

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

Amazon.Runtime.AssumeRoleAWSCredentials Removed

The `Amazon.Runtime.AssumeRoleAWSCredentials` class was removed because it was in a core namespace but had a dependency on the AWS Security Token Service, and because it has been obsolete in the SDK for some time. Use the `Amazon.SecurityToken.AssumeRoleAWSCredentials` class instead.

SetACL Method Removed from S3Link

The `S3Link` class is part of the `Amazon.DynamoDBv2` package and is used for storing objects in Amazon S3 that are references in a DynamoDB item. This is a useful feature, but we didn't want to
create a compile dependency on the Amazon.S3 package for DynamoDB. Consequently, we simplified the exposed Amazon.S3 methods from the S3Link class, replacing the SetACL method with the MakeS3ObjectPublic method. For more control over the access control list (ACL) on the object, use the Amazon.S3 package directly.

**Removal of Obsolete Result Classes**

For most services in the AWS SDK for .NET, operations return a response object that contains metadata for the operation, such as the request ID and a result object. Having a separate response and result class was redundant and created extra typing for developers. In version 2 of the AWS SDK for .NET, we put all the information in the result class into the response class. We also marked the result classes obsolete to discourage their use. In version 3 of the AWS SDK for .NET, we removed these obsolete result classes to help reduce the SDK's size.

**AWS Config Section Changes**

It is possible to do advanced configuration of the AWS SDK for .NET through the App.config or Web.config file. You do this through an <aws> config section like the following, which references the SDK assembly name.

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

In version 3 of the AWS SDK for .NET, the AWSSDK assembly no longer exists. We put the common code into the AWSSDK.Core assembly. As a result, you will need to change the references to the AWSSDK assembly in your App.config or Web.config file to the AWSSDK.Core assembly, as follows.

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

You can also manipulate the config settings with the Amazon.AWSConfigs class. In version 3 of the AWS SDK for .NET, we moved the config settings for DynamoDB from the Amazon.AWSConfigs class to the Amazon.AWSConfigsDynamoDB class.
Listing AWS Resources using AWS CloudFormation

The AWS SDK for .NET supports AWS CloudFormation, which creates and provisions AWS infrastructure deployments predictably and repeatedly. For more information, see AWS CloudFormation Getting Started Guide.

The following example shows how to use the low-level APIs to list accessible resources in AWS CloudFormation.

```csharp
// using Amazon.CloudFormation;
// using Amazon.CloudFormation.Model;

var client = new AmazonCloudFormationClient();
var request = new DescribeStacksRequest();
var response = client.DescribeStacks(request);

foreach (var stack in response.Stacks)
{
    // process stack information
}
```
Creating and Managing Encryption Keys Using AWS Key Management Service

The AWS SDK for .NET supports AWS Key Management Service (AWS KMS) as a managed service that makes it easy for you to create and control the encryption keys used to encrypt your data. AWS KMS is integrated with other AWS services including Amazon Elastic Block Store (Amazon EBS), Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon Elastic Transcoder, Amazon WorkMail, Amazon Relational Database Service (Amazon RDS), and others to make it simple to encrypt your data with encryption keys that you manage. AWS KMS is also integrated with AWS CloudTrail to provide you with key usage logs to help meet your auditing, regulatory and compliance needs.

AWS KMS lets you create master keys that can never be exported from the service and which can be used to encrypt and decrypt data based on policies you define.

You can perform the following management actions on master keys by using AWS KMS:

- Create, describe, and list master keys
- Enable and disable master keys
- Set and retrieve master key usage policies (access control)
- Create, delete, list, and update aliases, which are friendly names that point to your master keys
- Delete master keys to complete the key lifecycle

With AWS KMS you can also perform the following cryptographic functions using master keys:

- Encrypt, decrypt, and re-encrypt data
- Generate data encryption keys that you can export from the service in plaintext or encrypted under a master key that doesn’t leave the service
- Generate random numbers suitable for cryptographic applications

By using AWS KMS, you gain more control over access to data you encrypt. You can use the key management and cryptographic features directly in your applications or through AWS services that
are integrated with AWS KMS. Whether you are writing applications for AWS or using AWS services, AWS KMS enables you to maintain control over who can use your master keys and gain access to your encrypted data.

AWS KMS is integrated with AWS CloudTrail, a service that delivers log files to an Amazon S3 bucket that you designate. By using CloudTrail you can monitor and investigate how and when your master keys have been used and by whom.

For a more detailed introduction to AWS KMS, see AWS KMS Concepts.

To learn more about how AWS KMS uses cryptography and secures master keys, see the AWS Key Management Service Cryptographic Details whitepaper.

The following examples demonstrate how to use the AWS SDK for .NET to create and use AWS KMS topics.

The sample code is written in C#, but you can use the AWS SDK for .NET with any compatible language. The AWS SDK for .NET installs a set of C# project templates. So the simplest way to start this project is to open Visual Studio, and then choose File, New Project, AWS Sample Projects, AWS Console Project.

**Prerequisite Tasks**

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

For related API reference information, see Amazon.KeyManagementService, and Amazon.KeyManagementService.Model in the AWS SDK for .NET API Reference.

**Topics**

- AWS Key Management Service Customer Master Key (CMK) Examples (p. 46)
- AWS Key Management Service Data Key Examples (p. 52)
- AWS Key Management Service Alias Examples (p. 56)
- AWS Key Management Service Grant Examples (p. 59)
- AWS Key Management Service Policy Examples (p. 63)
- AWS Key Management Service Key Material Examples (p. 66)
- AWS Key Management Service Tag Examples (p. 69)

**AWS Key Management Service Customer Master Key (CMK) Examples**

AWS KMS lets you create master keys that can never be exported from the service and which can be used to encrypt and decrypt data based on policies you define.

You can perform the following management actions on master keys by using AWS KMS:

- Create, describe, and list master keys
- Enable and disable master keys
- Set and retrieve master key usage policies (access control)
- Create, delete, list, and update aliases, which are friendly names that point to your master keys
- Delete master keys to complete the key lifecycle

With AWS KMS you can also perform the following cryptographic functions using master keys:
• Encrypt, decrypt, and re-encrypt data
• Generate data encryption keys that you can export from the service in plaintext or encrypted under a
  master key that doesn't leave the service
• Generate random numbers suitable for cryptographic applications

By using AWS KMS, you gain more control over access to data you encrypt. You can use the key
management and cryptographic features directly in your applications or through AWS services that
are integrated with AWS KMS. Whether you are writing applications for AWS or using AWS services,
AWS KMS enables you to maintain control over who can use your master keys and gain access to your
encrypted data.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For
more information, see Getting Started with the AWS SDK for .NET (p. 3).

Create a CMK

In this example, you create a customer master key (CMK). You can use a CMK to encrypt small amounts of
data (4 KiB or less) directly, but CMKs are more commonly used to encrypt data encryption keys (DEKs),
which are used to encrypt raw data. For more information about DEKs and the difference between CMKs
and DEKs, see the following:
• The GenerateDataKey operation
• AWS Key Management Service Concepts in the AWS Key Management Service Developer Guide

This example uses the default client constructor which constructs a
AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in
Credential and Profile Resolution (p. 17).

Call the CreateKey method with an CreateKeyRequest with one or more property tags. Each tag consists
of a tag key and a tag value. Tag keys and tag values are both required, but tag values can be empty
(null) strings. This parameter is used to tag the CMK when it is created.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.CreateKey(new CreateKeyRequest
{
    Tags = new List<Tag> {
        new Tag {
            TagKey = "CreatedBy",
            TagValue = "ExampleUser"
        }
    } // One or more tags. Each tag consists of a tag key and a tag value.
});

KeyMetadata keyMetadata = response.KeyMetadata; // An object that contains information
// about the CMK created by this operation.
```

Describe a Key

In this example, you describe the specified key. See the KeyMetadata class for the complete description
of the data that is returned.
This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `DescribeKey` method with an `DescribeKeyRequest` with the identifier of the CMK that you want information about. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.DescribeKey(new DescribeKeyRequest
{
    KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK that you want information about. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
});
KeyMetadata keyMetadata = response.KeyMetadata; // An object that contains information about the specified CMK.
```

**List Keys**

In this example, you get a list of all customer master keys (CMKs) in the caller’s AWS account and region.

The request parameters are *Limit* (optional), the maximum number of items to return. When this value is present, AWS KMS does not return more than the specified number of items, but it might return fewer. *Marker* (optional) use in a subsequent request after you receive a response with truncated results. Set it to the value of `NextMarker` from the truncated response you just received.

The response values are *Keys*, a list of keys. *NextMarker*, when `Truncated` is `true`, this element is present and contains the value to use for the `Marker` parameter in a subsequent request. *Truncated*, a flag that indicates whether there are more items in the list. When this value is `true`, the list in this response is truncated. To get more items, pass the value of the `NextMarker` element in this response to the `Marker` parameter in a subsequent request.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `ListKeys` method with a `ListKeysRequest`.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.ListKeys(new ListKeysRequest
{
});
List<KeyListEntry> keys = response.Keys; // A list of CMKs, including the key ID and Amazon Resource Name (ARN) of each one.
bool truncated = response.Truncated; // A boolean that indicates whether there are more items in the list. Returns true when there are more items, or false when there are not.
```

**Update a Key Description**

In this example, you update the description of a customer master key (CMK). To see the description of a CMK, use `DescribeKey`. 
You cannot perform this operation on a CMK in a different AWS account.

This example uses the default client constructor which constructs a
AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in
Credential and Profile Resolution (p. 17).

Call the UpdateKeyDescription method with an UpdateKeyDescriptionRequest with the new description
and the identifier of the CMK that you want to update. You can use the key ID or the Amazon Resource
Name (ARN) of the CMK.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.UpdateKeyDescription(new UpdateKeyDescriptionRequest
{
    Description = "Example description that indicates the intended use of this CMK."; // The updated description.
    KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK whose
description you are updating. You can use the key ID or the Amazon Resource Name (ARN) of
the CMK.
});
```

Enable a Key

In this example, you set the state of a customer master key (CMK) to enabled, thereby permitting its use
for cryptographic operations. You cannot perform this operation on a CMK in a different AWS account.

This example uses the default client constructor which constructs a
AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in
Credential and Profile Resolution (p. 17).

Call the EnableKey method with an EnableKeyRequest with the identifier of the CMK whose key you
want to enable. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.EnableKey(new EnableKeyRequest
{
    KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK to enable.
    You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
});
```

Disable a Key

In this example, you set the state of a customer master key (CMK) to disabled, thereby preventing its use
for cryptographic operations. For more information about how key state affects the use of a CMK, see
How Key State Affects the Use of a Customer Master Key in the AWS Key Management Service Developer
Guide.

This example uses the default client constructor which constructs a
AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in
Credential and Profile Resolution (p. 17).

Call the DisableKey method with an DisableKeyRequest with the identifier of the CMK whose key you
want to disable. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
Schedule Key Deletion

In this example, you schedule the deletion of a customer master key (CMK). You may provide a waiting period, specified in days, before deletion occurs. If you do not provide a waiting period, the default period of 30 days is used. When this operation is successful, the state of the CMK changes to PendingDeletion. Before the waiting period ends, you can use CancelKeyDeletion to cancel the deletion of the CMK. After the waiting period ends, AWS KMS deletes the CMK and all AWS KMS data associated with it, including all aliases that refer to it.

You cannot perform this operation on a CMK in a different AWS account.

**Note**
Deleting a CMK is a destructive and potentially dangerous operation. When a CMK is deleted, all data that was encrypted under the CMK is rendered unrecoverable. To restrict the use of a CMK without deleting it, use DisableKey.

For more information about scheduling a CMK for deletion, see Deleting Customer Master Keys in the AWS Key Management Service Developer Guide.

This example uses the default client constructor which constructs a AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the ScheduleKeyDeletion method with an ScheduleKeyDeletionRequest with the pending window in days and the identifier of the CMK whose key you want to delete. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.ScheduleKeyDeletion(new ScheduleKeyDeletionRequest
{
   KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab", // The identifier of the CMK to schedule for deletion. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
   PendingWindowInDays = 7 // The waiting period, specified in number of days. After the waiting period ends, AWS KMS deletes the CMK.
});
DateTime deletionDate = response.DeletionDate; // The date and time after which AWS KMS deletes the CMK.
string keyId = response.KeyId; // The ARN of the CMK that is scheduled for deletion.
```

Enable Key Rotation

In this example, you enable automatic rotation of the key material for the specified customer master key (CMK). You cannot perform this operation on a CMK in a different AWS account.
This example uses the default client constructor which constructs a
AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in
Credential and Profile Resolution (p. 17).

Call the EnableKeyRotation method with an EnableKeyRotationRequest with the identifier of the CMK
whose key you want to enable rotation for. You can use the key ID or the Amazon Resource Name (ARN)
of the CMK.

Code

```
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.EnableKeyRotation(new EnableKeyRotationRequest
{
   KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK whose key
material will be rotated annually. You can use the key ID or the Amazon Resource Name (ARN)
of the CMK.
});
```

Get a Key Rotation Status

In this example, you get a Boolean value that indicates whether automatic rotation of the key material is
enabled for the specified customer master key (CMK).

To perform this operation on a CMK in a different AWS account, specify the key ARN in the value of the
KeyId parameter.

This example uses the default client constructor which constructs a
AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in
Credential and Profile Resolution (p. 17).

Call the GetKeyRotationStatus method with an GetKeyRotationStatusRequest with the identifier of the
CMK whose key rotation status you want to retrieve. You can use the key ID or the Amazon Resource
Name (ARN) of the CMK.

The return value is a boolean that indicates the key material rotation status. Returns true when
automatic annual rotation of the key material is enabled, or false when it is not.

Code

```
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.GetKeyRotationStatus(new GetKeyRotationStatusRequest
{
   KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK whose key
material rotation status you want to retrieve. You can use the key ID or the Amazon Resource Name (ARN)
of the CMK.
});
bool keyRotationEnabled = response.KeyRotationEnabled; // A boolean that indicates the key
material rotation status. Returns true when automatic annual rotation of the key material
is enabled, or false when it is not.
```

Disable Key Rotation

In this example, you disable automatic rotation of the key material for the specified customer master key
(CMK). You cannot perform this operation on a CMK in a different AWS account.
This example uses the default client constructor which constructs an
AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in
Credential and Profile Resolution (p. 17).

Call the DisableKeyRotation method with a DisableKeyRotationRequest with the identifier of the CMK
whose key rotation you want to disable. You can use the key ID or the Amazon Resource Name (ARN) of
the CMK.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.DisableKeyRotation(new DisableKeyRotationRequest
{
  KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK whose key
  material will no longer be rotated. You can use the key ID or the Amazon Resource Name
  (ARN) of the CMK.
});
```

AWS Key Management Service Data Key Examples

AWS KMS lets you create data keys, a symmetric key generated by AWS KMS for your application or
service. Inside of your custom application or service, the data key is used to encrypt or decrypt data. It
can be considered a resource by a service or application, or it can simply be metadata associated with the
encrypted data.

These examples show how to generate data keys and use the Encrypt, Decrypt, and ReEncrypt
operations in the AWS KMS API.

These operations are designed to encrypt and decrypt data keys. They use an AWS KMS customer master
key (CMK) in the encryption operations and they cannot accept more than 4 KB (4096 bytes) of data.
Although you might use them to encrypt small amounts of data, such as a password or RSA key, they are
not designed to encrypt application data.

To encrypt application data, use the server-side encryption features of an AWS service, or a client-side
encryption library, such as the AWS Encryption SDK or the Amazon S3 encryption client.

**Prerequisite Tasks**

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For
more information, see Getting Started with the AWS SDK for .NET (p. 3).

**Generate a Data Key**

In this example, you generate a data encryption key that you can use in your application to encrypt data
locally. You must specify the customer master key (CMK) under which to generate the data key. You also
specify the length of the data key using the KeySpec field.

This operation returns a plaintext copy of the data key in the Plaintext field of the response, and an
encrypted copy of the data key in the CiphertextBlob field. The data key is encrypted under the CMK
specified in the KeyId field of the request.

To encrypt data locally in your application:

- Use this operation (GenerateDataKey) to retrieve a data encryption key.
- Use the plaintext data encryption key (returned in the Plaintext field of the response) to encrypt data
  locally, then erase the plaintext data key from memory.
Store the encrypted data key (returned in the CiphertextBlob field of the response) alongside the locally encrypted data.

To decrypt data locally:

- Use the Decrypt operation to decrypt the encrypted data key into a plaintext copy of the data key.
- Use the plaintext data key to decrypt data locally, then erase the plaintext data key from memory.

Call the `GenerateDataKey` method with a `GenerateDataKeyRequest` with the key ID of the CMK to use to encrypt the key and the type of key to return. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

**Code**

```csharp
namespace AmazonKeyManagementService
{
    class KeyManagementService
    {
        public static void KeyManagementServiceGenerateDataKeyWithoutPlaintext()
        {
            AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
            var response = client.GenerateDataKey(new GenerateDataKeyRequest
            {
                KeyId = "alias/ExampleAlias", // The identifier of the CMK to use to encrypt the data key. You can use the key ID or Amazon Resource Name (ARN) of the CMK, or the name or ARN of an alias that refers to the CMK.
                KeySpec = "AES_256" // Specifies the type of data key to return.
            });

            MemoryStream ciphertextBlob = response.CiphertextBlob; // The encrypted data key.
            string keyId = response.KeyId; // The ARN of the CMK that was used to encrypt the data key.
            MemoryStream plaintext = response.Plaintext; // The unencrypted (plaintext) data key.
        }
    }
}
```

**Generate a Data Key Without Plain Text**

In this example, you generate a data encryption key encrypted under a customer master key (CMK). This operation is identical to `GenerateDataKey` but returns only the encrypted copy of the data key.

To perform this operation on a CMK in a different AWS account, specify the key ARN or alias ARN in the value of the `KeyId` parameter.

This operation is useful in a system that has multiple components with different degrees of trust. For example, consider a system that stores encrypted data in containers. Each container stores the encrypted data and an encrypted copy of the data key. One component of the system, called the control plane, creates new containers. When it creates a new container, it uses this operation (`GenerateDataKeyWithoutPlaintext`) to get an encrypted data key and then stores it in the container. Later, a different component of the system, called the data plane, puts encrypted data into the containers. To do this, it passes the encrypted data key to the `Decrypt` operation, then uses the returned plaintext data key to encrypt data, and finally stores the encrypted data in the container. In this system, the control plane never sees the plaintext data key.

Call the `GenerateDataKeyWithoutPlaintext` method with a `GenerateDataKeyWithoutPlaintext` with the key ID of the CMK to use to encrypt the key and the type of key to return. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

**Code**

```csharp
namespace AmazonKeyManagementService
{
    class KeyManagementService
    {
        public static void KeyManagementServiceGenerateDataKeyWithoutPlaintext()
        {
            AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
            var response = client.GenerateDataKeyWithoutPlaintext(new GenerateDataKeyWithoutPlaintextRequest
```
Encrypt

In this example, you encrypt plaintext into ciphertext by using a customer master key (CMK). The Encrypt operation has two primary use cases:

- To encrypt up to 4 kilobytes (4096 bytes) of arbitrary data such as an RSA key, a database password, or other sensitive information.
- To move encrypted data from one AWS region to another. You can use this operation to encrypt in the new region the plaintext data key that was used to encrypt the data in the original region. This provides you with an encrypted copy of the data key that can be decrypted in the new region and used there to decrypt the encrypted data.

To perform this operation on a CMK in a different AWS account, specify the key ARN or alias ARN in the value of the KeyId parameter.

Unless you are moving encrypted data from one region to another, you don't use this operation to encrypt a generated data key within a region. To get data keys that are already encrypted, call the GenerateDataKey or GenerateDataKeyWithoutPlaintext operation. Data keys don't need to be encrypted again by calling Encrypt.

To encrypt data locally in your application, use the GenerateDataKey operation to return a plaintext data encryption key and a copy of the key encrypted under the CMK of your choosing.

This example uses the default client constructor which constructs a AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the Encrypt method with an EncryptRequest with the identifier of the CMK whose key you want to use for encryption and the memory stream you want to encrypt. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();

UnicodeEncoding uniEncoding = new UnicodeEncoding();
byte[] binaryData = uniEncoding.GetBytes("The data to be encrypted");
var response = client.Encrypt(new EncryptRequest
{
    KeyId = "alias/ExampleAlias", // The identifier of the CMK to use to encrypt the data key. You can use the key ID or Amazon Resource Name (ARN) of the CMK, or the name or ARN of an alias that refers to the CMK.
    KeySpec = "AES_256" // Specifies the type of data key to return.
});

MemoryStream ciphertextBlob = response.CiphertextBlob; // The encrypted data key.
string keyId = response.KeyId; // The ARN of the CMK that was used to encrypt the data key.
```
Decrypt Data

In this example, you decrypt ciphertext. Ciphertext is plaintext that has been previously encrypted by using any of the following functions:

- GenerateDataKey
- GenerateDataKeyWithoutPlaintext
- Encrypt

Note that if a caller has been granted access permissions to all keys (through, for example, IAM user policies that grant Decrypt permission on all resources), then ciphertext encrypted by using keys in other accounts where the key grants access to the caller can be decrypted. To remedy this, we recommend that you do not grant Decrypt access in an IAM user policy. Instead grant Decrypt access only in key policies. If you must grant Decrypt access in an IAM user policy, you should scope the resource to specific keys or to specific trusted accounts.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `Decrypt` method with a `DecryptRequest` with the memory stream containing the encrypted data.

**Code**

```
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
UnicodeEncoding uniEncoding = new UnicodeEncoding();
byte[] binaryData = uniEncoding.GetBytes("The data to be encrypted");
var response = client.Decrypt(new DecryptRequest
{
    CiphertextBlob = new MemoryStream(binaryData) // The encrypted data (ciphertext).
});

string keyId = response.KeyId; // The Amazon Resource Name (ARN) of the CMK that was used to decrypt the data.
MemoryStream plaintext = response.Plaintext; // The decrypted (plaintext) data.
```

Reencrypt

In this example, you encrypt data on the server side with a new customer master key (CMK) without exposing the plaintext of the data on the client side. The data is first decrypted and then reencrypted. You can also use this operation to change the encryption context of a ciphertext.

You can reencrypt data using CMKs in different AWS accounts.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `ReEncrypt` method with a `ReEncryptRequest` with the binary data to reencrypt and the identifier of the CMK to use. You can use the key ID or Amazon Resource Name (ARN) of the CMK, or the name or ARN of an alias that refers to the CMK.

**Code**

```
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
```
UnicodeEncoding uniEncoding = new UnicodeEncoding();
byte[] binaryData = uniEncoding.GetBytes(
    "The data to be encrypted");
var response = client.ReEncrypt(new ReEncryptRequest
{
    CiphertextBlob = new MemoryStream(binaryData), // The data to reencrypt.
    DestinationKeyId = "0987dcba-09fe-87dc-65ba-ab0987654321" // The identifier of the CMK
        to use to reencrypt the data. You can use the key ID or Amazon Resource Name (ARN) of the
    CMK, or the name or ARN of an alias that refers to the CMK.
});
MemoryStream ciphertextBlob = response.CiphertextBlob; // The reencrypted data.
string keyId = response.KeyId; // The ARN of the CMK that was used to reencrypt the data.
string sourceKeyId = response.SourceKeyId; // The ARN of the CMK that was used to
    originally encrypt the data.

AWS Key Management Service Alias Examples

An alias is an optional display name for a customer master key (CMK).

Each CMK can have multiple aliases, but each alias points to only one CMK. The alias name must be
    unique in the AWS account and region. To simplify code that runs in multiple regions, you can use the
    same alias name, but point it to a different CMK in each region.

You can use AWS KMS API operations to list, create, and delete aliases. You can also update an alias,
    which associates an existing alias with a different CMK. There is no operation to edit or change an alias
    name. If you create an alias for a CMK that already has an alias, the operation creates another alias for
    the same CMK. To change an alias name, delete the current alias and then create a new alias for the CMK.

Because an alias is not a property of a CMK, it can be associated with and disassociated from an existing
CMK without changing the properties of the CMK. Deleting an alias does not delete the underlying CMK.

You can use an alias as the value of the KeyId parameter only in the following operations:

- DescribeKey
- Encrypt
- GenerateDataKey
- GenerateDataKeyWithoutPlaintext
- ReEncrypt

Aliases are created in an AWS account and are known only to the account in which you create them. You
cannot use an alias name or alias ARN to identify a CMK in a different AWS account.

To specify an alias, use the alias name or alias ARN, as shown in the following example. In either case, be
sure to prepend "alias/" to the alias name.

// Fully specified ARN

// Alias name (prefixed with "alias/")
alias/ExampleAlias

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For
more information, see Getting Started with the AWS SDK for .NET (p. 3).
Create an Alias

In this example, you create a display name for a customer master key. An alias can be used to identify a key and should be unique. The console enforces a one-to-one mapping between the alias and a key. An alias name can contain only alphanumeric characters, forward slashes (/), underscores (_), and dashes (-). An alias must start with the word "alias" followed by a forward slash (alias/).

This example uses the default client constructor which constructs a *AmazonSimpleNotificationServiceClient* using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the *CreateAlias* method with an *CreateAliasRequest* with the name of the alias to create and either the key ID or the Amazon Resource Name (ARN) of the CMK whose alias you are creating.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.CreateAlias(new CreateAliasRequest
{
    AliasName = "alias/ExampleAlias", // The alias to create. Aliases must begin with 'alias/'. Do not use aliases that begin with 'alias/aws' because they are reserved for use by AWS.
    TargetKeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK whose alias you are creating. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
});
```

List Aliases

In this example, you get a list of all aliases in the caller's AWS account and region. You cannot list aliases in other accounts. For more information about aliases, see CreateAlias.

The response might include several aliases that do not have a `TargetKeyId` field because they are not associated with a CMK. These are predefined aliases that are reserved for CMKs managed by AWS services. If an alias is not associated with a CMK, the alias does not count against the alias limit for your account.

The request parameters are *Limit* (optional), the maximum number of items to return. When this value is present, AWS KMS does not return more than the specified number of items, but it might return fewer. *Marker* (optional) use in a subsequent request after you receive a response with truncated results. Set it to the value of `NextMarker` from the truncated response you just received.

The response values are * `Aliases`, a list of aliases; * `NextMarker`, when `Truncated` is true, this element is present and contains the value to use for the `Marker` parameter in a subsequent request. * `Truncated`, a flag that indicates whether there are more items in the list. When this value is true, the list in this response is truncated. To get more items, pass the value of the `NextMarker` element in this response to the `Marker` parameter in a subsequent request.

This example uses the default client constructor which constructs a *AmazonSimpleNotificationServiceClient* using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the *ListAliases* method with an *ListAliasesRequest*.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
```
var response = client.ListAliases(new ListAliasesRequest
{
});
List<AliasListEntry> aliases = response.Aliases; // A list of aliases, including the key ID of the customer master key (CMK) that each alias refers to.
bool truncated = response.Truncated; // A boolean that indicates whether there are more items in the list. Returns true when there are more items, or false when there are not.

# Update an Alias

In this example, you update an alias to map it to a different key.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `UpdateAlias` method with an `UpdateAliasRequest` with the name of the alias to update and the identifier of the CMK that the alias will refer to after this operation succeeds. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.UpdateAlias(new UpdateAliasRequest
{
    AliasName = "alias/ExampleAlias", // The alias to update.
    TargetKeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK that the alias will refer to after this operation succeeds. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
});
```

# Delete an Alias

In this example, you delete the specified alias. You cannot perform this operation on an alias in a different AWS account.

Because an alias is not a property of a CMK, you can delete and change the aliases of a CMK without affecting the CMK. Also, aliases do not appear in the response from the DescribeKey operation. To get the aliases of all CMKs, use the `ListAliases` operation.

Each CMK can have multiple aliases. To change the alias of a CMK, use `DeleteAlias` to delete the current alias and `CreateAlias` to create a new alias. To associate an existing alias with a different customer master key (CMK), call `UpdateAlias`.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `DeleteAlias` method with an `DeleteAliasRequest` with the name of the alias to delete.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.DeleteAlias(new DeleteAliasRequest
{
});
```
AWS Key Management Service Grant Examples

AWS KMS supports two resource-based access control mechanisms: key policies and grants. Grants enable you to programmatically delegate the use of KMS customer master keys (CMKs) to other AWS principals. You can also use key policies to allow other principals to access a CMK, but key policies work best for relatively static assignments of permissions.

Key policy changes follow the same permissions model used for policy editing elsewhere in AWS. That is, users either have permission to change the key policy or they do not. Users with the kms:PutKeyPolicy permission for a CMK can completely replace the key policy for a CMK with a different key policy of their choice. To enable more granular permissions management, use grants.

You call the CreateGrant API operation to create a grant. You pass the identifier of the CMK for which the grant is to be created, the grantee principal being given permission to use the CMK, and a list of operations to be allowed. The CreateGrant operation returns a grant ID that you can use to identify the grant in subsequent operations. To further customize the grant permissions, you can also pass optional parameters that define grant constraints. After the grant has been created, the principal identified in the grant can execute the permitted operations, subject to the defined constraints, for as long as the grant is active. Grants can be explicitly revoked by a user who has the kms:RevokeGrant permission on the CMK, or they can be retired by the principal designated as the retiring principal for the grant.

There are two supported grant constraints: EncryptionContextEquals and EncryptionContextSubset. EncryptionContextEquals specifies that the grant applies only when the exact specified encryption context is present in the request. EncryptionContextSubset specifies that the grant applies as long as all the entries in the EncryptionContextSubset constraint are matched by the request. In this case, the request can contain additional encryption context entries. For example, a grant that allows the encrypt and decrypt operations with an EncryptionContextSubset constraint of{"Department":"Finance","Classification":"Public"} allows encryption and decryption when the request contains an encryption context of either{"Department":"Finance","Classification":"Public"} or{"Department":"Finance","Classification":"Public","Customer":"12345"}, but not when the request contains an encryption context of{"Department":"Finance"}.

When the grant includes CreateGrant as an allowed operation, the grant only allows creation of equally or more restrictive grants. That is, the grant operations passed with a subsequent CreateGrant API request can include any subset of the currently-allowed grant operations, and the grant constraints can be the same or more restrictive (fields can be added to an EncryptionContextSubset constraint, or an EncryptionContextSubset constraint can be turned into an EncryptionContextEquals constraint).

AWS Key Management Service provides the following APIs for you to create and manage grants:

- CreateGrant
- ListGrants
- RetireGrant
- ListRetirableGrants
- RevokeGrant

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).
Create a Grant

In this example, you add a grant to a customer master key (CMK). The grant specifies who can use the CMK and under what conditions. When setting permissions, grants are an alternative to key policies.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `CreateGrant` method with an `CreateGrantRequest` with the identity of the grantee, either the key ID or the Amazon Resource Name (ARN) of the CMK the grant applies to, and a list of the operations the grant allows.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.CreateGrant(new CreateGrantRequest
{
    GranteePrincipal = "arn:aws:iam::111122223333:role/ExampleRole", // The identity that is given permission to perform the operations specified in the grant.
    KeyId = "arn:aws:kms:us-east-2:444455556666:key/1234abcd-12ab-34cd-56ef-1234567890ab", // The identifier of the CMK to which the grant applies. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
    Operations = new List<string> {
        "Encrypt",
        "Decrypt"
    } // A list of operations that the grant allows.
});
string grantId = response.GrantId; // The unique identifier of the grant.
string grantToken = response.GrantToken; // The grant token.
```

List Grants

In this example, you get a list of all grants for the specified customer master key (CMK). You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

To perform this operation on a CMK in a different AWS account, specify the key ARN in the value of the KeyId parameter.

The request parameters are *KeyId (required), a unique identifier for the customer master key (CMK). *Limit (optional), the maximum number of items to return. When this value is present, AWS KMS does not return more than the specified number of items, but it might return fewer. *Marker (optional) use in a subsequent request after you receive a response with truncated results. Set it to the value of the NextMarker from the truncated response you just received.

The response values are *:code"Grants", a list of grants. *:code"NextMarker, when Truncated is true, this element is present and contains the value to use for the Marker parameter in a subsequent request. *Truncated, a flag that indicates whether there are more items in the list. When this value is true, the list in this response is truncated. To get more items, pass the value of the NextMarker element in this response to the Marker parameter in a subsequent request.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `ListGrants` method with an `ListGrantsRequest` specifying the key ID or the Amazon Resource Name (ARN) of the CMK whose grants you want to list.
Retire a Grant

In this example, you retire a grant. You can retire a grant when you're done using it. You should revoke a grant when you intend to actively deny operations that depend on it. The following are permitted to call this API:

- The AWS account (root user) under which the grant was created
- The RetiringPrincipal, if present in the grant
- The GranteePrincipal, if RetireGrant is an operation specified in the grant

You must identify the grant to retire by its grant token or by a combination of the grant ID and the Amazon Resource Name (ARN) of the customer master key (CMK).

This example uses the default client constructor which constructs an AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the RetireGrant method with an RetireGrantRequest with the grant ID and the identity of the grantee, either the key ID or the Amazon Resource Name (ARN) of the CMK the grant applies to.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.RetireGrant(new RetireGrantRequest
{
    GrantId = "0c237476b39febc44e45212e08498fbee3151305030726c0590d8d3e9f3d6a60", // The identifier of the grant to retire.
    KeyId = "arn:aws:kms:us-east-2:444455556666:key/1234abcd-12ab-34cd-56ef-1234567890ab" // The Amazon Resource Name (ARN) of the customer master key (CMK) associated with the grant.
});
```

List Resource Tags

In this example, you get a list of all grants for which the grant's RetiringPrincipal matches the one specified.

A typical use is to list all grants that you are able to retire. To retire a grant, use RetireGrant.

You cannot perform this operation on a CMK in a different AWS account.

The request parameters are * RetiringPrincipal (required), The retiring principal for which to list grants. To specify the retiring principal, use the Amazon Resource Name (ARN) of an AWS principal. Valid AWS principals include AWS accounts (root), IAM users, federated users, and assumed role users. For examples of the ARN syntax for specifying a principal, see AWS Identity and Access Management (IAM) in the Example ARNs section of the Amazon Web Services General Reference. * Limit (optional), the maximum number of items to return. When this value is present, AWS KMS does not return more than the specified number of items, but it might return fewer. * Marker (optional) use in a subsequent request after you receive a response with truncated results. Set it to the value of NextMarker from the truncated response you just received.

The response values are * :code"Grants", A list of grants. * NextMarker, when Truncated is true, this element is present and contains the value to use for the Marker parameter in a subsequent request. *
Truncated, a flag that indicates whether there are more items in the list. When this value is `true`, the list in this response is truncated. To get more items, pass the value of the `NextMarker` element in this response to the `Marker` parameter in a subsequent request.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `ListRetirableGrants` method with an `ListRetirableGrantsRequest` specifying the Amazon Resource Name (ARN) of the retiring principal whose grants you want to list.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.ListRetirableGrants(new ListRetirableGrantsRequest
{
    RetiringPrincipal = "arn:aws:iam::111122223333:role/ExampleRole" // The retiring principal whose grants you want to list. Use the Amazon Resource Name (ARN) of an AWS principal such as an AWS account (root), IAM user, federated user, or assumed role user.
});
List<GrantListEntry> grants = response.Grants; // A list of grants that the specified principal can retire.
bool truncated = response.Truncated; // A boolean that indicates whether there are more items in the list. Returns true when there are more items, or false when there are not.
```

Revoke a Grant

In this example, you revoke the specified grant for the specified customer master key (CMK). You can revoke a grant to actively deny operations that depend on it.

To perform this operation on a CMK in a different AWS account, specify the key ARN in the value of the `KeyId` parameter.

You must identify the grant to retire by its grant token or by a combination of the grant ID and the Amazon Resource Name (ARN) of the customer master key (CMK).

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `RevokeGrant` method with an `RevokeGrantRequest` with the grant ID and the identity of the grantee, either the key ID or the Amazon Resource Name (ARN) of the CMK the grant applies to.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.RetireGrant(new RetireGrantRequest
{
    GrantId = "0c237476b39f88bc44e45212e08498fbe3151305030726c0590dd8d3e9f3d6a60", // The identifier of the grant to retire.
    KeyId = "arn:aws:kms:us-east-2:44445556666:key/1234abcd-12ab-34cd-56ef-1234567890ab" // The Amazon Resource Name (ARN) of the customer master key (CMK) associated with the grant.
});
```
AWS Key Management Service Policy Examples

AWS KMS Key policies are the primary way to control access to customer master keys (CMKs) in AWS KMS. A key policy is a document that uses JSON (JavaScript Object Notation) to specify permissions. They are not the only way to control access, but you cannot control access without them. For more information, see Managing Access to AWS KMS CMKs.

When you create a CMK programmatically—that is, with the AWS KMS API (including through the AWS SDKs and command line tools)—you have the option of providing the key policy for the new CMK. If you don’t provide one, AWS KMS creates one for you. This default key policy has one policy statement that gives the AWS account (root user) that owns the CMK full access to the CMK. For more information about this policy statement, see Allows Access to the AWS Account and Enables IAM Policies.

AWS Key Management Service provides the following APIs for you to manage key policies:

- GetKeyPolicy
- ListKeyPolicies
- PutKeyPolicy

The sample code is written in C#, but you can use the AWS SDK for .NET with any compatible language. The AWS SDK for .NET installs a set of C# project templates. So the simplest way to start this project is to open Visual Studio, and then choose File, New Project, AWS Samples, AWS Console Project.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

Get a Key Policy

In this example, you get a key policy attached to the specified customer master key (CMK). You cannot perform this operation on a CMK in a different AWS account.

This example uses the default client constructor which constructs a AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the GetKeyPolicy method with an GetKeyPolicyRequest with the identifier of the CMK whose key policy you want to retrieve. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

Code

```
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.GetKeyPolicy(new GetKeyPolicyRequest
{
    KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab", // The identifier of the CMK whose key policy you want to retrieve. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
    PolicyName = "default" // The name of the key policy to retrieve.
});
string policy = response.Policy; // The key policy document.
```

List Policies

In this example, you get a list of all key policies for the specified customer master key (CMK). You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
To perform this operation on a CMK in a different AWS account, specify the key ARN in the value of the 
KeyId parameter.

The request parameters are *KeyId (required), a unique identifier for the customer master key (CMK). * 
Limit (optional), the maximum number of items to return. When this value is present, AWS KMS does 
not return more than the specified number of items, but it might return fewer. *Marker (optional) 
use in a subsequent request after you receive a response with truncated results. Set it to the value of 
NextMarker from the truncated response you just received.

The response values are *PolicyNames, a list of policy names. *NextMarker, when Truncated is 
true, this element is present and contains the value to use for the Marker parameter in a subsequent 
request. *Truncated, a flag that indicates whether there are more items in the list. When this value 
is true, the list in this response is truncated. To get more items, pass the value of the NextMarker 
element in this response to the Marker parameter in a subsequent request.

This example uses the default client constructor which constructs a 
AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in 
Credential and Profile Resolution (p. 17).

Call the ListKeyPolicies method with an ListKeyPoliciesRequest specifying the key ID or the Amazon 
Resource Name (ARN) of the CMK whose key policies you want to list.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.ListKeyPolicies(new ListKeyPoliciesRequest 
{
    KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK whose key 
policies you want to list. You can use the key ID or the Amazon Resource Name (ARN) of the 
CMK.
});
List<string> policyNames = response.PolicyNames; // A list of key policy names.
bool truncated = response.Truncated; // A boolean that indicates whether there are more 
items in the list. Returns true when there are more items, or false when there are not.
```

### Put a Key Policy

In this example, you attach a key policy to the specified customer master key (CMK). You cannot perform 
this operation on a CMK in a different AWS account.

For more information about key policies, see Key Policies in the AWS Key Management Service Developer 
Guide.

This example uses the default client constructor which constructs a 
AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in 
Credential and Profile Resolution (p. 17).

Call the PutKeyPolicy method with an PutKeyPolicyRequest with the policy, the name of the policy and 
the identifier of the CMK to attach the policy to. You can use the key ID or the Amazon Resource Name 
(ARN) of the CMK.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.PutKeyPolicy(new PutKeyPolicyRequest
```
{ 
    "KeyId": "1234abcd-12ab-34cd-56ef-1234567890ab", // The identifier of the CMK to attach the key policy to. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
    "Policy": "{" + 
        "Version": "2012-10-17"," + 
        "Id": "custom-policy-2016-12-07"," + 
        "Statement": [{" + 
            "Sid": "Enable IAM User Permissions"," + 
            "Effect": "Allow"," + 
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"," + 
                "Action": ["kms:*"]," + 
                "Resource": "*"}," + 
            }," + 
            "Sid": "Allow access for Key Administrators"," + 
            "Effect": "Allow"," + 
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ExampleAdminRole"," + 
                "Resource": "*"}," + 
            }," + 
            "Sid": "Allow use of the key"," + 
            "Effect": "Allow"," + 
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ExamplePowerUserRole"," + 
                "Action": ["kms:Encrypt","kms:Decrypt","kms:ReEncrypt*","kms:GenerateDataKey*","kms:DescribeKey","kms:CreateKey"]}," + 
                "Resource": "*"}," + 
            }," + 
            "Sid": "Allow attachment of persistent resources"," + 
            "Effect": "Allow"," + 
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/ExamplePowerUserRole"," + 
                "Action": ["kms:CreateGrant","kms:ListGrants","kms:RevokeGrant"]}," + 
            "Resource": "*"]," + 
        }," + 
        "Action": ["kms:CreateKey","kms:DescribeKey" + 
            "kms:EnableKey","kms:Encrypt","kms:Decrypt","kms:ReEncrypt*","kms:GenerateDataKey*","kms:DescribeKey","kms:CreateGrant","kms:ListGrants","kms:RevokeGrant"] + 
        "Resource": "*"} + 
    "}
AWS Key Management Service Key Material Examples

Customer master key (CMK) is a logical representation of a master key in AWS KMS. In addition to the master key's identifiers and other metadata including its creation date, description, and key state, a CMK contains the key material used to encrypt and decrypt data. When you create a CMK, by default AWS KMS generates the key material for that CMK. But you can choose to create a CMK without key material and then import your own key material into that CMK.

When you use imported key material, you remain responsible for the key material while allowing AWS KMS to use a copy of it. You might choose to do this for one or more of the following reasons:

- To prove that you generated the key material using a source of randomness that meets your requirements.
- To use key material from your own infrastructure with AWS services, and to use AWS KMS to manage the lifecycle of that key material within AWS.
- To gain the ability to set an expiration time for the key material in AWS and to manually delete it, but to also make it available again in the future. In contrast, scheduling key deletion requires a waiting period of 7 to 30 days, after which you cannot recover the deleted CMK.
- To own the original copy of the key material, and to keep it outside of AWS for additional durability and disaster recovery during the complete lifecycle of the key material.

For information about important differences between CMKs with imported key material and those with key material generated by AWS KMS, see Considerations for Imported Key Material.

The key material you import must be a 256-bit symmetric encryption key.

AWS Key Management Service provides the following APIs for you to create and manage key material:

- GetParametersForImport
- ImportKeyMaterial
- DeleteKeyMaterial

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

Get Parameters for Import

In this example, you get the items you need in order to import key material into AWS KMS from your existing key management infrastructure. For more information about importing key material into AWS KMS, see Importing Key Material in the WS Key Management Service Developer Guide.
You must specify the key ID of the customer master key (CMK) into which you will import key material. This CMK's Origin must be EXTERNAL. You must also specify the wrapping algorithm and type of wrapping key (public key) that you will use to encrypt the key material. You cannot perform this operation on a CMK in a different AWS account.

This operation returns a public key and an import token. Use the public key to encrypt the key material. Store the import token to send with a subsequent ImportKeyMaterial request. The public key and import token from the same response must be used together. These items are valid for 24 hours. When they expire, they cannot be used for a subsequent ImportKeyMaterial request. To get new ones, send another GetParametersForImport request.

This example uses the default client constructor which constructs a AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the GetParametersForImport method with an GetParametersForImportRequest with the identifier of the CMK whose key rotation status you want to retrieve, the algorithm that you will use to encrypt the key material before importing it, and the public key to use to encrypt the key material before importing it. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

The return values are ImportToken, the import token to send with a subsequent ImportKeyMaterial request; KeyId; the ARN of the CMK (the same CMK specified in the request) for which you are retrieving the public key and import token; ParametersValidTo; the time at which the import token and public key are no longer valid; and PublicKey, the public key to use to encrypt the key material before importing it.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.GetParametersForImport(new GetParametersForImportRequest
{
    KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab", // The identifier of the CMK for which to retrieve the public key and import token. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
    WrappingAlgorithm = "RSAES_OAEPS_SHA_1", // The algorithm that you will use to encrypt the key material before importing it.
    WrappingKeySpec = "RSA_2048" // The type of wrapping key (public key) to return in the response.
});
MemoryStream importToken = response.ImportToken; // The import token to send with a subsequent ImportKeyMaterial request.
string keyId = response.KeyId; // The ARN of the CMK for which you are retrieving the public key and import token. This is the same CMK specified in the request.
DateTime parametersValidTo = response.ParametersValidTo; // The time at which the import token and public key are no longer valid.
MemoryStream publicKey = response.PublicKey; // The public key to use to encrypt the key material before importing it.
```

**Get Parameters for Import**

In this example, you import key material into an existing AWS KMS customer master key (CMK) that was created without key material. You cannot perform this operation on a CMK in a different AWS account. For more information about creating CMKs with no key material and then importing key material, see Importing Key Material in the AWS Key Management Service Developer Guide.

Before using this operation, call GetParametersForImport. Its response includes a public key and an import token. Use the public key to encrypt the key material. Then, submit the import token from the same GetParametersForImport response.
When calling this operation, you must specify the following values:

- The key ID or key ARN of a CMK with no key material. Its Origin must be `EXTERNAL`.
  
  To create a CMK with no key material, call `CreateKey` and set the value of its `Origin` parameter to `EXTERNAL`. To get the Origin of a CMK, call `DescribeKey`.

- The encrypted key material. To get the public key to encrypt the key material, call `GetParametersForImport`.

- The import token that `GetParametersForImport` returned. This token and the public key used to encrypt the key material must have come from the same response.

- Whether the key material expires and if so, when. If you set an expiration date, you can change it only by reimporting the same key material and specifying a new expiration date. If the key material expires, AWS KMS deletes the key material and the CMK becomes unusable. To use the CMK again, you must reimport the same key material.

When this operation is successful, the CMK's key state changes from `PendingImport` to `Enabled`, and you can use the CMK. After you successfully import key material into a CMK, you can reimport the same key material into that CMK, but you cannot import different key material.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in `Credential and Profile Resolution` (p. 17).

Call the `ImportKeyMaterial` method with an `ImportKeyMaterialRequest` with `EncryptedKeyMaterial`, the encrypted key material to import; `ExpirationModel`, the value that specifies whether the key material expires; `ImportToken`, the import token that you received in the response to a previous `GetParametersForImport` request; `KeyId`, the identifier of the CMK to import the key material into. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.GetParametersForImport(new GetParametersForImportRequest
{
  KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab", // The identifier of the CMK for which to retrieve the public key and import token. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
  WrappingAlgorithm = "RSAES_OAEP_SHA_1", // The algorithm that you will use to encrypt the key material before importing it.
  WrappingKeySpec = "RSA_2048" // The type of wrapping key (public key) to return in the response.
});
MemoryStream importToken = response.ImportToken; // The import token to send with a subsequent ImportKeyMaterial request.
string keyId = response.KeyId; // The ARN of the CMK for which you are retrieving the public key and import token. This is the same CMK specified in the request.
DateTime parametersValidTo = response.ParametersValidTo; // The time at which the import token and public key are no longer valid.
MemoryStream publicKey = response.PublicKey; // The public key to use to encrypt the key material before importing it.
```

Delete Imported Key Material

In this example, you delete key material that you previously imported and makes the specified customer master key (CMK) unusable. For more information about importing key material into AWS KMS, see `Importing Key Material` in the `AWS Key Management Service Developer Guide`. 
When the specified CMK is in the `PendingDeletion` state, this operation does not change the CMK's state. Otherwise, it changes the CMK's state to `PendingImport`.

After you delete key material, you can use `ImportKeyMaterial` method with an `ImportKeyMaterialRequest` to reimport the same key material into the CMK.

When you import key material, you have the option of specifying a time at which the key material expires. When the key material expires, AWS KMS deletes the key material and the customer master key (CMK) becomes unusable. You can also delete key material on demand. Whether you wait for the key material to expire or you delete it manually, the effect is the same. AWS KMS deletes the key material, the CMK's key state changes to pending import, and the CMK is unusable. To use the CMK again, you must reimport the same key material.

Deleting key material affects the CMK right away, but you can reverse the deletion of key material by reimporting the same key material into the CMK. In contrast, scheduling key deletion for a CMK is irreversible. It deletes the key material and all metadata associated with the CMK, and requires a waiting period of between 7 and 30 days.

This example uses the default client constructor which constructs an `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in `Credential and Profile Resolution (p. 17)`.

Call the `DeleteKeyMaterial` method with an `DeleteKeyMaterialRequest` with the identifier of the CMK whose imported key material you are deleting. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.DeleteImportedKeyMaterial(new DeleteImportedKeyMaterialRequest
{
    KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK whose imported key material you are deleting. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
});
```

**AWS Key Management Service Tag Examples**

You can use the operations in the AWS Key Management Service (AWS KMS) API to manage CMK tags. You can use the APIs to add, delete, and list tags for the CMKs that you manage. You cannot tag AWS managed CMKs.

Each tag consists of a tag key and a tag value, both of which you define. For example, the tag key might be "Cost Center" and the tag value might be "87654."

You might use a tag to categorize and track your AWS costs. You can apply tags that represent business categories (such as cost centers, application names, or owners) to organize your costs across multiple services. When you add tags to your AWS resources, AWS generates a cost allocation report with usage and costs aggregated by tags. You can use this report to view your AWS KMS costs in terms of projects or applications, instead of viewing all AWS KMS costs as a single line item.

For more information about using tags for cost allocation, see `<problematic>` in the `AWS Billing and Cost Management User Guide`.

AWS Key Management Service provides the following APIs for you to create and manage tags:
• TagResource
• ListResourceTags
• UntagResource

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

Create a Grant

In this example, you add or overwrite one or more tags for the specified customer master key (CMK). You cannot perform this operation on a CMK in a different AWS account.

Each tag consists of a tag key and a tag value. Tag keys and tag values are both required, but tag values can be empty (null) strings.

You cannot use the same tag key more than once per CMK.

For more information, see Tagging Keys in the AWS Key Management Service Developer Guide.

This example uses the default client constructor which constructs a AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the TagResource method with an TagResourceRequest with the identity of the CMK you are tagging, either the key ID or the Amazon Resource Name (ARN) of the CMK the grant applies to, and a list of tag keys and tag values.

Code

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.TagResource(new TagResourceRequest
{
    KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab", // The identifier of the CMK you are tagging. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
    Tags = new List<Tag>
    {
        new Tag
        {
            TagKey = "Purpose",
            TagValue = "Test"
        }
    } // A list of tags.
});
```

List Resource Tags

In this example, you get a list of all tags for the specified customer master key (CMK).

You cannot perform this operation on a CMK in a different AWS account.

The request parameters are * KeyId (required), a unique identifier for the customer master key (CMK). * Limit (optional), the maximum number of items to return. When this value is present, AWS KMS does not return more than the specified number of items, but it might return fewer. * Marker (optional) use in a subsequent request after you receive a response with truncated results. Set it to the value of NextMarker from the truncated response you just received.
The response values are * :code"Tags*, a list of tags. Each tag consists of a tag key and a tag value. *NextMarker*, when :code:`Truncated` is :code:`true`, this element is present and contains the value to use for the Marker parameter in a subsequent request. *Truncated*, a flag that indicates whether there are more items in the list. When this value is :code:`true`, the list in this response is truncated. To get more items, pass the value of the NextMarker element in this response to the Marker parameter in a subsequent request.

This example uses the default client constructor which constructs an :class:`AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the :meth:`ListResourceTags` method with an :class:`ListResourceTagsRequest` specifying the key ID or the Amazon Resource Name (ARN) of the CMK whose key policies you want to list.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.ListResourceTags(new ListResourceTagsRequest
{
    KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab" // The identifier of the CMK whose tags
    you are listing. You can use the key ID or the Amazon Resource Name (ARN) of the CMK.
});
List<Tag> tags = response.Tags; // A list of tags.
bool truncated = response.Truncated; // A boolean that indicates whether there are more
items in the list. Returns true when there are more items, or false when there are not.
```

**Create a Grant**

In this example, you remove the specified tag or tags from the specified customer master key (CMK). You cannot perform this operation on a CMK in a different AWS account.

To remove a tag, you specify the tag key for each tag to remove. You do not specify the tag value. To overwrite the tag value for an existing tag, use :meth:`TagResource`.

For more information, see Tagging Keys in the AWS Key Management Service Developer Guide.

This example uses the default client constructor which constructs an :class:`AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the :meth:`UntagResource` method with an :class:`UntagResourceRequest` with the identity of the CMK you are untagging, either the key ID or the Amazon Resource Name (ARN) of the CMK the grant applies to, and a list of tag keys.

**Code**

```csharp
AmazonKeyManagementServiceClient client = new AmazonKeyManagementServiceClient();
var response = client.UntagResource(new UntagResourceRequest
{
    KeyId = "1234abcd-12ab-34cd-56ef-1234567890ab", // The identifier of the CMK whose tags
    you are removing.
    TagKeys = new List<string> {
        "Purpose",
        "CostCenter"
    } // A list of tag keys. Provide only the tag keys, not the tag values.
});
```
Using AWS KMS Keys with the AmazonS3EncryptionClient in the AWS SDK for .NET

The AmazonS3EncryptionClient class implements the same interface as the standard AmazonS3Client. This means it's easy to switch to the AmazonS3EncryptionClient class. In fact, your application code won't be aware of the encryption and decryption happening automatically in the client.

You can use an AWS KMS key as your master key when you use the AmazonS3EncryptionClient class for client-side encryption. All you have to do is create an EncryptionMaterials object that contains a KMS key ID. Then you pass the EncryptionMaterials object to the constructor of the AmazonS3EncryptionClient.

The main 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. It's done by AWS. A second advantage is that it makes the AWS SDK for .NET's AmazonS3EncryptionClient class interoperable with the AWS SDK for Java's AmazonS3EncryptionClient class. This means you can encrypt with the AWS SDK for Java and decrypt with the AWS SDK for .NET, and vice versa.

**Note**

The AWS SDK for .NET's AmazonS3EncryptionClient supports KMS master keys only when run in metadata mode. The instruction file mode of the AWS SDK for .NET's AmazonS3EncryptionClient is still incompatible with the AWS SDK for Java's AmazonS3EncryptionClient.

For more information about client-side encryption with the AmazonS3EncryptionClient class, and how envelope encryption works, see Client Side Data Encryption with AWS SDK for .NET and Amazon S3.

The following example demonstrates how to use AWS KMS keys with the AmazonS3EncryptionClient class. Your project must reference the latest version of the AWSSDK.KeyManagementService Nuget package to use this feature.

```csharp
using System;
using System.Collections.Specialized;
using System.Collections.Generic;
using System.Configuration;
using System.IO;
using System.Linq;
using System.Net;
using Amazon;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.S3.Encryption;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

namespace S3Sample1
{
    class S3Sample
    {
        public static void Main(string[] args)
        {
            string kmsKeyId = null;
```
using (var kmsClient = new AmazonKeyManagementServiceClient())
{
    var response = kmsClient.CreateKey(new CreateKeyRequest());
    kmsKeyId = response.KeyMetadata.KeyId;

    var keyMetadata = response.KeyMetadata; // An object that contains information about the CMK created by this operation.
    var bucketName = "<s3bucket>",
    var objectKey = "key",
    var kmsEncryptionMaterials = new EncryptionMaterials(kmsKeyId);
    // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
    var config = new AmazonS3CryptoConfiguration()
    {
        StorageMode = CryptoStorageMode.ObjectMetadata
    };

    using (var s3Client = new AmazonS3EncryptionClient(config, kmsEncryptionMaterials))
    {
        // encrypt and put object
        var putRequest = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = objectKey,
            ContentBody = "object content"
        };
        s3Client.PutObject(putRequest);

        // get object and decrypt
        var getRequest = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = objectKey
        };

        using (var getResponse = s3Client.GetObject(getRequest))
        using (var stream = getResponse.ResponseStream)
        using (var reader = new StreamReader(stream))
        {
            Console.WriteLine(reader.ReadToEnd());
        }
    }

    Console.WriteLine("Press any key to continue...");
    Console.ReadKey();
}
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 `CreateTableResponse` method returns an instance of the `CreateTableResponse` class.

```csharp
// 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.
    }
}
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.

```csharp
// 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.

The following examples show you how to use the document model to insert items and get items in tables in DynamoDB.

Inserting an Item into a Table

In the following example, an item is inserted into the table through the PutItem method of the Table class. The PutItem method takes an instance of the Document class; the Document class is simply a collection of initialized attributes. To determine the table to insert the item into, call the LoadTable
method of the `Table` class, specifying an instance of the `AmazonDynamoDBClient` class and the name of the target table in DynamoDB.

```csharp
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;
var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = new Document();
item["Id"] = 3;
item["Type"] = "Horse";
item["Name"] = "Shadow";
table.PutItem(item);
```

### Getting an Item from a Table

In the following example, the item is retrieved through the `GetItem` method of the `Table` class. To determine the item to get, 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.

```csharp
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;
var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = table.GetItem(3, "Horse");
Console.WriteLine("Id = " + item["Id"]);
Console.WriteLine("Type = " + item["Type"]);
Console.WriteLine("Name = " + item["Name"]);
```

The preceding example implicitly converts the attribute values for `Id`, `Type`, and `Name` to strings for the `WriteLine` method. You can do explicit conversions by using the various `AsType` methods of the `DynamoDBEntry` class. For example, you could explicitly convert the attribute value for `Id` from a primitive data type to an integer through the `AsInt` method:

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

Or, you could simply perform an explicit cast here by using `(int)`:  

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

### 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.
The following examples show you how to define a .NET class that represents an item, use an instance of the .NET class to insert an item, and use an instance of a .NET object to get an item from a table in DynamoDB.

**Defining a .NET Class that Represents an Item in a Table**

In the following example, the `DynamoDBTable` attribute specifies the table name, while the `DynamoDBHashKey` and `DynamoDBRangeKey` attributes model the table's hash-and-range primary key.

```csharp
// using Amazon.DynamoDBv2.DataModel;
[DynamoDBTable("AnimalsInventory")]
class Item
{
    [DynamoDBHashKey]
    public int Id { get; set; }
    [DynamoDBRangeKey]
    public string Type { get; set; }
    public string Name { get; set; }
}
```

**Using an Instance of the .NET Class to Insert an Item into a Table**

In this example, the item is inserted through the `Save` method of the `DynamoDBContext` class, which takes an initialized instance of the .NET class that represents the item. (The instance of the `DynamoDBContext` class is initialized with an instance of the `AmazonDynamoDBClient` class.)

```csharp
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DataModel;
var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
var item = new Item
{
    Id = 4,
    Type = "Fish",
    Name = "Goldie"
};
context.Save(item);
```

**Using an Instance of a .NET Object to Get an Item from a Table**

In this example, the item is retrieved through the `Load` method of the `DynamoDBContext` class, which takes a partially initialized instance of the .NET class that represents the hash-and-range primary key of the item to be retrieved. (As shown previously, the instance of the `DynamoDBContext` class is initialized with an instance of the `AmazonDynamoDBClient` class.)

```csharp
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DataModel;
var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
var item = context.Load<Item>(4, "Fish");
Console.WriteLine("Id = {0}", item.Id);
Console.WriteLine("Type = {0}", item.Type);
Console.WriteLine("Name = {0}", item.Name);
```
More Info

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 (p. 79)
- JSON Support in Amazon DynamoDB with the AWS SDK for .NET (p. 89)
- Managing ASP.NET Session State with Amazon DynamoDB (p. 91)

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

Using Expressions with Amazon DynamoDB and the AWS SDK for .NET

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 (p. 80)
- Get a Single Item by Using Expressions and the Item's Primary Key (p. 82)
- Get Multiple Items by Using Expressions and the Table's Primary Key (p. 83)
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.

```json
{
    "Id": {
        "N": "205"
    },
    "Title": {
        "S": "20-Bicycle 205"
    },
    "Description": {
        "S": "205 description"
    },
    "BicycleType": {
        "S": "Hybrid"
    },
    "Brand": {
        "S": "Brand-Company C"
    },
    "Price": {
        "N": "500"
    },
    "Gender": {
        "S": "B"
    },
    "Color": {
        "SS": [
            "Red",
            "Black"
        ]
    },
    "ProductCategory": {
        "S": "Bike"
    },
    "InStock": {
        "BOOL": true
    },
    "QuantityOnHand": {
        "N": "1"
    },
    "RelatedItems": {
        "NS": [
            "341",
            "472",
            "649"
        ]
    },
    "Pictures": {
        "L": [
            { ...
```
"M": {
    "FrontView": {
        "S": "http://example/products/205_front.jpg"
    }
},
"M": {
    "RearView": {
        "S": "http://example/products/205_rear.jpg"
    }
},
"M": {
    "SideView": {
        "S": "http://example/products/205_left_side.jpg"
    }
},
"ProductReviews": {
    "M": {
        "FiveStar": {
            "SS": [
                "Excellent! Can't recommend it highly enough! Buy it!",
                "Do yourself a favor and buy this."
            ]
        },
        "OneStar": {
            "SS": [
                "Terrible product! Do not buy this."
            ]
        }
    }
},
"Id": {
    "N": "301"
},
"Title": {
    "S": "18-Bicycle 301"
},
"Description": {
    "S": "301 description"
},
"BicycleType": {
    "S": "Road"
},
"Brand": {
    "S": "Brand-Company C"
},
"Price": {
    "N": "185"
},
"Gender": {
    "S": "F"
},
"Color": {
    "SS": [
        "Blue",
        "Silver"
    ]
}
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.

```csharp
// 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 (p. 85).

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.

```csharp
// 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>
    {
```

83
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 (p. 85).

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.

```csharp
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 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` (p. 85).

### 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` (p. 82), `Get Multiple Items by Using Expressions and the Table's Primary Key` (p. 83), and `Get Multiple Items by Using Expressions and Other Item Attributes` (p. 84).

```csharp
// 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);
    }
}
```
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`, `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.

```csharp
// 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:

```csharp
// 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 with the AWS SDK for .NET

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 (p. 89)
- Insert JSON Format Data into a DynamoDB Table (p. 90)
- DynamoDB Data Type Conversions to JSON (p. 90)
- More Info (p. 90)

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",
```
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:

```csharp
// 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`:

```csharp
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
Managing ASP.NET Session State with Amazon DynamoDB

ASP.NET applications often store session state data in memory. However, this approach doesn't scale well. After the application grows beyond a single web server, the session state must be shared between servers. A common solution is to set up a dedicated session-state server with Microsoft SQL Server, but this approach also has drawbacks: you must administer another machine; the session-state server is a single point of failure; and the session-state server itself can become a performance bottleneck.

DynamoDB, a NoSQL database store from Amazon Web Services (AWS), provides an effective solution for sharing session state across web servers without incurring any of these drawbacks.

Note
Regardless of the solution you choose, be aware that Amazon DynamoDB enforces limits on the size of an item. None of the records you store in DynamoDB can exceed this limit. For more information, see Limits in DynamoDB in the Amazon DynamoDB Developer Guide.

The AWS SDK for .NET includes AWS.SessionProvider.dll, which contains an ASP.NET session state provider. It also includes the AmazonDynamoDBSessionProviderSample sample, which demonstrates how to use Amazon DynamoDB as a session state provider.

For more information about using session state with ASP.NET applications, go to the MSDN documentation.

Create the ASP.NET_SessionState Table

When your application starts, it looks for an Amazon DynamoDB table named, by default, ASP.NET_SessionState. We recommend you create this table before you run your application for the first time.

To create the ASP.NET_SessionState table

1. Choose Create Table. The Create Table wizard opens.
2. In the Table name text box, enter ASP.NET_SessionState.
3. In the Primary key field, enter SessionId and set the type to String.
4. When all your options are entered as you want them, choose Create.

The ASP.NET_SessionState table is ready for use when its status changes from CREATING to ACTIVE.

Note
If you decide not to create the table beforehand, the session state provider will create the table during its initialization. See the web.config options below for a list of attributes that act as configuration parameters for the session state table. If the provider creates the table, it will use these parameters.

Configure the Session State Provider

To configure an ASP.NET application to use DynamoDB as the session-state server

1. Add references to both AWSSDK.dll and AWS.SessionProvider.dll to your Visual Studio ASP.NET project. These assemblies are available by installing the AWS SDK for .NET (p. 5). You can also install them by using NuGet (p. 5).
In earlier versions of the SDK, the functionality for the session state provider was contained in `AWS.Extension.dll`. To improve usability, the functionality was moved to `AWS.SessionProvider.dll`. For more information, see the blog post `AWS.Extension Renaming`.

2. Edit your application's `Web.config` file. In the `system.web` element, replace the existing `sessionState` element with the following XML fragment:

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

The profile represents the AWS credentials that are used to communicate with DynamoDB to store and retrieve the session state. If you are using the AWS SDK for .NET and are specifying a profile in the `appSettings` section of your application's `Web.config` file, you do not need to specify a profile in the `providers` section; the AWS .NET client code will discover it at run time. For more information, see Configuring Your AWS SDK for .NET Application (p. 8).

If the web server is running on an Amazon EC2 instance configured to use IAM roles for EC2 instances, then you do not need to specify any credentials in the `Web.config` file. In this case, the AWS .NET client will use the IAM role credentials. For more information, see Granting Access Using an IAM Role (p. 148) and Security Considerations (p. 93).

**Web.config Options**

You can use the following configuration attributes in the `providers` section of your `Web.config` file:

- **AWSAccessKey**
  
  Access key ID to use. This can be set either in the `providers` or `appSettings` section. We recommend not using this setting. Instead, specify credentials by using `AWSProfileName` to specify a profile.

- **AWSSecretKey**
  
  Secret key to use. This can be set either in the `providers` or `appSettings` section. We recommend not using this setting. Instead, specify credentials by using `AWSProfileName` to specify a profile.

- **AWSProfileName**
  
  The profile name associated with the credentials you want to use. For more information, see Configuring Your AWS SDK for .NET Application (p. 8).

- **Region**
  
  Required string attribute. The AWS region in which to use Amazon DynamoDB. For a list of AWS regions, see Regions and Endpoints: DynamoDB.

- **Application**
  
  Optional string attribute. The value of the `Application` attribute is used to partition the session data in the table so that the table can be used for more than one application.

- **Table**
  
  Optional string attribute. The name of the table used to store session data. The default is `ASP.NET_SessionState`. 
ReadCapacityUnits

Optional int attribute. The read capacity units to use if the provider creates the table. The default is 10.

WriteCapacityUnits

Optional int attribute. The write capacity units to use if the provider creates the table. The default is 5.

CreateIfNotExist

Optional boolean attribute. The CreateIfNotExist attribute controls whether the provider will auto-create the table if it doesn’t exist. The default is true. If this flag is set to false and the table doesn’t exist, an exception will be thrown.

Security Considerations

After the DynamoDB table is created and the application is configured, sessions can be used as with any other session provider.

As a security best practice, we recommend you run your applications with the credentials of an IAM User Guide user. You can use either the IAM Management Console or the AWS Toolkit for Visual Studio to create IAM users and define access policies.

The session state provider needs to be able to call the DeleteItem, DescribeTable, GetItem, PutItem, and UpdateItem operations for the table that stores the session data. The sample policy below can be used to restrict the IAM user to only the operations needed by the provider for an instance of DynamoDB running in us-east-1:

```json
{
  "Version" : "2012-10-17",
  "Statement" : [
    {
      "Sid" : "1",
      "Effect" : "Allow",
      "Action" : [
        "dynamodb:DeleteItem",
        "dynamodb:DescribeTable",
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem"
      ],
    }
  ]
}
```

Elastic Beanstalk Examples

The AWS SDK for .NET supports AWS Elastic Beanstalk, which enables you to quickly deploy and manage applications in the AWS Cloud without worrying about the infrastructure that runs those applications. Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

The following examples demonstrate how to use the AWS SDK for .NET to create and use Elastic Beanstalk topics.
Prerequisite Tasks
Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

For related API reference information, see Amazon.ElasticBeanstalk and Amazon.ElasticBeanstalk.Model in the AWS SDK for .NET API Reference.

Elastic Beanstalk Environment Management Examples

These examples use the default client constructor which constructs a AmazonElasticBeanstalkClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Abort an Elastic Beanstalk Environment Update
In this example, you cancel in-progress environment configuration update or application version deployment.

Call the AbortEnvironmentUpdate method with an AbortEnvironmentUpdateRequest containing the environment name of the update to be aborted.

Code

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.AbortEnvironmentUpdate(new AbortEnvironmentUpdateRequest
{
    EnvironmentName = "my-env"
});
```

Check DNS Availability
In this example, you check if the specified Canonical Name record (CNAME) is available in the Domain Name System (DNS).

Call the CheckDNSAvailability method with an CheckDNSAvailabilityRequest containing the CNAME to check.

Code

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
```
Create an Application Environment

In this example, you launch an environment for the specified application using the specified configuration.

Call the `CreateEnvironment` method with a `CreateEnvironmentRequest` containing the application name, Canonical Name record (CNAME) prefix, environment name, the name of the solution stack and the version label.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.CreateEnvironment(new CreateEnvironmentRequest
{
    ApplicationName = "my-app",
    CNAMEPrefix = "my-app",
    EnvironmentName = "my-env",
    SolutionStackName = "64bit Amazon Linux 2015.03 v2.0.0 running Tomcat 8 Java 8",
    VersionLabel = "v1"
});
string applicationName = response.ApplicationName;
string cname = response.CNAME;
DateTime dateCreated = response.DateCreated;
DateTime dateUpdated = response.DateUpdated;
string environmentId = response.EnvironmentId;
string environmentName = response.EnvironmentName;
string health = response.Health;
string solutionStackName = response.SolutionStackName;
string status = response.Status;
EnvironmentTier tier = response.Tier;
string versionLabel = response.VersionLabel;
```

Delete an Environment Configuration

In this example, you delete the draft configuration associated with the running environment.

Updating a running environment with any configuration changes creates a draft configuration set. You can get the draft configuration using `DescribeConfigurationSettings` while the update is in progress or if the update fails. The `DeploymentStatus` for the draft configuration indicates whether the deployment is in progress or has failed. The draft configuration remains in existence until it is deleted with this action.

Call the `DeleteEnvironmentConfiguration` method with a `DeleteEnvironmentConfigurationRequest` containing the application name and environment name.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
```
Get the Descriptions of Elastic Beanstalk Environmental Health

In this example, you get information about the overall health of the specified environment. The DescribeEnvironmentHealth operation is only available with AWS Elastic Beanstalk Enhanced Health.

Call the DescribeEnvironmentHealth method with a DescribeEnvironmentHealthRequest containing the attribute names of the response elements to return. To retrieve all attributes, set to All. If no attribute names are specified, returns the name of the environment. You must also specify either the environment name or ID.

Code

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.DescribeEnvironmentHealth(new DescribeEnvironmentHealthRequest
{
    AttributeNames = new List<string> { "All" },
    EnvironmentName = "my-env"
});
ApplicationMetrics applicationMetrics = response.ApplicationMetrics;
List<string> causes = response.Causes;
string color = response.Color;
string environmentName = response.EnvironmentName;
string healthStatus = response.HealthStatus;
InstanceHealthSummary instancesHealth = response.InstancesHealth;
DateTime refreshedAt = response.RefreshedAt;
```

Get the Descriptions of Elastic Beanstalk Environmental Resources

In this example, you get the AWS resources for an environment.

Call the DescribeEnvironmentResources method with a DescribeEnvironmentResourcesRequest containing the environment name. The response contains the AutoScalingGroups used by the environment. the name of the environment, the Amazon EC2 instances used by the environment, the Auto Scaling launch configurations in use by the environment, the LoadBalancers in use by the environment, the queues used by the environment and the AutoScaling triggers in use by the environment.

Code

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.DescribeEnvironments(new DescribeEnvironmentsRequest
{
    EnvironmentName = "my-env"
});
```
Get the Descriptions of Elastic Beanstalk Environments

In this example, you get descriptions for existing environments.

Call the `DescribeEnvironments` method with a `DescribeEnvironmentsRequest` containing a list of environment names. The response contains a list of `EnvironmentDescription` objects describing the environments.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.DescribeEnvironments(new DescribeEnvironmentsRequest
{
    EnvironmentNames = new List<string> {
        "my-env"
    }
});
List<EnvironmentDescription> environments = response.Environments;
```

Get the Descriptions of Elastic Beanstalk Instances Health

In this example, you get detailed information about the health of instances in your AWS Elastic Beanstalk. This operation requires enhanced health reporting.

Call the `DescribeInstancesHealth` method with a `DescribeInstancesHealthRequest` containing an environment name and a list of attribute names. The response contains a list of `SingleInstanceHealth` objects describing the health of the instances.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.DescribeInstancesHealth(new DescribeInstancesHealthRequest
{
    AttributeNames = new List<string> {
        "All"
    },
    EnvironmentName = "my-env"
});
List<SingleInstanceHealth> instanceHealthList = response.InstanceHealthList;
DateTime refreshedAt = response.RefreshedAt;
```

List the Available Solution Stacks

In this example, you get a list of the available solution stack names, with the public version first and then in reverse chronological order.
Call the `ListAvailableSolutionStacks` method with a `ListAvailableSolutionStacksRequest`. The response contains a list of available solution stacks and a list of `SolutionStackDescription` objects describing them.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.ListAvailableSolutionStacks(new ListAvailableSolutionStacksRequest { });
List<SolutionStackDescription> solutionStackDetails = response.SolutionStackDetails;
List<string> solutionStacks = response.SolutionStacks;
```

**Request Environment Info**

In this example, you initiate a request to compile the specified type of information of the deployed environment. Setting the `InfoType` to "tail" compiles the last lines from the application server log files of every Amazon EC2 instance in your environment.

Setting the `InfoType` to "bundle" compresses the application server log files for every Amazon EC2 instance into a .zip file. Legacy and .NET containers do not support bundle logs.

Use `RetrieveEnvironmentInfo` to obtain the set of logs.

Call the `RequestEnvironmentInfo` method with a `RequestEnvironmentInfoRequest` containing the name of the environment and the info type, either "tail" or "bundle".

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.RequestEnvironmentInfo(new RequestEnvironmentInfoRequest { 
    EnvironmentName = "my-env",
    InfoType = "tail"
 });
```

**Restart an Appserver**

In this example, you cause the environment to restart the application container server running on each Amazon EC2 instance.

Call the `RestartAppServer` method with a `RestartAppServerRequest` containing the name of the environment.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.RestartAppServer(new RestartAppServerRequest { 
    EnvironmentName = "my-env"
 });
```

**Swap Environment CNAMEs**

In this example, you swap the Canonical Name record (CNAME) of two environments.
Call the `SwapEnvironmentCNAMEs` method with a `SwapEnvironmentCNAMEsRequest` containing the destination and source environment names to swap.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.SwapEnvironmentCNAMEs(new SwapEnvironmentCNAMEsRequest
{
    DestinationEnvironmentName = "my-env-green",
    SourceEnvironmentName = "my-env-blue"
});
```

**Terminate an Environment**

In this example, you terminate an environment.

Call the `TerminateEnvironment` method with a `TerminateEnvironmentRequest` containing the name of the environment to terminate.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.TerminateEnvironment(new TerminateEnvironmentRequest
{
    EnvironmentName = "my-env"
});
bool abortableOperationInProgress = response.AbortableOperationInProgress;
string applicationName = response.ApplicationName;
DateTime dateCreated = response.DateCreated;
DateTime dateUpdated = response.DateUpdated;
string endpointURL = response.EndpointURL;
string environmentId = response.EnvironmentId;
string environmentName = response.EnvironmentName;
string health = response.Health;
string solutionStackName = response.SolutionStackName;
string status = response.Status;
EnvironmentTier tier = response.Tier;
```

**Update an Environment**

In this example, you update the environment version and update select configuration option values in the specified environment.

Attempting to update both the release and configuration is not allowed and AWS Elastic Beanstalk returns an `InvalidParameterCombination` error.

When updating the configuration settings to a new template or individual settings, a draft configuration is created and `DescribeConfigurationSettings` for this environment returns two setting descriptions with different `DeploymentStatus` values.

You can update an environment to a new version. Call the `UpdateEnvironment` method with a `UpdateEnvironmentRequest` containing the name of the environment to update and the new version number.
Code

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();

var response = client.UpdateEnvironment(new UpdateEnvironmentRequest
{
    EnvironmentName = "my-env",
    VersionLabel = "v2"
});

string applicationName = response.ApplicationName;
string cname = response.CNAME;
DateTime dateCreated = response.DateCreated;
DateTime dateUpdated = response.DateUpdated;
string endpointURL = response.EndpointURL;
string environmentId = response.EnvironmentId;
string environmentName = response.EnvironmentName;
string health = response.Health;
string solutionStackName = response.SolutionStackName;
string status = response.Status;
EnvironmentTier tier = response.Tier;
string versionLabel = response.VersionLabel;

You can update environment configuration option settings. Call the UpdateEnvironment method with a UpdateEnvironmentRequest containing the name of the environment to update and the new configuration option settings.

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();

var response = client.UpdateEnvironment(new UpdateEnvironmentRequest
{
    EnvironmentName = "my-env",
    OptionSettings = new List<ConfigurationOptionSetting> {
        new ConfigurationOptionSetting {
            Namespace = "aws:elb:healthcheck",
            OptionName = "Interval",
            Value = "15"
        },
        new ConfigurationOptionSetting {
            Namespace = "aws:elb:healthcheck",
            OptionName = "Timeout",
            Value = "8"
        },
        new ConfigurationOptionSetting {
            Namespace = "aws:elb:healthcheck",
            OptionName = "HealthyThreshold",
            Value = "2"
        },
        new ConfigurationOptionSetting {
            Namespace = "aws:elb:healthcheck",
            OptionName = "UnhealthyThreshold",
            Value = "3"
        }
    }
});

bool abortableOperationInProgress = response.AbortableOperationInProgress;
string applicationName = response.ApplicationName;
string cname = response.CNAME;
DateTime dateCreated = response.DateCreated;
DateTime dateUpdated = response.DateUpdated;
string endpointURL = response.EndpointURL;
string environmentId = response.EnvironmentId;
```
Elastic Beanstalk Application Management Examples

The AWS SDK for .NET supports AWS Elastic Beanstalk, which enables you to quickly deploy and manage applications in the AWS Cloud without worrying about the infrastructure that runs those applications. Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

These examples use the default client constructor which constructs a `AmazonElasticBeanstalkClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

**Prerequisite Tasks**

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

For related API reference information, see `Amazon.ElasticBeanstalk` and `Amazon.ElasticBeanstalk.Model` in the AWS SDK for .NET API Reference.

**Create an Application**

In this example, you create an application that has one configuration template named default and no application versions.

Call the `CreateApplication` method with an `CreateApplicationRequest` containing the application name and its description.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();// to-create-a-
new-application-1456268895683

var response = client.CreateApplication(new CreateApplicationRequest
{
    ApplicationName = "my-app",
    Description = "my application"
});

ApplicationDescription application = response.Application;
```

**Create an Application Version**

In this example, you create an application version for the specified application. You can create an application version from a source bundle in Amazon S3, a commit in AWS CodeCommit, or the output of an AWS CodeBuild build as follows:

- Specify a commit in an AWS CodeCommit repository with `SourceBuildInformation`.
- Specify a build in an AWS CodeBuild with `SourceBuildInformation` and `BuildConfiguration`.
- Specify a source bundle in S3 with `SourceBundle`
• Omit both `SourceBuildInformation` and `SourceBundle` to use the default sample application.

Once you create an application version with a specified Amazon S3 bucket and key location, you cannot change that Amazon S3 location. If you change the Amazon S3 location, you receive an exception when you attempt to launch an environment from the application version.

Call the `CreateApplication` method with an `CreateApplicationRequest` containing the application name and description, `AutoCreateApplication` and `Process` set to `true` and `SourceBundle` set to the S3 location of the source bundle.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.CreateApplicationVersion(new CreateApplicationVersionRequest
{
    ApplicationName = "my-app",
    AutoCreateApplication = true,
    Description = "my-app-v1",
    Process = true,
    SourceBundle = new S3Location {
        S3Bucket = "my-bucket",
        S3Key = "sample.war"
    },
    VersionLabel = "v1"
});
ApplicationVersionDescription applicationVersion = response.ApplicationVersion;
```

**Create a Storage Location**

In this example, you create the Amazon S3 storage location for the account.

The location is used to store user log files.

Call the `CreateStorageLocation` method with an `CreateStorageLocationRequest`. The name of the S3 bucket that was created is returned in the response.

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.CreateStorageLocation(new CreateStorageLocationRequest
{
});
string s3Bucket = response.S3Bucket;
```

**Delete an Application**

In this example, you delete the specified application along with all associated versions and configurations. The application versions will not be deleted from your Amazon S3 bucket.

You cannot delete an application that has a running environment.

Call the `DeleteApplication` method with an `DeleteApplicationRequest` containing the application name.

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
```
Delete an Application Version

In this example, you delete the specified version from the specified application.

You cannot delete an application version that is associated with a running environment.

Call the `DeleteApplicationVersion` method with an `DeleteApplicationVersionRequest` containing the application name, version label and a flag to indicate if the source bundle should be deleted.

Code

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.DeleteApplicationVersion(new DeleteApplicationVersionRequest
{
    ApplicationName = "my-app",
    DeleteSourceBundle = true,
    VersionLabel = "22a0-stage-150819_182129"
});
```

Get the Descriptions of Elastic Beanstalk Applications

In this example, you get the descriptions of existing applications.

Call the `DescribeApplications` method with the default `DescribeApplicationsRequest`. A list of application descriptions is returned in the response.

Code

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.DescribeApplications(new DescribeApplicationsRequest
{
});
List<ApplicationDescription> applications = response.Applications;
```

Get the Descriptions of Elastic Beanstalk Application Versions

In this example, you get the descriptions of application versions.

Call the `DescribeApplicationVersions` method with a `DescribeApplicationVersionsRequest` containing the name of the application and a list of version labels. A list of application version descriptions is returned in the response.

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.DescribeApplicationVersions(new DescribeApplicationVersionsRequest
{
});
```
Elastic Beanstalk Configuration Management Examples

The AWS SDK for .NET supports AWS Elastic Beanstalk, which enables you to quickly deploy and manage applications in the AWS Cloud without worrying about the infrastructure that runs those applications. Elastic Beanstalk reduces management complexity without restricting choice or control. You simply upload your application, and Elastic Beanstalk automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring.

The following examples demonstrate how to use the AWS SDK for .NET to create and use Elastic Beanstalk topics.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

For related API reference information, see Amazon.ElasticBeanstalk and Amazon.ElasticBeanstalk.Model in the AWS SDK for .NET API Reference.

Create a Configuration Template

In this example, you create a configuration template. Templates are associated with a specific application and are used to deploy different versions of the application with the same configuration settings.

Related Topics

- DescribeConfigurationOptions
- DescribeConfigurationSettings
- ListAvailableSolutionStacks

Call the CreateConfigurationTemplate method with an CreateConfigurationTemplateRequest containing the application name, environment ID, and template name.

Code

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.CreateConfigurationTemplate(new CreateConfigurationTemplateRequest
{
    ApplicationName = "my-app",
    EnvironmentId = "e-rpqsewtp2j",
    TemplateName = "my-app-v1"
});

string applicationName = response.ApplicationName;
DateTime dateCreated = response.DateCreated;
DateTime dateUpdated = response.DateUpdated;
```
Delete a Configuration Template

In this example, you delete the specified configuration template.

When you launch an environment using a configuration template, the environment gets a copy of the template. You can delete or modify the environment's copy of the template without affecting the running environment.

Call the `DeleteConfigurationTemplate` method with a `DeleteConfigurationTemplateRequest` containing the application name and template name.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.DeleteConfigurationTemplate(new DeleteConfigurationTemplateRequest
{
    ApplicationName = "my-app",
    TemplateName = "my-template"
});
```

Get the Descriptions of Elastic Beanstalk Configuration Options

In this example, you get the configuration options that are used in a particular configuration template or environment, or that a specified solution stack defines. The description includes the values the options, their default values, and an indication of the required action on a running environment if an option value is changed.

Call the `DescribeConfigurationOptions` method with a `DescribeConfigurationOptionsRequest` containing the application and environment names. A list of configuration option descriptions is returned in the response.

**Code**

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.DescribeConfigurationOptions(new DescribeConfigurationOptionsRequest
{
    ApplicationName = "my-app",
    EnvironmentName = "my-env"
});
List<ConfigurationOptionDescription> options = response.Options;
```

Get the Descriptions of Elastic Beanstalk Configuration Settings

In this example, you get a description of the settings for the specified configuration set, that is, either a configuration template or the configuration set associated with a running environment.

When describing the settings for the configuration set associated with a running environment, it is possible to receive two sets of setting descriptions. One is the deployed configuration set, and the other
is a draft configuration of an environment that is either in the process of deployment or that failed to deploy.

Call the DescribeConfigurationSettings method with a DescribeConfigurationSettingsRequest containing the application and environment names. A list of configuration settings descriptions is returned in the response.

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.DescribeConfigurationSettings(new
   DescribeConfigurationSettingsRequest
   {
      ApplicationName = "my-app",
      EnvironmentName = "my-env"
   });
List<ConfigurationSettingsDescription> configurationSettings =
   response.ConfigurationSettings;
```

**Update a Configuration Template**

In this example, you update the properties or configuration option values for a configuration template.

If a property (for example, ApplicationName) is not provided, its value remains unchanged. To clear such properties, specify an empty string.

Related Topics

- DescribeConfigurationOptions

Call the UpdateConfigurationTemplate method with a UpdateConfigurationTemplateRequest containing the application name, template name, a new description (optional), a list of configuration options to remove (optional) and a list of configuration option settings to update and their new value (optional).

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.UpdateConfigurationTemplate(new UpdateConfigurationTemplateRequest
   {
      ApplicationName = "my-app",
      OptionsToRemove = new List<OptionSpecification> {
         new OptionSpecification {
            Namespace = "aws:elasticbeanstalk:healthreporting:system",
            OptionName = "ConfigDocument"
         }
      },
      TemplateName = "my-template"
   });
string applicationName = response.ApplicationName;
DateTime dateCreated = response.DateCreated;
DateTime dateUpdated = response.DateUpdated;
string solutionStackName = response.SolutionStackName;
string templateName = response.TemplateName;
```

**Validate Environment Configuration Option Settings**

In this example, you take a set of configuration settings and an environment, and determines whether those values are valid.
This action returns a list of messages indicating any errors or warnings associated with the selection of option values.

Call the ValidateConfigurationSettings method with a ValidateConfigurationSettingsRequest containing the name of the application and environment configuration option settings to validate.

```csharp
AmazonElasticBeanstalkClient client = new AmazonElasticBeanstalkClient();
var response = client.ValidateConfigurationSettings(new ValidateConfigurationSettingsRequest {
    ApplicationName = "my-app",
    EnvironmentName = "my-env",
    OptionSettings = new List<ConfigurationOptionSetting> {
        new ConfigurationOptionSetting {
            Namespace = "aws:elasticbeanstalk:healthreporting:system",
            OptionName = "ConfigDocument",
            Value = "{"CloudWatchMetrics": {"Environment": {
        }
    },
    Namespace = "aws:elasticbeanstalk:healthreporting:system",
    OptionName = "ConfigDocument",
    Value = "{ "CloudWatchMetrics": { "Environment": {
    }
});
List<ValidationMessage> messages = response.Messages;
```

Deploying Applications Using Amazon EC2

The AWS SDK for .NET supports Amazon EC2, which is a web service that provides resizable computing capacity—literally, servers in Amazon's data centers—that you use to build and host your software systems. The Amazon EC2 APIs are provided by the AWSSDK.EC2 assembly.

The Amazon EC2 instances examples are intended to help you get started with Amazon EC2.

The Amazon EC2 Spot instances examples show you how to use Spot instances, which enable you to bid on unused Amazon EC2 capacity and run any instances that you acquire for as long as your bid exceeds the current Spot price. Amazon EC2 changes the Spot price periodically based on supply and demand; customers whose bids meet or exceed it gain access to the available Spot instances. For more information, see Spot Instances in the Amazon EC2 User Guide for Linux Instances and Spot Instances in the Amazon EC2 User Guide for Windows Instances.

Topics

- Amazon EC2 Instances Examples (p. 108)
Amazon EC2 Instances Examples

You can access the features of Amazon EC2 using the AWS SDK for .NET. For example, you can create, start, and terminate Amazon EC2 instances.

The sample code is written in C#, but you can use the AWS SDK for .NET with any compatible language. When you install the AWS Toolkit for Visual Studio a set of C# project templates are installed. So the easiest way to start this project is to open Visual Studio, and then choose File, New Project, AWS Sample Projects, Compute and Networking, AWS EC2 Sample.

Prerequisites

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

Examples

Topics

- Creating an Amazon EC2 Client (p. 108)
- Creating a Security Group in Amazon EC2 (p. 108)
- Working with Amazon EC2 Key Pairs (p. 112)
- Launching an Amazon EC2 Instance (p. 114)
- Terminating an Amazon EC2 Instance (p. 119)
- Using Regions and Availability Zones with Amazon EC2 (p. 120)
- Using VPC Endpoints with Amazon EC2 (p. 121)
- Using Elastic IP Addresses in Amazon EC2 (p. 123)

Creating an Amazon EC2 Client

Create an Amazon EC2 client to manage your EC2 resources, such as instances and security groups. This client is represented by an AmazonEC2Client object, which you can create as follows.

```csharp
var ec2Client = new AmazonEC2Client();
```

The permissions for the client object are determined by the policy attached to the profile you specified in the App.config file. By default, we use the region specified in App.config. To use a different region, pass the appropriate RegionEndpoint value to the constructor. For more information, see Regions and Endpoints: EC2 in the Amazon Web Services General Reference.

Creating a Security Group 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, Amazon 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. You can create a security group by using the Amazon EC2 console or the AWS SDK for .NET.

You create a security group for use in either EC2-Classic or EC2-VPC. For more information about EC2-Classic and EC2-VPC, see Supported Platforms in the Amazon EC2 User Guide for Windows Instances.
Alternatively, you can create a security group using the Amazon EC2 console. For more information, see Amazon EC2 Security Groups in the Amazon EC2 User Guide for Windows Instances.

## Enumerate Your Security Groups

You can enumerate your security groups and check whether a security group exists.

### To enumerate your security groups

Get the complete list of your security groups using DescribeSecurityGroups with no parameters.

The following example enumerates all of the security groups in the region.

```csharp
static void EnumerateSecurityGroups(AmazonEC2Client ec2Client)
{
    var request = new DescribeSecurityGroupsRequest();
    var response = ec2Client.DescribeSecurityGroups(request);
    List<SecurityGroup> mySGs = response.SecurityGroups;
    foreach (SecurityGroup item in mySGs)
    {
        Console.WriteLine("Security group: " + item.GroupId);
        Console.WriteLine("\tGroupId: " + item.GroupId);
        Console.WriteLine("\tGroupName: " + item.GroupName);
        Console.WriteLine("\tVpcId: " + item.VpcId);
    }
}
```

### To enumerate your security groups for a particular VPC

Use DescribeSecurityGroups with a filter.

The following example retrieves only the security groups that belong to the specified VPC.

```csharp
static void EnumerateVpcSecurityGroups(AmazonEC2Client ec2Client, string vpcID)
{
    Filter vpcFilter = new Filter
    {
        Name = "vpc-id",
        Values = new List<string>() { vpcID }
    };
    var request = new DescribeSecurityGroupsRequest();
    request.Filters.Add(vpcFilter);
    var response = ec2Client.DescribeSecurityGroups(request);
    List<SecurityGroup> mySGs = response.SecurityGroups;
    foreach (SecurityGroup item in mySGs)
    {
        Console.WriteLine("Security group: " + item.GroupId);
        Console.WriteLine("\tGroupId: " + item.GroupId);
        Console.WriteLine("\tGroupName: " + item.GroupName);
        Console.WriteLine("\tVpcId: " + item.VpcId);
    }
}
```

If you attempt to create a security group with a name of an existing security group, CreateSecurityGroup will throw an exception. To avoid this, the following examples search for a security group with the specified name, and return the appropriate SecurityGroup object if one is found.
To create a security group for EC2-Classic

Create and initialize a `CreateSecurityGroupRequest` object. Assign a name and description to the `GroupName` and `Description` properties, respectively.

The `CreateSecurityGroup` method returns a `CreateSecurityGroupResponse` object. You can get the identifier of the new security group from the response and then use `DescribeSecurityGroups` with the security group identifier to get the `SecurityGroup` object for the security group.

```csharp
static SecurityGroup CreateEc2SecurityGroup(
    AmazonEC2Client ec2Client,
    string secGroupName)
{
    // See if a security group with the specified name already exists
    Filter nameFilter = new Filter();
    nameFilter.Name = "group-name";
    nameFilter.Values = new List<string>() { secGroupName };

    var describeRequest = new DescribeSecurityGroupsRequest();
    describeRequest.Filters.Add(nameFilter);
    var describeResponse = ec2Client.DescribeSecurityGroups(describeRequest);

    // If a match was found, return the SecurityGroup object for the security group
    if(describeResponse.SecurityGroups.Count > 0)
    {
        return describeResponse.SecurityGroups[0];
    }

    // Create the security group
    var createRequest = new CreateSecurityGroupRequest();
    createRequest.GroupName = secGroupName;
    createRequest.Description = "My sample security group for EC2-Classic";

    var createResponse = ec2Client.CreateSecurityGroup(createRequest);

    var Groups = new List<string>() { createResponse.GroupId };
    describeRequest = new DescribeSecurityGroupsRequest() { GroupIds = Groups };
    describeResponse = ec2Client.DescribeSecurityGroups(describeRequest);

    return describeResponse.SecurityGroups[0];
}
```

To create a security group for EC2-VPC

Create and initialize a `CreateSecurityGroupRequest` object. Assign values to the `GroupName`, `Description`, and `VpcId` properties.

The `CreateSecurityGroup` method returns a `CreateSecurityGroupResponse` object. You can get the identifier of the new security group from the response and then use `DescribeSecurityGroups` with the security group identifier to get the `SecurityGroup` object for the security group.

```csharp
static SecurityGroup CreateVpcSecurityGroup(
    AmazonEC2Client ec2Client,
    string vpcId,
    string secGroupName)
{
    // See if a security group with the specified name already exists
    Filter nameFilter = new Filter();
    nameFilter.Name = "group-name";
    nameFilter.Values = new List<string>() { secGroupName };

    var describeRequest = new DescribeSecurityGroupsRequest();
    describeRequest.Filters.Add(nameFilter);
    var describeResponse = ec2Client.DescribeSecurityGroups(describeRequest);
```

110
// If a match was found, return the SecurityGroup object for the security group
if (describeResponse.SecurityGroups.Count > 0)
{
    return describeResponse.SecurityGroups[0];
}

// Create the security group
var createRequest = new CreateSecurityGroupRequest();
createRequest.GroupName = secGroupName;
createRequest.Description = "My sample security group for EC2-VPC";
createRequest.VpcId = vpcId;
var createResponse = ec2Client.CreateSecurityGroup(createRequest);

var Groups = new List<string>() { createResponse.GroupId };
describeRequest = new DescribeSecurityGroupsRequest() { GroupIds = Groups };
describeResponse = ec2Client.DescribeSecurityGroups(describeRequest);
return describeResponse.SecurityGroups[0];

Use the following procedure to add a rule to allow inbound traffic on TCP port 3389 (RDP). This enables you to connect to a Windows instance. If you’re launching a Linux instance, use TCP port 22 (SSH) instead.

**Note**
You can use a service to get the public IP address of your local computer. For example, we provide the following service: [http://checkip.amazonaws.com/](http://checkip.amazonaws.com/). To locate another service that provides your IP address, use the search phrase "what is my IP address". If you are connecting through an ISP or from behind your firewall without a static IP address, you need to find out the range of IP addresses used by client computers.

The examples in this section follow from the examples in the previous sections. They assume `secGroup` is an existing security group.

**To add a rule to a security group**

1. Create and initialize an `IpPermission` object.

```csharp
string ipRange = "1.1.1.1/1";
List<string> ranges = new List<string>() { ipRange };

var ipPermission = new IpPermission();
ipPermission.IpProtocol = "tcp";
ipPermission.FromPort = 3389;
ipPermission.ToPort = 3389;
ipPermission.IpRanges = ranges;
```

**IpProtocol**
The IP protocol.

**FromPort and ToPort**
The beginning and end of the port range. This example specifies a single port, 3389, which is used to communicate with Windows over RDP.

**IpRanges**
The IP addresses or address ranges, in CIDR notation. For convenience, this example uses 72.21.198.64/24, which authorizes network traffic for a single IP address. You can use [http://checkip.amazonaws.com/](http://checkip.amazonaws.com/) to determine your own IP address.
2. Create and initialize an `AuthorizeSecurityGroupIngressRequest` object.

```csharp
var ingressRequest = new AuthorizeSecurityGroupIngressRequest();
ingressRequest.GroupId = secGroup.GroupId;
ingressRequest.IpPermissions.Add(ipPermission);
```

<table>
<thead>
<tr>
<th>GroupId</th>
<th>The identifier of the security group.</th>
</tr>
</thead>
<tbody>
<tr>
<td>IpPermissions</td>
<td>The <code>IpPermission</code> object from step 1.</td>
</tr>
</tbody>
</table>

3. (Optional) You can add additional rules to the `IpPermissions` collection before going to the next step.

4. Pass the `AuthorizeSecurityGroupIngressRequest` object to the `AuthorizeSecurityGroupIngress` method, which returns an `AuthorizeSecurityGroupIngressResponse` object. If a matching rule already exists, an `AmazonEC2Exception` is thrown.

```csharp
try
{
    var ingressResponse = ec2Client.AuthorizeSecurityGroupIngress(ingressRequest);
    Console.WriteLine("New RDP rule for: "+ ipRange);
}
catch (AmazonEC2Exception ex)
{
    // Check the ErrorCode to see if the rule already exists
    if ("InvalidPermission.Duplicate" == ex.ErrorCode)
    {
        Console.WriteLine("An RDP rule for: \{\} already exists.", ipRange);
    }
    else
    {
        // The exception was thrown for another reason, so re-throw the exception
        throw;
    }
}
```

**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, then the recipient uses the private key to decrypt the data. The public and private keys are known as a key pair. You must specify a key pair when you launch an EC2 instance and specify the private key of the keypair when you connect to the instance. You can create a key pair or use one you've used when launching other instances. For more information, see [Amazon EC2 Key Pairs](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html) in the *Amazon EC2 User Guide for Windows Instances*. This example shows how to create a key pair, describe key pairs and delete a key pair using these `AmazonEC2Client` methods:

- `CreateKeyPair`
- `DeleteKeyPair`
- `DescribeKeyPairs`

**Create a Key Pair and Save the Private Key**

When you create a new key pair, you must save the private key that is returned. You cannot retrieve the private key later.
Create and initialize a `CreateKeyPairRequest` object. Set the `KeyName` property to the name of the key pair.

Pass the request object to the `CreateKeyPair` method, which returns a `CreateKeyPairResponse` object. If a key pair with the specified name already exists, an `AmazonEC2Exception` is thrown.

The response object includes a `CreateKeyPairResponse` object that contains the new key’s `KeyPair` object. The `KeyPair` object’s `KeyMaterial` property contains the unencrypted private key for the key pair. Save the private key as a `.pem` file in a safe location. You’ll need this file when you connect to your instance. This example saves the private key in the specified file name.

```csharp
public static void CreateKeyPair(
    AmazonEC2Client ec2Client,
    string keyPairName,
    string privateKeyFile)
{
    var request = new CreateKeyPairRequest();
    request.KeyName = keyPairName;
    try
    {
        var response = ec2Client.CreateKeyPair(request);
        Console.WriteLine();
        Console.WriteLine("New key: "+ keyPairName);
        // Save the private key in a .pem file
        using (FileStream s = new FileStream(privateKeyFile, FileMode.Create))
        using (StreamWriter writer = new StreamWriter(s))
        {
            writer.WriteLine(response.KeyPair.KeyMaterial);
        }
    }
    catch (AmazonEC2Exception ex)
    {
        // Check the ErrorCode to see if the key already exists
        if("InvalidKeyPair.Duplicate" == ex.ErrorCode)
        {
            Console.WriteLine("The key pair "{0}" already exists.", keyPairName);
        }
        else
        {
            // The exception was thrown for another reason, so re-throw the exception.
            throw;
        }
    }
}
```

**Enumerate Your Key Pairs**

You can enumerate your key pairs and check whether a key pair exists.

Get the complete list of your key pairs using the `DescribeKeyPairs` method with no parameters.

```csharp
public static void EnumerateKeyPairs(AmazonEC2Client ec2Client)
{
    var request = new DescribeKeyPairsRequest();
    var response = ec2Client.DescribeKeyPairs(request);
    foreach (KeyPairInfo item in response.KeyPairs)
    {
```
Delete Key Pairs

You can delete a key pair by calling the `DeleteKeyPair` from your `AmazonEC2Client` instance.

Pass a `DeleteKeyPairRequest` containing the name of the key pair to the `DeleteKeyPair` method of the `AmazonEC2Client` object.

```csharp
public static void DeleteKeyPair(    AmazonEC2Client ec2Client,    KeyPair keyPair)    {        try        {            // Delete key pair created for sample            ec2Client.DeleteKeyPair(new DeleteKeyPairRequest { KeyName = keyPair.KeyName });        }        catch (AmazonEC2Exception ex)        {            // Check the ErrorCode to see if the key already exists            if ("InvalidKeyPair.NotFound" == ex.ErrorCode)            {                Console.WriteLine("The key pair "{0}" was not found.", keyPair.KeyName);            }            else            {                // The exception was thrown for another reason, so re-throw the exception                throw;            }        }    }
```

Launching an Amazon EC2 Instance

Use the following procedure to launch one or more identically configured Amazon EC2 instances from the same Amazon Machine Image (AMI). After you create your EC2 instances, you can check their status. When your EC2 instances are running, you can connect to them.

Launch an EC2 Instance in EC2-Classic or in a VPC

You can launch an instance in either EC2-Classic or in a VPC. For more information about EC2-Classic and EC2-VPC, see Supported Platforms in the *Amazon EC2 User Guide for Windows Instances*.

To launch an EC2 instance in EC2-Classic

1. Create and initialize a `RunInstancesRequest` object. Be sure the AMI, key pair, and security group you specify exist in the region you specified when you created the client object.

```csharp
string amiID = "ami-e189c8d1";    string keyPairName = "my-sample-key";

List<string> groups = new List<string>() { mySG.GroupId };    var launchRequest = new RunInstancesRequest()    {
```
ImageId

The ID of the AMI. For a list of public AMIs, see Amazon Machine Images.

InstanceType

An instance type that is compatible with the specified AMI. For more information, see Instance Types in the Amazon EC2 User Guide for Windows Instances.

MinCount

The minimum number of EC2 instances to launch. If this is more instances than Amazon EC2 can launch in the target Availability Zone, Amazon EC2 launches no instances.

MaxCount

The maximum number of EC2 instances to launch. If this is more instances than Amazon EC2 can launch in the target Availability Zone, Amazon EC2 launches the largest possible number of instances above MinCount. You can launch between 1 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.

KeyName

The name of the EC2 key pair. If you launch an instance without specifying a key pair, you can't connect to it. For more information, see Working with Amazon EC2 Key Pairs (p. 112).

SecurityGroupIds

The identifiers of one or more security groups. For more information, see Creating a Security Group in Amazon EC2 (p. 108).

2. (Optional) To launch the instance with an IAM role (p. 148), specify an IAM instance profile in the RunInstancesRequest object.

An IAM user can't launch an instance with an IAM role without the permissions granted by the following policy.

```json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "iam:PassRole",
      "iam:ListInstanceProfiles",
      "ec2:*"
    ],
    "Resource": "*"
  }]
}
```

For example, the following snippet instantiates and configures an IamInstanceProfileSpecification object for an IAM role named winapp-instance-role-1.

```csharp
var instanceProfile = new IamInstanceProfile();
```
instanceProfile.Id = "winapp-instance-role-1";

To specify this instance profile in the RunInstancesRequest object, add the following line.

launchRequest.IamInstanceProfile = instanceProfile;

3. Launch the instance by passing the request object to the RunInstances method. Save the ID of the instance because you need it to manage the instance.

Use the returned RunInstancesResponse object to get the instance IDs for the new instances. The Reservation.Instances property contains a list of Instance objects, one for each EC2 instance you successfully launched. You can retrieve the ID for each instance from the InstanceId property of the Instance object.

```csharp
var launchResponse = ec2Client.RunInstances(launchRequest);
var instances = launchResponse.Reservation.Instances;
var instanceIds = new List<string>();
foreach (Instance item in instances)
{
    instanceIds.Add(item.InstanceId);
    Console.WriteLine();
    Console.WriteLine("New instance: " + item.InstanceId);
    Console.WriteLine("Instance state: " + item.State.Name);
}
```

To launch an EC2 instance in a VPC

1. Create and initialize an elastic network interface in a subnet of the VPC.

```csharp
string subnetID = "subnet-cb663da2";
List<string> groups = new List<string>() { mySG.GroupId }
var eni = new InstanceNetworkInterfaceSpecification()
{
    DeviceIndex = 0,
    SubnetId = subnetID,
    Groups = groups,
    AssociatePublicIpAddress = true
};
List<InstanceNetworkInterfaceSpecification> enis = new
List<InstanceNetworkInterfaceSpecification>() {eni};
```

**DeviceIndex**

The index of the device on the instance for the network interface attachment.

**SubnetId**

The ID of the subnet where the instance will be launched.

**GroupIds**

One or more security groups. For more information, see Creating a Security Group in Amazon EC2 (p. 108).

**AssociatePublicIpAddress**

Indicates whether to auto-assign a public IP address to an instance in a VPC.

2. Create and initialize a RunInstancesRequest object. Be sure the AMI, key pair, and security group you specify exist in the region you specified when you created the client object.
string amiID = "ami-e189c8d1";
string keyPairName = "my-sample-key";

var launchRequest = new RunInstancesRequest()
{
    ImageId = amiID,
    InstanceType = InstanceType.T1Micro,
    MinCount = 1,
    MaxCount = 1,
    KeyName = keyPairName,
    NetworkInterfaces = enis
};

**ImageId**

The ID of the AMI. For a list of public AMIs provided by Amazon, see Amazon Machine Images.

**InstanceType**

An instance type that is compatible with the specified AMI. For more information, see Instance Types in the Amazon EC2 User Guide for Windows Instances.

**MinCount**

The minimum number of EC2 instances to launch. If this is more instances than Amazon EC2 can launch in the target Availability Zone, Amazon EC2 launches no instances.

**MaxCount**

The maximum number of EC2 instances to launch. If this is more instances than Amazon EC2 can launch in the target Availability Zone, Amazon EC2 launches the largest possible number of instances above MinCount. You can launch between 1 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.

**KeyName**

The name of the EC2 key pair. If you launch an instance without specifying a key pair, you can't connect to it. For more information, see Working with Amazon EC2 Key Pairs (p. 112).

**NetworkInterfaces**

One or more network interfaces.

3. (Optional) To launch the instance with an IAM role (p. 148), specify an IAM instance profile in the RunInstancesRequest object.

An IAM user can't launch an instance with an IAM role without the permissions granted by the following policy.

```json
{
    "Version": "2012-10-17",
    "Statement": [{
      "Effect": "Allow",
      "Action": [
        "iam:PassRole",
        "iam:ListInstanceProfiles",
        "ec2:*"
      ],
      "Resource": "*"
    }]
}
```
For example, the following snippet instantiates and configures an `IamInstanceProfileSpecification` object for an IAM role named `winapp-instance-role-1`.

```csharp
var instanceProfile = new IamInstanceProfileSpecification();
instanceProfile.Name = "winapp-instance-role-1";
```

To specify this instance profile in the `RunInstancesRequest` object, add the following line.

```csharp
launchRequest.IamInstanceProfile = instanceProfile;
```

4. Launch the instances by passing the request object to the `RunInstances` method. Save the IDs of the instances because you need them to manage the instances.

Use the returned `RunInstancesResponse` object to get a list of instance IDs for the new instances. The `Reservation.Instances` property contains a list of `Instance` objects, one for each EC2 instance you successfully launched. You can retrieve the ID for each instance from the `InstanceId` property of the `Instance` object.

```csharp
RunInstancesResponse launchResponse = ec2Client.RunInstances(launchRequest);
List<string> instanceIds = new List<string>();
foreach (Instance instance in launchResponse.Reservation.Instances)
{
    Console.WriteLine(instance.InstanceId);
    instanceIds.Add(instance.InstanceId);
}
```

**Check the State of Your Instance**

Use the following procedure to get the current state of your instance. Initially, your instance is in the pending state. You can connect to your instance after it enters the running state.

1. Create and configure a `DescribeInstancesRequest` object and assign your instance's instance ID to the `InstanceIds` property. You can also use the `Filter` property to limit the request to certain instances, such as instances with a particular user-specified tag.

```csharp
var instanceRequest = new DescribeInstancesRequest();
instanceRequest.InstanceIds = new List<string>();
instanceRequest.InstanceIds.Add(instanceId);
```

2. Call the `DescribeInstances` method, and pass it the request object from step 1. The method returns a `DescribeInstancesResponse` object that contains information about the instance.

```csharp
var response = ec2Client.DescribeInstances(instanceRequest);
```

3. The `DescribeInstancesResponse.Reservations` property contains a list of reservations. In this case, there is only one reservation. Each reservation contains a list of `Instance` objects. Again, in this case, there is only one instance. You can get the instance's status from the `State` property.

```csharp
Console.WriteLine(response.Reservations[0].Instances[0].State.Name);
```

**Connect to Your Running Instance**

After an instance is running, you can remotely connect to it by using the appropriate remote client.
For Linux instances, use an SSH client. You must ensure that the instance's SSH port (22) is open to traffic. You will need the instance's public IP address or public DNS name and the private portion of the key pair used to launch the instance. For more information, see Connecting to Your Linux Instance in the Amazon EC2 User Guide for Linux Instances.

For Windows instances, use an RDP client. You must ensure the instance's RDP port (3389) is open to traffic. You will need the instance's public IP address or public DNS name and the administrator password. The administrator password is obtained with the GetPasswordData and GetPasswordDataResult.GetDecryptedPassword methods, which require the private portion of the key pair used to launch the instance. For more information, see Connecting to Your Windows Instance Using RDP in the Amazon EC2 User Guide for Windows Instances. The following example demonstrates how to get the password for a Windows instance.

```csharp
public static string GetWindowsPassword(
    AmazonEC2Client ec2Client,
    string instanceId,
    FileInfo privateKeyFile)
{
    string password = "";
    var request = new GetPasswordDataRequest();
    request.InstanceId = instanceId;

    var response = ec2Client.GetPasswordData(request);
    if (null != response.PasswordData)
    {
        using (StreamReader sr = new StreamReader(privateKeyFile.FullName))
        {
            string privateKeyData = sr.ReadToEnd();
            password = response.GetDecryptedPassword(privateKeyData);
        }
    }
    else
    {
        Console.WriteLine("The password is not available. The password for " +
            "instance {0} is either not ready, or it is not a Windows instance.",
            instanceId);
    }

    return password;
}
```

When you no longer need your EC2 instance, see Terminating an Amazon EC2 Instance (p. 119).

**Terminating an Amazon EC2 Instance**

When you no longer need one or more of your Amazon EC2 instances, you can terminate them.

**To terminate an EC2 instance**

```csharp
public static void TerminateInstance(
    AmazonEC2Client ec2Client,
    string instanceId)
{
    var request = new TerminateInstancesRequest();
    request.InstanceIds = new List<string>() { instanceId };

    try
```
```csharp
var response = ec2Client.TerminateInstances(request);
foreach (InstanceStateChange item in response.TerminatingInstances)
{
    Console.WriteLine("Terminated instance: " + item.InstanceId);
    Console.WriteLine("Instance state: " + item.CurrentState.Name);
}
}

try
{
    catch(AmazonEC2Exception ex)
    {
        // Check the ErrorCode to see if the instance does not exist.
        if ("InvalidInstanceID.NotFound" == ex.ErrorCode)
        {
            Console.WriteLine("Instance {0} does not exist.", instanceId);
        }
        else
        {
            // The exception was thrown for another reason, so re-throw the exception.
            throw;
        }
    }
}
```

1. Create and initialize a TerminateInstancesRequest object.
2. Set the TerminateInstancesRequest.InstanceIds property to a list of one or more instance IDs for the instances to terminate.
3. Pass the request object to the TerminateInstances method. If the specified instance doesn't exist, an AmazonEC2Exception is thrown.
4. You can use the TerminateInstancesResponse object to list the terminated instances, as follows.

### Using Regions and Availability Zones with Amazon EC2

This .NET example shows you how to:

- Get details about Availability Zones
- Get details about regions

#### The Scenario

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. Amazon EC2 provides the ability to place instances and data in multiple locations.

You can use the AWS SDK for .NET to retrieve details about regions and Availability Zones by using the following methods of the AmazonEC2Client class:

- DescribeAvailabilityZones
- DescribeRegions

For more information about regions and Availability Zones, see Regions and Availability Zones in the Amazon EC2 User Guide for Windows Instances.

#### Describe Availability Zones

Create an AmazonEC2Client instance and call the DescribeAvailabilityZones method. The DescribeAvailabilityZonesResponse object that is returned contains a list of Availability Zones.
public static void DescribeAvailabilityZones()
{
    Console.WriteLine("Describe Availability Zones");
    AmazonEC2Client client = new AmazonEC2Client();
    DescribeAvailabilityZonesResponse response = client.DescribeAvailabilityZones();
    var availZones = new List<AvailabilityZone>();
    availZones = response.AvailabilityZones;
    foreach (AvailabilityZone az in availZones)
    {
        Console.WriteLine(az.ZoneName);
    }
}

Describe Regions

Create an AmazonEC2Client instance and call the DescribeRegions method. The DescribeRegionsResponse object that is returned contains a list of regions.

public static void DescribeRegions()
{
    Console.WriteLine("Describe Regions");
    AmazonEC2Client client = new AmazonEC2Client();
    DescribeRegionsResponse response = client.DescribeRegions();
    var regions = new List<Region>();
    regions = response.Regions;
    foreach (Region region in regions)
    {
        Console.WriteLine(region.RegionName);
    }
}

Using VPC Endpoints with Amazon EC2

This .NET example shows you how to create, describe, modify, and delete VPC endpoints.

The Scenario

An endpoint enables you to create a private connection between your VPC and another AWS service in your account. You can specify a policy to attach to the endpoint that will control access to the service from your VPC. You can also specify the VPC route tables that use the endpoint.

This example uses the following AmazonEC2Client methods:

- CreateVpcEndpoint
- DescribeVpcEndpoints
- ModifyVpcEndpoint
- DeleteVpcEndpoints

Create a VPC Endpoint

The following example creates a VPC endpoint for an Amazon Simple Storage Service (S3).

Create an AmazonEC2Client instance. You'll create a new VPC so that you can create a VPC endpoint.

Create a CreateVpcRequest object specifying an IPv4 CIDR block as its constructor's parameter. Using that CreateVpcRequest object, use the CreateVpc method to create a VPC. Use that VPC to instantiate a
CreateVpcEndpointRequest object, specifying the service name for the endpoint. Then, use that request object to call the CreateVpcEndpoint method and create the VpcEndpoint.

```csharp
public static void CreateVPCEndpoint()
{
    AmazonEC2Client client = new AmazonEC2Client();
    CreateVpcRequest vpcRequest = new CreateVpcRequest("10.32.0.0/16");
    CreateVpcResponse vpcResponse = client.CreateVpc(vpcRequest);
    Vpc vpc = vpcResponse.Vpc;
    CreateVpcEndpointRequest endpointRequest = new CreateVpcEndpointRequest();
    endpointRequest.VpcId = vpc.VpcId;
    endpointRequest.ServiceName = "com.amazonaws.us-west-2.s3";
    CreateVpcEndpointResponse cVpcErsp = client.CreateVpcEndpoint(endpointRequest);
    VpcEndpoint vpcEndPoint = cVpcErsp.VpcEndpoint;
}
```

Describe a VPC Endpoint

Create an AmazonEC2Client instance. Next, create a DescribeVpcEndpointsRequest object and limit the maximum number of results to return to 5. Use that DescribeVpcEndpointsRequest object to call the DescribeVpcEndpoints method. The DescribeVpcEndpointsResponse that is returned contains the list of VPC Endpoints.

```csharp
public static void DescribeVPCEndPoints()
{
    AmazonEC2Client client = new AmazonEC2Client();
    DescribeVpcEndpointsRequest endpointRequest = new DescribeVpcEndpointsRequest();
    endpointRequest.MaxResults = 5;
    DescribeVpcEndpointsResponse endpointResponse = client.DescribeVpcEndpoints(endpointRequest);
    List<VpcEndpoint> endpointList = endpointResponse.VpcEndpoints;
    foreach (VpcEndpoint vpc in endpointList)
    {
        Console.WriteLine("VpcEndpoint ID = " + vpc.VpcEndpointId);
        List<string> routeTableIds = vpc.RouteTableIds;
        foreach (string id in routeTableIds)
        {
            Console.WriteLine("\tRoute Table ID = " + id);
        }
    }
}
```

Modify a VPC Endpoint

The following example modifies attributes of a specified VPC endpoint. You can modify the policy associated with the endpoint, and you can add and remove route tables associated with the endpoint.

Create an AmazonEC2Client instance. Create a ModifyVpcEndpointRequest object using the ID of the VPC endpoint and the ID of the route table to add to it. Call the ModifyVpcEndpoint method using the ModifyVpcEndpointRequest object. The ModifyVpcEndpointResponse object that is returned contains an HTTP status code indicating whether the modify request succeeded.

```csharp
public static void ModifyVPCEndPoint()
{
    AmazonEC2Client client = new AmazonEC2Client();
    ModifyVpcEndpointRequest modifyRequest = new ModifyVpcEndpointRequest();
    modifyRequest.VpcEndpointId = "vpce-17b05a7e";
    modifyRequest.AddRouteTableIds = new List<string> { "rtb-c46f15a3" };
    ModifyVpcEndpointResponse modifyResponse = client.ModifyVpcEndpoint(modifyRequest);
}
Delete a VPC Endpoint

You can delete one or more specified VPC endpoints. Deleting the endpoint also deletes the endpoint routes in the route tables that were associated with the endpoint.

Create an AmazonEC2Client instance. Use the DescribeVpcEndpoints method to list the VPC endpoints associated with the EC2 client. Use the list of VPC endpoints to create a list of VPC endpoint IDs. Use that list to create a DeleteVpcEndpointsRequest object to be used by the DeleteVpcEndpoints method.

```csharp
private static void DeleteVPCEndPoint()
{
    AmazonEC2Client client = new AmazonEC2Client();
    DescribeVpcEndpointsRequest endpointRequest = new DescribeVpcEndpointsRequest();
    endpointRequest.MaxResults = 5;
    DescribeVpcEndpointsResponse endpointResponse = client.DescribeVpcEndpoints(endpointRequest);
    List<VpcEndpoint> endpointList = endpointResponse.VpcEndpoints;
    var vpcEndPointListIds = new List<string>();
    foreach (VpcEndpoint vpc in endpointList)
    {
        Console.WriteLine("VpcEndpoint ID = " + vpc.VpcEndpointId);
        vpcEndPointListIds.Add(vpc.VpcEndpointId);
    }
    DeleteVpcEndpointsRequest deleteRequest = new DeleteVpcEndpointsRequest();
    deleteRequest.VpcEndpointIds = vpcEndPointListIds;
    client.DeleteVpcEndpoints(deleteRequest);
}
```

Using Elastic IP Addresses in Amazon EC2

This .NET example shows you how to:

- Retrieve descriptions of your Elastic IP addresses
- Allocate and associate an Elastic IP address with an Amazon EC2 instance
- Release an Elastic IP address

The Scenario

An Elastic IP address is a static IP address designed for dynamic cloud computing. An Elastic IP address is associated with your AWS account, and is a public IP address reachable from the Internet.

If your Amazon EC2 instance doesn't have a public IP address, you can associate an Elastic IP address with your instance to enable communication with the Internet.

In this example, you use the AWS SDK for .NET to manage Elastic IP addresses by using these methods of the Amazon EC2 client class:

- DescribeAddresses
- AllocateAddress
- AssociateAddress
- ReleaseAddress
For more information about Elastic IP addresses in Amazon EC2, see Elastic IP Addresses in the Amazon EC2 User Guide for Windows Instances.

Describe Elastic IP Addresses

Create an AmazonEC2Client object. Next, create a DescribeAddressesRequest object to pass as a parameter, filtering the addresses returned by those in your VPC. To retrieve descriptions of all your Elastic IP addresses, omit the filter from the parameters. Then call the DescribeAddresses method of the AmazonEC2Client object.

```csharp
public void DescribeElasticIps()
{
    using (var client = new AmazonEC2Client(RegionEndpoint.USWest2))
    {
        var addresses = client.DescribeAddresses(new DescribeAddressesRequest
        {
            Filters = new List<Filter>
            {
                new Filter
                {
                    Name = "domain",
                    Values = new List<string> { "vpc" }
                }
            }
        }).Addresses;
        foreach(var address in addresses)
        {
            Console.WriteLine(address.PublicIp);
            Console.WriteLine("Allocation Id: " + address.AllocationId);
            Console.WriteLine("Private IP Address: " + address.PrivateIpAddress);
            Console.WriteLine("Association Id: " + address.AssociationId);
            Console.WriteLine("Instance Id: " + address.InstanceId);
            Console.WriteLine("Network Interface Owner Id: " + address.NetworkInterfaceOwnerId);
        }
    }
}
```

Allocate and Associate an Elastic IP Address

Create an AmazonEC2Client object. Next, create an AllocateAddressRequest object for the parameter used to allocate an Elastic IP address, which in this case specifies that the domain is a VPC. Call the AllocateAddress method of the AmazonEC2Client object.

Upon success, the returned AllocateAddressResponse object has an AllocationId property that identifies the allocated Elastic IP address.

Create an AssociateAddressRequest object for the parameters used to associate an Elastic IP address to an Amazon EC2 instance. Include the AllocationId from the newly allocated address and the InstanceId of the Amazon EC2 instance. Then call the AssociateAddress method of the AmazonEC2Client object.

```csharp
public void AllocateAndAssociate(string instanceId)
{
    using (var client = new AmazonEC2Client(RegionEndpoint.USWest2))
    {
        var allocationId = client.AllocateAddress(new AllocateAddressRequest
        {
            Domain = DomainType.Vpc
        }).AllocationId;
    }
}
```
Release an Elastic IP Address

Create an `AmazonEC2Client` object. Next, create a `ReleaseAddressRequest` object for the parameters used to release an Elastic IP address, which in this case specifies the `AllocationId` for the Elastic IP address. Releasing an Elastic IP address also disassociates it from any Amazon EC2 instance. Call the `ReleaseAddress` method of the Amazon EC2 service object.

```csharp
public void Release(string allocationId)
{
    using (var client = new AmazonEC2Client(RegionEndpoint.USWest2))
    {
        client.ReleaseAddress(new ReleaseAddressRequest
        {
            AllocationId = allocationId
        });
    }
}
```

Amazon EC2 Spot Instances Examples

This topic describes how to use the AWS SDK for .NET with Amazon EC2 Spot Instances.

**Topics**

- Overview (p. 125)
- Prerequisites (p. 126)
- Setting Up Your Credentials (p. 126)
- Submitting Your Spot Request (p. 126)
- Determining the State of Your Spot Request (p. 128)
- Cleaning Up Your Spot Requests and Instances (p. 128)

**Overview**

Spot Instances enable you to bid on unused Amazon EC2 capacity and run any instances that you acquire for as long as your bid exceeds the current *Spot Price*. Amazon EC2 changes the Spot Price periodically based on supply and demand; customers whose bids meet or exceed it gain access to the available Spot Instances. Like On-Demand Instances and Reserved Instances, Spot Instances provide another option for obtaining more compute capacity.

Spot Instances can significantly lower your Amazon EC2 costs for applications such as batch processing, scientific research, image processing, video encoding, data and web crawling, financial analysis, and testing. Additionally, Spot Instances are an excellent option when you need large amounts of computing capacity, but the need for that capacity is not urgent.
To use Spot Instances, place a Spot Instance request specifying the maximum price you are willing to pay per instance hour; this is your bid. If your bid exceeds the current Spot Price, your request is fulfilled and your instances will run until either you choose to terminate them or the Spot Price increases above your bid (whichever is sooner). You can terminate a Spot Instance programmatically, as shown this tutorial, or by using the AWS Management Console or by using the AWS Toolkit for Visual Studio.

It's important to note two points:

1. You will often pay less per hour than your bid. Amazon EC2 adjusts the Spot Price periodically as requests come in and available supply changes. Everyone pays the same Spot Price for that period regardless of whether their bid was higher. Therefore, you might pay less than your bid, but you will never pay more than your bid.

2. If you're running Spot Instances and your bid no longer meets or exceeds the current Spot Price, your instances will be terminated. This means you will want to make sure that your workloads and applications are flexible enough to take advantage of this opportunistic—but potentially transient—capacity.

Spot Instances perform exactly like other Amazon EC2 instances while running, and like other Amazon EC2 instances, Spot Instances can be terminated when you no longer need them. If you terminate your instance, you pay for any partial hour used (as you would for On-Demand or Reserved Instances). However, if your instance is terminated by Amazon EC2 because the Spot Price goes above your bid, you will not be charged for any partial hour of use.

This tutorial provides an overview of how to use the .NET programming environment to do the following.

- Submit a Spot request
- Determine when the Spot request becomes fulfilled
- Cancel the Spot request
- Terminate associated instances

**Prerequisites**

This tutorial assumes you have signed up for AWS, set up your .NET development environment, and installed the AWS SDK for .NET. If you use the Microsoft Visual Studio development environment, we recommend you also install the AWS Toolkit for Visual Studio. For instructions on setting up your environment, see Getting Started with the AWS SDK for .NET (p. 3).

**Setting Up Your Credentials**

For information about how to use your AWS credentials with the SDK, see Configuring AWS Credentials (p. 11).

**Submitting Your Spot Request**

To submit a Spot request, you first need to determine the instance type, the Amazon Machine Image (AMI), and the maximum bid price you want to use. You must also include a security group, so that you can log into the instance if you want to. For more information about creating security groups, see Creating a Security Group in Amazon EC2 (p. 108).

There are several instance types to choose from; go to Amazon EC2 Instance Types for a complete list. For this tutorial, we will use t1.micro. You'll also want to get the ID of a current Windows AMI. For more information, see Finding an AMI in the Amazon EC2 User Guide for Windows Instances.

There are many ways to approach bidding for Spot instances. To get a broad overview of the various approaches, you should view the Bidding for Spot Instances video. However, to get started, we'll describe
three common strategies: bid to ensure cost is less than on-demand pricing; bid based on the value of the resulting computation; bid so as to acquire computing capacity as quickly as possible.

**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 using either the AWS Management Console or the Amazon EC2 API. For more information, go to Viewing Spot Price History. After you've analyzed the price history for your desired instance type in a given Availability Zone, you have two alternative approaches for your bid:

- You could bid at the upper end of the range of Spot Prices (which are still below the On-Demand price), anticipating that your one-time Spot request would most likely be fulfilled and run for enough consecutive compute time to complete the job.
- Or, you could bid 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. (We will explain how to automate this task later in this tutorial.)

**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 are worth in terms of computing costs. After you've analyzed the Spot Price history for your instance type, you choose a bid price at which the cost of the computing time is no more than the value of the job's results. You create a persistent bid and allow it to run intermittently as the Spot Price fluctuates at or below your bid.

**Acquire Computing Capacity Quickly**

You have an unanticipated, short-term need for additional capacity that is not available through On-Demand Instances. After you've analyzed the Spot Price history for your instance type, you bid above the highest historical price to greatly improve the likelihood your request will be fulfilled quickly and continue computing until it is complete.

After you choose your bid price, you are ready to request a Spot Instance. For the purposes of this tutorial, we will set our bid price equal to the On-Demand price ($0.03) to maximize the chances the bid will be fulfilled. You can determine the types of available instances and the On-Demand prices for instances by going to Amazon EC2 Pricing page.

To request a Spot Instance, you need to build your request with the parameters we have specified so far. Start by creating a RequestSpotInstanceRequest object. The request object requires the bid price and the number of instances you want to start. Additionally, you need to set the LaunchSpecification for the request, which includes the instance type, AMI ID, and the name of the security group you want to use for the Spot Instances. After the request is populated, call the RequestSpotInstances method to create the Spot Instance request. The following example demonstrates how to request a Spot Instance.

```csharp
public static SpotInstanceRequest RequestSpotInstance(
            AmazonEC2Client ec2Client,
            string amiId,
            string securityGroupName,
            InstanceType instanceType,
            string spotPrice,
            int instanceCount)
{
    var request = new RequestSpotInstancesRequest();
    request.SpotPrice = spotPrice;
    request.InstanceCount = instanceCount;
    request.LaunchSpecifications.Add(new LaunchSpecification()
```
var launchSpecification = new LaunchSpecification();
launchSpecification.ImageId = amiId;
launchSpecification.InstanceType = instanceType;

launchSpecification.SecurityGroups.Add(securityGroupName);
request.LaunchSpecification = launchSpecification;

var result = ec2Client.RequestSpotInstances(request);
return result.SpotInstanceRequests[0];
}

The Spot request ID is contained in the SpotInstanceRequestId member of the SpotInstanceRequest object.

Running this code will launch a new Spot Instance request.

**Note**
You will be charged for any Spot Instances that are launched, so make sure you cancel any requests and terminate any instances you launch to reduce any associated fees.

There are other options you can use to configure your Spot requests. To learn more, see RequestSpotInstances in the AWS SDK for .NET.

### Determining the State of Your Spot Request

Next, we need to wait until the Spot request reaches the **Active** state before proceeding to the last step. To determine the state of your Spot request, we use the DescribeSpotInstanceRequests method to obtain the state of the Spot request ID we want to monitor.

```csharp
public static SpotInstanceState GetSpotRequestState(
    AmazonEC2Client ec2Client,
    string spotRequestId)
{
    // Create the describeRequest object with all of the request ids
    // to monitor (e.g. that we started).
    var request = new DescribeSpotInstanceRequestsRequest();
    request.SpotInstanceRequestIds.Add(spotRequestId);

    // Retrieve the request we want to monitor.
    var describeResponse = ec2Client.DescribeSpotInstanceRequests(request);

    SpotInstanceRequest req = describeResponse.SpotInstanceRequests[0];
    return req.State;
}
```

### Cleaning Up Your Spot Requests and Instances

The final step is to clean up your requests and instances. It is important to both cancel any outstanding requests and terminate any instances. Just canceling your requests will not terminate your instances, which means that you will continue to be charged for them. If you terminate your instances, your Spot requests may be canceled, but there are some scenarios, such as if you use persistent bids, where terminating your instances is not sufficient to stop your request from being re-fulfilled. Therefore, it is a best practice to both cancel any active bids and terminate any running instances.

You use the **CancelSpotInstanceRequests** method to cancel a Spot request. The following example demonstrates how to cancel a Spot request.
public static void CancelSpotRequest(
    AmazonEC2Client ec2Client,
    string spotRequestId)
{
    var cancelRequest = new CancelSpotInstanceRequestsRequest();
    cancelRequest.SpotInstanceRequestIds.Add(spotRequestId);
    ec2Client.CancelSpotInstanceRequests(cancelRequest);
}

You use the TerminateInstances method to terminate an instance. The following example demonstrates how to obtain the instance identifier for an active Spot Instance and terminate the instance.

public static void TerminateSpotInstance(
    AmazonEC2Client ec2Client,
    string spotRequestId)
{
    var describeRequest = new DescribeSpotInstanceRequestsRequest();
    describeRequest.SpotInstanceRequestIds.Add(spotRequestId);

    // Retrieve the request we want to monitor.
    var describeResponse = ec2Client.DescribeSpotInstanceRequests(describeRequest);

    if (SpotInstanceState.Active == describeResponse.SpotInstanceRequests[0].State)
    {
        string instanceId = describeResponse.SpotInstanceRequests[0].InstanceId;

        var terminateRequest = new TerminateInstancesRequest();
        terminateRequest.InstanceIds = new List<string>() { instanceId };

        try
        {
            var terminateResponse = ec2Client.TerminateInstances(terminateRequest);
        }
        catch (AmazonEC2Exception ex)
        {
            // Check the ErrorCode to see if the instance does not exist.
            if ("InvalidInstanceID.NotFound" == ex.ErrorCode)
            {
                Console.WriteLine("Instance {0} does not exist.", instanceId);
            }
            else
            {
                // The exception was thrown for another reason, so re-throw the exception.
                throw;
            }
        }
    }
}

For more information about terminating active instances, see Terminating an Amazon EC2 Instance (p. 119).

Storing Archival Data Using Amazon Glacier

The AWS SDK for .NET supports Amazon Glacier, which is a storage service optimized for infrequently used data, or *cold data*. The service provides durable and extremely low-cost storage with security features for data archiving and backup. For more information, see Amazon Glacier Developer Guide.
Programming Models

The AWS SDK for .NET provides two programming models for working with Amazon Glacier. The following information describes these models and why and how to use them.

Topics

- Low-Level APIs (p. 130)
- High-Level APIs (p. 132)

Low-Level APIs

The AWS SDK for .NET provides low-level APIs for programming with Amazon Glacier. These low-level APIs map closely to the underlying REST API supported by Amazon Glacier. For each Amazon Glacier REST operation, the low-level APIs provide a corresponding method, a request object for you to provide request information, and a response object for you to process the Amazon Glacier response. The low-level APIs are the most complete implementation of the underlying Amazon Glacier operations.

The following example shows how to use the low-level APIs to list accessible vaults in Amazon Glacier:

```csharp
// using Amazon.Glacier;
// using Amazon.Glacier.Model;

var client = new AmazonGlacierClient();
var request = new ListVaultsRequest();
var response = client.ListVaults(request);

foreach (var vault in response.VaultList)
{
    Console.WriteLine("Vault: {0}", vault.VaultName);
    Console.WriteLine("  Creation date: {0}", vault.CreationDate);
    Console.WriteLine("  Size in bytes: {0}", vault.SizeInBytes);
    Console.WriteLine("  Number of archives: {0}", vault.NumberOfArchives);

    try
    {
        var requestNotifications = new GetVaultNotificationsRequest
        {
            VaultName = vault.VaultName
        };
        var responseNotifications =
            client.GetVaultNotifications(requestNotifications);
        Console.WriteLine("  Notifications:");
        Console.WriteLine("    Topic: {0}", responseNotifications.VaultNotificationConfig.SNSTopic);
        var events = responseNotifications.VaultNotificationConfig.Events;
        if (events.Any())
        {
            Console.WriteLine("    Events:");
            foreach (var e in events)
            {
                Console.WriteLine("{0}", e);
            }
        }
    }
```
else
{
    Console.WriteLine("   No events set.");
}
}

} catch (ResourceNotFoundException)
{
    Console.WriteLine("   No notifications set.");
}

var requestJobs = new ListJobsRequest{
    VaultName = vault.VaultName
};
var responseJobs = client.ListJobs(requestJobs);
var jobs = responseJobs.JobList;

if (jobs.Any())
{
    Console.WriteLine("   Jobs:");

    foreach (var job in jobs)
    {
        Console.WriteLine("   For job ID: ",
                 job.JobId);
        Console.WriteLine("   Archive ID: ",
                 job.ArchiveId);
        Console.WriteLine("   Archive size in bytes: ",
                 job.ArchiveSizeInBytes.ToString());
        Console.WriteLine("   Completed: ",
                 job.Completed);
        Console.WriteLine("   Completion date: ",
                 job.CompletionDate);
        Console.WriteLine("   Creation date: ",
                 job.CreationDate);
        Console.WriteLine("   Inventory size in bytes: ",
                 job.InventorySizeInBytes);
        Console.WriteLine("   Job description: ",
                 job.JobDescription);
        Console.WriteLine("   Status code: ",
                 job.StatusCode.Value);
        Console.WriteLine("   Status message: ",
                 job.StatusMessage);
    }
} else
{
    Console.WriteLine("   No jobs.");
}

For more examples, see:

- Using the AWS SDK for .NET
- Creating a Vault
- Retrieving Vault Metadata
- Downloading a Vault Inventory
- Configuring Vault Notifications
- Deleting a Vault
- Uploading an Archive in a Single Operation
• Uploading Large Archives in Parts
• Downloading an Archive
• Deleting an Archive

For related API reference information, see Amazon.Glacier and Amazon.Glacier.

High-Level APIs

The AWS SDK for .NET provides high-level APIs for programming with Amazon Glacier. To further simplify application development, these high-level APIs offer a higher-level abstraction for some of the operations, including uploading an archive and downloading an archive or vault inventory.

For examples, see the following topics in the Amazon Glacier Developer Guide:
• Using the AWS SDK for .NET
• Creating a Vault
• Deleting a Vault
• Upload an Archive to a Vault
• Uploading an Archive
• Uploading Large Archives in Parts
• Download an Archive from a Vault
• Downloading an Archive
• Delete an Archive from a Vault
• Deleting an Archive

For related API reference information, see Amazon.Glacier.Transfer in the AWS SDK for .NET API Reference.

Managing Users With AWS Identity and Access Management (IAM)

The AWS SDK for .NET supports IAM, which is a web service that enables AWS customers to manage users and user permissions in AWS.

The sample code is written in C#, but you can use the AWS SDK for .NET with any compatible language. When you install the AWS Toolkit for Visual Studio a set of C# project templates are installed. So the simplest way to start this project is to open Visual Studio, and then choose File, New Project, AWS Sample Projects, Deployment and Management, AWS Identity and Access Management User.

For related API reference information, see Amazon.IdentityManagement and Amazon.IdentityManagement.Model.

Prerequisites

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

Topics
• Managing IAM Account Aliases (p. 133)
Managing IAM Account Aliases

These .NET examples show you how to:

- Create an account alias for your AWS account ID
- List an account alias for your AWS account ID
- Delete an account alias for your AWS account ID

The Scenario

If you want the URL for your sign-in page to contain your company name or other friendly identifier instead of your AWS account ID, you can create an alias for your AWS account ID. If you create an AWS account alias, your sign-in page URL changes to incorporate the alias.

The following examples demonstrate how to manage your AWS account alias by using these methods of the AmazonIdentityManagementServiceClient class:

- CreateAccountAlias
- ListAccountAliases
- DeleteAccountAlias

For more information about IAM account aliases, see Your AWS Account ID and Its Alias in the IAM User Guide.

Create an Account Alias

Create an AmazonIdentityManagementServiceClient object. Next, create a CreateAccountAliasRequest object containing the new account alias you want to use. Call the CreateAccountAlias method of the AmazonIAMClient object. If the account alias is created, display the new alias on the console. If the name already exists, write the exception message to the console.

```csharp
public static void CreateAccountAlias()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new CreateAccountAliasRequest();
        request.AccountAlias = "my-aws-account-alias-2017";
        var response = iamClient.CreateAccountAlias(request);
        if (response.HttpStatusCode.ToString() == "OK")
            Console.WriteLine(request.AccountAlias + " created.");
        else
    }
    catch (Exception e)
    {
```
List Account Aliases

Create an `AmazonIdentityManagementServiceClient` object. Next, create a `ListAccountAliasesRequest` object. Call the `ListAccountAliases` method of the `AmazonIAMClient` object. If there is an account alias, display it on the console.

If there is no account alias, write the exception message to the console.

Note
There can be only one account alias.

```csharp
public static void ListAccountAliases()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new ListAccountAliasesRequest();
        var response = iamClient.ListAccountAliases(request);
        List<string> aliases = response.AccountAliases;
        foreach (string account in aliases)
        {
            Console.WriteLine("The account alias is: " + account);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Delete an Account Alias

Create an `AmazonIdentityManagementServiceClient` object. Next, create a `DeleteAccountAliasRequest` object containing the account alias you want to delete. Call the `DeleteAccountAlias` method of the `AmazonIAMClient` object. If the account alias is deleted, display the delete information on the console. If the name doesn't exist, write the exception message to the console.

```csharp
public static void DeleteAccountAlias()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new DeleteAccountAliasRequest();
        request.AccountAlias = "my-aws-account-alias-2017";
        var response = iamClient.DeleteAccountAlias(request);
        if (response.HttpStatusCode.ToString() == "OK")
            Console.WriteLine(request.AccountAlias + " deleted.");
        else
    }
    catch (NoSuchEntityException e)
    {
        Console.WriteLine(e.Message);
    }
}
Managing IAM Users

This .NET example shows you how to retrieve a list of IAM users, create and delete IAM users, and update an IAM user name.

You can create and manage users in IAM using these methods of the AmazonIdentityManagementServiceClient class:

- CreateUser
- ListUsers
- UpdateUser
- GetUser
- DeleteUser

For more information about IAM users, see IAM Users in the IAM User Guide.

For information about limitations on the number of IAM users you can create, see Limitations on IAM Entities in the IAM User Guide.

Create a User for Your AWS Account

Create an AmazonIdentityManagementServiceClient object. Next, create a CreateUserRequest object containing the user name you want to use for the new user. Call the CreateUser method of the AmazonIAMClient object. If the user name doesn't currently exist, display the name and the ARN for the user on the console. If the name already exists, write a message to that effect to the console.

```csharp
var client = new AmazonIdentityManagementServiceClient();
var request = new CreateUserRequest
{
    UserName = "DemoUser"
};
try
{
    var response = client.CreateUser(request);
    Console.WriteLine("User Name = '{0}', ARN = '{1}'", response.User.UserName, response.User.Arn);
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine("User 'DemoUser' already exists.");
}
```

List Users in Your AWS Account

This example lists the IAM users that have the specified path prefix. If no path prefix is specified, the action returns all users in the AWS account. If there are no users, the action returns an empty list.

Create an AmazonIdentityManagementServiceClient object. Next, create a ListUsersRequest object containing the parameters needed to list your users. Limit the number returned by setting the MaxItems parameter to 10. Call the ListUsers method of the AmazonIdentityManagementServiceClient object. Write each user's name and creation date to the console.

```csharp
public static void ListUsers()
{
    var iamClient = new AmazonIdentityManagementServiceClient();
};
```
var requestUsers = new ListUsersRequest() { MaxItems = 10 }; var responseUsers = iamClient.ListUsers(requestUsers);

foreach (var user in responseUsers.Users)
{
    Console.WriteLine("User " + user.UserName + " Created: " + user.CreateDate.ToShortDateString());
}

Update a User's Name

This example shows how to update the name or the path of the specified IAM user. Be sure you understand the implications of changing an IAM user’s path or name. For more information, see Renaming an IAM User in the IAM User Guide.

Create an AmazonIdentityManagementServiceClient object. Next, create an UpdateUserRequest object, specifying both the current and new user names as parameters. Call the UpdateUser method of the AmazonIdentityManagementServiceClient object.

```csharp
public static void UpdateUser()
{
    var client = new AmazonIdentityManagementServiceClient();
    var request = new UpdateUserRequest
    {
        UserName = "DemoUser",
        NewUserName = "NewUser"
    };
    try
    {
        var response = client.UpdateUser(request);
    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine("User 'NewUser' already exists.");
    }
}
```

Get Information about a User

This example shows how to retrieve information about the specified IAM user, including the user’s creation date, path, unique ID, and ARN. If you don’t specify a user name, IAM determines the user name implicitly based on the AWS access key ID used to sign the request to this API.

Create an AmazonIdentityManagementServiceClient object. Next, create a GetUserRequest object containing the user name you want to get information about. Call the GetUser method of the AmazonIdentityManagementServiceClient object to get the information. If the user doesn’t exist, an exception is thrown.

```csharp
public static void GetUser()
{
    var client = new AmazonIdentityManagementServiceClient();
    var request = new GetUserRequest
    {
        UserName = "DemoUser"
    };
```
try
{
    var response = client.GetUser(request);
    Console.WriteLine("Creation date: " +
    response.User.CreateDate.ToShortDateString());
    Console.WriteLine("Password last used: " +
    response.User.PasswordLastUsed.ToShortDateString());
    Console.WriteLine("UserId = " + response.User.UserId);
}
catch (NoSuchEntityException)
{
    Console.WriteLine("User 'DemoUser' does not exist.");
}

Delete a User

This example shows how to delete the specified IAM user. The user must not belong to any groups or
have any access keys, signing certificates, or attached policies.

Create an AmazonIdentityManagementServiceClient object. Next, create a DeleteUserRequest object
containing the parameters needed, which consists of the user name you want to delete. Call the
DeleteUser method of the AmazonIdentityManagementServiceClient object to delete it. If the user
doesn't exist, an exception is thrown.

```csharp
public static void DeleteUser()
{
    var client = new AmazonIdentityManagementServiceClient();
    var request = new DeleteUserRequest()
    {
        UserName = "DemoUser"
    };

    try
    {
        var response = client.DeleteUser(request);
    }
    catch (NoSuchEntityException)
    {
        Console.WriteLine("User DemoUser' does not exist.");
    }
}
```

Managing IAM Access Keys

These .NET examples shows you how to:

- Create an access key for a user
- Get the date that an access key was last used
- Update the status for an access key
- Delete an access key

The Scenario

Users need their own access keys to make programmatic calls to AWS from the AWS SDK for .NET. To
meet this need, you can create, modify, view, or rotate access keys (access key IDs and secret access keys)
for IAM users. When you create an access key, its status is Active by default, which means the user can use the access key for API calls.

The C# code uses the AWS SDK for .NET to manage IAM access keys using these methods of the `AmazonIdentityManagementServiceClient` class:

- `CreateAccessKey`
- `ListAccessKeys`
- `GetAccessKeyLastUsed`
- `UpdateAccessKey`
- `DeleteAccessKey`

For more information about IAM access keys, see Managing Access Keys for IAM Users in the IAM User Guide.

### Create Access Keys for a User

Call the `CreateAccessKey` method to create an access key named `S3UserReadOnlyAccess` for the IAM access keys examples. The `CreateAccessKey` method first creates a user named `S3UserReadOnlyAccess` with read-only access rights by calling the `CreateUser` method. It then creates an `AmazonIdentityManagementServiceClient` object and a `CreateAccessKeyRequest` object containing the `UserName` parameter needed to create new access keys. It then calls the `CreateAccessKey` method of the `AmazonIdentityManagementServiceClient` object.

```csharp
public static void CreateAccessKey()
{
    try
    {
        CreateUser();
        var iamClient = new AmazonIdentityManagementServiceClient();
        // Create an access key for the IAM user that can be used by the SDK
        var accessKey = iamClient.CreateAccessKey(new CreateAccessKeyRequest
        {
            // Use the user created in the CreateUser example
            UserName = "S3UserReadOnlyAccess"
        }).AccessKey;
    }
    catch (LimitExceededException e)
    {
        Console.WriteLine(e.Message);
    }
}

public static User CreateUser()
{
    var iamClient = new AmazonIdentityManagementServiceClient();
    try
    {
        // Create the IAM user
        var readOnlyUser = iamClient.CreateUser(new CreateUserRequest
        {
            UserName = "S3UserReadOnlyAccess"
        }).User;
        // Assign the read-only policy to the new user
        iamClient.PutUserPolicy(new PutUserPolicyRequest
        {
            UserName = readOnlyUser.UserName,
            PolicyName = "S3ReadOnlyAccess",
        });
    }
    catch (LimitExceededException e)
    {
        Console.WriteLine(e.Message);
    }
}```
List a User's Access Keys

Create an `AmazonIdentityManagementServiceClient` object and a `ListAccessKeysRequest` object containing the parameters needed to retrieve the user's access keys. This includes the IAM user's name and, optionally, the maximum number of access key pairs you want to list. Call the `ListAccessKeys` method of the `AmazonIdentityManagementServiceClient` object.

```csharp
public static void ListAccessKeys()
{
    var iamClient = new AmazonIdentityManagementServiceClient();
    var requestAccessKeys = new ListAccessKeysRequest
    {
        // Use the user created in the CreateAccessKey example
        UserName = "S3UserReadOnlyAccess",
        MaxItems = 10
    };
    var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);
    Console.WriteLine("  Access keys:");
    foreach (var accessKey in responseAccessKeys.AccessKeyMetadata)
    {
        Console.WriteLine("    {0}", accessKey.AccessKeyId);
    }
}
```

Get the Last Used Date for Access Keys

Create an `AmazonIdentityManagementServiceClient` object and a `ListAccessKeysRequest` object containing the `UserName` parameter needed to list the access keys. Call the `ListAccessKeys` method of the `AmazonIdentityManagementServiceClient` object. Loop through the access keys returned, displaying the `AccessKeyId` of each key and using it to create a `GetAccessKeyLastUsedRequest` object. Call the `GetAccessKeyLastUsed` method and display the time that the key was last used on the console.

```csharp
public static void GetAccessKeysLastUsed()
{
    var iamClient = new AmazonIdentityManagementServiceClient();
    var requestAccessKeys = new ListAccessKeysRequest
    {
        // Use the user we created in the CreateUser example
        UserName = "S3UserReadOnlyAccess"
    };
    var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);
    foreach (var accessKey in responseAccessKeys.AccessKeyMetadata)
    {
        var requestLastUsed = new GetAccessKeyLastUsedRequest
        {
            AccessKeyId = accessKey.AccessKeyId
        };
        var responseLastUsed = iamClient.GetAccessKeyLastUsed(requestLastUsed);
        Console.WriteLine("    Last used: {0}".FormatTimeDifference(responseLastUsed.LastUsedBefore), accessKey.AccessKeyId);
    }
}
```
var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);
Console.WriteLine("  Access keys:");
foreach (var accessKey in responseAccessKeys.AccessKeyMetadata)
{
    Console.WriteLine("    {0}", accessKey.AccessKeyId);
    GetAccessKeyLastUsedRequest request = new GetAccessKeyLastUsedRequest()
    {
        AccessKeyId = accessKey.AccessKeyId
    }
    var response = iamClient.GetAccessKeyLastUsed(request);
    Console.WriteLine("Key last used " +
    response.AccessKeyLastUsed.LastUsedDate.ToLongDateString());
}
}

Update the Status of an Access Key

Create an AmazonIdentityManagementServiceClient object and a ListAccessKeysRequest object containing the user name to list the keys for. The user name in this example is the user created for the other examples. Call the ListAccessKeys method of the AmazonIdentityManagementServiceClient. The ListAccessKeysResponse that is returned contains a list of the access keys for that user. Use the first access key in the list. Create an UpdateAccessKeyRequest object, providing the UserName, AccessKeyId, and Status parameters. Call the UpdateAccessKey method of the AmazonIdentityManagementServiceClient object.

```csharp
public static void UpdateKeyStatus()
{
    // This example changes the status of the key specified by its index in the list of
    // access keys
    // Optionally, you could change the keynumber parameter to be an AccessKey ID
    var iamClient = new AmazonIdentityManagementServiceClient();
    var requestAccessKeys = new ListAccessKeysRequest()
    {
        UserName = "S3UserReadOnlyAccess"
    };
    var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);
    UpdateAccessKeyRequest updateRequest = new UpdateAccessKeyRequest
    {
        UserName = "S3UserReadOnlyAccess",
        AccessKeyId = responseAccessKeys.AccessKeyMetadata[0].AccessKeyId,
        Status = StatusType.Active
    };
    iamClient.UpdateAccessKey(updateRequest);
    Console.WriteLine("  Access key " + updateRequest.AccessKeyId + " updated");
}
```

Delete Access Keys

Create an AmazonIdentityManagementServiceClient object and a ListAccessKeysRequest object containing the name of the user as a parameter. Call the ListAccessKeys method of the AmazonIdentityManagementServiceClient. The ListAccessKeysResponse that is returned contains a list of the access keys for that user. Delete each access key in the list by calling the DeleteAccessKey method of the AmazonIdentityManagementServiceClient.

```csharp
public static void DeleteAccessKeys()
{
    // Delete all the access keys created for the examples
    var iamClient = new AmazonIdentityManagementServiceClient();
    var requestAccessKeys = new ListAccessKeysRequest
    {
        UserName = "S3UserReadOnlyAccess"
    };
    var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);
    foreach (var accessKey in responseAccessKeys)
    {
        // Use the user created in the CreateUser example
```
Working with IAM Policies

The following examples show you how to:

• Create and delete IAM policies
• Attach and detach IAM policies from roles

The Scenario

You grant permissions to a user by creating a policy, which is a document that lists the actions that a user can perform and the resources those actions can affect. Any actions or resources that are not explicitly allowed are denied by default. You can create policies and attach them to users, groups of users, roles assumed by users, and resources.

Use the AWS SDK for .NET to create and delete policies and attach and detach role policies by using these methods of the AmazonIdentityManagementServiceClient class:

• CreatePolicy
• GetPolicy
• ListAttachedRolePolicies
• AttachRolePolicy
• DetachRolePolicy

For more information about IAM users, see Overview of Access Management: Permissions and Policies in the IAM User Guide.

Create an IAM Policy

Create an AmazonIdentityManagementServiceClient object. Next, create a CreatePolicy object containing the parameters needed to create a new policy, which consists of the name you want to use for the new policy and a policy document. You create the policy document by calling the provided GenerateRolePolicyDocument method. Upon returning from the CreatePolicy method call, the CreatePolicyResponse contains the policy ARN, which is displayed on the console. Please make a note of it so you can use it in the following examples.

```csharp
public static void CreatePolicyExample()
{
    UserName = "S3UserReadOnlyAccess"
};
var responseAccessKeys = iamClient.ListAccessKeys(requestAccessKeys);
Console.WriteLine(" Access keys:");

foreach (var accessKey in responseAccessKeys.AccessKeyMetadata)
{
    Console.WriteLine("   {0}", accessKey.AccessKeyId);
    iamClient.DeleteAccessKey(new DeleteAccessKeyRequest
    {
        UserName = "S3UserReadOnlyAccess",
        AccessKeyId = accessKey.AccessKeyId
    });
    Console.WriteLine("Access Key " + accessKey.AccessKeyId + " deleted");
}
}```
```csharp
{
    var client = new AmazonIdentityManagementServiceClient();
    // GenerateRolePolicyDocument is a custom method
    string policyDoc = GenerateRolePolicyDocument();

    var request = new CreatePolicyRequest
    {
        PolicyName = "DemoEC2Permissions",
        PolicyDocument = policyDoc
    };
    try
    {
        var createPolicyResponse = client.CreatePolicy(request);
        Console.WriteLine("Make a note, Policy named " +
        createPolicyResponse.Policy.PolicyName +
        " has Arn: : " + createPolicyResponse.Policy.Arn);
    }
    catch (EntityAlreadyExistsException)
    {
        Console.WriteLine("Policy 'DemoEC2Permissions' already exits.");
    }
}

public static string GenerateRolePolicyDocument()
{
    // using Amazon.Auth.AccessControlPolicy;
    // Create a policy that looks like this:
    /*
    "Version" : "2012-10-17",
    "Id" : "DemoEC2Permissions",
    "Statement" : [
        {
            "Sid" : "DemoEC2PermissionsStatement",
            "Effect" : "Allow",
            "Action" : [
                "s3:Get*",
                "s3:List*
            ],
            "Resource" : "**"
        }
    ]
    */
    var actionGet = new ActionIdentifier("s3:Get*");
    var actionList = new ActionIdentifier("s3:List*");
    var actions = new List<ActionIdentifier>();
    actions.Add(actionGet);
    actions.Add(actionList);
    var resource = new Resource("**");
    var resources = new List<Resource>();
    resources.Add(resource);
    var statement = new
    {
        Actions = actions,
        Id = "DemoEC2PermissionsStatement",
```
Resources = resources
};
var statements = new List<Amazon.Auth.AccessControlPolicy.Statement>();
statements.Add(statement);
var policy = new Policy
{
    Id = "DemoEC2Permissions",
    Version = "2012-10-17",
    Statements = statements
};
return policy.ToJson();

Get an IAM Policy

Create an AmazonIdentityManagementServiceClient object. Next, create a GetPolicyRequest object containing the parameter needed to get the policy, the policy ARN, which was returned by the CreatePolicy method in the previous example.

Call the GetPolicy method.

public static void GetPolicy()
{
    var client = new AmazonIdentityManagementServiceClient();
    var request = new GetPolicyRequest
    {
        PolicyArn = "arn:aws:iam::123456789:policy/DemoEC2Permissions"
    };
    try
    {
        var response = client.GetPolicy(request);
        Console.WriteLine("Policy " + response.Policy.PolicyName + " successfully retrieved");
    }
    catch (NoSuchEntityException)
    {
        Console.WriteLine("Policy 'DemoEC2Permissions' does not exist.");
    }
}

Attach a Managed Role Policy

Create an AmazonIdentityManagementServiceClient object. Next, create an AttachRolePolicy object containing the parameters needed to attach the policy to the role, the role name, and the JSON policy returned by the GenerateRolePolicyDocument method. Be sure to use a valid role from the roles associated with your AWS account.

public static void AttachRolePolicy()
{
    var client = new AmazonIdentityManagementServiceClient();
    string policy = GenerateRolePolicyDocument();
    CreateRoleRequest roleRequest = new CreateRoleRequest()
    {

RoleName = "tester",
AssumeRolePolicyDocument = policy
};

var request = new AttachRolePolicyRequest()
{
    RoleName = "tester"
};
try
{
    var response = client.AttachRolePolicy(request);
    Console.WriteLine("Policy DemoEC2Permissions attached to Role TestUser");
} catch (NoSuchEntityException)
{
    Console.WriteLine("Policy 'DemoEC2Permissions' does not exist");
} catch (InvalidInputException)
{
    Console.WriteLine("One of the parameters is incorrect");
}

## Detach a Managed Role Policy

Create an AmazonIdentityManagementServiceClient object. Next, create a DetachRolePolicy object containing the parameters needed to attach the policy to the role, the role name, and the JSON policy returned by the GenerateRolePolicyDocument method. Be sure to use the role you used to attach the policy in the previous example.

```csharp
public static void DetachRolePolicy()
{
    var client = new AmazonIdentityManagementServiceClient();
    string policy = GenerateRolePolicyDocument();
    CreateRoleRequest roleRequest = new CreateRoleRequest()
    {
        RoleName = "tester",
        AssumeRolePolicyDocument = policy
    };
    var request = new DetachRolePolicyRequest()
    {
        RoleName = "tester"
    };
    try
    {
        var response = client.DetachRolePolicy(request);
        Console.WriteLine("Policy DemoEC2Permissions detached from Role 'tester'");
    } catch (NoSuchEntityException e)
    {
        Console.WriteLine(e.Message);
    } catch (InvalidInputException i)
    {
        Console.WriteLine(i.Message);
    }
}
```
Working with IAM Server Certificates

These .NET examples show you how to:

- List server certificates
- Get server certificates
- Update server certificates
- Delete server certificates

The Scenario

In these examples, you'll basic tasks for managing server certificates for HTTPS connections. To enable HTTPS connections to your website or application on AWS, you need an SSL/TLS server certificate. To use a certificate that you obtained from an external provider with your website or application on AWS, you must upload the certificate to IAM or import it into AWS Certificate Manager.

These examples use the AWS SDK for .NET to send and receive messages by using these methods of the AmazonIdentityManagementServiceClient class:

- ListServerCertificates
- GetServerCertificate
- UpdateServerCertificate
- DeleteServerCertificate

For more information about server certificates, see Working with Server Certificates in the IAM User Guide.

List Your Server Certificates

Create an AmazonIdentityManagementServiceClient object. Next, create a ListServerCertificatesRequest object.

There are no required parameters. Call the ListServerCertificates method of the AmazonIdentityManagementServiceClient object.

```csharp
public static void ListCertificates()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new ListServerCertificatesRequest();
        var response = iamClient.ListServerCertificates(request);
        foreach (KeyValuePair<string, string> kvp in response.ResponseMetadata.Metadata)
        {
            Console.WriteLine("Key = {0}, Value = {1}",
                             kvp.Key, kvp.Value);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```
Get a Server Certificate


```csharp
public static void GetCertificate()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new GetServerCertificateRequest();
        request.ServerCertificateName = "CERTIFICATE_NAME";
        var response = iamClient.GetServerCertificate(request);
        Console.WriteLine("CertificateName = " +
        response.ServerCertificate.ServerCertificateMetadata.ServerCertificateName);
        Console.WriteLine("Certificate Arn = " +
        response.ServerCertificate.ServerCertificateMetadata.Arn);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Update a Server Certificate


```csharp
public static void UpdateCertificate()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new UpdateServerCertificateRequest();
        request.ServerCertificateName = "CERTIFICATE_NAME";
        request.NewServerCertificateName = "NEW_Certificate_NAME";
        var response = iamClient.UpdateServerCertificate(request);
        if (response.HttpStatusCode.ToString() == "OK")
            Console.WriteLine("Update successful");
        else
            Console.WriteLine("HTtpStatusCode returned = " +
            response.HttpStatusCode.ToString());
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Delete a Server Certificate

public static void DeleteCertificate()
{
    try
    {
        var iamClient = new AmazonIdentityManagementServiceClient();
        var request = new DeleteServerCertificateRequest();
        request.ServerCertificateName = "CERTIFICATE_NAME";
        var response = iamClient.DeleteServerCertificate(request);
        if (response.HttpStatusCode.ToString() == "OK")
            Console.WriteLine(request.ServerCertificateName + " deleted");
        else
            Console.WriteLine("HTTPStatusCode returned = " +
                    response.HttpStatusCode.ToString());
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}

public static void ListUsers()
{
    var iamClient = new AmazonIdentityManagementServiceClient();
    var requestUsers = new ListUsersRequest();
    var responseUsers = iamClient.ListUsers(requestUsers);
    foreach (var user in responseUsers.Users)
    {
        Console.WriteLine("For user {0}:", user.UserName);
        Console.WriteLine("  In groups:");

        var requestGroups = new ListGroupsForUserRequest
        {
            UserName = user.UserName
        };
        var responseGroups = iamClient.ListGroupsForUser(requestGroups);
        foreach (var group in responseGroups.Groups)
        {
            Console.WriteLine("    {0}", group.GroupName);
        }
        Console.WriteLine("  Policies:");

        var requestPolicies = new ListUserPoliciesRequest
        {
            UserName = user.UserName
        };
        var responsePolicies = iamClient.ListUserPolicies(requestPolicies);
        foreach (var policy in responsePolicies.PolicyNames)
        {
            Console.WriteLine("    {0}", policy);
        }
    }
}
Granting Access Using an IAM Role

This .NET example shows you how to:

- Create a sample program that retrieves an object from Amazon S3
- Create an IAM role
- Launch an Amazon EC2 instance and specify the IAM role
- Run the sample on the Amazon EC2 instance

The Scenario

All requests to AWS must be cryptographically signed by using credentials issued by AWS. Therefore, you need a strategy to manage credentials for software that runs on Amazon EC2 instances. You have to distribute, store, and rotate these credentials securely, but also keep them accessible to the software.

IAM roles enable you to effectively manage AWS credentials for software running on EC2 instances. You create an IAM role and configure it with the permissions the software requires. For more information about the benefits of using IAM roles, see IAM Roles for Amazon EC2 in the Amazon EC2 User Guide for Windows Instances and Roles (Delegation and Federation) in the IAM User Guide.

To use the permissions, the software constructs a client object for the AWS service. The constructor searches the credentials provider chain for credentials. For .NET, the credentials provider chain is as follows:

- The App.config file
- The instance metadata associated with the IAM role for the EC2 instance

If the client doesn't find credentials in App.config, it retrieves temporary credentials that have the same permissions as those associated with the IAM role from instance metadata. The credentials are stored by the constructor on behalf of the application software, and are used to make calls to AWS from that client object. Although the credentials are temporary and eventually expire, the SDK client periodically refreshes them so that they continue to enable access. This periodic refresh is completely transparent to the application software.

The following examples show a sample program that retrieves an object from Amazon S3 using the AWS credentials you configure. You create an IAM role to provide the AWS credentials. Finally, you launch an instance with an IAM role that provides the AWS credentials to the sample program running on the instance.
Create a Sample that Retrieves an Object from Amazon S3

The following sample code requires a text file in an Amazon S3 bucket that you have access to, and AWS credentials that provide you with access to the Amazon S3 bucket.

For more information about creating an Amazon S3 bucket and uploading an object, see the Amazon S3 Getting Started Guide. For more information about AWS credentials, see Configuring AWS Credentials (p. 11).

```csharp
using System;
using System.Collections.Specialized;
using System.IO;

using Amazon;
using Amazon.S3;
using Amazon.S3.Model;

namespace Aws3Sample
{
    class S3Sample
    {
        public static void Main(string[] args)
        {
            ReadS3File("bucket-name", "s3-file-name", "output-file-name");

            Console.WriteLine("Press enter to continue");
            Console.ReadLine();
        }

        public static void ReadS3File(
            string bucketName,
            string keyName,
            string filename)
        {
            string responseBody = "";

            try
            {
                using (var s3Client = new AmazonS3Client())
                {
                    Console.WriteLine("Retrieving (GET) an object");

                    var request = new GetObjectRequest()
                    {
                        BucketName = bucketName,
                        Key = keyName
                    };

                    using (var response = s3Client.GetObject(request))
                    using (var responseStream = response.ResponseStream)
                    using (var reader = new StreamReader(responseStream))
                    {
                        responseBody = reader.ReadToEnd();
                    }

                    using (var s = new FileStream(filename, FileMode.Create))
                    using (var writer = new StreamWriter(s))
                    {
                        writer.WriteLine(responseBody);
                    }
                }
            }
            catch (AmazonS3Exception s3Exception)
            {
                throw;
            }
        }
    }
}
```
To test the sample code
1. Open Visual Studio and create an AWS Console project.
2. Add the AWSSDK.S3 package to your project.
3. Replace the code in the Program.cs file with the sample code.
4. Replace the following values:
   
   bucket-name
   
   The name of your Amazon S3 bucket.

   s3-file-name
   
   The path and name of a text file in the bucket.

   output-file-name
   
   The path and file name to write the file to.

5. Compile and run the sample program. If the program succeeds, it displays the following output and creates a file named s3Object.txt on your local drive that contains the text it retrieved from the text file in Amazon S3.

   Retrieving (GET) an object

   If the program fails, be sure you're using credentials that provide you with access to the bucket.

6. (Optional) Transfer the sample program to a running Windows instance on which you haven't set up credentials. Run the program and verify that it fails because it can't locate credentials.

Create an IAM Role

Create an IAM role that has the appropriate permissions to access Amazon S3.

To create the IAM role

You can use the Amazon EC2 console or the AWS SDK for .NET to launch an EC2 instance with an IAM role.

- Using the console: Follow the directions in Launching a Windows Instance in the Amazon EC2 User Guide for Windows Instances. When you reach the Review Instance Launch page, choose Edit instance details. In IAM role, specify the IAM role you created previously. Complete the procedure as directed. You'll need to create or use an existing security group and key pair to connect to the instance.
- Using the AWS SDK for .NET: See Launching an Amazon EC2 Instance (p. 114).

An IAM user can't launch an instance with an IAM role without the permissions granted by the following policy.
To transfer the sample program to your EC2 instance, connect to the instance using the AWS Management Console as described in the following procedure.

**Note**
Alternatively, connect using the Toolkit for Visual Studio (see Connecting to an Amazon EC2 Instance in the AWS Toolkit for Visual Studio) and then copy the files from your local drive to the instance. The Remote Desktop session is automatically configured so that your local drives are available to the instance.

1. Open the IAM console.
2. In the navigation pane, choose Roles, and then choose Create New Role.
3. Type a name for the role, and then choose Next Step. Remember this name because you'll need it when you launch your EC2 instance.
4. Under AWS Service Roles, choose Amazon EC2. Under Select Policy Template, choose Amazon S3 Read Only Access. Review the policy, and then choose Next Step.
5. Review the role information, and then choose Create Role.

**To run the sample program on the EC2 instance**

1. Open the Amazon EC2 console.
2. Get the password for your EC2 instance:
   - In the navigation pane, choose Instances. Choose the instance, and then choose

**Connect.**

**a.** In the Connect To Your Instance dialog box, choose Get Password. (It will take a few minutes after the instance is launched before the password is available.)

**b.** Choose Browse and navigate to the private key file you created when you launched the instance. Choose the file, and then choose Open to copy the file's contents into the contents box.

**c.** Choose Decrypt Password. The console displays the default administrator password for the instance in the Connect To Your Instance dialog box, replacing the link to Get Password shown earlier with the password.

**d.** Record the default administrator password or copy it to the clipboard. You need this password to connect to the instance.

3. Connect to your EC2 instance:
   - Choose Download Remote Desktop File. When your browser prompts you, save the .rdp file. When you finish, choose Close to close the Connect To Your Instance dialog box.

   - Navigate to your downloads directory, right-click the .rdp file, and then choose Edit. On the Local Resources tab, under Local devices and resources, choose More. Choose Drives to make your local drives available to your instance. Then choose OK.
c. Choose Connect to connect to your instance. You may get a warning that the publisher of the remote connection is unknown.

d. Sign in to the instance when prompted, using the default Administrator account and the default administrator password you recorded or copied previously.

Sometimes copying and pasting content can corrupt data. If you encounter a "Password Failed" error when you sign in, try typing in the password manually. For more information, see Connecting to Your Windows Instance Using RDP and Troubleshooting Windows Instances in the Amazon EC2 User Guide for Windows Instances.

4. Copy the program and the AWS assemblies (AWSSDK.Core.dll and AWSSDK.S3.dll) from your local drive to the instance.

5. Run the program and verify that it succeeds using the credentials provided by the IAM role.

Running Serverless Web Applications Using AWS Lambda

The AWS SDK for .NET supports AWS Lambda, a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second. You pay only for the compute time you consume - there is no charge when your code is not running. With AWS Lambda, you can run code for virtually any type of application or backend service - all with zero administration. AWS Lambda runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging.

You can use AWS Lambda to run your code in response to events, such as changes to data in an Amazon S3 bucket or an Amazon DynamoDB table; to run your code in response to HTTP requests using Amazon API Gateway; or invoke your code using API calls made using AWS SDKs. With these capabilities, you can use Lambda to easily build data processing triggers for AWS services like Amazon S3 and Amazon DynamoDB process streaming data stored in Kinesis, or create your own back end that operates at AWS scale, performance, and security.

You can also build serverless applications composed of functions that are triggered by events and automatically deploy them using AWS CodePipeline and AWS CodeBuild. For more information, see Deploying Lambda-based Applications.

For more information about the AWS Lambda execution environment, see Amazon Lambda Execution Environment and Available Libraries. For information about how AWS Lambda determines compute resources required to execute your code, see Compute Requirements – Amazon Lambda Function Configuration.

The following examples demonstrate how to use the AWS SDK for .NET to create and use AWS Lambda applications.

The sample code is written in C#, but you can use the AWS SDK for .NET with any compatible language. The AWS SDK for .NET installs a set of C# project templates. So the simplest way to start this project is to open Visual Studio, and then choose File, New Project, AWS Samples, AWS Console Project.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).
AWS Lambda Function Examples

To create a Lambda function, you first package your code and dependencies in a deployment package. Then, you upload the deployment package to AWS Lambda to create your Lambda function.

Instructions to create a deployment package vary depending on the language you choose to author the code. For example, you can use build plugins such as Jenkins (for Node.js and Python), and Maven (for Java) to create the deployment packages. For more information, see Creating a Deployment Package.

AWS Lambda provides the CreateFunction operation, which is what you use to create a Lambda function. You can use the AWS Lambda console, AWS CLI, and AWS SDKs to create a Lambda function. Internally, all of these interfaces call the CreateFunction operation.

In addition to providing your deployment package, you can provide configuration information when you create your Lambda function including the compute requirements of your Lambda function, the name of the handler method in your Lambda function, and the runtime, which depends on the language you chose to author your code. For more information, see Lambda Functions.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

These examples use the default client constructor which constructs a AmazonLambdaClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Create a Lambda Function

In this example, you create a new Lambda function. The function metadata is created from the request parameters, and the code for the function is provided by a .zip file in the request body. If the function name already exists, the operation will fail. Note that the function name is case-sensitive.

If you are using versioning, you can also publish a version of the Lambda function you are creating using the Publish parameter. For more information about versioning, see AWS Lambda Function Versioning and Aliases.

This operation requires permission for the lambda:CreateFunction action.

Call the CreateFunction method with a CreateFunctionRequest containing a zip file of the function code and the properties described in the request set to identify the function.

```csharp
byte[] zip = File.ReadAllBytes("hello-world-nodejs-285b4fcc-4995-4026-92cd-8fec015ac1db"); // Function to upload
MemoryStream stream = new MemoryStream(zip);
FunctionCode code = new FunctionCode();
code.ZipFile = stream;
```
```csharp
var response = client.CreateFunction(new CreateFunctionRequest
{
    Code = code,  // Code to upload
    Description = "Test NodeJS function",
    FunctionName = "hello-world-nodejs",
    Handler = "index.handler",
    MemorySize = 128,
    Publish = true,
    Role = "arn:aws:iam::123456789012:role/lambda-sample_invoke_role",  // replace with the
    actual arn of the execution role you created
    Runtime = "nodejs6.10",
    Timeout = 15,
    VpcConfig = new VpcConfig { }
});

string codeSha256 = response.CodeSha256;
long codeSize = response.CodeSize;
Console.WriteLine("description = " + response.Description);
Console.WriteLine("functionArn = " + response.FunctionArn);
Console.WriteLine("functionName = " + response.FunctionName);
Console.WriteLine("handler = " + response.Handler);
Console.WriteLine("lastModified = " + response.LastModified);
Console.WriteLine("memorySize = " + response.MemorySize.ToString());
Console.WriteLine("role = " + response.Role);
Console.WriteLine("runtime = " + response.Runtime);
Console.WriteLine("timeout = " + response.Timeout.ToString());
Console.WriteLine("version = " + response.Version);

// Test function
InvokeResponse iResponse = client.Invoke(new InvokeRequest()
{ FunctionName = "hello-world-nodejs", InvocationType="Event", LogType="Tail"});

string functionError = iResponse.FunctionError;
string logResult = iResponse.LogResult;
MemoryStream payload = iResponse.Payload;
int statusCode = iResponse.StatusCode;

// Read the returned payload
var sr = new StreamReader(iResponse.Payload);
string result = sr.ReadToEnd();
```

### List Lambda Functions

In this example, you get a list of your Lambda functions. For each function, the response includes the function configuration information. You must use `GetFunction` to retrieve the code for your function.

This operation requires permission for the `lambda:ListFunctions` action.

If you are using the versioning feature, you can list all of your functions or only `:code;` `$LATEST` versions. For information about the versioning feature, see AWS Lambda Function Versioning and Aliases.

Call the `ListFunctions` method with an `ListFunctionsRequest` containing the name of the function to be listed. The returned list of `FunctionConfiguration` objects contains the descriptive information for the functions.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.ListFunctions(new ListFunctionsRequest
{
    Marker = "",
    MaxItems = 123
});
```
List<FunctionConfiguration> functions = response.Functions;
string nextMarker = response.NextMarker;

Update Lambda Function Code

In this example, you update the code for the specified Lambda function. This operation must only be used on an existing Lambda function and cannot be used to update the function configuration.

If you are using the versioning feature, note this API will always update the $LATEST version of your Lambda function. For information about the versioning feature, see AWS Lambda Function Versioning and Aliases.

This operation requires permission for the lambda:UpdateFunctionCode action.

Call the UpdateFunctionCode method with an UpdateFunctionCodeRequest containing the containing a zip file of the function code and the properties described in the request set to identify the function.

AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.UpdateFunctionCode(new UpdateFunctionCodeRequest
{
    FunctionName = "myFunction",
    Publish = true,
    S3Bucket = "myBucket",
    S3Key = "myKey",
    S3ObjectVersion = "1",
    ZipFile = new MemoryStream(fileb://file-path/file.zip)
});

string codeSha256 = response.CodeSha256;
long codeSize = response.CodeSize;
string description = response.Description;
string functionArn = response.FunctionArn;
string functionName = response.FunctionName;
string handler = response.Handler;
string lastModified = response.LastModified;
int memorySize = response.MemorySize;
string role = response.Role;
string runtime = response.Runtime;
int timeout = response.Timeout;
string version = response.Version;
VpcConfigDetail vpcConfig = response.VpcConfig;

Update a Lambda Function Configuration

In this example, you update the configuration parameters for the specified Lambda function by using the values provided in the request. You provide only the parameters you want to change. This operation must only be used on an existing Lambda function and cannot be used to update the function's code.

If you are using the versioning feature, note this API will always update the $LATEST version of your Lambda function. For information about the versioning feature, see AWS Lambda Function Versioning and Aliases.

This operation requires permission for the lambda:UpdateFunctionConfiguration action.

Call the UpdateFunctionConfiguration method with an UpdateFunctionConfigurationRequest containing the containing a zip file of the function code and the properties to change (described in the request) set to identify the function.
Get Function

In this example, you get the configuration information of the Lambda function and a presigned URL link to the .zip file you uploaded with CreateFunction so you can download the .zip file. Note that the URL is valid for up to 10 minutes. The configuration information is the same information you provided as parameters when uploading the function.

Using the optional Qualifier parameter, you can specify a specific function version for which you want this information. If you don't specify this parameter, the API uses unqualified function ARN which return information about the $LATEST version of the Lambda function. For more information, see AWS Lambda Function Versioning and Aliases.

This operation requires permission for the lambda:GetFunction action.

Call the GetFunction method with an GetFunctionRequest containing the function name and optionally the version.

Get Function Information

In this example, you get the configuration information of the Lambda function. This the same information you provided as parameters when uploading the function by using CreateFunction.
If you are using the versioning feature, you can retrieve this information for a specific function version by using the optional Qualifier parameter and specifying the function version or alias that points to it. If you don't provide it, the API returns information about the $LATEST version of the function. For more information about versioning, see AWS Lambda Function Versioning and Aliases.

This operation requires permission for the `lambda:GetFunctionConfiguration` operation.

Call the `GetFunctionConfiguration` method with an `GetFunctionConfigurationRequest` containing the function name and optionally the version.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.GetFunctionConfiguration(new GetFunctionConfigurationRequest
{
    FunctionName = "myFunction",
    //Qualifier = "1"
});
string codeSha256 = response.CodeSha256;
long codeSize = response.CodeSize;
DeadLetterConfig deadLetterConfig = response.DeadLetterConfig;
string description = response.Description;
EnvironmentResponse environment = response.Environment;
string functionArn = response.FunctionArn;
string functionName = response.FunctionName;
string handler = response.Handler;
string kmsKeyArn = response.KMSKeyArn;
string lastModified = response.LastModified;
int memorySize = response.MemorySize;
string role = response.Role;
string runtime = response.Runtime;
int timeout = response.Timeout;
string version = response.Version;
VpcConfigDetail vpcConfig = response.VpcConfig;
```

Invoke a Lambda Function

In this example, you invoke a specific Lambda function. If you are using the versioning feature, you can invoke the specific function version by providing function version or alias name that is pointing to the function version using the `Qualifier` parameter in the request. If you don't provide the `Qualifier` parameter, the `$LATEST` version of the Lambda function is invoked. Invocations occur at least once in response to an event and functions must give the same result if invoked more than once. For information about the versioning feature, see AWS Lambda Function Versioning and Aliases.

This operation requires permission for the `lambda:InvokeFunction` action.

Call the `InvokeFunction` method with an `InvokeFunctionRequest` containing the name of the function to be invoked.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.Invoke(new InvokeRequest
{
    FunctionName = "MyFunction",
    InvocationType = "Event",
    LogType = "Tail",
    //Qualifier = "1"
});
string functionError = response.FunctionError;
string logResult = response.LogResult;
MemoryStream payload = response.Payload;
```
Delete a Lambda Function

In this example, you delete the specified Lambda function code and configuration.

If you are using the versioning feature and you don't specify a function version in your DeleteFunction request, AWS Lambda will delete the function, including all its versions, and any aliases pointing to the function versions. To delete a specific function version, you must provide the function version via the Qualifier parameter. For information about function versioning, see AWS Lambda Function Versioning and Aliases.

When you delete a function the associated resource policy is also deleted. You will need to delete the event source mappings explicitly.

This operation requires permission for the lambda:DeleteFunction action.

Call the DeleteFunction method with a DeleteFunctionRequest containing the name of the function to be deleted.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.DeleteFunction(new DeleteFunctionRequest
{   
    FunctionName = "myFunction",
    //Qualifier = "1"
});
```

AWS Lambda Authentication Examples

Access to AWS Lambda requires credentials that AWS can use to authenticate your requests. Those credentials must have permissions to access AWS resources, such as an AWS Lambda function or an Amazon S3 bucket. The credentials for Lambda functions are determined by the Role property set when the function was created.

An IAM role is similar to a user, in that it is an AWS identity with permission policies that determine what the identity can and cannot do in AWS. However, instead of being uniquely associated with one person, a role is intended to be assumable by anyone who needs it. Also, a role does not have any credentials (password or access keys) associated with it. Instead, if a user is assigned to a role, access keys are created dynamically and provided to the user.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

These examples use the default client constructor which constructs a AmazonLambdaClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Get Account Settings

In this example, you get your account settings.

You can use this operation to retrieve Lambda limits information, such as code size and concurrency limits. For more information about limits, see AWS Lambda Limits. You can also retrieve resource usage statistics, such as code storage usage and function count.
Call the `GetAccountSettings` method with a `GetAccountSettingsRequest` containing a zip file of the function code and properties set to identify the function.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.GetAccountSettings(new GetAccountSettingsRequest
{
});
AccountLimit accountLimit = response.AccountLimit;
AccountUsage accountUsage = response.AccountUsage;
```

### Get Policy Information

In this example, you get the resource policy associated with the specified Lambda function. If you are using the versioning feature, you can get the resource policy associated with the specific Lambda function version or alias by specifying the version or alias name using the `:code;` `Qualifier` parameter. For more information about versioning, see AWS Lambda Function Versioning and Aliases.

You need permission for the `lambda:GetPolicy` action.

Call the `GetPolicy` method with a `GetPolicyRequest` containing the function name and optionally the version.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.GetPolicy(new GetPolicyRequest
{
    FunctionName = "myFunction",
    //Qualifier = "1"
});
string policy = response.Policy;
```

### Add a Permission

In this example, you add a permission to the resource policy associated with the specified AWS Lambda function. You use resource policies to grant permissions to event sources that use push model. In a push model, event sources (such as Amazon S3 and custom applications) invoke your Lambda function. Each permission you add to the resource policy allows an event source, permission to invoke the Lambda function.

For information about the push model, see AWS Lambda: How it Works.

If you are using versioning, the permissions you add are specific to the Lambda function version or alias you specify in the `AddPermission` request via the `Qualifier` parameter. For more information about versioning, see AWS Lambda Function Versioning and Aliases.

This operation requires permission for the `lambda:AddPermission` action.

This operation requires permission for the `lambda:CreateFunction` action.

Call the `AddPermission` method with a `AddPermissionRequest` containing the permission, principal, source account, source ARN, and statement ID.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.AddPermission(new AddPermissionRequest
{
    Action = "lambda:InvokeFunction",
```
Remove a Lambda Function Permission

In this example, you remove individual permissions from an resource policy associated with a Lambda function by providing a statement ID that you provided when you added the permission.

If you are using versioning, the permissions you remove are specific to the Lambda function version or alias you specify in the AddPermission request via the Qualifier parameter. For more information about versioning, see AWS Lambda Function Versioning and Aliases.

Note that removal of a permission will cause an active event source to lose permission to the function.

You need permission for the lambda:RemovePermission action.

Call the RemovePermission method with an RemovePermissionRequest containing the name of the permission to be removed and the statement ID associated with that permission.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.RemovePermission(new RemovePermissionRequest
{
    FunctionName = "myFunction",
    //Qualifier = "1",
    StatementId = "role-statement-id"
});

string statement = response.Statement;
```

AWS Lambda Alias Examples

You can create aliases for your Lambda function. An AWS Lambda alias is like a pointer to a specific Lambda function version. For more information about versioning, see Introduction to AWS Lambda Versioning. By using aliases, you can access the Lambda function it is pointing to (for example, to invoke the function) without the caller having to know the specific version the alias is pointing to.

AWS Lambda aliases enable the following use cases:

- Easier support for promotion of new versions of Lambda functions and roll back when needed
- Simplify management of event source mappings

An AWS Lambda alias is a resource similar to a Lambda function. However, you can't create an alias independently. You create an alias for an existing Lambda function. If a Lambda function is a resource, you can think of an AWS Lambda alias as a subresource that is associated with a Lambda function.

Both the Lambda function and alias are AWS Lambda resources, and like all other AWS resources they both have unique Amazon Resource Names (ARNs). The following example shows a Lambda function (the $LATEST version), with one published version. Each version has an alias pointing to it.

You can access the function using either the function ARN or the alias ARN.

**Prerequisite Tasks**
Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

Create a Lambda Function Alias

In this example, you create an alias for a Lambda function. For more information, see Introduction to AWS Lambda Aliases.

This requires permission for the `lambda:CreateAlias` action.

This example uses the default client constructor which constructs a `AmazonLambdaClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Call the `CreateAlias` method with an `CreateAliasRequest` containing the function name, version and alias name.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.CreateAlias(new CreateAliasRequest
{
    Description = "",
    FunctionName = "myFunction",
    FunctionVersion = "1",
    Name = "functionAlias"
});

string aliasArn = response.AliasArn;
string description = response.Description;
string functionVersion = response.FunctionVersion;
string name = response.Name;
```

Update a Lambda Function Alias

In this example, you update the function version to which the alias points and the alias description. For more information, see Introduction to AWS Lambda Aliases.

This requires permission for the `lambda:UpdateAlias` action.

Call the `UpdateAlias` method with an `UpdateAliasRequest` containing the function name, version and name of the alias to be updated.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.UpdateAlias(new UpdateAliasRequest
{
    Description = "",
    FunctionName = "myFunction",
    FunctionVersion = "1",
    Name = "functionAlias"
});

string aliasArn = response.AliasArn;
string description = response.Description;
string functionVersion = response.FunctionVersion;
string name = response.Name;
```

Get Alias Information

In this example, you get the specified alias information such as the alias ARN, description, and function version it is pointing to. For more information, see Introduction to AWS Lambda Aliases.
This requires permission for the `lambda:GetAlias` action.

Call the `GetAlias` method with a `GetAliasRequest` containing the name of the function and the alias.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.GetAlias(new GetAliasRequest
{
    FunctionName = "myFunction",
    Name = "myFunctionAlias"
});
string aliasArn = response.AliasArn;
string description = response.Description;
string functionVersion = response.FunctionVersion;
string name = response.Name;
```

### List Lambda Aliases

In this example, you get a list of aliases created for a Lambda function. For each alias, the response includes information such as the alias ARN, description, alias name, and the function version to which it points. For more information, see Introduction to AWS Lambda Aliases.

This requires permission for the `lambda:ListAliases` action.

Call the `DeleteAlias` method with a `DeleteAliasRequest` containing the function name to be listed. The returned list of `AliasConfiguration` objects contains the descriptive information for the function aliases.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.ListAliases(new ListAliasesRequest
{
    FunctionName = "myFunction",
    FunctionVersion = "1",
    Marker = "",
    MaxItems = 123
});
List<AliasConfiguration> aliases = response.Aliases;
string nextMarker = response.NextMarker;
```

### Delete a Lambda Function Alias

In this example, you delete the specified Lambda function alias. For more information, see Introduction to AWS Lambda Aliases.

This requires permission for the `lambda:DeleteAlias` action.

Call the `DeleteAlias` method with a `DeleteAliasRequest` containing the function name and alias to be deleted.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.DeleteAlias(new DeleteAliasRequest
{
    FunctionName = "myFunction",
    Name = "alias"
});
```
AWS Lambda Versioning Examples

When you create a Lambda function, there is only one version. It is the $LATEST version.

You can refer to this function using its Amazon Resource Name (ARN).

When you publish a version, AWS Lambda makes a snapshot copy of the Lambda function code (and configuration) in the $LATEST version. A published version is immutable. That is, you can't change the code or configuration information. The new version has a unique ARN that includes a version number suffix.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

These examples use the default client constructor which constructs a AmazonLambdaClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Publish a Version of a Lambda Function

In this example, you publish a version of your function from the current snapshot of $LATEST. That is, AWS Lambda takes a snapshot of the function code and configuration information from $LATEST and publishes a new version. The code and configuration cannot be modified after publication. For information about the versioning feature, see AWS Lambda Function Versioning and Aliases.

Call the PublishVersion method with an PublishVersionRequest containing the name of the function to be published. The metadata for the new version of the function is returned in the response.

```csharp
AmazonLambdaClient client = new AmazonLambdaClient();
var response = client.PublishVersion(new PublishVersionRequest
{
  CodeSha256 = "",
  Description = "",
  FunctionName = "myFunction"
});

string codeSha256 = response.CodeSha256;
long codeSize = response.CodeSize;
string description = response.Description;
string functionArn = response.FunctionArn;
string functionName = response.FunctionName;
string handler = response.Handler;
string lastModified = response.LastModified;
int memorySize = response.MemorySize;
string role = response.Role;
string runtime = response.Runtime;
int timeout = response.Timeout;
string version = response.Version;
VpcConfigDetail vpcConfig = response.VpcConfig;
```

List Lambda Aliases

In this example, you list all versions of a function. For information about the versioning feature, see AWS Lambda Function Versioning and Aliases.

Call the ListVersionsByFunction method with an ListVersionsByFunctionRequest containing the function name to be listed. The returned list of FunctionConfiguration objects contains the descriptive information for the functions.
Detecting faces with Amazon Rekognition

Amazon Rekognition makes it easy to add image and video analysis to your applications. You just provide an image or video to the Rekognition API, and the service can identify objects, people, text, scenes, and activities. You can detect, analyze, and compare faces for a wide variety of use cases.

The DetectFaces operation looks for key facial features such as eyes, nose, and mouth to detect faces in an input image. Use RecognizeCelebrities to recognize thousands of celebrities in a wide range of categories, such as entertainment and media, sports, business, and politics.

The following example demonstrates both of these APIs in self-contained functions.

```
static void IdentifyFaces(string filename)
{
    // Using USWest2, not the default region
    AmazonRekognitionClient rekoClient = new AmazonRekognitionClient(Amazon.RegionEndpoint.USWest2);

    DetectFacesRequest dfr = new DetectFacesRequest();

    // Request needs image butes, so read and add to request
    byte[] data = null;
    using (FileStream fs = new FileStream(filename, FileMode.Open, FileAccess.Read))
    {
        data = new byte[fs.Length];
        fs.Read(data, 0, data.Length);
        img.ImageBytes = data;
        dfr.Image = img;
    }
    dfr.Max heltions = 123;
    dfr.Region = Amazon.RegionEndpoint.USWest2;

    DetectFacesResponse dfrs = rekoClient.DetectFaces(dfr);
    int[] faces = dfrs.Faces.Select(f => f.Location).ToArray();
    foreach (int face in faces)
    {
        // Process each face
    }

    // Similar code for RecognizeCelebrities
```
data = new byte[fs.Length];
fs.Read(data, 0, (int)fs.Length);
}
img.Bytes = new MemoryStream(data);
dfr.Image = img;
var outcome = rekoClient.DetectFaces(dfr);
if (outcome.FaceDetails.Count > 0)
{
    // Load a bitmap to modify with face bounding box rectangles
    Pen pen = new Pen(Color.Black, 3);

    // Create a graphics context
    using (var graphics = Graphics.FromImage(facesHighlighted))
    {
        // Get the bounding box
        foreach (var fd in outcome.FaceDetails)
        {
            BoundingBox bb = fd.BoundingBox;
            graphics.DrawRectangle(pen, x: facesHighlighted.Width * bb.Left,
                                   y: facesHighlighted.Height * bb.Top,
                                   width: facesHighlighted.Width * bb.Width,
                                   height: facesHighlighted.Height * bb.Height);
        }
    }
    // Save the image with highlights as PNG
    string fileout = filename.Replace(Path.GetExtension(filename), "_faces.png");
    facesHighlighted.Save(fileout, System.Drawing.Imaging.ImageFormat.Png);
    Console.WriteLine(">>> " + outcome.FaceDetails.Count + " face(s) highlighted in file " + fileout);
    } else
    Console.WriteLine(">>> No faces found");
}

static void IdentifyCelebrityFaces(string filename)
{
    // Using USWest2, not the default region
    AmazonRekognitionClient rekoClient = new AmazonRekognitionClient(Amazon.RegionEndpoint.USWest2);

    RecognizeCelebritiesRequest dfr = new RecognizeCelebritiesRequest();

    // Request needs image bytes, so read and add to request
    byte[] data = null;
    using (FileStream fs = new FileStream(filename, FileMode.Open, FileAccess.Read))
    {
        data = new byte[fs.Length];
        fs.Read(data, 0, (int)fs.Length);
    }
    img.Bytes = new MemoryStream(data);
dfr.Image = img;
var outcome = rekoClient.RecognizeCelebrities(dfr);
if (outcome.CelebrityFaces.Count > 0)
{
    // Load a bitmap to modify with face bounding box rectangles
    Pen pen = new Pen(Color.Black, 3);
Font drawFont = new Font("Arial", 12);

// Create a graphics context
using (var graphics = Graphics.FromImage(facesHighlighted))
{
    foreach (var fd in outcome.CelebrityFaces)
    {
        // Get the bounding box
        BoundingBox bb = fd.Face.BoundingBox;
        // Draw the rectangle using the bounding box values
        // They are percentages so scale them to picture
        graphics.DrawRectangle(pen, x: facesHighlighted.Width * bb.Left,
                                  y: facesHighlighted.Height * bb.Top,
                                  width: facesHighlighted.Width * bb.Width,
                                  height: facesHighlighted.Height * bb.Height);
        graphics.DrawString(fd.Name, drawFont, Brushes.White,
                                  x: facesHighlighted.Width * bb.Left,
                                  y: facesHighlighted.Height * bb.Top + facesHighlighted.Height * bb.Height);
    }
}

// Save the image with highlights as PNG
string fileout = filename.Replace(Path.GetExtension(filename), 
                               
                               "_celebrityfaces.png");
facesHighlighted.Save(fileout, System.Drawing.Imaging.ImageFormat.Png);
Console.WriteLine(">>> " + outcome.CelebrityFaces.Count + " celebrity face(s)
highlighted in file " + fileout);
else
    Console.WriteLine(">>> No celebrity faces found");
}

For more information, see Detecting and Analyzing Faces and Recognizing Celebrities. See the complete example on GitHub.

Managing Domain Name System (DNS) Resources Using Amazon Route 53

The AWS SDK for .NET supports Amazon Route 53, which is a Domain Name System (DNS) web service that provides secure and reliable routing to your infrastructure that uses Amazon Web Services (AWS) products, such as Amazon Elastic Compute Cloud (Amazon EC2), Elastic Load Balancing, or Amazon Simple Storage Service (Amazon S3). You can also use Route 53 to route users to your infrastructure outside of AWS. This topic describes how to use the AWS SDK for .NET to create an Route 53 hosted zone and add a new resource record set to that zone.

Note
This topic assumes you are already familiar with how to use Route 53 and have already installed the AWS SDK for .NET. For more information about Route 53, see the Amazon Route 53 Developer Guide. For information about how to install the AWS SDK for .NET, see Getting Started with the AWS SDK for .NET (p. 3).

The basic procedure is as follows.

To create a hosted zone and update its record sets

1. Create a hosted zone.
2. Create a change batch that contains one or more record sets, and instructions on which action to take for each set.

3. Submit a change request to the hosted zone that contains the change batch.

4. Monitor the change to verify it is complete.

The example is a simple console application that shows how to use the AWS SDK for .NET to implement this procedure for a basic record set.

**To run this example**

1. In the Visual Studio File menu, choose New, and then choose Project.

2. Choose the AWS Empty Project template and specify the project's name and location.

3. Specify the application's default credentials profile and AWS region, which are added to the project's App.config file. This example assumes the region is set to US East (N. Virginia) and the profile is set to default. For more information on profiles, see Configuring AWS Credentials (p. 11).

4. Open program.cs and replace the using declarations and the code in Main with the corresponding code from the following example. If you are using your default credentials profile and region, you can compile and run the application as-is. Otherwise, you must provide an appropriate profile and region, as discussed in the notes that follow the example.

```csharp
using System;
using System.Collections.Generic;
using System.Threading;
using Amazon;
using Amazon.Route53;
using Amazon.Route53.Model;

namespace Route53_RecordSet
{
    //Create a hosted zone and add a basic record set to it
class recordset
    {
        public static void Main(string[] args)
        {
            string domainName = "www.example.org";

            //Create an Amazon Route 53 client object
            var route53Client = new AmazonRoute53Client();

            //Create a hosted zone
            var zoneRequest = new CreateHostedZoneRequest()
            {
                Name = domainName,
                CallerReference = "my_change_request"
            };

            var zoneResponse = route53Client.CreateHostedZone(zoneRequest);

            //Create a resource record set change batch
            var recordSet = new ResourceRecordSet()
            {
                Name = domainName,
                TTL = 60,
                Type = RRType.A,
                ResourceRecords = new List<ResourceRecord>
                {
                    new ResourceRecord { Value = "192.0.2.235" } 
                }
            }
```
var change1 = new Change()
{
    ResourceRecordSet = recordSet,
    Action = ChangeAction.CREATE
};

var changeBatch = new ChangeBatch()
{
    Changes = new List<Change> { change1 }
};

//[4] Update the zone's resource record sets
var recordsetRequest = new ChangeResourceRecordSetsRequest()
{
    HostedZoneId = zoneResponse.HostedZone.Id,
    ChangeBatch = changeBatch
};

var recordsetResponse = route53Client.ChangeResourceRecordSets(recordsetRequest);

//[5] Monitor the change status
var changeRequest = new GetChangeRequest()
{
    Id = recordsetResponse.ChangeInfo.Id
};

while (ChangeStatus.PENDING ==
    route53Client.GetChange(changeRequest).ChangeInfo.Status)
{
    Console.WriteLine("Change is pending.");
    Thread.Sleep(15000);
}

Console.WriteLine("Change is complete.");
Console.ReadKey();

The numbers in the following sections are keyed to the comments in the preceding example.

[1] Create a Client Object

The object must have the following information:

**An AWS region**

When you call a client method, the underlying HTTP request is sent to this endpoint.

**A credentials profile**

The profile must grant permissions for the actions you intend to use—the Route 53 actions in this case. Attempts to call actions that lack permissions will fail. For more information, see Configuring AWS Credentials (p. 11).

The **AmazonRoute53Client** class supports a set of public methods that you use to invoke Amazon Route 53 actions. You create the client object by instantiating a new instance of the AmazonRoute53Client class. There are multiple constructors.

[2] Create a hosted zone

A hosted zone serves the same purpose as a traditional DNS zone file. It represents a collection of resource record sets that are managed together under a single domain name.
To create a hosted zone

1. Create a `CreateHostedZoneRequest` object and specify the following request parameters. There are also two optional parameters that aren't used by this example.

   **Name**
   
   (Required) The domain name you want to register, `www.example.com` for this example. This domain name is intended only for examples. It can't be registered with a domain name registrar, but you can use it to create a hosted zone for learning purposes.

   **CallerReference**
   
   (Required) An arbitrary user-defined string that serves as a request ID and can be used to retry failed requests. If you run this application multiple times, you must change the `CallerReference` value.

1. Pass the `CreateHostedZoneRequest` object to the client object's `CreateHostedZone` method. The method returns a `CreateHostedZoneResponse` object that contains information about the request, including the `HostedZone.Id` property that identifies zone.

[3] Create a resource record set change batch

A hosted zone can have multiple resource record sets. Each set specifies how a subset of the domain's traffic, such as email requests, should be routed. You can update a zone's resource record sets with a single request. The first step is to package all the updates in a `ChangeBatch` object. This example specifies only one update, adding a basic resource record set to the zone, but a `ChangeBatch` object can contain updates for multiple resource record sets.

To create a `ChangeBatch` object

1. Create a `ResourceRecordSet` object for each resource record set you want to update. The group of properties you specify depends on the type of resource record set. For a complete description of the properties used by the different resource record sets, see Values that You Specify When You Create or Edit Amazon Route 53 Resource Record Sets. The example `ResourceRecordSet` object represents a basic resource record set, and specifies the following required properties.

   **Name**
   
   The domain or subdomain name, `www.example.com` for this example.

   **TTL**
   
   The amount of time, in seconds, the DNS recursive resolvers should cache information about this resource record set, 60 seconds for this example.

   **Type**
   
   The DNS record type, `A` for this example. For a complete list, see Supported DNS Resource Record Types.

   **ResourceRecords**
   
   A list of one or more `ResourceRecord` objects, each of which contains a DNS record value that depends on the DNS record type. For an `A` record type, the record value is an IPv4 address, which for this example is set to a standard example address, `192.0.2.235`.

2. Create a `Change` object for each resource record set, and set the following properties.

   **ResourceRecordSet**
   
   The `ResourceRecordSet` object you created in the previous step.

   **Action**
   
   The action to be taken for this resource record set: CREATE, DELETE, or UPSERT. For more information about these actions, see Elements. This example creates a new resource record set in the hosted zone, so Action is set to CREATE.
3. Create a `ChangeBatch` object and set its `Changes` property to a list of the `Change` objects that you created in the previous step.

**[4] Update the zone's resource record sets**

To update the resource record sets, pass the `ChangeBatch` object to the hosted zone, as follows.

**To update a hosted zone's resource record sets**

1. Create a `ChangeResourceRecordSetsRequest` object with the following property settings.
   - `HostedZoneId`
     - The hosted zone's ID, which the example sets to the ID that was returned in the `CreateHostedZoneResponse` object. To get the ID of an existing hosted zone, call `ListHostedZones`.  
   - `ChangeBatch`
     - A `ChangeBatch` object that contains the updates.

2. Pass the `ChangeResourceRecordSetsRequest` object to the `ChangeResourceRecordSets` method of the client object. It returns a `ChangeResourceRecordSetsResponse` object, which contains a request ID you can use to monitor the request's progress.

**[5] Monitor the update status**

Resource record set updates typically take a minute or so to propagate through the system. You can monitor the update's progress and verify that it is complete as follows.

**To monitor update status**

1. Create a `GetChangeRequest` object and set its `Id` property to the request ID that was returned by `ChangeResourceRecordSets`.  
2. Use a wait loop to periodically call the `GetChange` method of the client object. `GetChange` returns `PENDING` while the update is in progress and `INSYNC` after the update is complete. You can use the same `GetChangeRequest` object for all of the method calls.

---

**Using Amazon Simple Storage Service Internet Storage**

The AWS SDK for .NET supports Amazon Simple Storage Service (Amazon S3), which is storage for the Internet. It is designed to make web-scale computing easier for developers. For more information, see [Amazon S3](https://aws.amazon.com/s3/).

The following links provide examples of programming Amazon S3 with the AWS SDK for .NET:

- Using the AWS SDK for .NET for Amazon S3 Programming
- Making Requests Using AWS Account or IAM User Credentials
- Making Requests Using IAM User Temporary Credentials
- Making Requests Using Federated User Temporary Credentials
- Managing ACLs
- Creating a Bucket
- Upload an Object
- Multipart Upload with the High-Level API
- Multipart Upload with the Low-Level API
- Listing Objects
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 and receive notifications from the cloud. When using Amazon SNS, you (as the owner) create a topic and control access to it by defining policies that determine which publishers and subscribers can communicate with the topic. A publisher sends messages to topics that they have created or to topics they have permission to publish to. Instead of including a specific destination address in each message, a publisher sends a message to the topic. Amazon SNS matches the topic to a list of subscribers who have subscribed to that topic, and delivers the message to each of those subscribers. Each topic has a unique name that identifies the Amazon SNS endpoint for publishers to post messages and subscribers to register for notifications. Subscribers receive all messages published to the topics to which they subscribe, and all subscribers to a topic receive the same messages.

For more information, see Amazon SNS.

The following examples demonstrate how to use the AWS SDK for .NET to create and use Amazon SNS topics.

The sample code is written in C#, but you can use the AWS SDK for .NET with any compatible language. The AWS SDK for .NET installs a set of C# project templates. So the simplest way to start this project is to open Visual Studio, and then choose File, New Project, AWS Sample Projects, App Services, AWS SNS Sample.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

Topics

- Adding a Permission Using the AWS SDK for .NET (p. 172)
- Confirming a Subscription Using the AWS SDK for .NET (p. 172)
- Deleting a Topic Using the AWS SDK for .NET (p. 173)
Adding a Permission Using the AWS SDK for .NET

In this example, you add a statement to a topic's access control policy, granting access for the specified AWS accounts to the specified actions.

This example uses the default client constructor which constructs an AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

**Add a Permission**

Call the AddPermission method with an AddPermissionRequest with the action you want to allow for the specified principal, the AWS account IDs of the users, the label for the new policy statement ad the Amazon Resource Name (ARN) of the topic whose access control policy you wish to modify.

```csharp
var snsClient = new AmazonSimpleNotificationServiceClient();
var request = new AddPermissionRequest
{
    TopicArn = "arn:aws:sns:us-east-1:80398EXAMPLE:CodingTestResults",
    ActionName = new List<string>() { "Subscribe" },
    AWSAccountId = new List<string>() { "80398EXAMPLE" },
    Label = "SubscribePolicy"
};
snsClient.AddPermission(request);
```

Confirming a Subscription Using the AWS SDK for .NET

In this example, you verify an endpoint owner's intent to receive messages by validating the token sent to the endpoint by an earlier Subscribe action. If the token is valid, the action creates a new subscription and returns its Amazon Resource Name (ARN).

This example uses the default client constructor which constructs an AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

**Confirm a Subscription**

Call the ConfirmSubscription method with a ConfirmSubscriptionRequest with the topic ARN and the token from the confirmation message.

```csharp
var snsClient = new AmazonSimpleNotificationServiceClient();
```
Deleting a Topic Using the AWS SDK for .NET

In this example, you delete a topic and all its subscriptions. Deleting a topic might prevent some messages previously sent to the topic from being delivered to subscribers. Deleting a topic that does not exist does not result in an error.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Delete a Topic

Call the `DeleteTopic` method with an `DeleteTopicRequest` with the Amazon Resource Name (ARN) of the topic to be deleted.

```csharp
var snsClient = new AmazonSimpleNotificationServiceClient();
var request = new DeleteTopicRequest
{
    TopicArn = "arn:aws:sns:us-east-1:80398EXAMPLE:CodingTestResults"
};
snsClient.DeleteTopic(request);
```

Listing Subscriptions Using the AWS SDK for .NET

In this example, you get a list of the subscriptions. Each call returns a limited list of subscriptions, up to 100. If there are more subscriptions, a `NextToken` is also returned. Use the `NextToken` parameter in a new `ListSubscriptions` call to get further results.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

List Subscriptions

Call the `ListSubscriptions` method with an `ListSubscriptionsRequest`. The returned `ListSubscriptionsResponse` contains the list of subscriptions for the client.

```csharp
var snsClient = new AmazonSimpleNotificationServiceClient();
var request = new ListSubscriptionsRequest();
var response = new ListSubscriptionsResponse();
do
{
    response = snsClient.ListSubscriptions(request);
    if (response.NextToken !="")
    {
        request.NextToken = response.NextToken;
    }
    else
    {
        break;
    }
}
```
foreach (var sub in response.Subscriptions)
{
    Console.WriteLine("Subscription: {0}", sub.SubscriptionArn);
}
request.NextToken = response.NextToken;
} while (!string.IsNullOrEmpty(response.NextToken));
Console.ReadLine();

Listing Subscriptions by Topic Using the AWS SDK for .NET

In this example, you get a list of the subscriptions to a specific topic. Each call returns a limited list of subscriptions, up to 100. If there are more subscriptions, a NextToken is also returned. Use the NextToken parameter in a new ListSubscriptionsByTopic call to get further results.

This example uses the default client constructor which constructs a AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

List Subscriptions by Topic

Call the ListSubscriptionsByTopic method with an ListSubscriptionsByTopicRequest. The returned ListSubscriptionsByTopicResponse contains the list of subscriptions by topic for the client.

```
var snsClient = new AmazonSimpleNotificationServiceClient();
var request = new ListSubscriptionsByTopicRequest();
var response = new ListSubscriptionsByTopicResponse();
request.TopicArn = "arn:aws:sns:us-east-1:80398EXAMPLE:CodingTestResults";

do
{
    response = snsClient.ListSubscriptionsByTopic(request);
    foreach (var sub in response.Subscriptions)
    {
        Console.WriteLine("Subscription: {0}", sub.SubscriptionArn);
        var subAttrs = snsClient.GetSubscriptionAttributes(new GetSubscriptionAttributesRequest
            {
                SubscriptionArn = sub.SubscriptionArn
            }).Attributes;
        if (subAttrs.Count > 0)
        {
            foreach (var subAttr in subAttrs)
            {
                Console.WriteLine(" -{0} : {1}", subAttr.Key, subAttr.Value);
            }
        }
        Console.WriteLine();
    }
    request.NextToken = response.NextToken;
}
```
while (!string.IsNullOrEmpty(response.NextToken));

Console.ReadLine();

Listing Topics Using the AWS SDK for .NET

In this example, you get a list of your topics. Each call returns a limited list of topics, up to 100. If there are more topics, a NextToken is also returned. Use the NextToken parameter in a new ListTopics call to get further results.

This example uses the default client constructor which constructs a AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

List Subscriptions by Topic

Call the ListTopics method with an ListTopicsRequest. The returned ListTopicsResponse contains the list of topics for the client.

Sending an SMS Message Using the AWS SDK for .NET

In this example, you send an SMS message to a phone number. By specifying the PhoneNumber property rather than a TargetArn for an endpoint or a TopicArn for a topic you send an SMS message.

This example uses the default client constructor which constructs a AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).
Send an SMS Message

Call the `Publish` method with an `PublishRequest` with the message to be sent, the phone number to send it to and a default message attributes object. The contents of the `Message` property will be sent to the specified phone number.

```csharp
var snsClient = new AmazonSimpleNotificationServiceClient();

String message = "My SMS message";
String phoneNumber = "+1xxxxxxx";
byte[] data = UTF8Encoding.UTF8.GetBytes("mySenderID");
Dictionary<System.String, MessageAttribute> smsAttributes = new Dictionary<string, MessageAttribute>();
snsClient.Publish(new PublishRequest()
{
    Message = message,
    PhoneNumber = phoneNumber,
    MessageAttributes = smsAttributes
});
Console.ReadLine();
```

Creating a Subscription and Publishing a Message Using the AWS SDK for .NET

In this example, you subscribe to an endpoint by creating a topic, setting topic attributes, subscribing to the topic, confirming the subscription and then publishing a message.

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Create a Subscription and Publish a Message

Call the `CreateTopic` method with an `CreateTopicRequest`. The returned `CreateTopicResponse` contains a `TopicArn` property. Call the `SetTopicAttributes` method using the `TopicArn` then call the `Subscribe` method to subscribe that topic to a specified end point. Call the `ListSubscriptionsByTopic` method with an `ListSubscriptionsByTopicRequest` method with an `PublishRequest` to publish the message.

```csharp
var snsClient = new AmazonSimpleNotificationServiceClient();

var topicRequest = new CreateTopicRequest
{
    Name = "CodingTestResults"
};
var topicResponse = snsClient.CreateTopic(topicRequest);
var topicAttrRequest = new SetTopicAttributesRequest
{
    TopicArn = topicResponse.TopicArn,
    AttributeName = "DisplayName",
    AttributeValue = "Coding Test Results"
};
snsClient.SetTopicAttributes(topicAttrRequest);
snsClient.Subscribe(new SubscribeRequest
{
    Protocol = "sms",
    Endpoint = "+1xxxxxxx",
    Attributes = new Dictionary<String, String>
    {
        { "AWS_Cloud_Hosted_SMS_Sender", "mySenderID" }
    }
});
```

This example uses the default client constructor which constructs a `AmazonSimpleNotificationServiceClient` using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).
Deleting a Subscription Using the AWS SDK for .NET

In this example, you delete a subscription by unsubscribing. If the subscription requires authentication for deletion, only the owner of the subscription or the topic's owner can unsubscribe, and an AWS signature is required. If unsubscribing does not require authentication and the requester is not the owner, a subscription final cancellation message is delivered to the endpoint, so that the endpoint owner can easily resubscribe to the topic if the request to unsubscribe was unintended.

This example uses the default client constructor which constructs a AmazonSimpleNotificationServiceClient using the credentials searched for in the order described in Credential and Profile Resolution (p. 17).

Delete a Subscription

To delete a subscription call the Unsubscribe method with an UnsubscribeRequest with the Amazon Resource Name (ARN) of the subscription to be deleted.

```csharp
var snsClient = new AmazonSimpleNotificationServiceClient();
var request = new UnsubscribeRequest();
request.SubscriptionArn = "arn:aws:sns:us-east-1:80398EXAMPLE:CodingTestResults:" +
```
Messaging Using Amazon SQS

The AWS SDK for .NET supports Amazon SQS, which is a message queuing service that handles messages or workflows between components in a system. For more information, see Amazon SQS.

The following examples demonstrate how to use the AWS SDK for .NET to create and use Amazon SQS queues.

The sample code is written in C#, but you can use the AWS SDK for .NET with any compatible language. The AWS SDK for .NET installs a set of C# project templates. So the simplest way to start this project is to open Visual Studio, and then choose File, New Project, AWS Sample Projects, App Services, AWS SQS Sample.

Prerequisite Tasks

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).

For related API reference information, see Amazon.SQS, Amazon.SQS.Model, and Amazon.SQS.Util in the AWS SDK for .NET Reference.

Topics

- Creating an Amazon SQS Client (p. 178)
- Creating an Amazon SQS Queue (p. 179)
- Constructing Amazon SQS Queue URLs (p. 180)
- Sending an Amazon SQS Message (p. 180)
- Sending an Amazon SQS Message Batch (p. 181)
- Receiving a Message from an Amazon SQS Queue (p. 182)
- Deleting a Message from an Amazon SQS Queue (p. 182)
- Enabling Long Polling in Amazon SQS (p. 183)
- Using Amazon SQS Queues (p. 184)
- Using Amazon SQS Dead Letter Queues (p. 185)

Creating an Amazon SQS Client

You need an Amazon SQS client in order to create and use an Amazon SQS queue. Before configuring your client, you should create an App.config file to specify your AWS credentials.

You specify your credentials by referencing the appropriate profile in the appSettings section of the file.

The following example specifies a profile named my_profile. For more information about credentials and profiles, see Configuring Your AWS SDK for .NET Application (p. 8).

```xml
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
</configuration>
```
Creating an Amazon SQS Queue

Creating an Amazon SQS queue is an administrative task that you can do by using the SQS Management Console. However, you can also use the AWS SDK for .NET to programatically create an Amazon SQS queue.

To create an Amazon SQS queue

For information about how queues work in Amazon SQS, see How SQS Queues Work.

For information about your queue URL, see Constructing Amazon SQS Queue URLs (p. 180).

1. Create and initialize a CreateQueueRequest instance. Provide the name of your queue and specify a visibility timeout for your queue messages, as follows.

```csharp
var createQueueRequest = new CreateQueueRequest();
createQueueRequest.QueueName = "MySQSQueue";
var attrs = new Dictionary<string, string>();
attrs.Add(QueueAttributeName.VisibilityTimeout, "10");
createQueueRequest.Attributes = attrs;
```

Your queue name must be composed of only alphanumeric characters, hyphens, and underscores. Any message in the queue remains in the queue unless the specified visibility timeout is exceeded. The default visibility timeout for a queue is 30 seconds. For more information about visibility timeouts, see Visibility Timeout. For more information about different queue attributes you can set, see SetQueueAttributes.

2. After you create the request, pass it as a parameter to the CreateQueue method. The method returns a CreateQueueResponse object, as follows.
Constructing Amazon SQS Queue URLs

You need a queue URL to send, receive, and delete queue messages. You can get your queue URL using the `GetQueueUrl` method.

```csharp
var createQueueResponse = sqsClient.CreateQueue(createQueueRequest);

var client = new AmazonSQSClient();
var request = new GetQueueUrlRequest
{
    QueueName = "MyTestQueue",
    QueueOwnerAWSAccountId = "80398EXAMPLE"
};
var response = client.GetQueueUrl(request);
Console.WriteLine("Queue URL: " + response.QueueUrl);

To find your AWS account number, go to Security Credentials. Your account number is located under Account Number at the top-right of the page.

For information about sending a message to a queue, see Sending an Amazon SQS Message (p. 180).

For information about receiving messages from a queue, see Receiving a Message from an Amazon SQS Queue (p. 182).

For information about deleting messages from a queue, see Deleting a Message from an Amazon SQS Queue (p. 182).

Sending an Amazon SQS Message

You can use the AWS SDK for .NET to send a message to an Amazon SQS queue.

Important
Due to the distributed nature of the queue, Amazon SQS can't guarantee you will receive messages in the precise order they are sent. If you need to preserve the message order, place sequencing information in each message so you can reorder the messages upon receipt.

To end a message to an Amazon SQS queue

For information about deleting messages from your queue, see Deleting a Message from an Amazon SQS Queue (p. 182).

For information about receiving messages from your queue, see Receiving a Message from an Amazon SQS Queue (p. 182).

1. Create and initialize a `SendMessageRequest` instance. Specify the queue name and the message you want to send, as follows.

   ```csharp
   sendMessageRequest.QueueUrl = myQueueURL; sendMessageRequest.MessageBody = "{YOUR_QUEUE_MESSAGE}";
   ```

For more information about your queue URL, see Constructing Amazon SQS Queue URLs (p. 180).

Each queue message must be composed of Unicode characters only, and can be up to 64 KB in size. For more information about queue messages, see SendMessage in the Amazon SQS API Reference.
2. After you create the request, pass it as a parameter to the `SendMessage` method. The method returns a `SendMessageResponse` object, as follows.

```csharp
var sendMessageResponse = sqsClient.SendMessage(sendMessageRequest);
```

The sent message will stay in your queue until the visibility timeout is exceeded, or until it is deleted from the queue. For more information about visibility timeouts, go to Visibility Timeout.

## Sending an Amazon SQS Message Batch

You can use the AWS SDK for .NET to send batch messages to an Amazon SQS queue. The `SendMessageBatch` method delivers up to 10 messages to the specified queue. This is a batch version of `SendMessage`.

For a FIFO queue, multiple messages within a single batch are enqueued in the order they are sent.

For more information about sending batch messages, see `SendMessageBatch` in the Amazon SQS API Reference.

### To send batch messages to an Amazon SQS queue

1. Create an `AmazonSQSClient` instance and initialize a `SendMessageBatchRequest` object. Specify the queue name and the message you want to send, as follows.

```csharp
AmazonSQSClient client = new AmazonSQSClient();
var sendMessageBatchRequest = new SendMessageBatchRequest
{
    Entries = new List<SendMessageBatchRequestEntry>
    {
        new SendMessageBatchRequestEntry("message1", "FirstMessageContent"),
        new SendMessageBatchRequestEntry("message2", "SecondMessageContent"),
        new SendMessageBatchRequestEntry("message3", "ThirdMessageContent")
    },
    QueueUrl = "SQS_QUEUE_URL"
};
```

For more information about your queue URL, see Constructing Amazon SQS Queue URLs (p. 180).

Each queue message must be composed of Unicode characters only, and can be up to 64 KB in size. For more information about queue messages, see `SendMessage` in the Amazon SQS API Reference.

2. After you create the request, pass it as a parameter to the `SendMessageBatch` method. The method returns a `SendMessageBatchResponse` object, which contains the unique ID of each message and the message content for each successfully sent message. It also returns the message ID, message content, and a sender's fault flag if the message failed to send.

```csharp
SendMessageBatchResponse response = client.SendMessageBatch(sendMessageBatchRequest);
Console.WriteLine("Messages successfully sent:");
foreach (var success in response.Successful)
{
    Console.WriteLine("    Message id : {0}", success.MessageId);
    Console.WriteLine("    Message content MD5 : {0}", success.MD5OfMessageBody);
}
Console.WriteLine("Messages failed to send:");
foreach (var failed in response.Failed)
{
    Console.WriteLine("    Message id : {0}", failed.Id);
    Console.WriteLine("    Message content : {0}", failed.Message);
}
```
Receiving a Message from an Amazon SQS Queue

You can use the AWS SDK for .NET to receive messages from an Amazon SQS queue.

**To receive a message from an Amazon SQS queue**

For information about sending a message to your queue, see Sending an Amazon SQS Message (p. 180).

For more information about deleting a message from the queue, see Deleting a Message from an Amazon SQS Queue (p. 182).

1. Create and initialize a `ReceiveMessageRequest` instance. Specify the queue URL to receive a message from, as follows.

   ```csharp
   var receiveMessageRequest = new ReceiveMessageRequest();
   receiveMessageRequest.QueueUrl = myQueueURL;
   ```

   For more information about your queue URL, see Your Amazon SQS Queue URL (p. 180).

2. Pass the request object as a parameter to the `ReceiveMessage` method, as follows.

   ```csharp
   var receiveMessageResponse = sqsClient.ReceiveMessage(receiveMessageRequest);
   ```

   The method returns a `ReceiveMessageResponse` instance, containing the list of messages the queue contains.

3. The `ReceiveMessageResponse.ReceiveMessageResult` property contains a `ReceiveMessageResponse` object, which contains a list of the messages that were received. Iterate through this list and call the `ProcessMessage` method to process each message.

   ```csharp
   foreach (var message in result.Messages)
   {
     ProcessMessage(message);  // Go to a method to process messages.
   }
   ```

   The `ProcessMessage` method can use the `ReceiptHandle` property to obtain a receipt handle for the message. You can use this receipt handle to change the message visibility timeout or to delete the message from the queue. For more information about how to change the visibility timeout for a message, see ChangeMessageVisibility.

Deleting a Message from an Amazon SQS Queue

You can use the AWS SDK for .NET to delete messages from an Amazon SQS queue.

**To delete a message from an Amazon SQS queue**

For information about sending a message to a queue, see Sending an Amazon SQS Message (p. 180).

For information about receiving messages from a queue, see Receiving a Message from an Amazon SQS Queue (p. 182).
1. Create and initialize a `DeleteMessageRequest` object. Specify the Amazon SQS queue to delete a message from and the receipt handle of the message to delete, as follows.

   ```csharp
   var deleteMessageRequest = new DeleteMessageRequest();
   deleteMessageRequest.QueueUrl = queueUrl;
   deleteMessageRequest.ReceiptHandle = receiptHandle;
   ```

2. Pass the request object as a parameter to the `DeleteMessage` method. The method returns a `DeleteMessageResponse` object, as follows.

   ```csharp
   var response = sqsClient.DeleteMessage(deleteMessageRequest);
   ```

Calling `DeleteMessage` unconditionally removes the message from the queue, regardless of the visibility timeout setting. For more information about visibility timeouts, see Visibility Timeout.

### Enabling Long Polling in Amazon SQS

Long polling reduces the number of empty responses by allowing Amazon SQS to wait a specified time for a message to become available in the queue before sending a response. Also, long polling eliminates false empty responses by querying all the servers instead of a sampling of servers. To enable long polling, you must specify a non-zero wait time for received messages. You can do this by setting the `ReceiveMessageWaitTimeSeconds` parameter of a queue or by setting the `WaitTimeSeconds` parameter on a message when it's received. This .NET example shows you how to enable long polling in Amazon SQS for a newly created or existing queue, or upon receipt of a message.

These examples use the following methods of the `AmazonSQSClient` class to enable long polling:

- `CreateQueue`
- `SetQueueAttributes`
- `ReceiveMessage`

For more information about long polling, see Amazon SQS Long Polling in the Amazon SQS Developer Guide.

### Enable Long Polling When Creating a Queue

Create an `AmazonSQSClient` service object. Create a `CreateQueueRequest` object containing the properties needed to create a queue, including a non-zero value for the `ReceiveMessageWaitTimeSeconds` property.

Call the `CreateQueue` method. Long polling is then enabled for the queue.

```csharp
AmazonSQSClient client = new AmazonSQSClient();
var request = new CreateQueueRequest{
    QueueName = "SQS_QUEUE_NAME",
    Attributes = new Dictionary<string, string>
    {
        { "ReceiveMessageWaitTimeSeconds", "20" }
    }
};
var response = client.CreateQueue(request);
Console.WriteLine("Created a queue with URL : {0}", response.QueueUrl);
```
Enable Long Polling on an Existing Queue

Create an AmazonSQSClient service object. Create a SetQueueAttributesRequest object containing the properties needed to set the attributes of the queue, including a non-zero value for the ReceiveMessageWaitTimeSeconds property and the URL of the queue. Call the SetQueueAttributes method. Long polling is then enabled for the queue.

```
AmazonSQSClient client = new AmazonSQSClient();
var request = new SetQueueAttributesRequest
{
    Attributes = new Dictionary<string, string>
    {
        { "ReceiveMessageWaitTimeSeconds", "20"}
    },
    QueueUrl = "SQS_QUEUE_URL"
};
var response = client.SetQueueAttributes(request);
```

Receive a Message

Create an AmazonSQSClient service object. Create a ReceiveMessageRequest object containing the properties needed to receive a message, including a non-zero value for the WaitTimeSeconds parameter and the URL of the queue. Call the ReceiveMessage method.

```
public void OnMessageReceipt()
{
    AmazonSQSClient client = new AmazonSQSClient();

    var request = new ReceiveMessageRequest
    {
        AttributeNames = { "SentTimestamp" },
        MaxNumberOfMessages = 1,
        MessageAttributeNames = { "All" },
        QueueUrl = "SQS_QUEUE_URL",
        WaitTimeSeconds = 20
    };

    var response = client.ReceiveMessage(request);
}
```

Using Amazon SQS Queues

Amazon SQS offers standard as the default queue type. A standard queue enables you to have a nearly-unlimited number of transactions per second. Standard queues support at-least-once message delivery. However, occasionally more than one copy of a message might be delivered out of order. Standard queues provide best-effort ordering, which ensures that messages are generally delivered in the same order as they’re sent.

You can use standard message queues in many scenarios, as long as your application can process messages that arrive more than once and out of order.

This code example demonstrates how to use queues by using these methods of the :code:`AmazonSQSClient` class:

- **ListQueues**: Gets a list of your message queues
• GetQueueUrl: Obtains the URL for a particular queue
• DeleteQueue: Deletes a queue

For more information about Amazon SQS messages, see How Amazon SQS Queues Work in the Amazon SQS Developer Guide.

### List Your Queues

Create a `ListQueuesRequest` object containing the properties needed to list your queues, which by default is an empty object. Call the `ListQueues` method with the `ListQueuesRequest` as a parameter to retrieve the list of queues. The `ListQueuesResponse` returned by the call contains the URLs of all queues.

```csharp
AmazonSQSClient client = new AmazonSQSClient();
ListQueuesResponse response = client.ListQueues(new ListQueuesRequest());
foreach (var queueUrl in response.QueueUrls)
{
    Console.WriteLine(queueUrl);
}
```

### Get the URL for a Queue

Create a `GetQueueUrlRequest` object containing the properties needed to identify your queue, which must include the name of the queue whose URL you want. Call the `GetQueueUrl` method using the `GetQueueUrlRequest` object as a parameter. The call returns a `GetQueueUrlResponse` object containing the URL of the specified queue.

```csharp
AmazonSQSClient client = new AmazonSQSClient();
var request = new GetQueueUrlRequest
{
    QueueName = "SQS_QUEUE_NAME"
};
GetQueueUrlResponse response = client.GetQueueUrl(request);
Console.WriteLine("The SQS queue's URL is {0}", response.QueueUrl);
```

### Delete a Queue

Create a `DeleteQueueRequest` object containing the URL of the queue you want to delete. Call the `DeleteQueue` method with the `DeleteQueueRequest` object as the parameter.

```csharp
AmazonSQSClient client = new AmazonSQSClient();
var request = new DeleteQueueRequest
{
    QueueUrl = "SQS_QUEUE_URL"
};
client.DeleteQueue(request);
```

### Using Amazon SQS Dead Letter Queues

This example shows you how to use a queue to receive and hold messages from other queues that the queues can't process.
A dead letter queue is one that other (source) queues can target for messages that can't be processed successfully. You can set aside and isolate these messages in the dead letter queue to determine why their processing did not succeed. You must individually configure each source queue that sends messages to a dead letter queue. Multiple queues can target a single dead letter queue.

In this example, an AmazonSQSClient object uses the SetQueueAttributesRequest method to configure a source queue to use a dead letter queue.

For more information about Amazon SQS dead letter queues, see Using Amazon SQS Dead Letter Queues in the Amazon SQS Developer Guide.

**Configure a Source Queue**

This code example assumes you have created a queue to act as a dead letter queue. See Creating an Amazon SQS Queue (p. 179) for information about creating a queue. After creating the dead letter queue, you must configure the other queues to route unprocessed messages to the dead letter queue. To do this, specify a redrive policy that identifies the queue to use as a dead letter queue and the maximum number of receives by individual messages before they are routed to the dead letter queue.

Create an AmazonSQSClient object to set the queue attributes. Create a SetQueueAttributesRequest object containing the properties needed to update queue attributes, including the RedrivePolicy property that specifies both the Amazon Resource Name (ARN) of the dead letter queue, and the value of maxReceiveCount. Also specify the URL source queue you want to configure. Call the SetQueueAttributes method.

```csharp
AmazonSQSClient client = new AmazonSQSClient();

var setQueueAttributeRequest = new SetQueueAttributesRequest
{
    Attributes = new Dictionary<string, string>
    {
        {"RedrivePolicy", @"{ "deadLetterTargetArn" : "DEAD_LETTER_QUEUE_ARN", "maxReceiveCount" : "10" }" }
    },
    QueueUrl = "SOURCE_QUEUE_URL"
};

client.SetQueueAttributes(setQueueAttributeRequest)
```

**Monitoring Your AWS Resources Using Amazon CloudWatch**

Amazon CloudWatch is a web service that monitors your AWS resources and the applications you run on AWS in real time. You can use CloudWatch to collect and track metrics, which are variables you can measure for your resources and applications. CloudWatch alarms send notifications or automatically make changes to the resources you're monitoring based on rules that you define.

The code for these examples is written in C#, but you can use the AWS SDK for .NET with any compatible language. When you install the AWS Toolkit for Visual Studio, a set of C# project templates are installed. The simplest way to start this project is to open Visual Studio, and then choose File, New Project, AWS Sample Projects, Deployment and Management, AWS CloudWatch Example.

**Prerequisite Tasks**

Before you begin, be sure that you have created an AWS account and set up your AWS credentials. For more information, see Getting Started with the AWS SDK for .NET (p. 3).
Describing, Creating, and Deleting Alarms in Amazon CloudWatch

This .NET example show you how to:

- Describe a CloudWatch alarm
- Create a CloudWatch alarm based on a metric
- Delete a CloudWatch alarm

The Scenario

An alarm watches a single metric over a time period you specify. It performs one or more actions based on the value of the metric, relative to a given threshold over a number of time periods. The following examples show how to describe, create, and delete alarms in CloudWatch using these methods of the AmazonCloudWatchClient class:

- DescribeAlarms
- PutMetricAlarm
- DeleteAlarms

For more information about CloudWatch alarms, see Creating Amazon CloudWatch Alarms in the Amazon CloudWatch User Guide.

Prerequisite Tasks

To set up and run this example, you must first:

- Get set up to use Amazon CloudWatch.
- Set up and configure the AWS SDK for .NET.

Describing an Alarm

Create an AmazonCloudWatchClient instance and a DescribeAlarmsRequest object, limiting the alarms that are returned to those with a state of INSUFFICIENT_DATA. Then call the DescribeAlarms method of the AmazonCloudWatchClient object.

```csharp
using (var cloudWatch = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    var request = new DescribeAlarmsRequest();
    request.StateValue = "INSUFFICIENT_DATA";
    request.AlarmNames = new List<string> { "Alarm1", "Alarm2" };
    do
```
Creating an Alarm Based on a Metric

Create an `AmazonCloudWatchClient` instance and a `PutMetricAlarmRequest` object for the parameters needed to create an alarm that is based on a metric, in this case, the CPU utilization of an Amazon EC2 instance.

The remaining parameters are set to trigger the alarm when the metric exceeds a threshold of 70 percent.

Then call the `PutMetricAlarm` method of the `AmazonCloudWatchClient` object.

```csharp
var client = new AmazonCloudWatchClient(RegionEndpoint.USWest2);
client.PutMetricAlarm(
    new PutMetricAlarmRequest
    {
        AlarmName = "Web_Server_CPU_Utilization",
        ComparisonOperator = ComparisonOperator.GreaterThanThreshold,
        EvaluationPeriods = 1,
        MetricName = "CPUUtilization",
        Namespace = "AWS/EC2",
        Period = 60,
        Statistic = Statistic.Average,
        Threshold = 70.0,
        ActionsEnabled = true,
        AlarmActions = new List<string> { "arn:aws:swf:us-west-2:" + "customerAccount" + ":action/actions/AWS_EC2.InstanceId.Reboot/1.0" },
        AlarmDescription = "Alarm when server CPU exceeds 70%",
        Dimensions = new List<Dimension>
        {
            new Dimension { Name = "InstanceId", Value = "INSTANCE_ID" }
        },
        Unit = StandardUnit.Seconds
    });
```

Deleting an Alarm

Create an `AmazonCloudWatchClient` instance and a `DeleteAlarmsRequest` object to hold the names of the alarms you want to delete. Then call the `DeleteAlarms` method of the `AmazonCloudWatchClient` object.

```csharp
using (var cloudWatch = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    var response = cloudWatch.DeleteAlarms(
        new DeleteAlarmsRequest
        {
            AlarmNames = new List<string> { "Alarm1", "Alarm2" }
        });
}
```
Using Alarms in Amazon CloudWatch

This .NET example shows you how to change the state of your Amazon EC2 instances automatically based on a CloudWatch alarm.

The Scenario

Using alarm actions, you can create alarms that automatically stop, terminate, reboot, or recover your Amazon EC2 instances. You can use the stop or terminate actions when you no longer need an instance to be running. You can use the reboot and recover actions to automatically reboot those instances.

In this example, .NET is used to define an alarm action in CloudWatch that triggers the reboot of an Amazon EC2 instance. The methods use the AWS SDK for .NET to manage Amazon EC2 instances using these methods of the AmazonCloudWatchClient class:

- EnableAlarmActions
- DisableAlarmActions

For more information about CloudWatch alarm actions, see Create Alarms to Stop, Terminate, Reboot, or Recover an Instance in the Amazon CloudWatch User Guide.

Prerequisite Tasks

To set up and run this example, you must first:

- Get set up to use Amazon CloudWatch.
- Set up and configure the AWS SDK for .NET.

Create and Enable Actions on an Alarm

1. Create an AmazonCloudWatchClient instance and a PutMetricAlarmRequest object to hold the parameters for creating an alarm, specifying ActionsEnabled as true and an array of ARNs for the actions the alarm will trigger. Call the PutMetricAlarm method of the AmazonCloudWatchClient object, which creates the alarm if it doesn't exist or updates it if the alarm does exist.

```csharp
using (var client = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    client.PutMetricAlarm(new PutMetricAlarmRequest
    {
        AlarmName = "Web_Server_CPU_Utilization",
        ComparisonOperator = ComparisonOperator.GreaterThanThreshold,
        EvaluationPeriods = 1,
        MetricName = "CPUUtilization",
        Namespace = "AWS/EC2",
        Period = 60,
        Statistic = Statistic.Average,
        Threshold = 70.0,
        ActionsEnabled = true,
        AlarmActions = new List<string> { "arn:aws:swf:us-west-2:" + "customerAccount" + ":action/actions/AWS_EC2.InstanceId.Reboot/1.0" },
        AlarmDescription = "Alarm when server CPU exceeds 70%",
        Dimensions = new List<Dimension>
        {
            new Dimension { Name = "InstanceId", Value = "instanceId" }
        },
        Unit = StandardUnit.Seconds
```
2. When `PutMetricAlarm` completes successfully, create an `EnableAlarmActionsRequest` object containing the name of the CloudWatch alarm. Call the `EnableAlarmActions` method to enable the alarm action.

```csharp
client.EnableAlarmActions(new EnableAlarmActionsRequest
{
   AlarmNames = new List<string> { "Web_Server_CPU_Utilization" }
});
```

3. Create a `MetricDatum` object containing the CPUUtilization custom metric. Create a `PutMetricDataRequest` object containing the `MetricData` parameter needed to submit a data point for the CPUUtilization metric. Call the `PutMetricData` method.

```csharp
MetricDatum metricDatum = new MetricDatum
{ MetricName = "CPUUtilization" };
PutMetricDataRequest putMetricDatarequest = new PutMetricDataRequest
{
   MetricData = new List<MetricDatum> { metricDatum }
};
client.PutMetricData(putMetricDatarequest);
```

### Disable Actions on an Alarm

Create an `AmazonCloudWatchClient` instance and a `DisableAlarmActionsRequest` object containing the name of the CloudWatch alarm. Call the `DisableAlarmActions` method to disable the actions for this alarm.

```csharp
using (var client = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
   client.DisableAlarmActions(new DisableAlarmActionsRequest
   {
      AlarmNames = new List<string> { "Web_Server_CPU_Utilization" }
   });
}
```

### Getting Metrics from Amazon CloudWatch

This example shows you how to:

- Retrieve a list of CloudWatch metrics
- Publish CloudWatch custom metrics

### The Scenario

Metrics are data about the performance of your systems. You can enable detailed monitoring of some resources such as your Amazon EC2 instances or your own application metrics. In this example, you use .NET to retrieve a list of published CloudWatch metrics and publish data points to CloudWatch metrics using these methods of the `AmazonCloudWatchClient` class:

- `ListMetrics`
- `PutMetricData`
For more information about CloudWatch metrics, see Using Amazon CloudWatch Metrics in the Amazon CloudWatch User Guide.

Prerequisite Tasks

To set up and run this example, you must first:

- Get set up to use Amazon CloudWatch.
- Set up and configure the AWS SDK for .NET.

List Metrics

Create a ListMetricsRequest object containing the parameters needed to list metrics within the AWS/Logs namespace. Call the ListMetrics method from a AmazonCloudWatchClient instance to list the IncomingLogEvents metric.

```csharp
var logGroupName = "LogGroupName";
DimensionFilter dimensionFilter = new DimensionFilter()
{
    Name = logGroupName
};
var dimensionFilterList = new List<DimensionFilter>();
dimensionFilterList.Add(dimensionFilter);
var dimension = new Dimension
{
    Name = "UniquePages",
    Value = "URLs"
};
using (var cw = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    var listMetricsResponse = cw.ListMetrics(new ListMetricsRequest
    {
        Dimensions = dimensionFilterList,
        MetricName = "IncomingLogEvents",
        Namespace = "AWS/Logs"
    });
    Console.WriteLine(listMetricsResponse.Metrics);
}
```

Submit Custom Metrics

Create a PutMetricDataRequest object containing the parameters needed to submit a data point for the PAGES_VISITED custom metric. Call the PutMetricData method from the AmazonCloudWatchClient instance.

```csharp
using (var cw = new AmazonCloudWatchClient(RegionEndpoint.USWest2))
{
    cw.PutMetricData(new PutMetricDataRequest
    {
        MetricData = new List<MetricDatum>{new MetricDatum
        {
            MetricName = "PagesVisited",
            Dimensions = new List<Dimension>(dimension),
            Unit = "None",
            Value = 1.0
        }},
        Namespace = "SITE/TRAFFIC"
    });
}
Sending Events to Amazon CloudWatch Events

This .NET code example shows you how to:

- Create and update a scheduled rule to trigger an event
- Add a AWS Lambda function target to respond to an event
- Send events that are matched to targets

The Scenario

Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in AWS resources to various targets. Using simple rules, you can match events and route them to one or more target functions or streams. This .NET example shows you how to create and update a rule used to trigger an event, define one or more targets to respond to an event, and send events that are matched to targets for handling.

The code manages instances using these methods of the AmazonCloudWatchEventsClient class:

- PutRule
- PutTargets
- PutEvents

For more information about Amazon CloudWatch Events, see Adding Events with PutEvents in the Amazon CloudWatch Events User Guide.

Prerequisite Tasks

To set up and run this example, you must first:

- Get set up to use Amazon CloudWatch.
- Set up and configure the AWS SDK for .NET.
- Create a Lambda function using the hello-world blueprint to serve as the target for events. To learn how, see Step 1: Create an AWS Lambda function in the Amazon CloudWatch Events User Guide.

Create an IAM Role to Run the Examples

The following examples require an IAM role whose policy grants permission to CloudWatch Events and that includes events.amazonaws.com as a trusted entity. This example creates a role named CWEEvents, setting it's trust relationship and role policy.

```csharp
static void Main()
{
    var client = new AmazonIdentityManagementServiceClient();
    // Create a role and it's trust relationship policy
    var role = client.CreateRole(new CreateRoleRequest
    {
        RoleName = "CWEEvents",
        AssumeRolePolicyDocument = @"{"Statement":[{"Principal":{"Service":"events.amazonaws.com"}}]," + 
            @"Effect":"Allow","Action":["sts:AssumeRole"]}}").Role;
    // Create a role policy and add it to the role
```
string policy = GenerateRolePolicyDocument();
var request = new CreatePolicyRequest
{
    PolicyName = "DemoCWPermissions",
    PolicyDocument = policy
};
try
{
    var createPolicyResponse = client.CreatePolicy(request);
}
catch (EntityAlreadyExistsException)
{
    Console.WriteLine
    ("Policy 'DemoCWPermissions' already exits.");
}
var request2 = new AttachRolePolicyRequest()
{
    RoleName = "CWEvents"
};
try
{
    var response = client.AttachRolePolicy(request2); //managedpolicy
    Console.WriteLine("Policy DemoCWPermissions attached to Role TestUser");
}
catch (NoSuchEntityException)
{
    Console.WriteLine
    ("Policy 'DemoCWPermissions' does not exist");
}
catch (InvalidInputException)
{
    Console.WriteLine
    ("One of the parameters is incorrect");
}
}

public static string GenerateRolePolicyDocument()
{
    /* This method produces the following managed policy:
    "Version": "2012-10-17",
    "Statement": [
      {
        "Sid": "CloudWatchEventsFullAccess",
        "Effect": "Allow",
        "Action": "events:*",
        "Resource": "*"
      },
      {
        "Sid": "IAMPassRoleForCloudWatchEvents",
        "Effect": "Allow",
        "Action": "iam:PassRole",
        "Resource": "arn:aws:iam::*:role/AWS_Events_Invoke_Targets"
      }
    ]
    */
    var actionList = new ActionIdentifier("events:*");
    var actions = new List<ActionIdentifier>();
    actions.Add(actionList);
    var resource = new Resource("*");
    var resources = new List<Resource>();
    resources.Add(resource);
    var statement = new Amazon.Auth.AccessControlPolicy.Statement
    {

Create a Scheduled Rule

Create an `AmazonCloudWatchEventsClient` instance and a `PutRuleRequest` object containing the parameters needed to specify the new scheduled rule, which include the following:

- A name for the rule
- The ARN of the IAM role you created previously
- An expression to schedule triggering of the rule every five minutes

Call the `PutRule` method to create the rule. The `PutRuleResponse` returns the ARN of the new or updated rule.

```csharp
AmazonCloudWatchEventsClient client = new AmazonCloudWatchEventsClient();

var putRuleRequest = new PutRuleRequest
{
    Name = "DEMO_EVENT",
    RoleArn = "IAM_ROLE_ARN",
    ScheduleExpression = "rate(5 minutes)",
    State = RuleState.ENABLED
};

var putRuleResponse = client.PutRule(putRuleRequest);
Console.WriteLine("Successfully set the rule {0}", putRuleResponse.RuleArn);
```

Add a Lambda Function Target

Create an `AmazonCloudWatchEventsClient` instance and a `PutTargetsRequest` object containing the parameters needed to specify the rule to which you want to attach the target, including the ARN of the Lambda function you created. Call the `PutTargets` method of the `AmazonCloudWatchClient` instance.
Using Subscription Filters in Amazon CloudWatch Logs

This .NET example shows you how to:

- List a subscription filter in CloudWatch Logs
- Create or delete a subscription filter in CloudWatch Logs

The Scenario

Subscriptions provide access to a real-time feed of log events from CloudWatch Logs and deliver that feed to other services such as an Amazon Kinesis Data Streams or AWS Lambda for custom processing, analysis, or loading to other systems. A subscription filter defines the pattern to use for filtering which log events are delivered to your AWS resource. This example shows how to list, create, and delete a subscription filter in CloudWatch Logs. The destination for the log events is a Lambda function.

This example uses the AWS SDK for .NET to manage subscription filters using these methods of the AmazonCloudWatchLogsClient class:
Using Subscription Filters in Amazon CloudWatch Logs

- DescribeSubscriptionFilters
- PutSubscriptionFilter
- DeleteSubscriptionFilter

For more information about CloudWatch Logs subscriptions, see Real-time Processing of Log Data with Subscriptions in the Amazon CloudWatch Logs User Guide.

Prerequisite Tasks

To set up and run this example, you must first:

- Get set up to use Amazon CloudWatch.
- Set up and configure the AWS SDK for .NET.

Describe Existing Subscription Filters

Create an AmazonCloudWatchLogsClient object. Create a DescribeSubscriptionFiltersRequest object containing the parameters needed to describe your existing filters. Include the name of the log group and the maximum number of filters you want described. Call the DescribeSubscriptionFilters method.

```csharp
public static void DescribeSubscriptionFilters()
{
    var client = new AmazonCloudWatchLogsClient();
    var request = new Amazon.CloudWatchLogs.Model.DescribeSubscriptionFiltersRequest()
    {
        LogGroupName = "GROUP_NAME",
        Limit = 5
    };
    try
    {
        var response = client.DescribeSubscriptionFilters(request);
    }
    catch (Amazon.CloudWatchLogs.Model.ResourceNotFoundException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Create a Subscription Filter

Create an AmazonCloudWatchLogsClient object. Create a PutSubscriptionFilterRequest object containing the parameters needed to create a filter, including the ARN of the destination Lambda function, the name of the filter, the string pattern for filtering, and the name of the log group. Call the PutSubscriptionFilter method.

```csharp
public static void PutSubscriptionFilters()
{
    var client = new AmazonCloudWatchLogsClient();
    var request = new Amazon.CloudWatchLogs.Model.PutSubscriptionFilterRequest()
    {
        DestinationArn = "LAMBDA_FUNCTION_ARN",
        FilterName = "FILTER_NAME",
        FilterPattern = "ERROR",
        LogGroupName = "Log_Group"
    };
    try
    {
```
var response = client.PutSubscriptionFilter(request);
}
catch (InvalidOperationException e)
{
    Console.WriteLine(e.Message);
}

Delete a Subscription Filter

Create an AmazonCloudWatchLogsClient object. Create a DeleteSubscriptionFilterRequest object containing the parameters needed to delete a filter, including the names of the filter and the log group. Call the DeleteSubscriptionFilter method.

```csharp
public static void DeleteSubscriptionFilter()
{
    var client = new AmazonCloudWatchLogsClient();
    var request = new Amazon.CloudWatchLogs.Model.DeleteSubscriptionFilterRequest()
    {
        LogGroupName = "GROUP_NAME",
        FilterName = "FILTER"
    };
    try
    {
        var response = client.DeleteSubscriptionFilter(request);
    }
    catch (Amazon.CloudWatchLogs.Model.ResourceNotFoundException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Programming AWS OpsWorks to Work with Stacks and Applications

The AWS SDK for .NET supports AWS OpsWorks, which provides a simple and flexible way to create and manage stacks and applications. With AWS OpsWorks, you can provision AWS resources, manage their configuration, deploy applications to those resources, and monitor their health. For more information, see OpsWorks.

The SDK provides APIs for programming with AWS OpsWorks. These APIs typically consist of sets of matching request-and-response objects that correspond to HTTP-based API calls focusing on their corresponding service-level constructs.

For related API reference information, see Amazon.OpsWorks and Amazon.OpsWorks.Model in the AWS SDK for .NET Reference.

Programming Support for Additional AWS Services

The AWS SDK for .NET supports programming AWS services in addition to those described in the code examples. For information about programming specific services with the AWS SDK for .NET, 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:
<table>
<thead>
<tr>
<th>Area</th>
<th>Description</th>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Support</td>
<td>Programmatic access to AWS Support cases and Trusted Advisor features.</td>
<td>See Amazon.AWSSupport and Amazon.AWSSupport.Model.</td>
</tr>
<tr>
<td>General</td>
<td>Helper classes and enumerations.</td>
<td>See Amazon and Amazon.Util.</td>
</tr>
</tbody>
</table>

Other general programming information for the AWS SDK for .NET includes the following:

- Overriding Endpoints in the AWS SDK for .NET
- .NET Object Lifecycles
Additional Resources

Home Page for AWS SDK for .NET

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

SDK Reference Documentation

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

AWS Forums

Visit the AWS forums to ask questions or provide feedback about AWS. Each documentation page has a Go to the forums button at the top of the page that takes you to the associated forum. AWS engineers monitor the forums and respond to questions, feedback, and issues. You can also subscribe to RSS feeds for any of the forums.

AWS Toolkit for Visual Studio

If you use the Microsoft Visual Studio IDE, you should check out the Toolkit for Visual Studio User Guide.
Document History

The following table describes the important changes since the last release of the *AWS SDK for .NET Developer Guide*.

*Last major documentation update: November 1, 2017*

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update</td>
<td>Added examples and clarifications.</td>
<td>November 1, 2017</td>
</tr>
</tbody>
</table>