AWS SDK for .NET

Developer Guide

Version v1.0.0

AWS SDK for .NET: Developer Guide

AWS SDK for .NET Developer Guide

The AWS SDK for .NET is a single, downloadable package that includes Visual Studio project templates, the AWS .NET library, C# code samples, and documentation. 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 infrastructure services such as Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2).

The AWS SDK for .NET supports development on any platform that supports the .NET Framework 2.0 (or later). You can develop with the SDK using any edition of Visual Studio 2008 or 2010. To use the AWS Toolkit for Visual Studio, you will need the Standard Edition (or higher). Most developers can start building solutions for the AWS cloud without having to upgrade to a new operating system, .NET Framework, or Visual Studio version.

The AWS SDK for .NET has the following features:

AWS .NET Library
Build .NET applications on top of APIs that take the complexity out of coding directly against a web service interface. The library provides APIs that hide much of the lower-level plumbing, including authentication, request retries, and error handling.

Examples
Examples in C# that show how to use the library to build applications.

Documentation
Information about how to use the library and code samples as well as online video tutorials and reference documentation.

Visual Studio Support
Visual Studio templates for console and web applications to help you get started quickly on your AWS application. If you are developing with Microsoft Visual Studio 2008 (or 2010) Standard Edition, you should also install the AWS Toolkit for Visual Studio.

Getting Started (p. 4)

If you are just starting out with the AWS SDK for .NET, you should first read through the Getting Started (p. 4) section. It will guide you through setting up your development environment and introduce the samples that are included with the SDK.

Programming AWS (p. 10)

General programming techniques and information for developing software with the AWS SDK for .NET.

Tutorials: Accessing AWS Services from .NET

This developer guide includes tutorials that walk you through how to use some of the AWS services.

- **Tutorial: Amazon EC2 Spot Instances (p. 49)**
  Explains how to set up requests for Amazon EC2 Spot Instances, how to determine when they have completed, and how to clean up afterward.

How-to: Useful Code for Programming AWS

These topics are shorter than the preceding tutorials and deal with discrete programming tasks related to Tutorial: Create Amazon EC2 Instances (p. 40).

Supported Services

The AWS SDK for .NET supports the following AWS infrastructure products:

- **Compute**
  - Amazon EC2
  - Auto Scaling
  - Elastic Load Balancing
  - Amazon Elastic MapReduce
  - VM Import
- **Content Delivery**
  - Amazon CloudFront
- **Database**
  - Amazon SimpleDB
  - Amazon Relational Database Service
- **Deployment & Management**
  - AWS Elastic Beanstalk
  - AWS CloudFormation
- **Messaging**
  - Amazon Simple Notification Service (Amazon SNS)
  - Amazon Simple Queue Service (Amazon SQS)
  - Amazon Simple Email Service (Amazon SES)
- **Monitoring**
  - Amazon CloudWatch
- **Networking**

  - Amazon Virtual Private Cloud(Amazon VPC)
- **Security**
  - AWS Identity and Access Management(IAM)
  - AWS Security Token Service
- **Storage**
  - Amazon S3
  - AWS Import/Export

Revision History for the AWS SDK for .NET

We regularly release updates to the AWS SDK for .NET to support new services and new service features. To see what changed with a given release, you can check the release notes history.

Additional Resources

The Additional Resources section has pointers to other resources to assist you in programming AWS.

About Amazon Web Services

Amazon Web Services (AWS) is a collection of digital infrastructure services that developers can leverage when developing their applications. The services include computing, storage, database, and application synchronization (messaging and queuing). AWS uses a pay-as-you-go service model. You are charged only for the services that you—or your applications—use. Also, to make AWS more approachable as a platform for prototyping and experimentation, AWS offers a free usage tier. On this tier, services are free below a certain level of usage. For more information about AWS costs and the free tier go to Test-Driving AWS in the Free Usage Tier. To obtain an AWS account, go to the AWS home page and click **Sign Up Now**.

Getting Started

To get started with the AWS SDK for .NET, you need to set up the following:

- AWS Account and Credentials
- .NET Development Environment
- AWS SDK for .NET

AWS Account and Credentials

To access AWS, you will need to sign up for an AWS account.

To sign up for an AWS account

1. Go to http://aws.amazon.com, and then click **Sign Up**.
2. Follow the on-screen instructions.

   Part of the sign-up procedure involves receiving a phone call and entering a PIN using the phone keypad.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account by going to http://aws.amazon.com and clicking **My Account/Console**.

When you sign up, AWS provides you with security credentials that are specific to your account. Two of these credentials, your *access key ID* and your *secret key*, are used by the SDK whenever it accesses the services provided by AWS. The security credentials authenticate requests to the service and identify you as the sender of a request. The following list shows examples of these credentials.

- Access key ID example: AKIAIOSFODNN7EXAMPLE
- Secret access key example: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

To view your AWS access credentials

1. Go to the Amazon Web Services website at http://aws.amazon.com.
2. Click **My Account/Console**, and then click **Security Credentials**.

3. Under **Your Account**, click **Security Credentials**.
4. In the spaces provided, type your user name and password, and then click **Sign in using our secure server**.
5. Under **Access Credentials**, on the **Access Keys** tab, your access key ID is displayed. To view your secret key, under **Secret Access Key**, click **Show**.

Your secret key must remain a secret that is known only by you and AWS. Keep it confidential in order to protect your account. Store it securely in a safe place, and never email it. Do not share it outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

.NET Development Environment

To use the AWS SDK for .NET, you'll need the following software:

- Microsoft .NET Framework 3.5 or later
- To use the project templates, you will need one of the following Visual Studio editions:
  - Microsoft Visual Studio 2008 Standard Edition or Visual Studio 2010 Standard Edition
  - Microsoft Visual C# 2008 Express Edition
  - Microsoft Visual Web Developer 2008 Express Edition

  To use the AWS Toolkit for Visual Studio, you will need Visual Studio 2008 Standard Edition or Visual Studio 2010 Standard Edition.

To confirm installation

1. Open your Visual Studio product.
2. On the **Help** menu, click **About**. A dialog box opens that lists Microsoft Visual Studio and .NET Framework versions.

Configuring the .NET CLR

For the best performance of your server-based applications with the AWS SDK for .NET, we recommend that you use Server mode garbage collection (GC).

To enable Server mode GC, add the following to your app.config file:

```
<runtime>
    <gcServer enabled="true"/>
    <gcConcurrent enabled="true"/>
</runtime>
```

Note
Server mode GC works only on systems with multiple processors or processor cores. Enabling Server mode GC has no effect otherwise.

Installing the AWS SDK for .NET

To install the AWS SDK for .NET

1. Go to http://aws.amazon.com/sdkfornet.
2. Click the **Download** button in the upper right corner of the page. Your browser will prompt you to save the install file.
3. To begin the install process, open the saved install file and follow the on-screen instructions.

By default, the AWS SDK for .NET is installed in the *Program Files* directory, which requires administrator privileges. You can install the AWS SDK for .NET as a non-administrator if you choose a different installation directory.

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

Install AWS Assemblies with NuGet

NuGet is a package management system for the .NET platform. With NuGet, you can add the AWSSDK assembly and the AWS.Extensions assembly to your application without first installing the SDK.

NuGet always has the most recent versions of the AWS .NET assemblies, and also enables you to install previous versions. NuGet is aware of dependencies between assemblies and installs required assemblies automatically. Assemblies that are installed with NuGet are stored with your solution rather than in a central location such as `Program Files`. This enables you to install assembly versions specific to a given application without creating compatibility issues for other applications.

For more information on NuGet, go to the NuGet documentation.

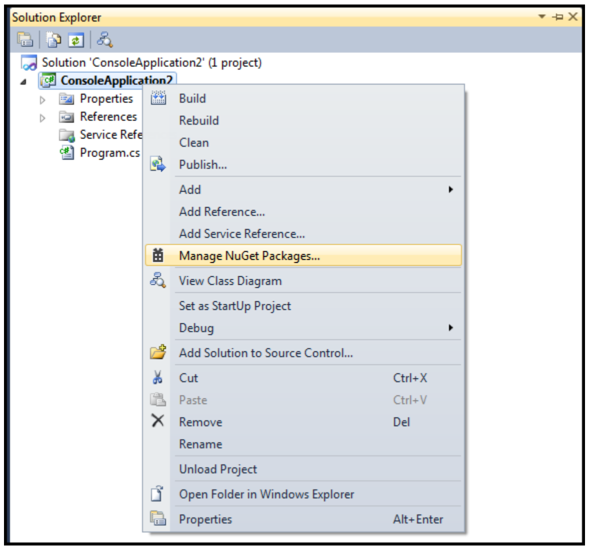
Installation

To use NuGet, install it from the Visual Studio Gallery on MSDN. If you are using Visual Studio 2012 or later, NuGet is installed automatically with Visual Studio.

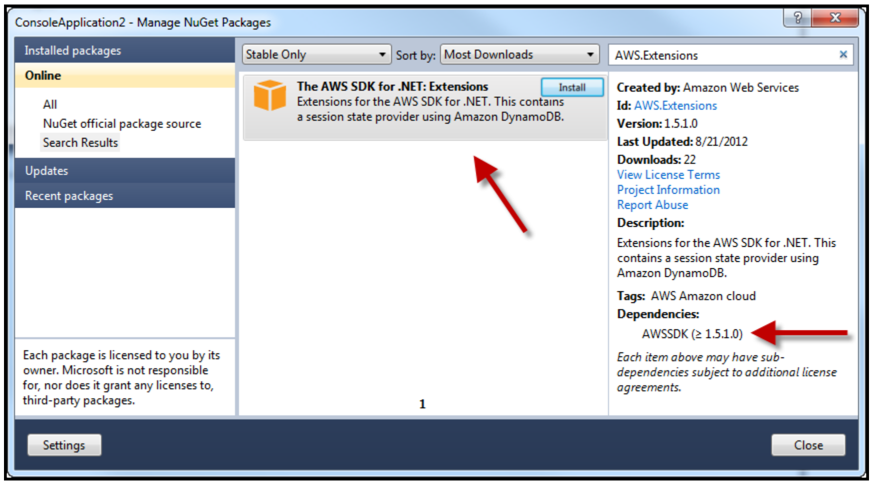
You can use NuGet either from **Solution Explorer** or from the **Package Manager Console**.

NuGet from Solution Explorer

To use NuGet from Solution Explorer, right-click on your project and select **Manage NuGet Packages...** from the context menu.



From the **Manage NuGet Packages** dialog box, select **Online** in the left pane. You can then search for the package that you want to install using the search box in the upper right corner. The screenshot shows the `AWS.Extensions` assembly package. Notice that NuGet is aware that this package has a dependency on the `AWSSDK` assembly package; NuGet will therefore install the `AWSSDK` package if it is not already installed.



The AWS SDK for .NET: Extensions
Extensions for the AWS SDK for .NET. This contains a session state provider using Amazon DynamoDB.

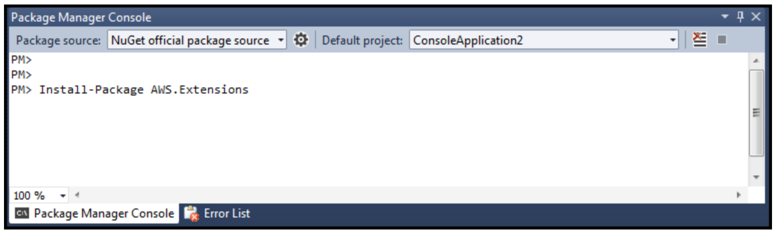
Created by: Amazon Web Services
Id: AWS.Extensions
Version: 1.5.1.0
Last Updated: 8/21/2012
Downloads: 22
View License Terms
Project Information
Report Abuse
Description:
Extensions for the AWS SDK for .NET. This contains a session state provider using Amazon DynamoDB.
Tags: AWS Amazon cloud
Dependencies:
AWSSDK (≥ 1.5.1.0)

Each item above may have sub-dependencies subject to additional license agreements.

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

NuGet Package Manager Console

To use NuGet from the Package Manager Console within Visual Studio, from the **View** menu, select **Other Windows**, and click **Package Manager Console**.


Package Manager Console
Package source: NuGet official package source
Default project: ConsoleApplication2
100 %
Package Manager Console
Error List

```
PM>
PM>
PM> Install-Package AWS.Extensions
```

From the console, you can install the AWS assemblies using the **`Install-Package`** command. For example, to install the AWS SDK for .NET assembly, use the following command line:

```
Install-Package AWSSDK
```

To install an earlier version of a package, use the `-Version` option and specify the desired package version. For example, to install version 1.5.1.0 of the AWS SDK for .NET assembly, use the following command line:

```
Install-Package AWSSDK -Version 1.5.1.0
```

The NuGet website provides a page for every package that is available through NuGet such as the AWSSDK and AWS.Extensions assemblies. The page for each package includes a sample command line for installing the package using the console. Each page also includes a list of the previous versions of the package that are available through NuGet.

Starting a New Project

The AWS SDK for .NET provides the following project template(s) you can use to build a new project on Amazon Web Services:

AWS Console Project
Creates a console application that references the AWS .NET library (AWSSDK.dll) with a sample Program.cs file that makes a basic request to Amazon Simple Storage Service (Amazon S3), Amazon SimpleDB, and Amazon Elastic Compute Cloud (Amazon EC2).

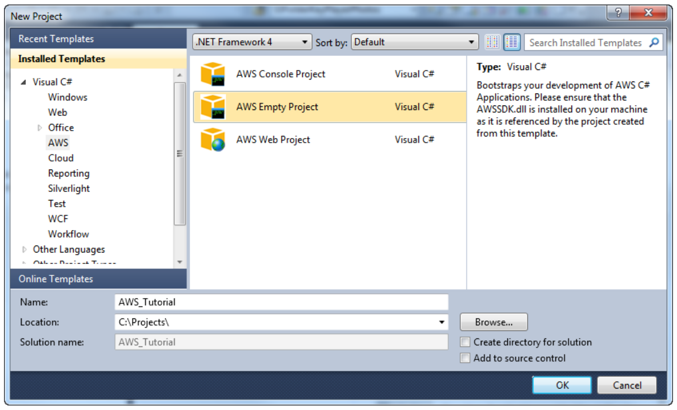
AWS Empty Project
Creates a console application that references the AWS .NET library (AWSSDK.dll), but does not include any code in the Program.cs file.

AWS Web Project
Creates an ASP.NET application that references the AWS .NET library (AWSSDK.dll) with a sample Program.cs file that makes a basic request to Amazon S3, Amazon SimpleDB, and Amazon EC2.

The general process to create a new project based on a project template is similar across Visual Studio editions, but we'll go through the steps for Visual Studio 2010 Professional Edition.

To create a new project in Visual Studio 2010 Professional Edition

1. Launch Visual Studio.
2. On the **File** menu, select **New**, and then click **Project**. The **New Project** dialog box opens.
3. Select **AWS** from the list of installed templates, and then select the AWS project template you want to use. Enter a project name, and then click **OK**.



.NET Framework 4 Sort by: Default

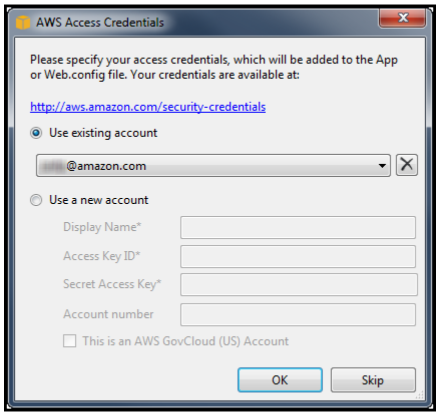
Type: Visual C#

Bootstraps your development of AWS C# Applications. Please ensure that the AWSSDK.dll is installed on your machine as it is referenced by the project created from this template.

Name: AWS_Tutorial
Location: C:\Projects\
Solution name: AWS_Tutorial
Create directory for solution
Add to source control

4. On project creation, the template prompts you for the security credentials that your code should use to access AWS. To use the credentials for one of the accounts that was added with the AWS Toolkit for Visual Studio, click **Use existing account** and select the account from the drop-down list. To add a new set of credentials, click **Use a new account** and enter the credentials information. Once you have specified the credentials to use, click **OK**.

   If you do not want to specify any credentials at this time, click **Skip**. Note, however, that your code will not be able to access AWS without a valid set of credentials.



Please specify your access credentials, which will be added to the App or Web.config file. Your credentials are available at:

http://aws.amazon.com/security-credentials

[✓ Use existing account]

[redacted]@amazon.com

Use a new account

Display Name*
Access Key ID*
Secret Access Key*
Account number

This is an AWS GovCloud (US) Account

Running the Project

You can run the project immediately after the project is created by pressing **F5** (or clicking **Start Debugging** on the **Debug** menu).

Where Do I Go from Here?

From here, you can check out the tutorials included in this developer guide.

- **Tutorial: Amazon EC2 Spot Instances (p. 49)**

  This tutorial explains how to set up requests for Amazon EC2 Spot Instances, how to determine when they have completed, and how to clean up afterward.

The Additional Resources section has pointers to other resources to assist you in programming AWS.

Programming AWS

General programming techniques and information for developing software with the AWS SDK for .NET.

Configuring Credentials for Your Application

For the AWS Console Project and AWS Empty Project, your security credentials are added to the App.config file. For the AWS Web Project, your security credentials are added to the Web.config file.

To add your credentials to the App.config file

1. With your Visual Studio solution open, in **Solution Explorer**, right-click the App.config file, and then click **View Code**.
2. The App.config file is an XML file. In the **appSettings** element, edit the **value** attributes to add your access key ID and secret key, as shown.

```
<appSettings>
  <add key="AWSAccessKey" value="YOUR ACCESS KEY ID" />
  <add key="AWSSecretKey" value="YOUR SECRET KEY" />
  <add key="ClientSettingsProvider.ServiceUri" value="" />
</appSettings>
```

3. Save your changes.

To add your credentials to the Web.config file

1. With your Visual Studio solution open, in **Solution Explorer**, right-click the Web.config file, and then click **Open**.
2. The Web.config file is an XML file. In the **appSettings** element, edit the **value** attributes to add your access key ID and secret key, as shown below.

```
<appSettings>
  <!-- AWS CREDENTIALS -->
  <!-- Uncomment the following section and add your credentials if you are
 not using a deployment mechanism that manages credentials. -->
  <!--
  <add key="AWSAccessKey" value="YOUR ACCESS KEY ID"/>
  <add key="AWSSecretKey" value="YOUR SECRET KEY"/>
  -->
</appSettings>
```

Securely Managing Your Credentials

For experimentation, prototyping, and initial development, you could embed your personal AWS credentials in your application as described above. However, your personal credentials provide complete access to all AWS resources associated with your account. They also provide access to the account information itself, such as your billing information.

Therefore, in order to keep your credentials as secure as possible, we recommend that, for production code, you instead use credentials that provide only the access that your application requires. To do this, you would use the AWS Identity and Access Management (IAM) service to generate an *IAM user account*. Then, instead of using your own credentials in your software, you would use the credentials associated with the IAM user. By default, IAM users have no access to AWS resources. However, by attaching an IAM policy to the user, you can grant permissions to access specific resources. For example, you could attach a policy that grants the user access only to a specific Amazon S3 bucket. For more information, go to the Getting Started Guide for Identity and Access Management.

For applications that run on Amazon EC2 instances, the most secure way to manage credentials is to use IAM roles for EC2 Instances. See the following topic for information about how to use this technology with the AWS SDK for .NET.

- Using IAM Roles for EC2 Instances with the SDK for .NET (p. 19)

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

- 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 above refers specifically to mobile applications, the article itself contains information that is useful for any AWS application that is deployed outside of your organization.

Proxy Credentials

If your software communicates with AWS through a proxy, you should specify credentials for the proxy using the `ProxyCredentials` property on the ClientConfig class for the service. For example, for Amazon S3, you could use code similar to the following, where `foo` and `bar` are the proxy username and password specified in a NetworkCredential object.

```
AmazonS3Config config = new AmazonS3Config();
config.ProxyCredentials = new NetworkCredential("foo", "bar");
```

Earlier versions of the SDK used `ProxyUsername` and `ProxyPassword`, but these properties have been deprecated.

AWS Region Selection

AWS regions allow you to access AWS services that reside physically in a specific geographic region. This can be useful both for redundancy and to keep your data and applications running close to where you and your users will access them. To select a particular region, configure the AWS client object with an endpoint that corresponds to that region.

For example, to instantiate an Amazon Elastic Compute Cloud(Amazon EC2) client that connects to the EU (Ireland) Region region, use the following code:

```
AmazonEC2Config config = new AmazonEC2Config();
config.ServiceURL = "https://ec2.eu-west-1.amazonaws.com";
AmazonEC2Client ec2 = new AmazonEC2Client("ACCESS_KEY", "SECRET_KEY", config);
```

Be aware that regions are logically isolated from each other, so for example, you won't be able to access US East resources when communicating with the EU West endpoint. If you're code accesses multiple AWS regions, we recommend that you instantiate a specific client for each region.

The AWS SDK for .NET uses US East (Northern Virginia) Region as the default region if you do not specify a region in your code. However, the AWS Management Console uses US West (Oregon) Region as its default. Therefore, when using the AWS Management Console in conjunction with your development, be sure to specify the same region in both your code and the console.

Go to Regions and Endpoints for the current list of regions and corresponding endpoints for each of the services offered by AWS.

AWS SDK for .NET Version 2 (Preview)

The AWS SDK for .NET Version 2 is the next major version of our AWS SDK for .NET. The Version 2 SDK adds support for Windows Store and Windows Phone apps, enabling developers to use AWS from these new platforms. The new SDK also adopts the new .NET task-based asynchronous pattern, which simplifies making asynchronous calls with the new `async` and `await` keywords.

- Download Version 2 of the SDK as a zip file
- Get Version 2 of the SDK using nuget from inside the Visual Studio IDE

This section includes topics on how to migrate your code to use the new SDK—including information on breaking changes—and on differences in functionality between the newly supported platforms. Also below is a link to the new API documentation for Version 2.

- Migration Guide (p. 13)
- Platform Differences (p. 17)
- API Reference

To provide feedback on the new SDK, contact us through the Forums or through our GitHub project.

Migration Guide for Version 2 of the AWS SDK for .NET (Preview)

This guide describes how the new AWS SDK for .NET version 2 differs from the first version of the SDK and how to migrate your code to use the new SDK.

Introduction

The AWS SDK for .NET was first released in November 2009 and was designed for .NET Framework 2.0. Since then, .NET has had many improvements with .NET 4.0 and .NET 4.5. Also, .NET has added the new platforms WinRT and Windows Phone. For version 2, the SDK has been updated to take advantage of the new features of .NET and to work on the new .NET platforms as well.

What's New

- Support for `Task`-based asynchronous API
- Support for Windows Store apps
- Support for Windows Phone 8
- Ability to configure service region via `app.config` or `web.config`
- Collapsed `Response` and `Result` classes
- Updated names for classes and properties to follow .NET conventions

What's Different

Architecture

Inside the .NET SDK, there is a common runtime that services use to make requests. In version 1 of the SDK, this runtime was added after the initial release, and several of the older services do not run on it. In version 2 of the SDK, all services have moved to run on the common runtime. Doing this enables us to add new features to the core runtime, which then propagate to all the services. For example, in version 1 of the SDK, Amazon Simple Queue Service (Amazon SQS) and Amazon Simple Notification Service (Amazon SNS) never had asynchronous operations, but now that they have been moved to the common runtime, all the operations can be called asynchronously.

In version 2 of the SDK, there are separate runtimes for .NET 3.5 and .NET 4.5. The version 2 runtime for .NET 3.5 is similar to the existing version 1 runtime, which is based on the `System.Net.HttpWebRequest` class and uses the `Begin` and `End` pattern for asynchronous methods. The version 2 runtime for .NET 4.5 is based on the new `System.Net.Http.HttpClient` class and uses `Tasks` for asynchronous methods, which enables users to use the new `async/await` keywords in C# 5.0. The WinRT and Windows Phone 8 versions of the SDK reuse the runtime for .NET 4.5, with the exception that they support only asynchronous methods. Windows Phone 8 doesn't natively support `System.Net.Http.HttpClient`, so the SDK depends on Microsoft's portable class implementation of `HttpClient`, which is hosted on nuget at the following URL:

```
http://nuget.org/packages/Microsoft.Net.Http/2.1.10
```

Removal of the "With" methods

In version 1 of the SDK, classes had "With" methods for every property to support a fluent style for setting properties. This was useful for the original .NET 2.0 target of the SDK, but in .NET 3.0, we added constructor initializers, which made the "With" methods redundant. The "With" methods also added significant overhead to the API design and worked poorly in cases of inheritance. As a result, we removed the "With" methods from version 2 of the SDK because they were nonstandard and offered no benefit over constructor initializers.

Using the "With" methods in version 1 of the SDK to set up a `TransferUtilityUploadRequest` object would look like this:

```
TransferUtilityUploadRequest uploadRequest = new TransferUtilityUploadRequest()

    .WithBucketName("my-bucket")
    .WithKey("test")
    .WithFilePath("c:\test.txt")
    .WithServerSideEncryptionMethod(ServerSideEncryptionMethod.AES256);
```

Using constructor initializers in version 2 of the SDK would change the code to this:

```
TransferUtilityUploadRequest uploadRequest = new TransferUtilityUploadRequest()
{
    BucketName = "my-bucket",
    Key = "test",
    FilePath = "c:\test.txt",
    ServerSideEncryptionMethod = ServerSideEncryptionMethod.AES256
};
```

Removal of SecureString

The use of `System.Security.SecureString` was removed in version 2 of the SDK because it is not available on the WinRT and Windows Phone 8 platforms.

Breaking Changes

Many classes and properties were changed to either meet .NET naming conventions or more closely follow service documentation. Amazon Simple Storage Service (Amazon S3) and Amazon Elastic Compute Cloud (Amazon EC2) were the most affected by this because they are the oldest services in the SDK and were moved to the new common runtime. Below are the most visible changes.

- All client interfaces have been renamed to follow the .NET convention of starting with the letter "I".
- Properties for collections have been properly pluralized.
- `AWSClientFactory.CreateAmazonSNSClient` has been renamed `CreateAmazonSimpleNotificationServiceClient`.
- `AWSClientFactory.CreateAmazonIdentityManagementClient` has been renamed `CreateAmazonIdentityManagementServiceClient`.

Amazon DynamoDB

- The `amazon.dynamodb` namespace has been removed; only the `amazon.dynamodbv2` namespace remains.
- Service-response collections that were set to null in version 1 are now set to an empty collection. For example, `queryrequest.lastevaluatedkey` and `scanresponse.lastevaluatedkey` will be set to empty collections when there are no more items to query/scan. If your code depends on `lastevaluatedkey` to be null, it now has to check the collection `count` field to avoid a possible infinite loop.

Amazon EC2

- `Amazon.EC2.Model.RunningInstance` has been renamed `Instance`.
- `Amazon.EC2.Model.IpPermissionSpecification` has been renamed `IpPermission`.
- `Amazon.EC2.Model.Volume.Status` has been renamed `State`.
- The `Amazon.EC2.Model.Instance` properties `GroupName` and `GroupId` have been combined into the `GroupIdentifier` property.
- `AuthorizeSecurityGroupIngress` removed root properties for `ToPort` and `FromPort` in favor of always using `IpPermission`. This was done because the root properties were silently ignored when set for an instance running in a VPC.

Amazon Redshift

- The `ClusterVersion.Name` property has been renamed `ClusterVersion.Version`.

Amazon S3

- All "Set" operations were renamed to "Put".
- `AmazonS3Config.CommunicationProtocol` was removed to be consistent with other services where `ServiceURL` contains the protocol.
- The `PutACLRequest.ACL` property has been renamed `AccessControlList` to make it consistent with `GetACLRequest`.
- `GetNotificationConfiguration` and `SetNotificationConfiguration` have been renamed `GetBucketNotification` and `PutBucketNotification`, respectively.
- `EnableBucketLogging` and `DisableBucketLogging` were consolidated into `PutBucketLogging`.
- The `GenerateMD5` property has been removed from `PutObject` and `UploadPart` because this is now computed as the object is being written to Amazon S3 and compared against the MD5 returned in the response from Amazon S3.
- `PutBucketTagging` has been simplified to remove the extra `TagSet` collection.
- The parameter order for the utility methods `DoesS3BucketExist`, `SetObjectStorageClass`, `SetServerSideEncryption`, `SetWebsiteRedirectLocation`, and `DeleteS3BucketWithObjects` in `AmazonS3Util` were changed to pass the `IAmazonS3` client first to be consistent with other high-level APIs in the SDK.
- Only responses that return a `Stream` like `GetObjectResponse` are `IDisposable`. In version 1, all responses were `IDisposable`.

Amazon Simple Workflow Service

- The `DomainInfos.Name` property has been renamed `DomainInfos.Infos`.

Configuring Region

In version 1 of the SDK, the access key ID and secret key can be configured in the app.config/web.config. With version 2 of the SDK, the AWS region can be configured as well. For example, the following app.config configures all clients that don't explicitly set the region to point to us-west-2.

```
<configuration>
  <appSettings>
    <add key="AWSAccessKey" value="YOUR_ACCESS_KEY"/>
    <add key="AWSSecretKey" value="YOUR_SECRET_KEY"/>
    <add key="AWSRegion" value="us-west-2"/>
  </appSettings>
</configuration>
```

Response and Result classes

To help simplify developers' code, the `Response` and `Result` classes that are returned by a service call have been collapsed. For example, to get an SQS Queue URL in version 1, the code would look like this:

```
GetQueueUrlResponse response = SQSClient.GetQueueUrl(request);

Console.WriteLine(createResponse.CreateQueueResult.QueueUrl);
```

But in version 2, the extra call to retrieve `CreateQueueResult` has been removed, changing the second line to:

```
Console.WriteLine(createResponse.QueueUrl);
```

The `CreateQueueResult` property still exists, but has been marked as obsolete.

Platform Differences in Version 2 of the AWS SDK for .NET (Preview)

The AWS SDK for .NET provides four distinct assemblies for developers to target different platforms. However, not all SDK functionality is available on each of these platforms. This topic describes the differences in what is supported across these platforms.

AWS SDK for .NET Framework 3.5

This version of the new AWS SDK for .NET is the one most similar to version 1 of the SDK. This version, compiled against .NET Framework 3.5, supports the same set of services as version 1. It also uses the same pattern for making asynchronous calls (p. 28). Note, however, that this version still contains a number of breaking changes. For more information about these, see the Migration Guide (p. 13).

AWS SDK for .NET Framework 4.5

The version of the .NET SDK v2 compiled against .NET Framework 4.5 supports the same set of services as version 1 of .NET SDK. However, it uses a different pattern for asynchronous calls. Instead of the Begin/End pattern, it uses the task-based pattern which allows developers to use the new `async` and `await` keywords introduced in C# 5.0.

AWS SDK for Windows RT

The version of the .NET SDK v2 compiled for Windows RT supports only asynchronous method calls using async and await.

This version does not provide all of the functionality for Amazon S3 and Amazon DynamoDB available in version 1 of the SDK. The following Amazon S3 functionality is currently unavailable in the Windows RT version of SDK.

- Transfer Utility
- IO Namespace

The Windows RT version of the SDK supports only the Amazon DynamoDB low-level API. It does not support Amazon DynamoDB Helper classes or the .NET Object Persistence Model.

Also, the Windows RT version of the SDK does not support decryption of the Windows password using the GetDecryptedPassword method.

AWS SDK for Windows Phone 8

The version of the .NET SDK v2 compiled for Windows Phone 8 (and later versions) has a programming model similar to Winodws RT. As with the Windows RT version, it supports only asynchronous method calls using async and await. Also, because Windows Phone 8 doesn't natively support `System.Net.Http.HttpClient`, the SDK depends on Microsoft's portable class implementation of `HttpClient`, which is hosted on nuget at the following URL:

```
http://nuget.org/packages/Microsoft.Net.Http/2.1.10
```

This version of the AWS .NET SDK supports only the following services. This is the same set of services as those supported in the AWS SDK for Android and the AWS SDK for iOS.

- Amazon EC2
- Elastic Load Balancing
- Auto Scaling
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon SES
- Amazon DynamoDB
- Amazon SimpleDB
- Amazon CloudWatch
- Amazon STS

This version does not provide all of the functionality for Amazon S3 and Amazon DynamoDB available in version 1 of the SDK. The following Amazon S3 functionality is currently unavailable in the Windows Phone 8 version of SDK.

- Transfer Utility
- IO Namespace

The Windows Phone 8 version of the SDK supports only the Amazon DynamoDB low-level API. It does not support Amazon DynamoDB Helper classes or the .NET Object Persistence Model.

Also, the Windows Phone 8 version of the SDK does not support decryption of the Windows password using the GetDecryptedPassword method.

Using IAM Roles for EC2 Instances with the AWS SDK for .NET

Note
For in-depth information about IAM roles for EC2 instances, go to Using Identity and Access Management.

Securely managing authentication credentials is one of the challenges developers face when writing software that accesses Amazon Web Services (AWS). All requests to AWS must be cryptographically signed using credentials issued by AWS. For software that runs on Amazon Elastic Compute Cloud (Amazon EC2) instances, developers must store these credentials in a way that keeps them secure but also accessible to the software, which needs them in order to make requests.

IAM roles for EC2 instances provides an effective way to manage credentials for AWS software running on EC2 instances. This section describes IAM roles for EC2 instances and shows how it works with a sample .NET program. But first, let's examine some common strategies for managing credentials and the issues that arise when using them.

One strategy is to first launch an Amazon EC2 instance and then securely transfer the credentials to the instance using a utility such as SCP (secure copy). However, this strategy doesn't scale well to large numbers of instances. It also doesn't work well for instances that are created by AWS on behalf of the customer, such as Spot Instances or instances in Auto Scaling groups.

Another strategy is to embed the credentials as literal strings in the software itself. However, this means that anyone who comes into possession of the software can scan through the code and retrieve the credentials.

Yet another strategy is to create a custom AMI (Amazon Machine Image) with the credentials, perhaps stored in a file on the AMI. However, with this approach anyone with access to the AMI automatically has access to the credentials—which again creates an unnecessary security risk.

All of the above strategies also make it cumbersome to rotate (update) the credentials. The new credentials either have to be re-copied to the EC2 instance or compiled into a new build of the software or incorporated into the creation of a new AMI.

Use IAM Roles for EC2 Instances to Manage Your Credentials

IAM roles for EC2 instances provides a solution. With IAM roles, a developer can develop software and deploy it to an EC2 instance without having to manage the credentials the software is using.

You use the IAM console to create the IAM role and configure it with all the permissions that the software requires. Permissions for IAM roles are specified in a way that is similar to permissions for IAM users. For more information about specifying permissions, go to Using Identity and Access Management.

Amazon EC2 instances support the concept of an instance profile, which is a logical container for the IAM role. At the time that you launch an EC2 instance, you can associate the instance with an instance profile, which in turn corresponds to the IAM role. Any software that runs on the EC2 instance is able to access AWS using the permissions associated with the IAM role.

If you are using the AWS Management Console, you don't need to worry about instance profiles. The IAM console creates one for you in the background whenever you create an IAM role.

To use the permissions associated with the instance profile, the software constructs a client object for an AWS service, say Amazon Simple Storage Service (Amazon S3), using an overload of the constructor that does not take any parameters. When this parameterless constructor executes, it searches the "credentials provider chain." The credentials provider chain is the set of places where the constructor will attempt to find credentials if they are not specified explicitly as parameters. For .NET, the credentials provider chain is:

- App.config file
- Instance Metadata Service that provides the credentials associated with the IAM role for the EC2 instance

If the client does not find credentials in the App.config file, it retrieves temporary credentials that have the same permissions as those associated with the IAM role. The credentials are retrieved from Instance Metadata. The credentials are stored by the constructor on behalf of the customer 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.

Note
AWS CloudFormation does not support calling its API with an IAM role. You must call the AWS CloudFormation API as a regular IAM user.

Walkthrough: Using IAM Roles to Retrieve an Amazon S3 Object from an EC2 Instance

In this walkthrough, we consider a program that retrieves an object from Amazon S3 using regular credentials. We then modify the program so that it uses IAM roles for EC2 Instances.

Sample Program with Credentials

Here is our starting program, which retrieves an object from an Amazon S3 bucket. The code as shown here accesses credentials using the App.config file.

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.IO;

using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.retrieveobject
{
  class S3Sample
  {
    static string bucketName = "text-content";
    static string keyName    = "text-object.txt";

    static AmazonS3 client;

    public static void Main(string[] args) {

      NameValueCollection appConfig = ConfigurationManager.AppSettings;

      string accessKeyID = appConfig["AWSAccessKey"];
      string secretAccessKeyID = appConfig["AWSSecretKey"];

      string responseBody;

      try {
        using ( client = Amazon.AWSClientFactory.CreateAmazonS3Client(
                accessKeyID, secretAccessKeyID ) ) {

          Console.WriteLine("Retrieving (getting) an object");

          GetObjectRequest request = new GetObjectRequest()
            .WithBucketName(bucketName)
            .WithKey(keyName);

          using (GetObjectResponse response = client.GetObject(request)) {
            using (Stream responseStream = response.ResponseStream) {
              using (StreamReader reader = new StreamReader(responseStream)) {

                  responseBody = reader.ReadToEnd();
              }
            }
          }
        }

        FileStream s  = new FileStream( "s3Object.txt", FileMode.Create );
        StreamWriter writer = new StreamWriter( s );
        writer.WriteLine( responseBody );
      }
      catch (AmazonS3Exception s3Exception) {
        Console.WriteLine(s3Exception.Message, s3Exception.InnerException);
      }
    } // main
  } // class
} // namespace
```

You can test this program, by filling in your own credentials in an App.config. The following example shows the lines you would need to edit.

```
<?xml version="1.0"?>
<configuration>
 <appSettings>
    <add key="AWSAccessKey" value="AKIAIOSFODNN7EXAMPLE"/>
    <add key="AWSSecretKey" value="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"/>

 </appSettings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```

You should also specify the name of an Amazon S3 bucket and text object associated with your account.

```
static string bucketName = "text-content";
static string keyName    = "text-object.txt";
```

For instructions on how to create an Amazon S3 bucket and upload an object, go to the Amazon Simple Storage Service Getting Started Guide.

After you have configured the program for your environment (credentials and target object to retrieve), you can compile and run it. You can compile the program in Visual Studio or from a Visual Studio Command Prompt. Here is a sample command line for compiling the program (here called `get-object.cs`). You will need to adjust the path to the AWS assembly (AWSSDK.dll) to match the location of the assembly on your computer. When you run the program, the assembly should be in the General Assembly Cache (GAC) or in the same directory as the program.

```
csc get-object.cs /r:c:\aws-sdk-net\bin\AWSSDK.dll
```

Update the Sample Program for IAM Roles for EC2 Instances

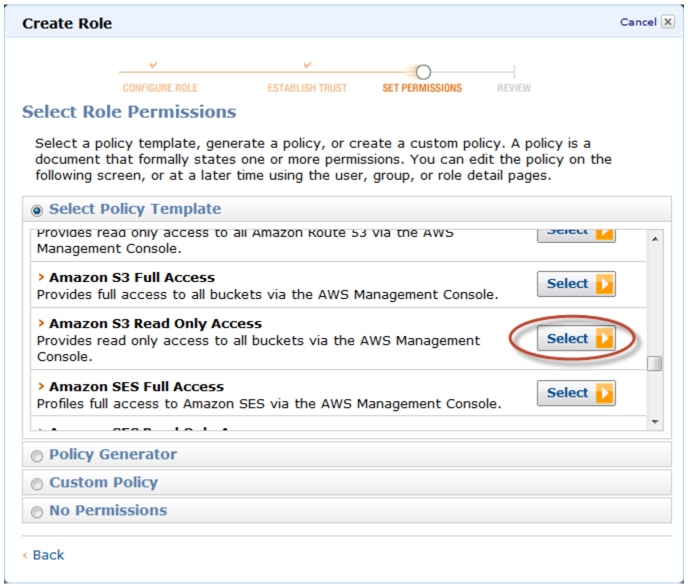
Our next stage is to update this program to run from an EC2 instance using IAM roles for EC2 instances. Here are the high-level steps.

1. **Create the Role**
2. **Launch an EC2 Instance with the Corresponding Instance Profile**
3. **Edit the Source File to Remove the Credentials**
4. **Transfer the Compiled Program to Your EC2 Instance**
5. **Run the Program**

Create the Role

The first step is to create an IAM role that has the appropriate permissions. To create the IAM role, follow the procedure Creating an IAM Role in *Using IAM*. When you create the IAM role, specify that the trusted entity is Amazon EC2 and that the role has read access to Amazon S3.

The IAM console provides ready-made policy templates for specific AWS services. When you create the IAM role, specify the **Amazon S3 Read Only Access** policy template. The following screen shot from the IAM role creation wizard shows this policy template.



Select Role Permissions

Select a policy template, generate a policy, or create a custom policy. A policy is a document that formally states one or more permissions. You can edit the policy on the following screen, or at a later time using the user, group, or role detail pages.

Select Policy Template

Provides read only access to all Amazon Route 53 via the AWS Management Console.

Amazon S3 Full Access
Provides full access to all buckets via the AWS Management Console.

Amazon S3 Read Only Access
Provides read only access to all buckets via the AWS Management Console.

Amazon SES Full Access
Profiles full access to Amazon SES via the AWS Management Console.

Policies can also be represented in JSON format. The following JSON block describes the policy for Amazon S3 Ready Only Access.

```
{
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

Note down the name of the role that you create so that you can specify it when you create your EC2 Instance in the next step.

Launch an EC2 Instance with the Corresponding Instance Profile

To create an EC2 instance, follow the procedure Running an Instance in the *Amazon Elastic Compute Cloud User Guide*. We recommend that you specify a recent **Windows Server 2008 R2** for your EC2 instance. When you create the EC2 instance, specify the IAM role that you created previously in the IAM console.

When you create your EC2 instance, you will also need to specify a key pair and a security group. Specify a key pair for which you have the private key (PEM file) stored on your local computer. Specify a security group that will enable you to connect to your EC2 instance using RDP (port 3389). Information about key pairs and security groups is provided in the *Amazon Elastic Compute Cloud User Guide*.

Edit the Source File to Remove the Credentials

Edit the source for the program so that it does **not** specify any credentials in the call that creates the Amazon S3 client. In this new version of the program, the call to `CreateAmazonS3Client` no longer takes any parameters.

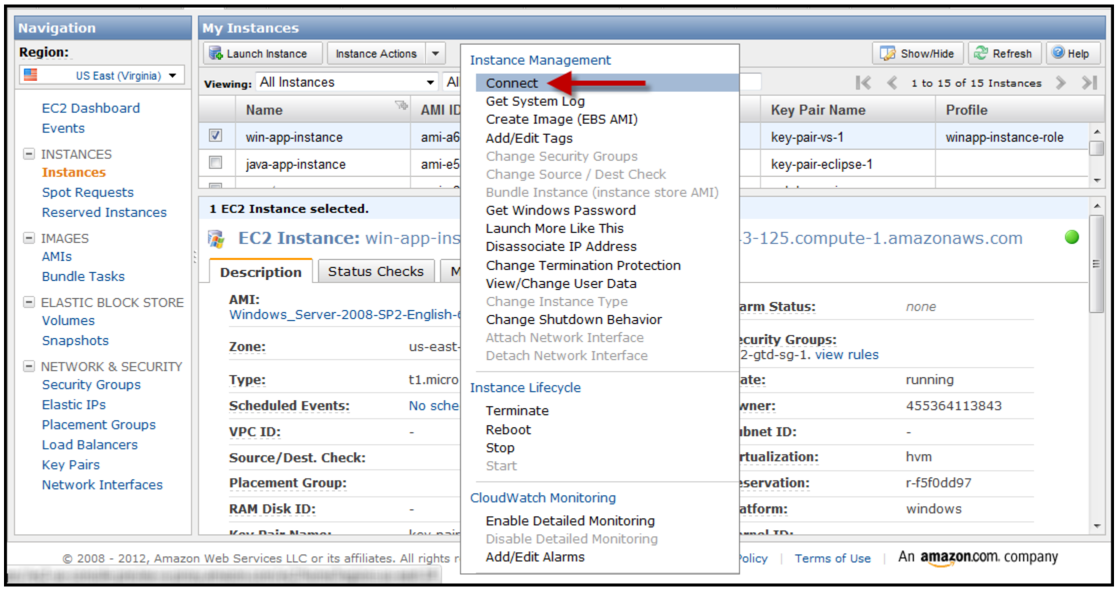
```
using ( client = Amazon.AWSClientFactory.CreateAmazonS3Client() ) {
```

Build the modified program. You might actually run the program on your local computer to verify that it does not work without credentials; you will get an Amazon Service Exception.

```
using System;
using System.Configuration;
using System.Collections.Specialized;
using System.IO;

using Amazon.S3;
using Amazon.S3.Model;

namespace s3.amazon.com.docsamples.retrieveobject
{
  class S3Sample
  {
    static string bucketName = "text-content";
    static string keyName = "text-object.txt";

    static AmazonS3 client;

    public static void Main(string[] args) {

      NameValueCollection appConfig = ConfigurationManager.AppSettings;

      string responseBody;

      try {
        using ( client = Amazon.AWSClientFactory.CreateAmazonS3Client() ) {

          Console.WriteLine("Retrieving (getting) an object");

          GetObjectRequest request = new GetObjectRequest()
            .WithBucketName(bucketName)
            .WithKey(keyName);

          using (GetObjectResponse response = client.GetObject(request)) {
            using (Stream responseStream = response.ResponseStream) {
              using (StreamReader reader = new StreamReader(responseStream)) {

                  responseBody = reader.ReadToEnd();
              }
            }
          }
        }

        FileStream s  = new FileStream( "s3Object.txt", FileMode.Create );
        StreamWriter writer = new StreamWriter( s );
        writer.WriteLine( responseBody );
      }
      catch (AmazonS3Exception s3Exception) {
        Console.WriteLine(s3Exception.Message, s3Exception.InnerException);
      }
    } // main
  } // class
} // namespace
```

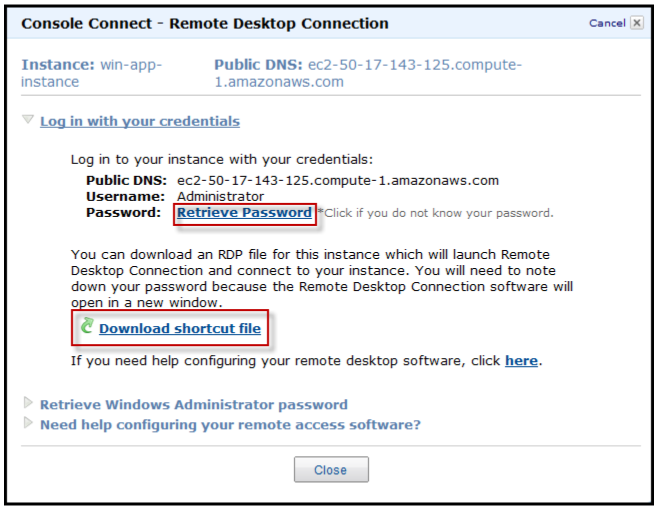
Transfer the Compiled Program to Your EC2 Instance

To transfer the program to your EC2 instance, connect to your EC2 instance using Remote Desktop. In the **Instances** view in the AWS Management Console, either click **Instance Actions** or right-click your EC2 instance and select **Connect**.



You need to retrieve the administrator's password for your EC2 instance in order to connect. To retrieve the password, you need to specify the private key file for the key pair. When you create a key pair, you have an opportunity to download the private key to your local computer. This file ends in a PEM extension.

You can invoke the Remote Desktop client from the **Start** menu on your computer or you can download a shortcut from the **Connect** dialog box in the AWS Management Console.


Console Connect - Remote Desktop Connection

Instance: win-app-instance
Public DNS: ec2-50-17-143-125.compute-1.amazonaws.com

Log in with your credentials

Log in to your instance with your credentials:
Public DNS: ec2-50-17-143-125.compute-1.amazonaws.com
Username: Administrator
Password: Retrieve Password *Click if you do not know your password.

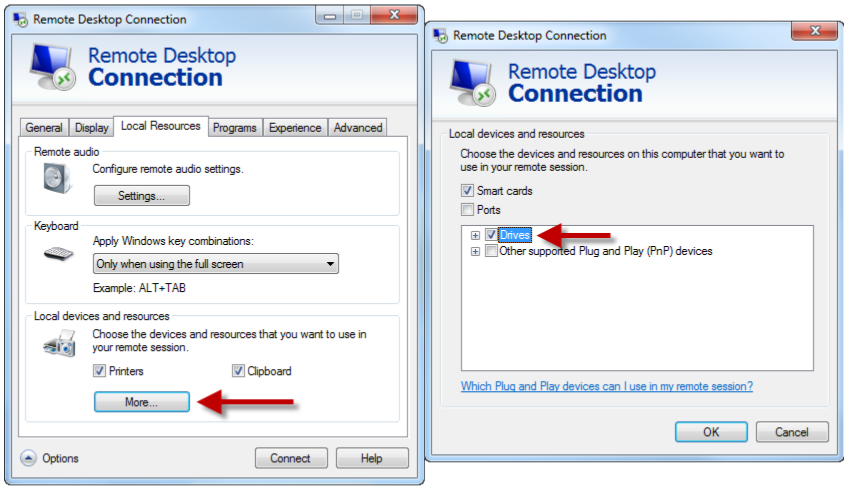
You can download an RDP file for this instance which will launch Remote Desktop Connection and connect to your instance. You will need to note down your password because the Remote Desktop Connection software will open in a new window.

Download shortcut file

If you need help configuring your remote desktop software, click here.

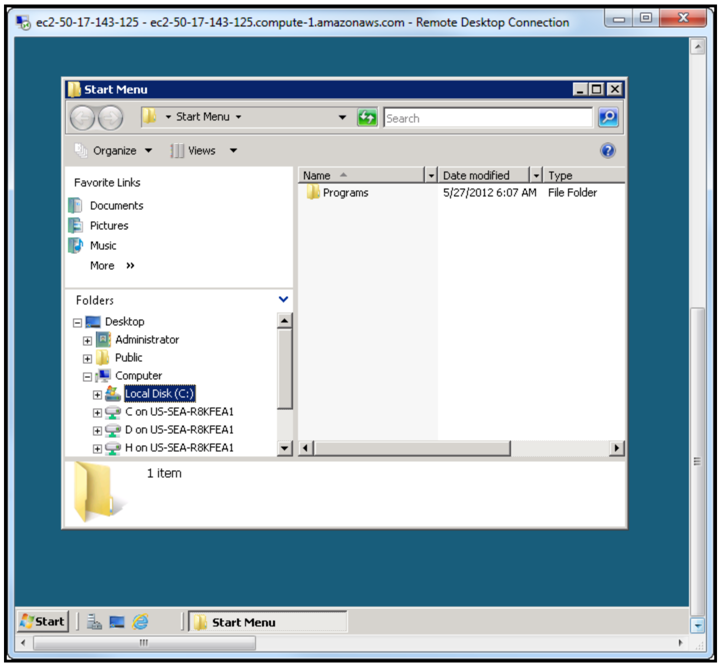
Retrieve Windows Administrator password
Need help configuring your remote access software?

You should configure your Remote Desktop session so that you can access your local drives from the EC2 instance. If you use the shortcut provided by the AWS Management Console, right-click the shortcut and select **Edit**. On the **Local Resources** tab, in the **Local devices and resources** section, click the **More** button. In the following dialog box, select the **Drives** check box to make your local drives available.


Remote Desktop Connection
Remote Desktop Connection
General
Display
Local Resources
Programs
Experience
Advanced
Remote audio
Configure remote audio settings.
Settings...
Keyboard
Apply Windows key combinations:
Only when using the full screen
Example: ALT+TAB
Local devices and resources
Choose the devices and resources that you want to use in your remote session.
Printers
Clipboard
More...
Options
Connect
Help
Remote Desktop Connection
Remote Desktop Connection
Local devices and resources
Choose the devices and resources on this computer that you want to use in your remote session.
Smart cards
Ports
Drives
Other supported Plug and Play (PnP) devices
Which Plug and Play devices can I use in my remote session?
OK
Cancel

You can also bring up a Remote Desktop session to your EC2 instance from the AWS Toolkit for Visual Studio. If you connect from the Toolkit, the Remote Desktop session is already configured so that your local drives are available.

Connect to your EC2 instance as Administrator using the password that you retrieved from the AWS Management Console. Once you are connected, copy both the program and the AWS assembly (`AWSSDK.dll`) to the instance.



Run the Program

Run the program. Ensure that the AWS assembly is either in the same directory as the program or that you have installed it in the General Assembly Cache.

```
get-object.exe
```

The program downloads the object and writes its contents to a file called `s3Object.txt`.

AWS Asynchronous API for .NET

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 could make a request to write data to Amazon S3 or Amazon DynamoDB and then have your code continue to do other work while AWS processes the requests.

Which Services Are Supported

To determine whether the .NET client for a service supports an async API, look for methods that have a prefix of "Begin" and "End". For example, DynamoDB.BeginPutItem and DynamoDB.EndPutItem are the async methods that correspond to the synchronous method DynamoDB.PutItem

The following clients do not currently support asynchronous service request methods.

- Amazon EC2
- Amazon SNS
- Amazon SQS

Amazon CloudFront.2012.03.15 does not support asynchronous requests. However, the newer CloudFront client does.

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. Then, after some period of time, you would call the corresponding `End` method. This method retrieves the response from the service and also provides an opportunity to handle exceptions that might have occurred during the operation.

Note
It is not required that you call the `End` method. Assuming that no errors are encountered, the asynchronous operation will complete whether or not you call the `End` method.

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 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

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

Asynchronous Method

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

`AsyncCallback callback`

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

```
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 `asyncResult` parameter, that is, `asyncResult.AsyncState`.

Calling Patterns

The asynchronous `Begin` methods support any of the following call 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

There may be circumstances in which the code that calls the `Begin` method needs 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`.

End Method Syntax

In order to receive the response back from the service, you need to call the corresponding asynchronous End method. For example, if you called the AWS CloudFormation BeginCreateStack method, you would subsequently call the EndCreateStack method. The End methods take as a parameter the IAsyncResult that was returned by the previous corresponding Begin method. The End method returns the service response object for the request.

```
CreateStackResponse EndCreateStack(
  IAsyncResult asyncResult
)
```

If you call the End method before the operation has completed, you will block on the call until the operation completes. The IAsyncResult interface exposes a boolean property, IsCompleted, that you can test to determine whether the asynchronous operation has completed. If IsCompleted is true, the operation is complete, and you can call the End method to get the service response without blocking. Note that IsCompleted is set to true even if the operation fails.

It is not required that you call the End method. Assuming that no errors are encountered, the asynchronous operation will complete whether or not you call the End method. However, if an exception occurs during the asynchronous operation, you will not receive it unless you call the End method. Therefore, there are two reasons for calling the End method: to retrieve the service response and to catch any exceptions that were thrown during the asynchronous operation.

Examples

All of the following examples assume the following initialization code.

```
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, 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 that 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 completes, 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 that the following callback function has been defined.

```
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 completes, 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 that the following class and callback function have been defined.

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

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

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

```
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 completes, 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 various patterns that you can use when calling the asynchronous request methods.

```
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
  {
    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("-----------------");
        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("-----------------");
        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}", 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) {
        //
        // Code to process exception
        //
      }

      Console.WriteLine("Finished Async PutObject operation for {0}.", re
quest.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 Cli
entState { Client = client, Start = DateTime.Now } );
      Thread.Sleep( TimeSpan.FromSeconds(5) );
    }
  }
}
```

See Also

- Getting Started with the AWS SDK for .NET (p. 4)

- Programming with the AWS SDK for .NET (p. 10)

Manage ASP.NET Session State with Amazon DynamoDB

ASP.NET applications often store session-state data in memory. However, this approach doesn't scale well; once 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. However, 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.

Amazon 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

The AWS SDK for .NET includes the *AWS.Extensions.dll* which contains an ASP.NET session state provider. Also included is 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.

Configure the Session State Provider

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

1. Add references to both *AWSSDK.dll* and *AWS.Extensions.dll* to your Visual Studio ASP.NET project. These assemblies are available by installing the AWS SDK for .NET (p. 6) or you can install them using NuGet (p. 6).
2. Edit your application's *web.config* file: in the `system.web` element, replace the existing `sessionState` element with the following XML fragment:

```
<sessionState
  timeout="20"
  mode="Custom"
  customProvider="DynamoDBSessionStoreProvider">
  <providers>
    <add name="DynamoDBSessionStoreProvider"
```

```
            type="Amazon.SessionProvider.DynamoDBSessionStateStore"
            AWSAccessKey="YOUR-ACCESS-KEY"
            AWSSecretKey="YOUR-SECRET-KEY"
            />
    </providers>
</sessionState>
```

You need to provide a valid access key ID and secret key for the `AWSAccessKey` and `AWSSecretKey` attributes. These credentials are used to communicate with Amazon DynamoDB to store and retrieve the session state. If you are using the AWS SDK for .NET and are setting the access key ID and secret key in the `appSettings` section of your application's *web.config* file, you do not need to specify the access key ID and secret key in the `providers` section; the AWS .NET client code will discover the keys at run time.

If the web server is running on an Amazon EC2 instance that is 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 roles credentials. For more information, see Using IAM Roles for EC2 Instances with the AWS SDK for .NET (p. 19) and Security Considerations (p. 39).

When your application starts, it looks for an Amazon DynamoDB table called, by default, `ASP.NET_SessionState`. The table should have a string hash key, no range key, and the desired values for *`ReadCapacityUnits`* and *`WriteCapacityUnits`*.

We recommend that you create this table before first running your application. However, if you don't create the table, the extension creates 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 extension creates the table, it will use these parameters.

For more information on working with Amazon DynamoDB tables and provisioned throughput, go to the Amazon DynamoDB Developer Guide.

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

Web.config Options

Below are the configuration attributes that can be used in the `providers` section in your *web.config* file:

AWSAccessKey
: Access key ID to use. This can be set either in the `providers` section or in the `appSettings` section.

AWSSecretKey
: Secret key to use. This can be set either in the `providers` section or in the `appSettings` section.

Application
: Optional `string` attribute. The value of the `Application` attribute is used to partition the session data in the table so that the same 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`.

Region
: Required `string` attribute. The AWS region in which to use Amazon DynamoDB. For a list of available AWS regions, go to the Regions and Endpoints documentation.

ReadCapacityUnits
: Optional `int` attribute. The read capacity units to use if the extension creates the table. The default is 10.

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

CreateIfNotExist
Optional `boolean` attribute. The `CreateIfNotExist` attribute controls whether the extension 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

As a security best practice, we recommend that you run your applications with the credentials of an AWS Identity and Access Management (IAM) user. You can use either the AWS 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`, `DescribeTables`, `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 Amazon DynamoDB running in the US East (Northern Virginia) Region:

```
{
    "Version" : "2008-10-17",
    "Statement" : [
        {
            "Sid" : "1",
            "Effect" : "Allow",
            "Action" : [
                "dynamodb:DeleteItem",
                "dynamodb:DescribeTable",
                "dynamodb:GetItem",
                "dynamodb:PutItem",
                "dynamodb:UpdateItem"
            ],
            "Resource" : "arn:aws:dynamodb:us-east-1:<YOUR-AWS-ACCOUNT-
ID>:table/ASP.NET_SessionState"
        }
    ]
}
```

Tutorial: Create Amazon EC2 Instances

This topic demonstrates how to use the AWS SDK for .NET to create, start, and terminate Amazon Elastic Compute Cloud (Amazon EC2) instances.

The sample code in this topic is written in C#, but you can use the AWS SDK for .NET with any language that is compatible with the Microsoft .NET Framework. The AWS SDK for .NET installs a set of C# project templates, so the simplest way to start this project is to click the Visual Studio **File** menu's **New Project** command and create a new AWS Empty Project.

Your project must include an `App.Config` file, which serves as a container for your AWS credentials and has the following content:

```
<?xml version="1.0"?>
<configuration>
    <appSettings>
        <add key="AWSAccessKey" value="Your_AWS_Access_Key"/>
        <add key="AWSSecretKey" value="Your_AWS_Secret_Key"/>
    </appSettings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```

Note
If create your project by using an AWS SDK for .NET template, Visual Studio creates this file for you.

To add your credentials to `App.Config`, replace the `Your_AWS_Access_Key` and `Your_AWS_Secret_Key` values with your AWS access and secret keys, respectively. To learn more about your AWS credentials, including where to find them, go to About AWS Security Credentials.

The following sections walk you through the basic procedure for creating, launching, and terminating an EC2 instance by using the AWS SDK for .NET.

Topics

Create an EC2 Client

The first step in setting up an Amazon EC2 instance is to create an *Amazon EC2 client*, which represents a set of EC2 instances and is used to configure, start, and terminate them. The client is represented by an AmazonEC2Client object, which you create as follows:

```
var ec2Client = new AmazonEC2Client(RegionEndpoint.USWest2);
```

To specify the service endpoint, pass the appropriate RegionEndpoint value to the constructor. If you omit the endpoint, the instance is created in the default region. For a list of Amazon EC2 service endpoints, see Regions and Endpoints.

Specify an EC2 Security Group

An Amazon EC2 *security group* controls which network traffic can flow to and from your Amazon EC2 instances, much like a firewall. For example, you can configure a security group to authorize inbound traffic from only a specified range of IP addresses, or perhaps only a single address. An AWS account can have up to 500 security groups, each of which is represented by a user-defined name.

By default, Amazon EC2 associates your instances with a security group that allows no inbound traffic, which means that you cannot communicate with them. To authorize your EC2 instances to accept inbound traffic you must explicitly associate them with a security group that authorizes *ingress*. For more information about security groups, go to Security Group Concepts.

If your account already has an appropriately configured security group, you can associate it with your instances, as follows:

To select an existing security group

1. Create and initialize a DescribeSecurityGroupsRequest object.

   ```
   var secGroupRequest = new DescribeSecurityGroupsRequest();
   secGroupRequest.WithGroupName("GroupName");
   ```

   The DescribeSecurityGroupsRequest object characterizes the request. This example sets the object's WithGroupName property to the name of the desired group.
2. Pass the request object to the Amazon EC2 client's DescribeSecurityGroups method, which returns a SecurityGroupResponse object.

```
DescribeSecurityGroupsResponse secGroupResponse = ec2Client.DescribeSecur
ityGroups(secGroupRequest);
SecurityGroup secGroup = secGroupResponse.DescribeSecurityGroupsResult.Se
curityGroup[0];
```

The object's DescribeSecurityGroupsResult.SecurityGroup property contains a list of the requested security groups, each of which is represented by a SecurityGroup object. This example requests a particular group, so the list has only one member.

Important
If the specified group name does not correspond to one of your account's security groups, `DescribeSecurityGroups` throws an exception.

A more robust approach is to enumerate your account's security groups, as follows

To enumerate existing security groups

1. Obtain a list of your account's security groups.

```
DescribeSecurityGroupsRequest secGroupRequest = new DescribeSecurity
GroupsRequest();
DescribeSecurityGroupsResponse secGroupResponse = ec2Client.DescribeSecur
ityGroups(secGroupRequest);
List<SecurityGroup> secGroups = secGroupResponse.DescribeSecurityGroupsRes
ult.SecurityGroup;
```

   This example but does not specify a group name, which directs DescribeSecurityGroups to return a list containing all of the account's security groups.

2. Enumerate the requested security groups and select the desired group by name.

```
SecurityGroup secGroup = null;
...
foreach(SecurityGroup item in secGroups)
{
    if (item.GroupName == "GroupName")
    {
        secGroup = item;
        break;
    }
}
```

If your account does not have a suitable security group, you can create a new one, as follows:

Important
Use this procedure only for new security groups. If you attempt to create a new security group with the same name as one of your account's existing groups, `CreateSecurityGroup` throws an exception.

To create a new Amazon EC2 security group

1. Create and initialize a CreateSecurityGroupRequest object.

```
var newGroupRequest = new CreateSecurityGroupRequest()
{
  GroupName = “YourSecurityGroupName”,
  GroupDescription = “YourSecurityGroupDescription”
};
```

Assign the group's name and description to the object's GroupName and GroupDescription properties, respectively. The two strings must contain only US-ASCII characters and the group name must be unique within the AWS region in which you initialized your Amazon EC2 client.

2. Create the new security group.

```
CreateSecurityGroupResponse newGroupResponse = ec2Client.CreateSecurity
Group(newGroupRequest);
```

Pass the request object to the EC2 client's CreateSecurityGroup method, which returns a CreateSecurityGroupResponse object.

3. Optionally, you can obtain the `SecurityGroup` object for the group that you just created by using the To select an existing security group (p. 41) procedure described earlier.

Authorize Security Group Ingress

By default, EC2 instances have no constraints on their outbound traffic but accept no inbound traffic. To receive inbound traffic, an instance must be associated with a security group that explicitly authorizes ingress. You can configure the ingress authorization to limit inbound traffic to individual IP addresses, ranges of IP addresses, specific protocols, and specific TCP/UDP ports.

You authorize ingress for a new security group, as follows.

To authorize security group ingress for a new security group

1. Create and initialize an IpPermissionSpecification object.

```
var ipPermission = new IpPermissionSpecification()
{
  IpProtocol = "tcp",
  FromPort = 3389,
  ToPort = 3389
};
ipPermission.IpRanges.Add("0.0.0.0/0");
```

To initialize the object:

- Specify the IP protocol by assigning it to the IpProtocol property.
- For the TCP or UDP protocol, authorize ingress for specified ports by assigning appropriate values to the FromPort and ToPort properties, which represent the beginning and end of the port range, respectively. This example specifies a single port, 3389, which is the port that you use to communicate with a Windows EC2 instance by using the remote desktop protocol.
- Authorize ingress for particular IP addresses or address ranges by adding them to the IpRanges collection. Use the CIDR notation to represent addresses or address ranges. For convenience,

this example uses 0.0.0.0/0, which authorizes all addresses. For production use, you typically specify a more restricted range or even a single address.

Incoming packets must meet all of these specifications.

2. Create and initialize an AuthorizeSecurityGroupIngressRequest object.

```
var ingressRequest = new AuthorizeSecurityGroupIngressRequest();
ingressRequest.GroupName = secGroupName;
ingressRequest.IpPermissions.Add(ipPermission);
```

   To initialize the object:

   - Set the GroupName property to the security group name of.
   - Add the `IpPermissionSpecification` object from Step 1 to the group's IpPermissions collection.

3. Authorize ingress.

```
AuthorizeSecurityGroupIngressResponse ingressResponse = ec2Client.Author
izeSecurityGroupIngress(ingressRequest);
```

   Pass the request object to EC2 client's AuthorizeSecurityGroupIngress method, which returns an AuthorizeSecurityGroupIngressResponse object.

   To authorize ingress for additional IP address ranges, ports, or protocols, initialize a new `IpPermissionSpecification` instance and add it to the `IpPermissions` collection before calling `AuthorizeSecurityGroupIngress`.

You can also use this procedure to add IP address ranges, ports, and protocols to existing security group. Each `AuthorizeSecurityGroupIngress` call adds a *rule* to the security group up to a maximum of 100 rules. For more information about security groups, see Security Group Concepts.

Important

If you attempt to authorize ingress for an IP address range that has already been authorized, `AuthorizeSecurityGroupIngress` throws an exception. The following example shows how to enumerate a security group's authorized IP address ranges to check for existing ranges, where `secGroup` is the group's `SecurityGroup` object. For a description of how to obtain this object, see Specify an EC2 Security Group (p. 41) .

```
foreach (IpPermission ipPerm in secGroup.IpPermission)
{
  foreach (String ipAddress in ipPerm.IpRange)
  {
    if (ipAddress == "Address Range")
      //...
  }
}
```

Specify an EC2 Key Pair

Public EC2 instances do not have a default password. Instead, you log into the instance by using a public/private key pair. EC2 key pairs are distinct from your AWS account's public and private keys. For more information, see Getting an SSH Key Pair.

Each of your account's Amazon EC2 key pairs is identified by a name. If you want to use an existing key pair, you can launch your EC2 instances by using that name, as described later.

Important
Key pair names must be unique. If you attempt to create a key pair with the same name as an existing key pair, `CreateKeyPair` returns an exception. You can enumerate your account's existing key pairs to check for an existing key name as follows:

```
DescribeKeyPairsRequest keyPairDescriptionRequest = new DescribeKeyPair
sRequest();
DescribeKeyPairsResponse keyPairDescriptionResponse = ec2Client.De
scribeKeyPairs(keyPairDescriptionRequest);
List<KeyPair> keyPairs = keyPairDescriptionResponse.DescribeKeyPairsRes
ult.KeyPair;
foreach (KeyPair item in keyPairs)
{
    if (item.KeyName == keyName)
    {
        keyPair = item;
        break;
    }
}
```

You can generate a new key pair, as follows:

To create a key pair and obtain the private key

1. Create and initialize a CreateKeyPairRequest object.

```
var newKeyRequest = new CreateKeyPairRequest();
newKeyRequest.KeyName = "Key Name";
```

   Set the KeyName property to the key pair name.

2. Pass the request object to the EC2 client object's CreateKeyPair method, which returns a CreateKeyPairResponse object.

```
CreateKeyPairResponse newKeyResponse = ec2Client.CreateKeyPair(newKeyRequest);
keyPair = newKeyResponse.CreateKeyPairResult.KeyPair;
privateKey = keyPair.KeyMaterial;
```

   The response object includes a CreateKeyPairResult property that contains the new key's KeyPair object. The `KeyPair` object's KeyMaterial property contains the unencrypted PEM-encoded private key. You should store this value for later use, because this is the only time that you can retrieve the private key from `KeyMaterial`. You can obtain an existing key pair's `KeyPair` object, as described earlier in this section, but `KeyMaterial` will be empty.

Launch EC2 Instances

You can now launch a set of one or more identically configured EC2 instances, as follows:

To launch EC2 instances

1. Create and initialize a RunInstanceRequest object.

```
var runInstanceRequest = new RunInstancesRequest()
{
  ImageId = "ami-62c44d52",
  InstanceType = "m1.small",
  MinCount = 1,
  MaxCount = 1,
  KeyName = keyName
};
runInstanceRequest.SecurityGroup.Add(secGroupName);
```

   To configure the request object:

   - Specify which Amazon Machine Image (AMI) your instances will use by setting the ImageId property to an appropriate public or privately-provided image ID. For a list of public AMIs provided by Amazon, go to Amazon Machine Images.
   - Specify the instance type by assigning it to the InstanceType property. It must be compatible with the specified AMI. For more information on instance types, see Instance Families and Types.
   - Set MinCount to the minimum number of EC2 instances to be launched. If `MinCount` is larger than your authorized maximum, AWS does not launch any instances. The default maximum is 20.
   - Set MaxCount to the number of EC2 instances that you want to launch, which cannot be larger than your authorized maximum. If `MaxCount` exceeds the number of available instances, AWS launches as many as possible.
   - Set KeyName to the EC2 key name.
   - Add one or more security group names to the list of authorized security groups.

   Important
   Make sure that the specified AMI, key name, and security group names exist in the region that you specified when you created the client object.

2. Launch the instances by passing the request object to the RunInstances method, as follows:

```
RunInstancesResponse runResponse = ec2Client.RunInstances(runInstanceRequest);
```

3. You can use the returned RunInstancesResponse object to get a list of instance IDs for the new instances.

```
List<RunningInstance> instances = runResponse.RunInstancesResult.Reserva
tion.RunningInstance;
List<String> instanceIDs = new List<string>();
foreach (RunningInstance item in instances)
{
  instanceIDs.Add(item.InstanceId);
}
```

You need the instance IDs, in particular, to check status and terminate instances. This enumeration also allows you to determine how many instances were actually launched. The response object's runResponse.RunInstancesResult.Reservation.RunningInstance property contains a list of RunningInstance objects, one for each EC2 instance that you successfully launched. You can retrieve the ID for each instance from the `RunningInstance` object's InstanceId property.

After you create your Amazon EC2 instances, you can check their status programmatically.

To check EC2 status

1. Create and configure a DescribeInstancesRequest object.

```
var instancesRequest = new DescribeInstancesRequest();
instancesRequest.InstanceId = instanceIDs;
```

   To configure the object, assign a list of instance IDs to the InstanceId property. You can use the object's Filter property to limit the request to certain instances, such as instances with a particular user-specified tag.

2. Call the EC2 client's DescribeInstances method, and pass it the request object from Step 1.

```
DescribeInstancesResponse statusResponse = ec2Client.DescribeInstances(in
stancesRequest);
```

   The method returns a DescribeInstancesResponse object with the descriptions.

3. Enumerate the running instances and determine their status.

```
List<RunningInstance> runningInstances = statusResponse.DescribeInstances
Result.Reservation[0].RunningInstance;
foreach (RunningInstance instance in runningInstances)
{
  Console.WriteLine("Instance Status: " + instance.InstanceState.Name);
}
```

   The `DescribeInstancesResponse` object's DescribeInstancesResult.Reservation property contains a list of reservations. In this case, there is only one. Each reservation contains a list of RunningInstance objects, each of which represents one of the reservation's EC2 instances. You can get the instance's status from it's `RunningInstance` object's InstanceState.Name property, which can have one of the following values: "pending", "running", "shutting-down", "terminated", "stopping", "stopped".

After an instance is running, you can remotely connect to it by using an SSH/RDP client on your computer. Before connecting to your Amazon EC2 instance, you must ensure that the instance’s SSH/RDP port is open to traffic. To connect, you will need the Amazon EC2 instance ID and the private key from instance's key pair. For information on how to obtain the private key, see Create a Key Pair (p. 45). For more information on how to connect to an EC2 instance, see Connecting to Instances.

Next: Terminate EC2 Instances (p. 48)

Terminate EC2 Instances

When you no longer need one or more of your EC2 instances, you can terminate them, as follows:

To terminate EC2 instances

1. Create and initialize a TerminateInstancesRequest object.

```
var termRequest = new TerminateInstancesRequest();
termRequest.InstanceId = instanceIDs;
```

   Set the object's InstanceId property to a list of the instance IDs that you want to terminate.
2. Pass the request object to the Amazon EC2 client object's TerminateInstances method.

```
TerminateInstancesResponse termResponse = ec2Client.TerminateInstances(ter
mRequest);
```

You can use the response object to list the terminated instances, as follows:

```
List<InstanceStateChange> terminatedInstances = termResponse.TerminateInstances
Result.TerminatingInstance;
foreach(InstanceStateChange item in terminatedInstances)
{
  Console.WriteLine("Terminated Instance: " + item.InstanceId);
}
```

Next: Additional EC2 Resources (p. 48)

Additional EC2 Resources

The following table lists related resources that you'll find useful when using Amazon EC2 with the AWS SDK for .NET.

Resource	Description
Windows & .NET Developer Center	Sample code, documentation, tools, and additional resources to help you build applications on Amazon Web Services.
Amazon Elastic Compute Cloud (EC2) Documentation	Amazon EC2 service documentation.

Tutorial: Amazon EC2 Spot Instances

Overview

Spot Instances allow you to bid on unused Amazon Elastic Compute Cloud (Amazon EC2) capacity and run the acquired instances for as long as your bid exceeds the current *Spot Price*. Amazon EC2 changes the Spot Price periodically based on supply and demand, and 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 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 that 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 usage.

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 that 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 that you also install the AWS Toolkit for Visual Studio. For instructions for setting up your environment, see Getting Started (p. 4).

Step 1: Setting Up Your Credentials

To begin using this code sample, you need to populate the `App.config` file with your access key and secret key. Your access keys identify you to Amazon Web Services. You should have stored your keys in a safe place when you created them. Although you can retrieve your access key ID from the Your Security Credentials page, you can't retrieve your secret access key. Therefore, if you can't find your secret access key, you'll need to create new keys before you can use this sample.

Copy and paste your access key ID and secret key into the `App.config` file.

```
<appSettings>
    <add key="AWSAccessKey" value="your access key here" />
    <add key="AWSSecretKey" value="your secret key here" />
    <add key="ClientSettingsProvider.ServiceUri" value="" />
</appSettings>
```

To learn more about setting up security credentials, see the Amazon Elastic Compute Cloud User Guide.

Now that you have configured your settings, you can get started using the code in the example.

Step 2: Setting Up a Security Group

A *security group* acts as a firewall that controls the traffic allowed in and out of a group of instances. By default, an instance is started without any security group, which means that all incoming IP traffic, on any TCP port will be denied. So, before submitting your Spot Request, you will set up a security group that allows the necessary network traffic. For the purposes of this tutorial, we will create a new security group called "GettingStarted" that allows connection using the Windows Remote Desktop Protocol (RDP) from the IP address of the local computer, that is, the computer where you are running the application.

To set up a new security group, you need to include or run the following code sample that sets up the security group programmatically. You only need to run this code once to create the new security group. However, the code is designed so that it is safe to run even if the security group already exists. In this case, the code catches and ignores the "InvalidGroup.Duplicate" exception.

In the code below, we first use **AWSClientFactoryClass** to create an **AmazonEC2** client object. We then create a `CreateSecurityGroupRequest` object with the name, "GettingStarted" and a description for the security group. Finally, we call the `ec2.createSecurityGroup` API to create the group.

```
AmazonEC2 ec2 = AWSClientFactory.CreateAmazonEC2Client();

try
{
    CreateSecurityGroupRequest securityGroupRequest = new CreateSecurityGroupRequest();
    securityGroupRequest.GroupName = "GettingStartedGroup";
   securityGroupRequest.GroupDescription = "Getting Started Security Group";

    ec2.CreateSecurityGroup(securityGroupRequest);
}
catch (AmazonEC2Exception ae)
{
   if (string.Equals(ae.ErrorCode, "InvalidGroup.Duplicate", StringComparison.InvariantCulture))
    {
        Console.WriteLine(ae.Message);
    }
    else
    {
        throw;
    }
}
```

To enable access to the group, we create an `ipPermission` object with the IP address set to the CIDR representation of the IP address of the local computer. The "/32" suffix on the IP address indicates that the security group should accept traffic *only* from the local computer. We also configure the `ipPermission` object with the TCP protocol and port 3389 (RDP). You will need to fill in the IP address of the local computer. If your connection to the Internet is mediated by a firewall or some other type of proxy, you will need to determine the external IP address that is used by the proxy. One technique is to query a search engine such as Google or Bing with the string: "what is my IP address".

```

// TODO - Change the code below to use your external IP address.
String ipSource = "XXX.XXX.XXX.XX/32";

List<String> ipRanges = new List<String>();
ipRanges.Add(ipSource);

List<IpPermissionSpecification> ipPermissions = new List<IpPermissionSpecification>();
IpPermissionSpecification ipPermission = new IpPermissionSpecification();
ipPermission.IpProtocol = "tcp";
ipPermission.FromPort = 3389;
ipPermission.ToPort = 3389;
ipPermission.IpRanges = ipRanges;
ipPermissions.Add(ipPermission);
```

The final step is to call `ec2.authorizeSecurityGroupIngress` with the name of our security group and the `ipPermission` object.

```
try {
       // Authorize the ports to be used.
       AuthorizeSecurityGroupIngressRequest ingressRequest = new AuthorizeSecurityGroupIngressRequest();
       ingressRequest.IpPermissions = ipPermissions;
       ingressRequest.GroupName = "GettingStartedGroup";
       ec2.AuthorizeSecurityGroupIngress(ingressRequest);
   } catch (AmazonEC2Exception ae) {
       if (String.Equals(ae.ErrorCode, "InvalidPermission.Duplicate", StringComparison.InvariantCulture))
       {
           Console.WriteLine(ae.Message);
       }
       else
       {
           throw;
       }
   }
```

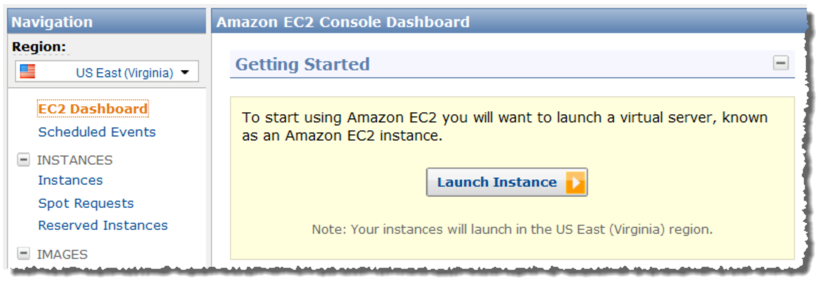
You can also create the security group using the AWS Toolkit for Visual Studio. Go to the toolkit documentation for more information.

Step 3: Submitting Your Spot Request

To submit a Spot request, you first need to determine the instance type, Amazon Machine Image (AMI), and maximum bid price you want to use. You must also include the security group we configured previously, so that you can log into the instance if you want to.

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, the least expensive instance type available. Next, we will determine the type of AMI we would like to use. We'll use ami-fbf93092, the most up-to-date Windows AMI available when we wrote this tutorial. The latest AMI may change over time, but you can always determine the latest version AMI by doing the following:

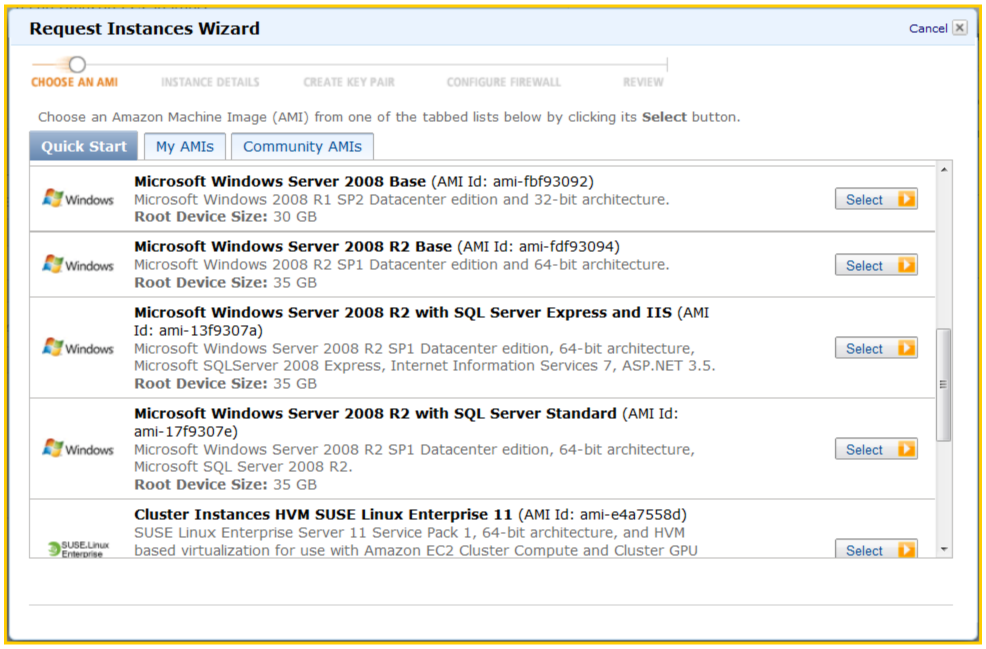
1. Log into the AWS Management Console, open the Amazon EC2 console, and from the **Amazon EC2 Console Dashbord**, attempt to launch an instance.



To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Note: Your instances will launch in the US East (Virginia) region.

2. In the window that displays AMIs, just use the AMI ID as shown in the following image. Alternatively, you can use the `DescribeImages` API, but leveraging that command is outside the scope of this tutorial.



Choose an Amazon Machine Image (AMI) from one of the tabbed lists below by clicking its **Select** button.

Microsoft Windows Server 2008 Base (AMI Id: ami-fbf93092)
Microsoft Windows 2008 R1 SP2 Datacenter edition and 32-bit architecture.
Root Device Size: 30 GB

Microsoft Windows Server 2008 R2 Base (AMI Id: ami-fdf93094)
Microsoft Windows 2008 R2 SP1 Datacenter edition and 64-bit architecture.
Root Device Size: 35 GB

Microsoft Windows Server 2008 R2 with SQL Server Express and IIS (AMI Id: ami-13f9307a)
Microsoft Windows Server 2008 R2 SP1 Datacenter edition, 64-bit architecture, Microsoft SQLServer 2008 Express, Internet Information Services 7, ASP.NET 3.5.
Root Device Size: 35 GB

Microsoft Windows Server 2008 R2 with SQL Server Standard (AMI Id: ami-17f9307e)
Microsoft Windows Server 2008 R2 SP1 Datacenter edition, 64-bit architecture, Microsoft SQL Server 2008 R2.
Root Device Size: 35 GB

Cluster Instances HVM SUSE Linux Enterprise 11 (AMI Id: ami-e4a7558d)
SUSE Linux Enterprise Server 11 Service Pack 1, 64-bit architecture, and HVM based virtualization for use with Amazon EC2 Cluster Compute and Cluster GPU

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 when it completes. You want to see if you can complete it for less cost than with 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 provide a high likelihood that your request will be fulfilled quickly and continue computing until it completes.

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 that 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 simply need to build your request with the parameters we have specified so far. We start by creating a `RequestSpotInstanceRequest` object. The request object requires the number of instances you want to start (2) and the bid price ($0.03). Additionally, you need to set the `LaunchSpecification` for the request, which includes the instance type, AMI ID, and security group you want to use. Once the request is populated, you call the `requestSpotInstances` method on the `AmazonEC2Client` object. An example of how to request a Spot Instance is shown below.

```
RequestSpotInstancesRequest requestRequest = new RequestSpotInstances
Request();

requestRequest.SpotPrice = "0.03";
requestRequest.InstanceCount = 2;

LaunchSpecification launchSpecification = new LaunchSpecification();
launchSpecification.ImageId = "ami-fbf93092";   // latest Windows AMI as
of this writing
launchSpecification.InstanceType = "t1.micro";

launchSpecification.SecurityGroup.Add("GettingStartedGroup");

requestRequest.LaunchSpecification = launchSpecification;

RequestSpotInstancesResponse requestResult = ec2.RequestSpotInstances(re
questRequest);
```

There are other options you can use to configure your Spot Requests. To learn more, see RequestSpotInstances in the AWS SDK for .NET.

Running this code will launch a new Spot Instance Request.

Note
You will be charged for any Spot Instances that are actually launched, so make sure that you cancel any requests and terminate any instances you launch to reduce any associated fees.

Step 4: Determining the State of Your Spot Request

Next, we want to create code to wait until the Spot request reaches the "active" state before proceeding to the last step. To determine the state of our Spot request, we poll the describeSpotInstanceRequests method for the state of the Spot request ID we want to monitor.

The request ID created in Step 2 is embedded in the result of our `requestSpotInstances` request. The following example code gathers request IDs from the `requestSpotInstances` result and uses them to populate the `SpotInstanceRequestId` member of a `describeRequest` object. We will use this object in the next part of the sample.

```
// Call the RequestSpotInstance API.
RequestSpotInstancesResponse requestResult = ec2.RequestSpotInstances(re
questRequest);

// Create the describeRequest object with all of the request ids
// to monitor (e.g. that we started).
DescribeSpotInstanceRequestsRequest describeRequest = new DescribeSpotIn
stanceRequestsRequest();
foreach (SpotInstanceRequest spotInstanceRequest in requestResult.Request
SpotInstancesResult.SpotInstanceRequest)
{
    describeRequest.SpotInstanceRequestId.Add(spotInstanceRequest.SpotIn
stanceRequestId);
}
```

```
// Create a variable that will track whether there are any
// requests still in the open state.
bool anyOpen;

// Create a list to store any instances that were activated.
List<String> instanceIds = new List<String>();

do
{
    // Initialize the anyOpen variable to false, which assumes there
    // are no requests open unless we find one that is still open.
    anyOpen = false;
    instanceIds.Clear();

    try
    {
        // Retrieve all of the requests we want to monitor.
        DescribeSpotInstanceRequestsResponse describeResponse = ec2.De
scribeSpotInstanceRequests(describeRequest);

        // Look through each request and determine if they are all in
        // the active state.
        foreach (SpotInstanceRequest spotInstanceRequest in de
scribeResponse.DescribeSpotInstanceRequestsResult.SpotInstanceRequest)
        {
            // If the state is open, it hasn't changed since we attempted
            // to request it. There is the potential for it to transition
            // almost immediately to closed or canceled, so we compare
            // against open instead of active.
            if (spotInstanceRequest.State.Equals("open", StringComparis
on.InvariantCulture))
            {
                anyOpen = true;
                break;
            }
           else if (spotInstanceRequest.State.Equals("active", StringCom
parison.InvariantCulture))
            {
                // Add the instance id to the list we will
                // eventually terminate.
                instanceIds.Add(spotInstanceRequest.InstanceId);
            }
        }
    }
    catch (AmazonEC2Exception e)
    {
        // If we have an exception, ensure we don't break out of
```

```
            // the loop. This prevents the scenario where there was
            // blip on the wire.
            anyOpen = true;

            Console.WriteLine(e.Message);
        }

        if (anyOpen)
        {
            // Wait for the requests to go active.
            Console.WriteLine("Requests still in open state, will retry in 60
seconds.");
            Thread.Sleep((int)TimeSpan.FromMinutes(1).TotalMilliseconds);
        }
    } while (anyOpen);
```

If you just ran the code up to this point, your Spot Instance Request would complete—or possibly fail with an error. For the purposes of this tutorial, we'll add some code that cleans up the requests after all of them have transitioned out of the open state.

Step 5: Cleaning up Your Spot Requests and Instances

The final step is to clean up our 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 pay 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.

The following code demonstrates how to cancel your requests.

```
    try
    {
        // Cancel requests.
       CancelSpotInstanceRequestsRequest cancelRequest = new CancelSpotInstan
ceRequestsRequest();

        foreach (SpotInstanceRequest spotInstanceRequest in requestResult.Re
questSpotInstancesResult.SpotInstanceRequest)
        {
           cancelRequest.SpotInstanceRequestId.Add(spotInstanceRequest.SpotIn
stanceRequestId);
        }

        ec2.CancelSpotInstanceRequests(cancelRequest);
    }
    catch (AmazonEC2Exception e)
    {
        // Write out any exceptions that may have occurred.
        Console.WriteLine("Error cancelling instances");
        Console.WriteLine("Caught Exception: " + e.Message);
        Console.WriteLine("Reponse Status Code: " + e.StatusCode);
        Console.WriteLine("Error Code: " + e.ErrorCode);
        Console.WriteLine("Request ID: " + e.RequestId);
    }
    }
```

To terminate any outstanding instances, we use the instanceIds array, which we populated with the instance IDs of those instances that transitioned to the active state. We terminate these instances by assigning this array to the `InstanceId` member of a `TerminateInstancesRequest` object, then passing that object to the `ec2.TerminateInstances` API.

```
    if (instanceIds.Count > 0)
    {
        try
        {
          TerminateInstancesRequest terminateRequest = new TerminateInstances
Request();
            terminateRequest.InstanceId = instanceIds;

            ec2.TerminateInstances(terminateRequest);
        }
        catch (AmazonEC2Exception e)
        {
            Console.WriteLine("Error terminating instances");
            Console.WriteLine("Caught Exception: " + e.Message);
            Console.WriteLine("Reponse Status Code: " + e.StatusCode);
            Console.WriteLine("Error Code: " + e.ErrorCode);
            Console.WriteLine("Request ID: " + e.RequestId);
        }
    }
```

Conclusion

Congratulations! You have just completed the getting started tutorial for developing Spot Instance software with the AWS SDK for .NET.

Create and Use an Amazon SQS Queue

This topic demonstrates how to use the AWS SDK for .NET to create and use an Amazon Simple Queue Service (Amazon SQS) queue.

The sample code in this topic is written in C#, but you can use the AWS SDK for .NET with any language that is compatible with the Microsoft .NET Framework.

Topics

Create an Amazon SQS Client

You will 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 store your AWS access key and your secret key.

The file looks like this:

```
<?xml version="1.0"?>
<configuration>
    <appSettings>
        <add key="AWSAccessKey" value="Your_AWS_Access_Key"/>
        <add key="AWSSecretKey" value="Your_AWS_Secret_Key"/>
    </appSettings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
</configuration>
```

Specify your AWS credentials as values for the `AWSAccessKey` and `AWSSecretKey` entries. For more information about getting credentials, see How Do I Get Security credentials? in the *AWS General Reference*.

After you create this file, you are ready to create and initialize your Amazon SQS client.

To create and initialize an Amazon SQS client

1. Create and initialize an AmazonSQSConfig instance, and set the ServiceURL property with the protocol and service endpoint, as follows:

   ```
   AmazonSQSConfig amazonSQSConfig =
       new AmazonSQSConfig();

   amazonSQSConfig.ServiceURL =
       "http://sqs.us-west-2.amazonaws.com";
   ```

   The AWS SDK for .NET uses US East (Northern Virginia) Region as the default region if you do not specify a region in your code. However, the AWS Management Console uses US West (Oregon) Region as its default. Therefore, when using the AWS Management Console in conjunction with your development, be sure to specify the same region in both your code and the console.

   Go to Regions and Endpoints for the current list of regions and corresponding endpoints for each of the services offered by AWS.
2. Use the `AmazonSQSConfig` instance to create and initialize an AmazonSQSClient instance, as follows:

   ```
   amazonSQSClient =
   new AmazonSQSClient(amazonSQSConfig);
   ```

You can now use the client to create an Amazon SQS queue. For information about creating a queue, see Create an Amazon SQS Queue (p. 59).

Create an Amazon SQS Queue

You can use the AWS SDK for .NET to programmatically create an Amazon SQS queue. Creating an Amazon SQS Queue is an administrative task. You can create a queue by using the AWS Management Console instead of creating a queue programmatically.

To create an Amazon SQS queue

1. Create and initialize a CreateQueueRequest instance. Provide the name of your queue and specify a visibility timeout for your queue messages, as follows:

   ```

   CreateQueueRequest createQueueRequest =
       new CreateQueueRequest();
   ```

```
createQueueRequest.QueueName = "MySQSQueue";
createQueueRequest.DefaultVisibilityTimeout = 10;
```

Your queue name must only be composed of 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, go to Visibility Timeout. For more information about different queue attributes you can set, go to setQueueAttributes.

2. After you create the request, pass it as a parameter to the CreateQueue method. The method returns a CreateQueueResponse object, as follows:

```
CreateQueueResponse createQueueResponse =
    amazonSQSClient.CreateQueue(createQueueRequest);
```

For information about how queues work in Amazon SQS, go to How SQS Queues Work.

For information about your queue URL, see Amazon SQS Queue URLs (p. 60).

Amazon SQS Queue URLs

You require the queue URL to send, receive, and delete queue messages. A queue URL is constructed in the following format:

```
https://queue.amazonaws.com/YOUR_ACCOUNT_NUMBER/YOUR_QUEUE_NAME
```

To find your AWS account number, go to Security Credentials . Your account number is located under **Account Number** in the upper right of the page.

For information on sending a message to a queue, see Send an Amazon SQS Message (p. 60).

For information about receiving messages from a queue, see Receive a Message from an Amazon SQS Queue (p. 61).

For information about deleting messages from a queue, see Delete a Message from an Amazon SQS Queue (p. 62).

Send an Amazon SQS Message

You can use the Amazon SDK for .NET to send a message to an Amazon SQS queue.

Important
Due to the distributed nature of the queue, Amazon SQS cannot guarantee you will receive messages in the exact order they are sent. If you require that message order be preserved, place sequencing information in each message so you can reorder the messages upon receipt.

To send a message to an Amazon SQS queue

1. Create and initialize a SendMessageRequest instance. Specify the queue name and the message you want to send, as follows:

   ```
   sendMessageRequest.QueueUrl = myQueueURL;
   sendMessageRequest.MessageBody = "YOUR_QUEUE_MESSAGE";
   ```

   For more information about your queue URL, see Amazon SQS Queue URLs (p. 60).

   Each queue message must be composed of only Unicode characters, and can be up to 64 kB in size. For more information about queue messages, go to SendMessage in the Amazon SQS service 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:

   ```
   SendMessageResponse sendMessageResponse =
       amazonSQSClient.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.

For information on deleting messages from your queue, see Delete a Message from an Amazon SQS Queue (p. 62).

For information on receiving messages from your queue, see Receive a Message from an Amazon SQS Queue (p. 61).

Receive a Message from an Amazon SQS Queue

You can use the Amazon SDK for .NET to receive messages from an Amazon SQS queue.

To receive a message from an Amazon SQS queue

1. Create and initialize a ReceiveMessageRequest instance. Specify the queue URL to receive a message from, as follows:

   ```
   ReceiveMessageRequest recieveMessageRequest =
       new ReceiveMessageRequest();

   recieveMessageRequest.QueueUrl = myQueueURL;
   ```

   For more information about your queue URL, see Amazon SQS Queue URLs (p. 60).
2. Pass the request object as a parameter to the ReceiveMessages method, as follows:

   ```
   ReceiveMessageResponse receiveMessageResponse =
       amazonSQSClient.ReceiveMessage(receiveMessageRequest);
   ```

The method returns a ReceiveMessageResponse instance, containing the list of messages the queue contains.

3. The response object contains a ReceiveMessageResult member. This member includes a Message list. Iterate through this list to find a specific message, and use the Body property to determine if the list contains a specified message, as follows:

```
if (result.Message.Count != 0)
{
    for (int i = 0; i < result.Message.Count; i++)
    {
        if (result.Message[i].Body == messageBody)
        {
            recieptHandle =
                result.Message[i].ReceiptHandle;
        }
    }
}
```

Once the message is found in the list, use the ReceiptHandle property to obtain a receipt handle for the message. You can use this receipt handle to change message visibility timeout or to delete the message from the queue. For more information about how to change the visibility timeout for a message, go to ChangeMessageVisibility.

For information about sending a message to your queue, see Send an Amazon SQS Message (p. 60).

For more information about deleting a message from the queue, see Delete a Message from an Amazon SQS Queue (p. 62).

Delete a Message from an Amazon SQS Queue

You can use the Amazon SDK for .NET to receive messages from an Amazon SQS queue.

To delete a message from an Amazon SQS queue

1. Create and initialize a DeleteQueueRequest instance. Specify the Amazon SQS queue to delete a message from and the receipt handle of the message to delete, as follows:

```
DeleteMessageRequest deleteMessageRequest =
    new DeleteMessageRequest();

deleteMessageRequest.QueueUrl = queueUrl;
deleteMessageRequest.ReceiptHandle = recieptHandle;
```

2. Pass the request object as a parameter to the DeleteMessage method. The method returns a DeleteMessageResponse object, as follows:

```
DeleteMessageResponse response =
    amazonSQSClient.DeleteMessage(deleteMessageRequest);
```

Calling `DeleteMessage` unconditionally removes the message from the queue, regardless of the visibility timeout setting. For more information about visibility timeouts, go to Visibility Timeout .

For information about sending a message to a queue, see Send an Amazon SQS Message (p. 60).

For information about receiving messages from a queue, see Receive a Message from an Amazon SQS Queue (p. 61).

Related Resources

The following table lists related resources that you'll find useful when using Amazon SQS with the AWS SDK for .NET.

Resource	Description
Windows & .NET Developer Center	Provides sample code, documentation, tools, and additional resources to help you build applications on Amazon Web Services.
AWS SDK for .NET Documentation	Provides documentation for the AWS SDK for .NET.
Amazon Simple Queue Service (SQS) Documentation	Provides documentation for the Amazon SQS service.

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/sdkfornet.

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. You can find it at http://docs.aws.amazon.com/sdkfornet/latest/apidocs/Index.html.

AWS Forums

Visit the AWS forums to ask questions or provide feedback about AWS. There is a forum specifically for AWS development in .NET as well as forums for individual services such as Amazon S3. AWS engineers monitor the forums and respond to questions, feedback, and issues. You can also subscribe 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 AWS Toolkit for Visual Studio and the accompanying User Guide.