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

Using the AWS Toolkit for Visual Studio ................................................................................................................ 1
  Toolkit for Visual Studio ........................................................................................................................................ 1
    AWS Explorer .............................................................................................................................................. 1
    Credential and Region Management ................................................................. 1
    Amazon EC2 ............................................................................................................................................... 1
    AWS Lambda ........................................................................................................................................... 1
    AWS CodeCommit ................................................................................................................................. 1
    Amazon DynamoDB ............................................................................................................................... 2
    Amazon S3 ............................................................................................................................................ 2
    Amazon RDS .......................................................................................................................................... 2
    AWS Elastic Beanstalk ......................................................................................................................... 2
    AWS CloudFormation .......................................................................................................................... 2
    AWS Identity and Access Management (IAM) ................................................................. 2

Setting Up the AWS Toolkit for Visual Studio ............................................................................. 3

Working with AWS Services ......................................................................................................... 9
  Managing Amazon EC2 Instances ............................................................................................................. 9
    The Amazon Machine Images and Amazon EC2 Instances Views .............................................. 9
    Launching an Amazon EC2 Instance ................................................................................................. 11
    Connecting to an Amazon EC2 Instance ........................................................................................... 14
    Ending an Amazon EC2 Instance .................................................................................................... 17
  Managing Amazon ECS Instances ............................................................................................... 19
    Modifying service properties ........................................................................................................... 19
    Stopping a task .................................................................................................................................. 20
    Deleting a service ............................................................................................................................. 19
    Deleting a cluster ............................................................................................................................. 20
    Creating a repository ......................................................................................................................... 20
    Deleting a repository ......................................................................................................................... 20
  Managing Security Groups from AWS Explorer ............................................................................. 21
    Creating a Security Group .................................................................................................................. 21
    Adding Permissions to Security Groups ......................................................................................... 21
  Create an AMI from an Amazon EC2 Instance ............................................................................... 23

Setting Launch Permissions on an Amazon Machine Image ........................................................... 24

Amazon Virtual Private Cloud (VPC) ........................................................................................................... 25

Deployment Using the AWS Toolkit ........................................................................................................ 29
  Deploying to Elastic Beanstalk ......................................................................................................... 30
  Deploying to Amazon EC2 Container Service ................................................................................. 69
  Standalone Deployment Tool ............................................................................................................ 80

Using the AWS CloudFormation Template Editor for Visual Studio .................................................. 92
  Creating an AWS CloudFormation Template Project in Visual Studio ......................................... 93
  Deploying a AWS CloudFormation Template in Visual Studio ...................................................... 95
  Estimating the Cost of Your AWS CloudFormation Template Project in Visual Studio ............... 97
  Formatting a AWS CloudFormation Template in Visual Studio ................................................... 99

Using Amazon S3 from AWS Explorer ............................................................................................. 100
# AWS Toolkit for Visual Studio User Guide

<table>
<thead>
<tr>
<th>Creating an Amazon S3 Bucket</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managing Amazon S3 Buckets from AWS Explorer</td>
<td>101</td>
</tr>
<tr>
<td>Uploading Files and Folders to Amazon S3</td>
<td>102</td>
</tr>
<tr>
<td>Amazon S3 File Operations from AWS Toolkit for Visual Studio</td>
<td>103</td>
</tr>
<tr>
<td>Using DynamoDB from AWS Explorer</td>
<td>107</td>
</tr>
<tr>
<td>Creating an DynamoDB Table</td>
<td>108</td>
</tr>
<tr>
<td>Viewing an DynamoDB Table as a Grid</td>
<td>109</td>
</tr>
<tr>
<td>Editing and Adding Attributes and Values</td>
<td>110</td>
</tr>
<tr>
<td>Scanning an DynamoDB Table</td>
<td>111</td>
</tr>
<tr>
<td>Using AWS CodeCommit with Visual Studio Team Explorer</td>
<td>113</td>
</tr>
<tr>
<td>Credential Types for AWS CodeCommit</td>
<td>113</td>
</tr>
<tr>
<td>Connecting to AWS CodeCommit</td>
<td>113</td>
</tr>
<tr>
<td>Creating a Repository</td>
<td>114</td>
</tr>
<tr>
<td>Setting up Git Credentials</td>
<td>115</td>
</tr>
<tr>
<td>Cloning a Repository</td>
<td>118</td>
</tr>
<tr>
<td>Working with Repositories</td>
<td>118</td>
</tr>
<tr>
<td>Amazon RDS from AWS Explorer</td>
<td>119</td>
</tr>
<tr>
<td>Launch an Amazon RDS Database Instance</td>
<td>119</td>
</tr>
<tr>
<td>Create a Microsoft SQL Server Database in an RDS Instance</td>
<td>125</td>
</tr>
<tr>
<td>Amazon RDS Security Groups</td>
<td>127</td>
</tr>
<tr>
<td>Using Amazon SimpleDB from AWS Explorer</td>
<td>130</td>
</tr>
<tr>
<td>Using Amazon SQS from AWS Explorer</td>
<td>131</td>
</tr>
<tr>
<td>Creating a Queue</td>
<td>132</td>
</tr>
<tr>
<td>Deleting a Queue</td>
<td>132</td>
</tr>
<tr>
<td>Managing Queue Properties</td>
<td>132</td>
</tr>
<tr>
<td>Sending a Message to a Queue</td>
<td>133</td>
</tr>
<tr>
<td>Identity and Access Management</td>
<td>134</td>
</tr>
<tr>
<td>Create and Configure an IAM User</td>
<td>134</td>
</tr>
<tr>
<td>Create an IAM Group</td>
<td>135</td>
</tr>
<tr>
<td>Add an IAM User to an IAM Group</td>
<td>136</td>
</tr>
<tr>
<td>Generate Credentials for an IAM User</td>
<td>137</td>
</tr>
<tr>
<td>Create an IAM Role</td>
<td>139</td>
</tr>
<tr>
<td>Create an IAM Policy</td>
<td>140</td>
</tr>
<tr>
<td>Using the AWS Lambda Templates in the AWS Toolkit for Visual Studio</td>
<td>142</td>
</tr>
<tr>
<td>Tutorial: Using the AWS Lambda Project in the AWS Toolkit for Visual Studio</td>
<td>142</td>
</tr>
<tr>
<td>Tutorial: Build and Test a Serverless Application with AWS Lambda</td>
<td>151</td>
</tr>
<tr>
<td>Tutorial: Creating an Amazon Rekognition Lambda Application</td>
<td>160</td>
</tr>
<tr>
<td>Tutorial: Using Amazon Logging Frameworks with AWS Lambda to Create Application Logs</td>
<td>167</td>
</tr>
<tr>
<td>Deploying an AWS Lambda Project with the .NET Core CLI</td>
<td>168</td>
</tr>
<tr>
<td>Listing the Lambda Commands Available through the CLI</td>
<td>169</td>
</tr>
<tr>
<td>Publishing a .NET Core Lambda Project from the .NET Core CLI</td>
<td>169</td>
</tr>
</tbody>
</table>

# Document History

- Earlier Updates | 172

# Earlier Updates

- Document History | 172

- Document History | 173
Using the AWS Toolkit for Visual Studio

Toolkit for Visual Studio

The AWS Toolkit for Visual Studio is a plugin for the Visual Studio IDE that makes it easier for you to develop, debug, and deploy .NET applications that use Amazon Web Services. The Toolkit for Visual Studio is supported for Visual Studio versions 2013 and later. For details about how to download and install the kit, see Install the Toolkit for Visual Studio (p. 3).

Note
The Toolkit for Visual Studio is also available for Visual Studio 2008, 2010, and 2012 versions. However, those versions are not supported. For more information, see Install the Toolkit for Visual Studio (p. 3).

The Toolkit for Visual Studio contains the following features to enhance your development experience.

AWS Explorer

The AWS Explorer tool window, available from the IDE's View menu, enables you to interact with many of the AWS services from inside the Visual Studio IDE. Supported data services include Amazon Simple Storage Service (Amazon S3), Amazon SimpleDB, Amazon Simple Notification Service (Amazon SNS), Amazon Simple Queue Service (Amazon SQS), and Amazon CloudFront. AWS Explorer also provides access to Amazon Elastic Compute Cloud (Amazon EC2) management, AWS Identity and Access Management (IAM) user and policy management, deployment of serverless applications and functions to AWS Lambda and deployment of web applications to AWS Elastic Beanstalk and AWS CloudFormation.

Credential and Region Management

AWS Explorer supports multiple AWS accounts (including IAM user accounts) and regions, and enables you to easily change the displayed view from one account to another or view and manage resources and services in different regions.

Amazon EC2

From AWS Explorer, you can view available Amazon Machine Images (AMIs), create Amazon EC2 instances from those AMIs, and then connect to those instances by using Windows Remote Desktop. AWS Explorer also enables supporting functionality, such as the capability to create and manage key pairs and security groups.

AWS Lambda

You can use Lambda to host your serverless .NET Core C# functions and serverless applications. Use blueprints to quickly create new serverless projects and get a head start in developing your serverless application.

AWS CodeCommit

CodeCommit is integrated with Visual Studio Team Explorer. This makes it easy to clone and create repositories held in CodeCommit, and to work with source code changes from within the IDE.
Amazon DynamoDB

DynamoDB is a fast, highly scalable, highly available, cost-effective, nonrelational database service. The Toolkit for Visual Studio provides functionality for working with Amazon DynamoDB in a development context. With the Toolkit for Visual Studio, you can create and edit attributes in DynamoDB tables and run scan operations on tables.

Amazon S3

You can quickly and easily upload content to Amazon S3 buckets by dragging and dropping, or download content from Amazon S3. You can also set permissions, metadata, and tags conveniently on objects in buckets.

Amazon RDS

AWS Explorer can help you create and manage Amazon RDS assets in Visual Studio. Amazon RDS instances that use Microsoft SQL Server can also be added to Visual Studio's Server Explorer.

AWS Elastic Beanstalk

You can use Elastic Beanstalk to deploy your .NET web application projects to AWS. You can deploy your application to a single instance environment or to a fully load balanced, automatically scaled environment from within the IDE. You can also deploy new versions of your application quickly and conveniently without leaving Visual Studio. If your application uses SQL Server in Amazon RDS, the deployment wizard can also set up the connectivity between your application environment in Elastic Beanstalk and the database instance in Amazon RDS. The Toolkit for Visual Studio also includes the standalone command-line deployment tool. Use the deployment tool to make deployment an automatic part of your build process, or to include deployment in other scripting scenarios outside of Visual Studio.

AWS CloudFormation

You can use the Toolkit for Visual Studio to edit AWS CloudFormation JSON-format templates with support for editor IntelliSense and syntax highlighting. With a AWS CloudFormation template you describe the resources you want to instantiate to host your application. From within the IDE you then deploy the template to AWS CloudFormation. The resources described in the template are provisioned for you, freeing you to focus on developing the application's functionality.

AWS Identity and Access Management (IAM)

From AWS Explorer, you can create IAM users, roles, and policies, and attach policies to users.
Setting Up the AWS Toolkit for Visual Studio

The topics in this section will help you set up and use the Toolkit for Visual Studio.

Topics
- Setting Up the AWS Toolkit for Visual Studio (p. 3)
- Providing AWS Credentials (p. 5)
- Using the Toolkit for Visual Studio (p. 8)

Setting Up the AWS Toolkit for Visual Studio

This topic describes how to install and configure the Toolkit for Visual Studio.

Prerequisites

To install and configure the Toolkit for Visual Studio, you must:

- Have an AWS account. This account enables you to use AWS services. To get an AWS account, on the AWS home page, choose Create an AWS Account.
- Run a supported operating system: Windows 10, Windows 8, or Windows 7.
  
  We recommend that you install the latest service packs and updates for the Windows version you're using.

- Visual Studio 2013 or later (including Community editions).
  
  We recommend that you install the latest service packs and updates.

Note

The Toolkit for Visual Studio is still available if you're using Visual Studio versions 2008, 2010, and 2012 (including Express editions where available). However, it is not supported. For Express editions, the installation includes only the AWS project templates and the standalone deployment tool (p. 80). Visual Studio Express editions don't support third-party extensions, such as AWS Explorer. Find links to these older versions of the Toolkit for Visual Studio below in Older Versions of the Toolkit for Visual Studio (p. 5).

Install the Toolkit for Visual Studio

Install for Visual Studio 2017 and Visual Studio 2019


Install for Visual Studio 2013 and Visual Studio 2015

The Toolkit for Visual Studio for Visual Studio 2013 and Visual Studio 2015 are part of the AWS Tools for Windows. You can install the AWS Tools for Windows for these versions as follows.
1. Navigate to the page AWS Toolkit for Visual Studio.
2. In the Download section, choose Toolkit for Visual Studio 2013-2015 to download the installer.
3. To start the installation, run the downloaded installer and follow the instructions.

**Note**
By default, the Toolkit for Visual Studio is installed in the Program Files directory, which requires administrator privileges. To install the Toolkit for Visual Studio as a non-administrator, specify a different installation directory.

### Uninstall the Toolkit for Visual Studio

#### Uninstall for Visual Studio 2017 and Visual Studio 2019

Uninstall the Toolkit for Visual Studio from within Visual Studio by using the Extensions and Updates dialog (Visual Studio 2017) or the Manage Extensions dialog (Visual Studio 2019).


To uninstall the Toolkit for Visual Studio, you must uninstall the AWS Tools for Windows.

1. In Control Panel, open Programs and Features.

   **Note**
   To open Programs and Features directly, run appwiz.cpl from a command prompt or the Windows Run dialog.

2. Choose AWS Tools for Windows, and then choose Uninstall.

3. If prompted, choose Yes.

Uninstalling the AWS Tools for Windows doesn't remove the Samples directory. This directory is preserved in case you have modified the samples. You have to manually remove this directory.
Older Versions of the Toolkit for Visual Studio


Providing AWS Credentials

Before you can use the Toolkit for Visual Studio, you must provide one or more sets of valid AWS credentials. These credentials allow you to access your AWS resources through the Toolkit for Visual Studio. They’re also used to sign programmatic web services requests, so AWS can verify that the request comes from an authorized source.

**Important**

AWS credentials consist of an access key and a secret key. We recommend that you do not use your account’s root credentials. Instead, create one or more IAM users, and then use those credentials. For more information, see Using IAM Users and Best Practices for Managing AWS Access Keys.

The Toolkit for Visual Studio supports multiple sets of credentials from any number of accounts. Each set is referred to as a *profile*. When you add a profile to Toolkit for Visual Studio, the credentials are encrypted and stored in the SDK Credential Store. This is also used by the AWS SDK for .NET and the AWS Tools for Windows PowerShell. The SDK Credential Store is specific to your Windows user account on your machine and can’t be decrypted or used elsewhere.

In addition to the encrypted SDK Credential Store, the Toolkit for Visual Studio can also read credentials from the plain-text shared credentials file used by other AWS SDKs and the AWS CLI. To use the Toolkit for Visual Studio, at least one credential profile must be available from either the SDK Credential Store or the shared credential file.

**Note**

Credential profiles created using the Toolkit for Visual Studio are saved only to the encrypted SDK Credential Store. Multi-Factor Authentication (MFA) profiles are not supported by the Toolkit for Visual Studio.

Adding a profile to the SDK Credential Store

To add a profile to the SDK Credential Store:

2. Choose the New Account Profile icon to the right of the Profile list.
Adding a profile to the SDK Credential Store

The New Account Profile dialog box opens.

3. To create a credential profile, enter the following data into the dialog box. When you create an account in the AWS Management Console, or create an IAM user and set up credentials for the user,
you are prompted to download and save the generated credentials. You can choose **Import from csv file** to browse to the file containing the access and secret key credentials, and automatically import them into the dialog box.

**Profile Name**

   (Required) The profile's display name.

**Access Key ID**

   (Required) The access key.

**Secret Access Key**

   (Required) The secret key.

**Account Number**

   (Optional) The credential's account number. The Toolkit for Visual Studio uses the account number to construct Amazon Resource Names (ARNs).

**Account Type**

   (Required) The account type. This entry determines which regions are displayed in AWS Explorer when you specify this profile. For **Standard AWS Account**:

   - If you choose AWS GovCloud (US) Account, AWS Explorer displays only the AWS GovCloud (US) region.
   - If you choose **Amazon AWS Account - China (Beijing) Region**, AWS Explorer displays only the China (Beijing) Region.

4. To add the profile to the SDK Credential Store, choose **OK**.

After you add the first profile, you can also do the following:

- To add another profile, repeat the procedure.
- To delete a profile, choose it, and then choose the **Delete Profile** icon.
- To edit a profile, choose the **Edit Profile** icon to open the **Edit Profile** dialog box.

For example, if you have **rotated an IAM user's credentials**—a recommended practice—you can edit the profile to update the user's credentials in the SDK Credential Store. For more information, see **IAM Credential Rotation**.

You can also add profiles to the SDK Credential Store when you create certain AWS projects. Before Visual Studio creates the project files, it displays the **AWS Access Credentials** dialog box. You can choose an existing profile from the SDK Credential Store or create one.

You can set your credentials in the AWS credentials profile file on your local system, located at `C:\Users\USERNAME\.aws\credentials` on Windows.

This file should contain lines in the following format:
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key

Substitute your own AWS credentials values for the values `your_access_key_id` and `your_secret_access_key`.

You can use a role by creating a profile for the role. The following example shows a role profile named `assumed-role` that is assumed by the default profile.

[assume-role-test]
role_arn = arn:aws:iam::123456789012:role/assumed-role
source_profile = default

In this case, the default profile is an IAM user with credentials and permission to assume a role named `assumed-role`. To access the role, you create a named profile, in this case `assume-role-test`. Instead of configuring this profile with credentials, you specify the ARN of the role and the name of the profile that has access to it.

For an EC2 instance, specify an IAM role and then give your EC2 instance access to that role. See IAM Roles for Amazon EC2 in the Amazon EC2 User Guide for Linux Instances for a detailed discussion about how this works.

Using the Toolkit for Visual Studio

Profiles and Toolkit for Visual Studio Window Binding

The AWS Explorer window is bound to a single profile and region at a time.

- Windows opened from the AWS Explorer use the current bound profile and region. Once the window is open, you can switch to another profile or region in the AWS Explorer.
- Publish and other wizards default to the profile and region of the AWS Explorer. You can change them. Any resources created by the wizard, or windows opened when the wizard closes, will continue to use the profile and region selected in the wizard.
- If you have multiple Visual Studio open, each can be bound to a different profile and region. The AWS Explorer saves the last-used profile and region. The last Visual Studio instance closed will have its values persisted.
Working with AWS Services

AWS Explorer gives you a view of, and allows you to manipulate, multiple Amazon Web Services simultaneously. This section provides information about how to access and use the AWS Explorer view in Visual Studio.

It assumes that you’ve already installed the AWS Toolkit for Visual Studio on your system.

Topics

• Managing Amazon EC2 Instances (p. 9)
• Managing Amazon ECS Instances (p. 19)
• Managing Security Groups from AWS Explorer (p. 20)
• Create an AMI from an Amazon EC2 Instance (p. 23)
• Setting Launch Permissions on an Amazon Machine Image (p. 24)
• Amazon Virtual Private Cloud (VPC) (p. 25)
• Deployment Using the AWS Toolkit (p. 29)
• Using the AWS CloudFormation Template Editor for Visual Studio (p. 92)
• Amazon S3 from AWS Explorer (p. 100)
• Using DynamoDB from AWS Explorer (p. 107)
• Using AWS CodeCommit with Visual Studio Team Explorer (p. 113)
• Amazon RDS from AWS Explorer (p. 119)
• Using Amazon SimpleDB from AWS Explorer (p. 130)
• Using Amazon SQS from AWS Explorer (p. 131)
• Identity and Access Management (p. 134)
• Using the AWS Lambda Templates in the AWS Toolkit for Visual Studio (p. 142)
• Deploying an AWS Lambda Project with the .NET Core CLI (p. 168)

Managing Amazon EC2 Instances

AWS Explorer provides detailed views of Amazon Machine Images (AMI) and Amazon Elastic Compute Cloud (Amazon EC2) instances. From these views, you can launch an Amazon EC2 instance from an AMI, connect to that instance, and either stop or terminate the instance, all from inside the Visual Studio development environment. You can use the instances view to create AMIs from your instances. For more information, see Create an AMI from an Amazon EC2 Instance (p. 23).

The Amazon Machine Images and Amazon EC2 Instances Views

From AWS Explorer, you can display views of Amazon Machine Images (AMIs) and Amazon EC2 instances. In AWS Explorer, expand the Amazon EC2 node.

To display the AMIs view, on the first subnode, AMIs, open the context (right-click) menu and then choose View.
To display the Amazon EC2 instances view, on the **Instances** node, open the context (right-click) menu and then choose **View**.

You can also display either view by double-clicking the appropriate node.

- The views are scoped to the region specified in AWS Explorer (for example, the US West (N. California) region).
- You can rearrange columns by clicking and dragging. To sort the values in a column, click the column heading.
- You can use the drop-down lists and filter box in **Viewing** to configure views. The initial view displays AMIs of any platform type (Windows or Linux) that are owned by the account specified in AWS Explorer.

**Show/Hide Columns**

You can also choose the **Show/Hide** drop-down at the top of the view to configure which columns are displayed. Your choice of columns will persist if you close the view and reopen it.

**Show/Hide Columns UI for AMI and Instances views**

**Tagging AMIs, Instances, and Volumes**

You can also use the **Show/Hide** drop-down list to add tags for AMIs, Amazon EC2 instances, or volumes you own. Tags are name-value pairs that enable you to attach metadata to your AMIs, instances, and volumes. Tag names are scoped both to your account and also separately to your AMIs and instances. For example, there would be no conflict if you used the same tag name for your AMIs and your instances. Tag names are not case-sensitive.

For more information about tags, go to **Using Tags** in the *Amazon EC2 User Guide for Linux Instances*.

**To add a tag**

1. In the **Add** box, type a name for the tag. Choose the green button with the plus sign (+), and then choose **Apply**.
Add a tag to an AMI or Amazon EC2 instance

The new tag is displayed in italic, which indicates no values have yet been associated with that tag.

In the list view, the tag name appears as a new column. When at least one value has been associated with the tag, the tag will be visible in the AWS Console.

2. To add a value for the tag, double-click a cell in the column for that tag, and type a value. To delete the tag value, double-click the cell and delete the text.

If you clear the tag in the Show/Hide drop-down list, the corresponding column disappears from the view. The tag is preserved, along with any tag values associated with AMIs, instances, or volumes.

**Note**

If you clear a tag in the Show/Hide drop-down list that has no associated values, the AWS Toolkit will delete the tag entirely. It will no longer appear in the list view or in the Show/Hide drop-down list. To use that tag again, use the Show/Hide dialog box to re-create it.

### Launching an Amazon EC2 Instance

AWS Explorer provides all of the functionality required to launch an Amazon EC2 instance. In this section, we'll select an Amazon Machine Image (AMI), configure it, and then start it as an Amazon EC2 instance.

**To launch a Windows Server Amazon EC2 instance**

1. At the top of the AMIs view, in the drop-down list on the left, choose **Amazon Images**. In the drop-down list on the right, choose **Windows**. In the filter box, type `ebs` for Elastic Block Storage. It may take a few moments for the view to be refreshed.

2. Choose an AMI in the list, open the context (right-click) menu, and then choose **Launch Instance**.
AMIs list

3. In the Launch New Amazon EC2 Instance dialog box, configure the AMI for your application.

**Instance Type**

Choose the type of the EC2 instance to launch. You can find a list of instance types and pricing information on the EC2 Pricing page.

**Name**

Type a name for your instance. This name cannot be more than 256 characters.

**Key Pair**

A key pair is used to obtain the Windows password that you use to log in to the EC2 instance using Remote Desktop Protocol (RDP). Choose a key pair for which you have access to the private key, or choose the option to create a key pair. If you create the key pair in the Toolkit, the Toolkit can store the private key for you.

Key pairs stored in the AWSToolkit are encrypted. You can find them at %LOCALAPPDATA%\AWSToolkit\keypairs (typically: C:\Users\<user>\AppData\Local\AWSToolkit\keypairs). You can export the encrypted key pair into a .pem file.

a. In Visual Studio, select View and click AWS Explorer.

b. Click on Amazon EC2 and select Key Pairs.

c. The key pairs will be listed, and those created/managed by the Toolkit marked as Stored in AWSToolkit.

d. Right click on the key pair you created and select Export Private Key. The private key will be unencrypted and stored in the location you specify.

**Security Group**

The security group controls the type of network traffic the EC2 instance will accept. Choose a security group that will allow incoming traffic on port 3389, the port used by RDP, so that you can connect to the EC2 instance. For information about how to use the Toolkit to create security groups, see Managing Security Groups from AWS Explorer (p. 20).

**Instance Profile**

The instance profile is a logical container for an IAM role. When you choose an instance profile, you associate the corresponding IAM role with the EC2 instance. IAM roles are configured with policies that specify access to AWS services and account resources. When an EC2 instance is associated with an IAM role, application software that runs on the instance runs with the
permissions specified by the IAM role. This enables the application software to run without having to specify any AWS credentials of its own, which makes the software more secure. For more information about IAM roles, go to the IAM User Guide.

**EC2 Launch AMI dialog box**

4. Choose **Launch**.

In AWS Explorer, on the **Instances** subnode of **Amazon EC2**, open the context (right-click) menu and then choose **View**. The AWS Toolkit displays the list of Amazon EC2 instances associated with the active account. You may need to choose **Refresh** to see your new instance. When the instance first appears, it may be in a pending state, but after a few moments, it transitions to a running state.
Connecting to an Amazon EC2 Instance

You can use Windows Remote Desktop to connect to a Windows Server instance. For authentication, the AWS Toolkit enables you to retrieve the administrator password for the instance, or you can simply use the stored key pair associated with the instance. In the following procedure, we'll use the stored key pair.

To connect to a Windows Server instance using Windows Remote Desktop

1. In the EC2 instance list, right-click the Windows Server instance to which you want to connect. From the context menu, choose **Open Remote Desktop**.

   If you want to authenticate using the administrator password, you would choose **Get Windows Passwords**.
EC2 Instance context menu

2. In the **Open Remote Desktop** dialog box, choose **Use EC2 keypair to log on**, and then choose **OK**.

If you did not store a key pair with the AWS Toolkit, specify the PEM file that contains the private key.

3. The **Remote Desktop** window will open. You do not need to sign in because authentication occurred with the key pair. You will be running as the administrator on the Amazon EC2 instance.
If the EC2 instance has only recently started, you may not be able to connect for two possible reasons:

- The Remote Desktop service might not yet be up and running. Wait a few minutes and try again.
- Password information might not yet have been transferred to the instance. In this case, you will see a message box similar to the following.

![Password not yet available message box](image)

Password not yet available

The following screenshot shows a user connected as administrator through Remote Desktop.

![Remote Desktop screenshot](image)

Remote Desktop
Ending an Amazon EC2 Instance

Using the AWS Toolkit, you can stop or terminate a running Amazon EC2 instance from Visual Studio. To stop the instance, the EC2 instance must be using an Amazon EBS volume. If the EC2 instance is not using an Amazon EBS volume, then your only option is to terminate the instance.

If you stop the instance, data stored on the EBS volume is retained. If you terminate the instance, all data stored on the local storage device of the instance will be lost. In either case, stop or terminate, you will not continue to be charged for the EC2 instance. However, if you stop an instance, you will continue to be charged for the EBS storage that persists after the instance is stopped.

Another possible way to end an instance is to use Remote Desktop to connect to the instance, and then from the Windows Start menu, use Shutdown. You can configure the instance to either stop or terminate in this scenario.

To stop an Amazon EC2 instance

1. In AWS Explorer, expand the Amazon EC2 node, open the context (right-click) menu for Instances, and then choose View. In the Instances list, right-click the instance you want to stop and choose Stop from the context menu. Choose Yes to confirm you want to stop the instance.

2. At the top of the Instances list, choose Refresh to see the change in the status of the Amazon EC2 instance. Because we stopped rather than terminated the instance, the EBS volume associated with the instance is still active.
Terminated Instances Remain Visible

If you terminate an instance, it will continue to appear in the **Instance** list alongside running or stopped instances. Eventually, AWS reclaims these instances and they disappear from the list. You are not charged for instances in a terminated state.

To specify the behavior of an EC2 instance at shutdown

The AWS Toolkit enables you to specify whether an Amazon EC2 instance will stop or terminate if **Shutdown** is selected from the **Start** menu.

1. In the **Instances** list, right-click an Amazon EC2 instance, and then choose **Change shutdown behavior**.

   ![Change Shutdown Behavior menu item](image)

   **Change Shutdown Behavior** menu item
Managing Amazon ECS Instances

AWS Explorer provides detailed views of Amazon Elastic Container Service (Amazon ECS) clusters and container repositories. You can create, delete and manage cluster and container details from within the Visual Studio development environment.

Modifying service properties

You can view service details, service events and service properties from the cluster view.

1. In AWS Explorer, open the context (right-click) menu for the cluster to manage, and then choose View.
2. In the ECS Cluster view, click Services on the left, and then click the Details tab in the details view. You can click Events to see event messages and Deployments to deployment status.
3. Click Edit. You can change the desired task count and the minimum and maximum healthy percent.
4. Click Save to accept changes or Cancel to revert to existing values.

Stopping a task

You can see the current status of tasks and stop one or more tasks in the cluster view.

To stop a task

1. In AWS Explorer, open the context (right-click) menu for the cluster with tasks you wish to stop, and then choose View.
2. In the ECS Cluster view, click Tasks on the left.
3. Make sure Desired Task Status is set to Running. Choose the individual tasks to stop and then click Stop or click Stop All to select and stop all running tasks.
4. In the Stop Tasks dialog box, choose Yes.

Deleting a service

You can delete services from a cluster from the cluster view.

To delete a cluster service
1. In AWS Explorer, open the context (right-click) menu for the cluster with a service you want to delete, and then choose View.
2. In the ECS Cluster view, click Services on the left, and then click Delete.
3. In the Delete Cluster dialog box, if there is a load balancer and target group in your cluster, you can choose to delete them with the cluster. They will not be used when the service is deleted.
4. In the Delete Cluster dialog box, choose OK. When the cluster is deleted, it will be removed from the AWS Explorer.

Deleting a cluster

You can delete an Amazon Elastic Container Service cluster from AWS Explorer.

To delete a cluster
1. In AWS Explorer, open the context (right-click) menu for the cluster you want to delete under the Clusters node of Amazon ECS, and then choose Delete.
2. In the Delete Cluster dialog box, choose OK. When the cluster is deleted, it will be removed from the AWS Explorer.

Creating a repository

You can create an Amazon Elastic Container Registry repository from AWS Explorer.

To create a repository
1. In AWS Explorer, open the context (right-click) menu of the Repositories node under Amazon ECS, and then choose Create Repository.
2. In the Create Repository dialog box, provide a repository name and then choose OK.

Deleting a repository

You can delete an Amazon Elastic Container Registry repository from AWS Explorer.

To delete a repository
1. In AWS Explorer, open the context (right-click) menu of the Repositories node under Amazon ECS, and then choose Delete Repository.
2. In the Delete Repository dialog box, you can choose to delete the repository even if it contains images. Otherwise, it will only be deleted if it is empty. Click Yes.

Managing Security Groups from AWS Explorer

The Toolkit for Visual Studio enables you to create and configure security groups to use with Amazon Elastic Compute Cloud (Amazon EC2) instances and AWS CloudFormation. When you launch Amazon EC2 instances or deploy an application to AWS CloudFormation, you specify a security group to associate with the Amazon EC2 instances. (Deployment to AWS CloudFormation creates Amazon EC2 instances.)

A security group acts like a firewall on incoming network traffic. The security group specifies which types of network traffic are allowed on an Amazon EC2 instance. It can also specify that incoming traffic will be accepted from certain IP addresses only or from specified users or other security groups only.
Creating a Security Group

In this section, we'll create a security group. After it has been created, the security group will not have any permissions configured. Configuring permissions is handled through an additional operation.

To create a security group

1. In AWS Explorer, under the Amazon EC2 node, open the context (right-click) menu on the Security Groups node, and then choose View.
3. In the Create Security Group dialog box, type a name and description for the security group, and then choose OK.

Adding Permissions to Security Groups

In this section, we'll add permissions to the security group to allow web traffic through the HTTP and HTTPS protocols. We'll also allow other computers to connect by using Windows Remote Desktop Protocol (RDP).

To add permissions to a security group

1. On the EC2 Security Groups tab, choose a security group and then choose the Add Permission button.
2. In the Add IP Permission dialog box, choose the Protocol, Port and Network radio button, and then from the Protocol drop-down list, choose HTTP. The port range automatically adjusts to port 80, the default port for HTTP. The Source CIDR field defaults to 0.0.0.0/0, which specifies that HTTP network traffic will be accepted from any external IP address. Choose OK.
Open port 80 (HTTP) for this security group

3. Repeat this process for HTTPS and RDP. Your security groups permissions should now look like the following.

You can also set permissions in the security group by specifying a user ID and security group name. In this case, Amazon EC2 instances in this security group will accept all incoming network traffic from Amazon EC2 instances in the specified security group. You must also specify the user ID as a way to disambiguate the security group name; security group names are not required to be unique across all of AWS. For more information about security groups, go to the EC2 documentation.
Create an AMI from an Amazon EC2 Instance

From the Amazon EC2 Instances view, you can create Amazon Machine Images (AMIs) from either running or stopped instances.

To create an AMI from an instance

1. Right-click the instance you want to use as the basis for your AMI, and choose Create Image from the context menu.

2. In the Create Image dialog box, type a unique name and description, and then choose Create Image.
Setting Launch Permissions on an Amazon Machine Image

You can set launch permissions on your Amazon Machine Images (AMIs) from the AMIs view in AWS Explorer. You can use the Set AMI Permissions dialog box to copy permissions from AMIs.

To set permissions on an AMI

1. In the AMIs view in AWS Explorer, open the context (right-click) menu on an AMI, and then choose Edit Permission.
2. There are three options available in the **Set AMI Permissions** dialog box:
   
   - To give launch permission, choose **Add**, and type the account number for the AWS user to whom you are giving launch permission.
   
   - To remove launch permission, choose the account number for the AWS user from whom you are removing launch permission, and choose **Remove**.
   
   - To copy permissions from one AMI to another, choose an AMI from the list, and choose **Copy from**. The users who have launch permissions on the AMI you chose will be given launch permissions on the current AMI. You can repeat this process with other AMIs in the **Copy-from** list to copy permissions from multiple AMIs into the target AMI.

   The **Copy-from** list contains only those AMIs owned by the account that was active when the **AMIs** view was displayed from AWS Explorer. As a result, the **Copy-from** list might not display any AMIs if no other AMIs are owned by the active account.

---

**Amazon Virtual Private Cloud (VPC)**

Amazon Virtual Private Cloud (Amazon VPC) enables you to launch Amazon Web Services (AWS) resources into a virtual network you've defined. This virtual network resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS. For more information, go to the [Amazon VPC User Guide](https://docs.aws.amazon.com/vpc/).

The Toolkit for Visual Studio enables a developer to access VPC functionality similar to that exposed by the **AWS Management Console** but from the Visual Studio development environment. The **Amazon VPC** node of AWS Explorer includes subnodes for the following areas:

- VPCs
- Subnets
Creating a Public-Private VPC for Deployment with AWS Elastic Beanstalk

This section describes how to create an Amazon VPC that contains both public and private subnets. The public subnet contains an Amazon EC2 instance that performs network address translation (NAT) to enable instances in the private subnet to communicate with the public internet. The two subnets must reside in the same Availability Zone (AZ).

This is the minimal VPC configuration required to deploy an AWS Elastic Beanstalk environment in a VPC. In this scenario, the Amazon EC2 instances that host your application reside in the private subnet; the Elastic Load Balancing load balancer that routes incoming traffic to your application resides in the public subnet.

For more information about network address translation (NAT), go to NAT Instances in the Amazon Virtual Private Cloud User Guide. For an example of how to configure your deployment to use a VPC, see Deploying to Elastic Beanstalk (p. 30).

To create a public-private subnet VPC

1. In the Amazon VPC node in AWS Explorer, open the VPCs subnode, then choose Create VPC.

2. Configure the VPC as follows:
   - Type a name for your VPC.
   - Select the With Public Subnet and the With Private Subnet check boxes.
   - From the Availability Zone drop-down list box for each subnet, choose an Availability Zone. Be sure to use the same AZ for both subnets.
   - For the private subnet, in NAT Key Pair Name, provide a key pair. This key pair is used for the Amazon EC2 instance that performs network address translation from the private subnet to the public Internet.
   - Select the Configure default security group to allow traffic to NAT check box.
to use the same AZ for both subnets. For the private subnet, in **NAT Key Pair Name**, provide a key pair. This key pair is used for the Amazon EC2 instance that performs network address translation from the private subnet to the public Internet. Select the **Configure default security group to allow traffic to NAT** check box.

Choose **OK**.

You can view the new VPC in the **VPCs** tab in AWS Explorer.

The NAT instance might take a few minutes to launch. When it is available, you can view it by expanding the **Amazon EC2** node in AWS Explorer and then opening the **Instances** subnode.
An AWS Elastic Beanstalk (Amazon EBS) volume is created for the NAT instance automatically. For more information about Elastic Beanstalk, go to AWS Elastic Beanstalk (EBS) in the Amazon EC2 User Guide for Linux Instances.

If you deploy an application to an AWS Elastic Beanstalk environment (p. 30) and choose to launch the environment in a VPC, the Toolkit will populate the Publish to AWS dialog box with the configuration information for your VPC.

The Toolkit populates the dialog box with information only from VPCs that were created in the Toolkit, not from VPCs created using the AWS Management Console. This is because when the Toolkit creates a VPC, it tags the components of the VPC so that it can access their information.

The following screenshot from the Deployment Wizard shows an example of a dialog box populated with values from a VPC created in the Toolkit.

To delete a VPC

To delete the VPC, you must first terminate any Amazon EC2 instances in the VPC.
1. If you have deployed an application to an AWS Elastic Beanstalk environment in the VPC, delete the environment. This will terminate any Amazon EC2 instances hosting your application along with the Elastic Load Balancing load balancer.

   If you attempt to directly terminate the instances hosting your application without deleting the environment, the Auto Scaling service will automatically create new instances to replace the deleted ones. For more information, go to the Auto Scaling Developer Guide.

2. Delete the NAT instance for the VPC.

   You do not need to delete the Amazon EBS volume associated with the NAT instance in order to delete the VPC. However, if you do not delete the volume, you will continue to be charged for it even if you delete the NAT instance and the VPC.

3. On the VPC tab, choose the Delete link to delete the VPC.

4. In the Delete VPC dialog box, choose OK.

Deployment Using the AWS Toolkit

The Toolkit for Visual Studio supports application deployment to AWS Elastic Beanstalk containers or AWS CloudFormation stacks.
• **Deploying to Elastic Beanstalk (p. 30)** describes how to use the Visual Studio IDE to deploy applications to Elastic Beanstalk.

• **Deploying to Amazon EC2 Container Service (p. 69)** describes how to use the Visual Studio IDE to deploy applications to Amazon ECS.

• **Standalone Deployment Tool (p. 80)** describes how to use the standalone deployment tool to deploy to either Elastic Beanstalk containers or AWS CloudFormation stacks from a command window.

**Note**

If you are using Visual Studio Express Edition:

- You can use the standalone deployment tool (p. 80) to deploy applications to Elastic Beanstalk containers.
- You can use the Docker CLI to deploy applications to Amazon ECS containers.
- You can use the AWS Management Console to deploy applications to Elastic Beanstalk containers.

For Elastic Beanstalk deployments, you must first create a web deployment package. For more information, see How to: Create a Web Deployment Package in Visual Studio. For Amazon ECS deployment, you must have a Docker image. For more information, see Visual Studio Tools for Docker.

**Topics**

- Deploying to Elastic Beanstalk (p. 30)
- Deploying to Amazon EC2 Container Service (p. 69)
- Standalone Deployment Tool (p. 80)

### Deploying to Elastic Beanstalk

AWS Elastic Beanstalk is a service that simplifies the process of provisioning AWS resources for your application. Elastic Beanstalk provides all of the AWS infrastructure required to deploy your application. This infrastructure includes:

- Amazon EC2 instances that host the executables and content for your application.
- An Auto Scaling group to maintain the appropriate number of Amazon EC2 instances to support your application.
- An Elastic Load Balancing load balancer that routes incoming traffic to the Amazon EC2 instance with the most bandwidth.

The Toolkit for Visual Studio provides a wizard that simplifies publishing applications through Elastic Beanstalk. This wizard is described in the following sections.

For more information about Elastic Beanstalk, go to the Elastic Beanstalk documentation.

**Topics**

- Deploy a Traditional ASP.NET Application to Elastic Beanstalk (p. 31)
- Deploying an ASP.NET Core Application to Elastic Beanstalk (p. 39)
- How to Specify the AWS Security Credentials for Your Application (p. 41)
- How to Republish Your Application to an Elastic Beanstalk Environment (p. 41)
- Custom Elastic Beanstalk Application Deployments (p. 43)
- Custom ASP.NET Core Elastic Beanstalk Deployments (p. 44)
Deploy a Traditional ASP.NET Application to Elastic Beanstalk

This section describes how to use the Publish to Elastic Beanstalk wizard, provided as part of the Toolkit for Visual Studio, to deploy an application through Elastic Beanstalk. To practice, you can use an instance of a web application starter project that is built in to Visual Studio or you can use your own project.

**Note**
This topic describes using the wizard to deploy traditional ASP.NET applications. The wizard also supports deploying ASP.NET Core applications. For information about ASP.NET Core, see Deploying an ASP.NET Core Application to Elastic Beanstalk (p. 39).

**Note**
Before you can use the Publish to Elastic Beanstalk wizard, you must download and install Web Deploy. The wizard relies on Web Deploy to deploy web applications and websites to Internet Information Services (IIS) web servers.

**To create a sample web application starter project**

1. In Visual Studio, from the File menu, choose New, and then choose Project.
2. In the navigation pane of the New Project dialog box, expand Installed, expand Templates, expand Visual C#, and then choose Web.
3. In the list of web project templates, choose any template containing the words Web and Application in its description. For this example, choose ASP.NET Web Forms Application.

4. In the Name box, type AEBWebAppDemo.
5. In the Location box, type the path to a solution folder on your development machine or choose Browse, and then browse to and choose a solution folder, and choose Select Folder.
6. Confirm the Create directory for solution box is selected. In the Solution drop-down list, confirm Create new solution is selected, and then choose OK. Visual Studio will create a solution and project based on the ASP.NET Web Forms Application project template. Visual Studio will then display Solution Explorer where the new solution and project appear.
To deploy an application by using the Publish to Elastic Beanstalk wizard

1. In Solution Explorer, open the context (right-click) menu for the AEBWebAppDemo project folder for the project you created in the previous section, or open the context menu for the project folder for your own application, and choose Publish to AWS Elastic Beanstalk.

The Publish to Elastic Beanstalk wizard appears.
2. In **Profile**, from the **Account profile to use for deployment** drop-down list, choose the AWS account profile you want to use for the deployment.

   Optionally, if you have an AWS account you want to use, but you haven't yet created an AWS account profile for it, you can choose the button with the plus symbol (+) to add an AWS account profile.

3. From the **Region** drop-down list, choose the region to which you want Elastic Beanstalk to deploy the application.

4. In **Deployment Target**, you can choose either **Create a new application environment** to perform an initial deployment of an application or **Redeploy to an existing environment** to redeploy a previously deployed application. (The previous deployments may have been performed with either the wizard or the **Standalone Deployment Tool** (p. 80).) If you choose **Redeploy to an existing environment**, there may be a delay while the wizard retrieves information from previous deployments that are currently running.

   **Note**
   If you choose **Redeploy to an existing environment**, choose an environment in the list, and then choose **Next**, the wizard will take you directly to the **Application Options** page. If you go this route, skip ahead to the instructions later in this section that describe how to use the **Application Options** page.

5. Choose **Next**.
6. On the Application Environment page, in the Application area, the Name drop-down list proposes a default name for the application. You can change the default name by choosing a different name from the drop-down list.

7. In the Environment area, in the Name drop-down list, type a name for your Elastic Beanstalk environment. In this context, the term environment refers to the infrastructure Elastic Beanstalk provisions for your application. A default name may already be proposed in this drop-down list. If a default name is not already proposed, you can type one or choose one from the drop-down list, if any additional names are available. The environment name cannot be longer than 23 characters.

8. In the URL area, the box proposes a default subdomain of .elasticbeanstalk.com that will be the URL for your web application. You can change the default subdomain by typing a new subdomain name.

9. Choose Check availability to make sure the URL for your web application is not already in use.

10. If the URL for your web application is okay to use, choose Next.
1. On the AWS Options page, in Amazon EC2 Launch Configuration, from the Container type drop-down list, choose an Amazon Machine Image (AMI) type that will be used for your application.

2. In the Instance type drop-down list, specify an Amazon EC2 instance type to use. For this example, we recommend you use Micro. This will minimize the cost associated with running the instance. For more information about Amazon EC2 costs, go to the EC2 Pricing page.

3. In the Key pair drop-down list, choose an Amazon EC2 instance key pair to use to sign in to the instances that will be used for your application.

4. Optionally, in the Use custom AMI box, you can specify a custom AMI that will override the AMI specified in the Container type drop-down list. For more information about how to create a custom AMI, go to Using Custom AMIs in the AWS Elastic Beanstalk Developer Guide and Create an AMI from an Amazon EC2 Instance (p. 23).

5. Optionally, if you want to launch your instances in a VPC, select the Use a VPC box.

6. Optionally, if you want to launch a single Amazon EC2 instance and then deploy your application to it, select the Single instance environment box.

   If you select this box, Elastic Beanstalk will still create an Auto Scaling group, but will not configure it. If you want to configure the Auto Scaling group later, you can use the AWS Management Console.

7. Optionally, if you want to control the conditions under which your application is deployed to the instances, select the Enable Rolling Deployments box. You can select this box only if you have not selected the Single instance environment box.

8. If your application uses AWS services such as Amazon S3 and DynamoDB, the best way to provide credentials is to use an IAM role. In the Deployed Application Permissions area, you can either choose an existing IAM role or create one the wizard will use to launch your environment. Applications using the AWS SDK for .NET will automatically use the credentials provided by this IAM role when making a request to an AWS service.

9. If your application accesses an Amazon RDS database, in the drop-down list in the Relational Database Access area, select the boxes next to any Amazon RDS security groups the wizard will update so that your Amazon EC2 instances can access that database.

10. Choose Next.
• If you selected **Use a VPC**, the **VPC Options** page will appear.

• If you selected **Enable Rolling Deployments**, but did not select **Use a VPC**, the **Rolling Deployments** page will appear. Skip ahead to the instructions later in this section that describe how to use the **Rolling Deployments** page.

• If you did not select **Use a VPC** or **Enable Rolling Deployments**, the **Application Options** page will appear. Skip ahead to the instructions later in this section that describe how to use the **Application Options** page.

If you selected **Use a VPC**, specify information on the **VPC Options** page to launch your application into a VPC.

The VPC must have already been created. If you created the VPC in the Toolkit for Visual Studio, the Toolkit for Visual Studio will populate this page for you. If you created the VPC in the **AWS Management Console**, type information about your VPC into this page.

### Key considerations for deployment to a VPC

- Your VPC needs at least one public and one private subnet.

- In the **ELB Subnet** drop-down list, specify the public subnet. The Toolkit for Visual Studio deploys the Elastic Load Balancing load balancer for your application to the public subnet. The public subnet is associated with a routing table that has an entry that points to an Internet gateway. You can recognize an Internet gateway because it has an ID that begins with `igw-` (for example, `igw-83cddaex`). Public subnets that you create by using the Toolkit for Visual Studio have tag values that identify them as public.

- In the **Instances Subnet** drop-down list, specify the private subnet. The Toolkit for Visual Studio deploys the Amazon EC2 instances for your application to the private subnet.

- The Amazon EC2 instances for your application communicate from the private subnet to the Internet through an Amazon EC2 instance in the public subnet that performs network address translation (NAT). To enable this communication, you will need a **VPC security group** that allows traffic to flow from the private subnet to the NAT instance. Specify this VPC security group in the **Security Group** drop-down list.
For more information about how to deploy an Elastic Beanstalk application to a VPC, go to the AWS Elastic Beanstalk Developer Guide.

1. After you have filled in all of the information on the VPC Options page, choose Next.
   - If you selected Enable Rolling Deployments, the Rolling Deployments page will appear.
   - If you did not select Enable Rolling Deployments, the Application Options page will appear. Skip ahead to the instructions later in this section that describe how to use the Application Options page.

2. If you selected Enable Rolling Deployments, you specify information on the Rolling Deployments page to configure how new versions of your applications are deployed to the instances in a load-balanced environment. For example, if you have four instances in your environment and you want to change the instance type, you can configure the environment to change two instances at a time. This helps ensure your application is still running while changes are being made.

3. In the Application Versions area, choose an option to control deployments to either a percentage or number of instances at a time. Specify either the desired percentage or number.

4. Optionally, in the Environment Configuration area, select the box if you want to specify the number of instances that remain in service during deployments. If you select this box, specify the maximum number of instances that should be modified at a time, the minimum number of instances that should remain in service at a time, or both.

5. Choose Next.

6. On the Application Options page, you specify information about build, Internet Information Services (IIS), and application settings.
7. In the **Build and IIS Deployment Settings** area, in the **Project build configuration** drop-down list, choose the target build configuration. If the wizard can find it, **Release** appears otherwise, the active configuration is displayed in this box.

8. In the **App pool** drop-down list, choose the version of the .NET Framework required by your application. The correct .NET Framework version should already be displayed.

9. If your application is 32-bit, select the **Enable 32-bit applications** box.

10. In the **App path** box, specify the path IIS will use to deploy the application. By default, **Default Web Site/** is specified, which typically translates to the path `c:\inetpub\wwwroot`. If you specify a path other than **Default Web Site/**, the wizard will place a redirect in the **Default Web Site/** path that points to the path you specified.

11. In the **Application Settings** area, in the **Health check URL** box, type a URL for Elastic Beanstalk to check to determine if your web application is still responsive. This URL is relative to the root server URL. The root server URL is specified by default. For example, if the full URL is `example.com/site-is-up.html`, you would type `/site-is-up.html`.

12. In the area for **Key** and **Value**, you can specify any key and value pairs you want to add to your application's `Web.config` file.

**Note**

Although not recommended, you can use the area for **Key** and **Value**, to specify AWS credentials under which your application should run. The preferred approach is to specify an IAM role in the **Identity and Access Management Role** drop-down list on the **AWS Options** page. However, if you must use AWS credentials instead of an IAM role to run your application, in the **Key** row, choose **AWSAccessKey**. In the **Value** row, type the access key. Repeat these steps for **AWSSecretKey**.

13. Choose **Next**.

---

38
14. On the Review page, review the options you configured, and select the **Open environment status window when wizard closes** box.

15. Optionally, you can save the deployment configuration to a text file that you can then use with the standalone deployment tool (p. 80). To save the configuration, select **Generate AWSDeploy configuration**, choose **Choose File**, and then specify a file to which to save the configuration. You can also save the deployment configuration to a text file after the deployment is complete. In AWS Explorer, open the context (right-click) menu for the deployment and then choose **Save Configuration**.

16. If everything looks correct, choose **Deploy**.

   **Note**
   When you deploy the application, the active account will incur charges for the AWS resources used by the application.

   Information about the deployment will appear in the Visual Studio status bar and the **Output** window. It may take several minutes. When the deployment is complete, a confirmation message will appear in the **Output** window.

17. To delete the deployment, in AWS Explorer, expand the **Elastic Beanstalk** node, open the context (right-click) menu for the subnode for the deployment, and then choose **Delete**. The deletion process might take a few minutes.

### Deploying an ASP.NET Core Application to Elastic Beanstalk

AWS Elastic Beanstalk is a service that simplifies the process of provisioning AWS resources for your application. AWS Elastic Beanstalk provides all of the AWS infrastructure required to deploy your application.

The Toolkit for Visual Studio supports deploying ASP.NET Core applications to AWS using Elastic Beanstalk. ASP.NET Core is the redesign of ASP.NET with a modularized architecture that minimizes dependency overhead and streamlines your application to run in the cloud.
AWS Elastic Beanstalk makes it easy to deploy applications in a variety of different languages to AWS. Elastic Beanstalk supports both traditional ASP.NET applications and ASP.NET Core applications. This topic describes deploying ASP.NET Core applications.

Using the Deployment Wizard

The easiest way to deploy ASP.NET Core applications to Elastic Beanstalk is with the Toolkit for Visual Studio.

If you have used the toolkit before to deploy traditional ASP.NET applications, you'll find the experience for ASP.NET Core to be very similar. In the steps below, we'll walk through the deployment experience.

If you have never used the toolkit before, the first thing you'll need to do after installing the toolkit is register your AWS credentials with the toolkit. See How to Specify the AWS Security Credentials for Your Application (p. 41) for Visual Studio documentation for details on how to do so.

To deploy an ASP.NET Core web application, right-click the project in the Solution Explorer and select Publish to AWS….

On the first page of the Publish to AWS Elastic Beanstalk deployment wizard, choose to create a new Elastic Beanstalk application. An Elastic Beanstalk application is a logical collection of Elastic Beanstalk components, including environments, versions, and environment configurations. The deployment wizard generates an application that in turn contains a collection of application versions and environments. The environments contain the actual AWS resources that run an application version. Every time you deploy an application, a new application version is created and the wizard points the environment to that version. You can learn more about these concepts in Elastic Beanstalk Components.

Next, set names for the application and its first environment. Each environment has a unique CNAME associated with it that you can use to access the application when the deployment is complete.

The next page, AWS Options, allows you to configure the type of AWS resources to use. For this example, leave the default values, except for the Key pair section. Key pairs allow you retrieve the Windows administrator password so you can log on to the machine. If you haven't already created a key pair you might want to select Create new key pair.

Permissions

The Permissions page is used for assigning AWS credentials to the EC2 instance(s) running your application. This is important if your application uses the AWS SDK for .NET to access other AWS services. If you are not using any other services from your application then you can leave this page at its default.

Application Options

The details on the Application Options page are different from those specified when deploying traditional ASP.NET applications. Here, you specify the build configuration and framework used to package the application, and also specify the IIS resource path for the application.

After completing the Application Options page, click Next to review the settings, then click Deploy to begin the deployment process.

Checking Environment Status

After the application is packaged and uploaded to AWS, you can check the status of the Elastic Beanstalk environment by opening the environment status view from the AWS Explorer in Visual Studio.

Events are displayed in the status bar as the environment is coming online. Once everything is complete, the environment status will move to healthy state. You can click on the URL to view the site. From here,
you can also pull the logs from the environment, or remote desktop into the Amazon EC2 instance(s) that are part of your Elastic Beanstalk environment.

The first deployment of any application will take a bit longer than subsequent re-deployments, as it creates new AWS resources. As you iterate on your application during development, you can quickly re-deploy by going back through the wizard, or selecting the Republish option when you right click the project.

Republish packages your application using the settings from the previous run through the deployment wizard and uploads the application bundle to the existing Elastic Beanstalk environment.

How to Specify the AWS Security Credentials for Your Application

The AWS account you specify in the Publish to Elastic Beanstalk wizard (or the legacy version of this wizard, Publish to Amazon Web Services) is the AWS account the wizard will use for deployment to Elastic Beanstalk.

Although not recommended, you may also need to specify AWS account credentials that your application will use to access AWS services after it has been deployed. The preferred approach is to specify an IAM role. In the Publish to Elastic Beanstalk wizard, you do this through the Identity and Access Management Role drop-down list on the AWS Options page. In the legacy Publish to Amazon Web Services wizard, you do this through the IAM Role drop-down list on the AWS Options page.

If you must use AWS account credentials instead of an IAM role, you can specify the AWS account credentials for your application in one of the following ways:

- Reference a profile corresponding to the AWS account credentials in the appSettings element of the project's Web.config file. (To create a profile, see Configuring AWS Credentials.) The following example specifies credentials whose profile name is myProfile.

```
<appSettings>
    <!-- AWS CREDENTIALS -->
    <add key="AWSProfileName" value="myProfile"/>
</appSettings>
```

- If you're using the Publish to Elastic Beanstalk wizard, on the Application Options page, in the Key row of the Key and Value area, choose AWSAccessKey. In the Value row, type the access key. Repeat these steps for AWSSecretKey.

- If you're using the legacy Publish to Amazon Web Services wizard, on the Application Options page, in the Application Credentials area, choose Use these credentials, and then type the access key and secret access key into the Access Key and Secret Key boxes.

How to Republish Your Application to an Elastic Beanstalk Environment

You can iterate on your application by making discrete changes and then republishing a new version to your already launched Elastic Beanstalk environment.

1. In Solution Explorer, open the context (right-click) menu for the AEBWebAppDemo project folder for the project you published in the previous section, and choose Publish to AWS Elastic Beanstalk.
2. Select **Redeploy to an existing environment** and choose the environment you previously published to. Click **Next**.

The **Review** wizard appears.
3. Click **Deploy**. The application will redploy to the same environment.

You cannot republish if your application is in the process of launching or terminating.

**Custom Elastic Beanstalk Application Deployments**

This topic describes how the deployment manifest for Elastic Beanstalk’s Microsoft Windows container supports custom application deployments.

Custom application deployments are a powerful feature for advanced users who want to leverage the power of Elastic Beanstalk to create and manage their AWS resources, but want complete control on how their application is deployed. For a custom application deployment, you create Windows PowerShell scripts for the three different actions Elastic Beanstalk performs. The install action is used when a deployment is initiated, restart is used when the **RestartAppServer** API is called from either the toolkit or the web console, and uninstall which is invoked on any previous deployment whenever a new deployment occurs.

For example, you might have an ASP.NET application that you want to deploy while your documentation team has written a static website that they want included with the deployment. You can do that by writing your deployment manifest like this:

```json
{
    "manifestVersion": 1,
    "deployments": {
        "msDeploy": [
            {
                "name": "app",
                "parameters": {
                    "appBundle": "CoolApp.zip",
                    "iisPath": "/"
                }
            }
        ]
    }
}
```
The scripts listed for each action must be in the application bundle relative to the deployment manifest file. For this example, the application bundle will also contain a documentation.zip file which contains a static website created by your documentation team.

The `install.ps1` script extracts the zip file and sets up the IIS Path.

```powershell
Add-Type -assembly "system.io.compression.filesystem"
[io.compression.zipfile]::ExtractToDirectory('./documentation.zip', 'c:\inetpub\wwwroot\documentation')
powershell.exe -Command {New-WebApplication -Name documentation -PhysicalPath c:\inetpub\wwwroot\documentation -Force}
```

Since your application is running in IIS, the restart action will invoke an IIS reset.

```powershell
iisreset /timeout:1
```

For uninstall scripts, it is important to clean up all settings and files used during the install stage. That way during the install phase for the new version, you can avoid any collision with previous deployments. For this example, you need to remove the IIS application for the static website and remove the website files.

```powershell
powershell.exe -Command {Remove-WebApplication -Name documentation}
Remove-Item -Recurse -Force 'c:\inetpub\wwwroot\documentation'
```

With these script files and the documentation.zip file included in your application bundle, the deployment creates the ASP.NET application and then deploys the documentation site.

For this example, we choose a simple example that deploys a simple static website, but with custom application deployment you can deploy any type of application and let Elastic Beanstalk manage the AWS resources for it.

### Custom ASP.NET Core Elastic Beanstalk Deployments

This topic describes how deployment works and what you can do customize deployments when creating ASP.NET Core applications with Elastic Beanstalk and the Toolkit for Visual Studio.

After you complete the deployment wizard in the Toolkit for Visual Studio, the toolkit bundles the application and sends it to Elastic Beanstalk. Your first step in creating the application bundle is to...
use the new dotnet CLI to prepare the application for publishing by using the `publish` command. The framework and configuration are passed down from the settings in the wizard to the `publish` command. So if you selected **Release** for configuration and **netcoreapp1.0** for the framework, the toolkit will execute the following command:

```
dotnet publish --configuration Release --framework netcoreapp1.0
```

When the `publish` command finishes, the toolkit writes the new deployment manifest into the publishing folder. The deployment manifest is a JSON file named **aws-windows-deployment-manifest.json**, which the Elastic Beanstalk Windows container (version 1.2 or later) reads to determine how to deploy the application. For example, for an ASP.NET Core application you want to be deploy at the root of IIS, the toolkit generates a manifest file that looks like this:

```json
{
    "manifestVersion": 1,
    "deployments": {
        "aspNetCoreWeb": [
            {
                "name": "app",
                "parameters": {
                    "appBundle": ".",
                    "iisPath": "/",
                    "iisWebSite": "Default Web Site"
                }
            }
        ]
    }
}
```

The `appBundle` property indicates where the application bits are in relation to the manifest file. This property can point to either a directory or a ZIP archive. The `iisPath` and `iisWebSite` properties indicate where in IIS to host the application.

### Customizing the Manifest

The toolkit only writes the manifest file if one doesn't already exist in the publishing folder. If the file does exist, the toolkit updates the `appBundle`, `iisPath` and `iisWebSite` properties in the first application listed under the `aspNetCoreWeb` section of the manifest. This allows you to add the **aws-windows-deployment-manifest.json** to your project and customize the manifest. To do this for an ASP.NET Core Web application in Visual Studio add a new JSON file to the root of the project and name it **aws-windows-deployment-manifest.json**.

The manifest must be named **aws-windows-deployment-manifest.json** and it must be at the root of the project. The Elastic Beanstalk container looks for the manifest in the root and if it finds it will invoke the deployment tooling. If the file doesn't exist, the Elastic Beanstalk container falls back to the older deployment tooling, which assumes the archive is an **msdeploy** archive.

To ensure the dotnet CLI `publish` command includes the manifest, update the `project.json` file to include the manifest file in the `include` section under `include` in `publishOptions`.

```json
{
    "publishOptions": {
        "include": [
            "wwwroot",
            "Views",
            "Areas/**/Views",
            "appsettings.json",
            "web.config",
            "aws-windows-deployment-manifest.json"
        ]
    }
}
```
Now that you've declared the manifest so that it's included in the app bundle, you can further configure how you want to deploy the application. You can customize deployment beyond what the deployment wizard supports. AWS has defined a JSON schema for the `aws-windows-deployment-manifest.json` file, and when you installed the Toolkit for Visual Studio, the setup registered the URL for the schema.

When you open `windows-deployment-manifest.json`, you'll see the schema URL selected in the Schema drop down box. You can navigate to the URL to get a full description of what can be set in the manifest. With the schema selected, Visual Studio will provide IntelliSense while you're editing the manifest.

One customization you can do is to configure the IIS application pool under which the application will run. The following example shows how you can define an IIS Application pool ("customPool") that recycles the process every 60 minutes, and assigns it to the application using "appPool": "customPool".

```json
{
  "manifestVersion": 1,
  "iisConfig": {
    "appPools": [
      {
        "name": "customPool",
        "recycling": {
          "regularTimeInterval": 60
        }
      }
    ],
  },
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "app",
        "parameters": {
          "appPool": "customPool"
        }
      }
    ],
  }
}
```

Additionally, the manifest can declare Windows PowerShell scripts to run before and after the install, restart and uninstall actions. For example, the following manifest runs the Windows PowerShell script `PostInstallSetup.ps1` to do further setup work after the ASP.NET Core application is deployed to IIS. When adding scripts like this, make sure the scripts are added to the include section under `publishOptions` in the `project.json` file, just as you did with the `aws-windows-deployment-manifest.json` file. If you don't, the scripts won't be included as part of the `dotnet CLI publish` command.

```json
{
  "manifestVersion": 1,
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "app",
        "scripts": {
          "postInstall": {
            "file": "SetupScripts/PostInstallSetup.ps1"
          }
        }
      }
    ],
  }
}
```
What about ebextensions?

The Elastic Beanstalk ebextensions configuration files are supported as with all the other Elastic Beanstalk containers. To include ebextensions in an ASP.NET Core application, add the .ebextensions directory to the include section under publishOptions in the project.json file. For further information about ebextensions checkout the Elastic Beanstalk Developer Guide.

Multiple Application Support for .NET and Elastic Beanstalk

Using the deployment manifest, you have the ability to deploy multiple applications to the same Elastic Beanstalk environment.

The deployment manifest supports ASP.NET Core web applications as well as msdeploy archives for traditional ASP.NET applications. Imagine a scenario where you have written a new amazing application using ASP.NET Core for the frontend and a Web API project for an extensions API. You also have an admin app that you wrote using traditional ASP.NET.

The toolkit’s deployment wizard focuses on deploying a single project. To take advantage of multiple application deployment, you have to construct the application bundle by hand. To start, write the manifest. For this example, you will write the manifest at the root of your solution.

The deployment section in the manifest has two children: an array of ASP.NET Core web applications to deploy, and an array of msdeploy archives to deploy. For each application, you set the IIS path and the location of the application’s bits relative to the manifest.

```json
{
  "manifestVersion": 1,
  "deployments": {
    "aspNetCoreWeb": [
      {
        "name": "frontend",
        "parameters": {
          "appBundle": "./frontend",
          "iisPath": "/frontend"
        }
      },
      {
        "name": "ext-api",
        "parameters": {
          "appBundle": "./ext-api",
          "iisPath": "/ext-api"
        }
      }
    ],
    "msDeploy": [
      {
        "name": "admin",
        "parameters": {
          "appBundle": "AmazingAdmin.zip",
          "iisPath": "/admin"
        }
      }
    ]
  }
}
```
With the manifest written, you’ll use Windows PowerShell to create the application bundle and update an existing Elastic Beanstalk environment to run it. The script is written assuming that it will be run from the folder containing your Visual Studio solution.

The first thing you need to do in the script is setup a workspace folder in which to create the application bundle.

```powershell
$publishFolder = "c:\temp\publish"
$publishWorkspace = [System.IO.Path]::Combine($publishFolder, "workspace")
$appBundle = [System.IO.Path]::Combine($publishFolder, "app-bundle.zip")

If (Test-Path $publishWorkspace){
    Remove-Item $publishWorkspace -Confirm:$false -Force
} If (Test-Path $appBundle){
    Remove-Item $appBundle -Confirm:$false -Force
}
```

Once you’ve created the folder, it is time to get the frontend ready. As with the deployment wizard, use the dotnet CLI to publish the application.

```powershell
Write-Host 'Publish the ASP.NET Core frontend'
$publishFrontendFolder = [System.IO.Path]::Combine($publishWorkspace, "frontend")
dotnet publish .\src\AmazingFrontend\project.json -o $publishFrontendFolder -c Release -f netcoreapp1.0

Notice that the subfolder "frontend" was used for the output folder, matching the folder you set in the manifest. Now you need to do the same for the Web API project.

```powershell
Write-Host 'Publish the ASP.NET Core extensibility API'
$publishExtAPIFolder = [System.IO.Path]::Combine($publishWorkspace, "ext-api")
dotnet publish .\src\AmazingExtensibleAPI\project.json -o $publishExtAPIFolder -c Release -f netcoreapp1.0
```

The admin site is a traditional ASP.NET application, so you can't use the dotnet CLI. For the admin application, you should use msbuild, passing the build target package to create the msdeploy archive. By default the package target creates the msdeploy archive under the \obj\Release\Package folder, so you will need to copy the archive to the publish workspace.

```powershell
Write-Host 'Create msdeploy archive for admin site'
msbuild .\src\AmazingAdmin\AmazingAdmin.csproj /t:package /p:Configuration=Release
Copy-Item .\src\AmazingAdmin\obj\Release\Package\AmazingAdmin.zip $publishWorkspace
```

To tell the Elastic Beanstalk environment what to do with all these applications, copy the manifest from your solution to the publish workspace and then zip up the folder.

```powershell
Write-Host 'Copy deployment manifest'
Copy-Item .\aws-windows-deployment-manifest.json $publishWorkspace

Write-Host 'Zipping up publish workspace to create app bundle'
Add-Type -assembly "system.io.compression.filesystem"
[io.compression.zipfile]::CreateFromDirectory( $publishWorkspace, $appBundle)
```

Now that you have the application bundle, you could go to the web console and upload the archive to a Elastic Beanstalk environment. Alternatively, you can continue to use the AWS PowerShell cmdlets to update the Elastic Beanstalk environment with the application bundle. Make sure you have set the current profile and region to the profile and region that contains your Elastic Beanstalk environment by using Set-AWSCredentials and Set-DefaultAWSRegion cmdlets.
Write-Host 'Write application bundle to S3'
# Determine S3 bucket to store application bundle
$s3Bucket = New-EBStorageLocation
Write-S3Object -BucketName $s3Bucket -File $appBundle

$appBundle = "app-bundle.zip"
Write-Host 'Update Beanstalk environment for new application bundle'
New-EBApplicationVersion -ApplicationName $applicationName -VersionLabel $versionLabel -SourceBundle_S3Bucket $s3Bucket -SourceBundle_S3Key $appBundle
Update-EBEnvironment -ApplicationName $applicationName -EnvironmentName $environmentName -VersionLabel $versionLabel

Now, check the status of the update using either the Elastic Beanstalk environment status page in either the toolkit or the web console. Once complete you will be able to navigate to each of the applications you deployed at the IIS path set in the deployment manifest.

Deploying to Elastic Beanstalk (Legacy)

**Note**
The information in this section refers to the Publish to Amazon Web Services wizard, which has been replaced by the Publish to Elastic Beanstalk wizard. The following information is provided for those who prefer to, or must, use the legacy wizard.

For information about using the Publish to Elastic Beanstalk wizard, see Deploying to Elastic Beanstalk (p. 30).

AWS Elastic Beanstalk is a service that simplifies the process of provisioning AWS resources for your application. Elastic Beanstalk provides all of the AWS infrastructure required to deploy your application. This infrastructure includes:

- Amazon EC2 instances that host the executables and content for your application.
- An Auto Scaling group to maintain the appropriate number of Amazon EC2 instances to support your application.
- An Elastic Load Balancing load balancer that routes incoming traffic to the Amazon EC2 instance with the most bandwidth.

For more information about Elastic Beanstalk, go to the Elastic Beanstalk documentation.

How to Deploy a Web Application Using Elastic Beanstalk (Legacy)

This section describes how to use the legacy Publish to Amazon Web Services wizard, provided as part of the Toolkit for Visual Studio, to deploy a web application through Elastic Beanstalk. To practice, you can use an instance of a web application starter project that is built in to Visual Studio or you can use your own project.

**Note**
Before you can use the legacy Publish to Amazon Web Services wizard, you must download and install Web Deploy. The wizard relies on Web Deploy to deploy web applications and websites to Internet Information Services (IIS) web servers.

To deploy an application by using the legacy Publish to Amazon Web Services wizard

**Note**
If you don't have a project ready to deploy, follow the steps in To create a sample web application starter project (p. 58) and then follow the steps below.
1. Specify the AWS security credentials for the web application. For instructions, see How to Specify the AWS Security Credentials for Your Application (p. 41).

These credentials might be different from the credentials you use to do the deployment. The credentials for the deployment are specified in the deployment wizard described later.

2. In Solution Explorer, open the context (right-click) menu for the AEBWebAppDemo project folder or for the project folder for your own application, and choose Publish to AWS.

3. On the Publish to AWS Elastic Beanstalk page, choose Use legacy wizard.

4. On the Template page of the wizard, choose the AWS account you want to use for the deployment. To add a new account, choose the button with the plus sign (+).

There are options to perform an initial deployment of an application or redeploy a previously deployed application. Previous deployments may have been performed with either the deployment wizard or the Standalone Deployment Tool (p. 80). If you choose a redeployment, there may be a delay while the wizard retrieves information from previous deployments that are currently running.

For this example, choose Deploy new application with template, choose AWS Elastic Beanstalk, and then choose Next.
5. On the **Application** page, the Toolkit has already provided a default name for the application. You can change the default name. You can also provide an optional description in the **Application Details** area.

The Toolkit also provides a deployment version label, which is based on the current date and time. You can change this version label, but the Toolkit checks it for uniqueness.

If you are using incremental deployment, **Deployment version label** is unavailable. For incremental deployments, the version label is formed from the Git commit ID. In this case, the version label is unique because the commit ID is derived from a SHA-1 cryptographic hash.

With incremental deployment, the first time that you deploy your application, all application files are copied to the server. If you later update some of your application files and redeploy, only the changed files are copied, which potentially reduces the amount of time required for redeployment. Without incremental deployment, all of your application files, whether they were changed or not, are copied to the server with each redeployment.

Select **Deploy application incrementally** and then choose **Next**.
6. On the **Environment** page, type a name and description for your Elastic Beanstalk environment. In this context, **environment** refers to the infrastructure Elastic Beanstalk provisions for your application. The Toolkit has already provided a default name, which you can change. The environment name cannot be longer than 23 characters. In **Description**, type any text you choose.

   You can also provide a subdomain of `.elasticbeanstalk.com` that will be the URL for your application. The Toolkit provides a default subdomain based on the environment name.

7. Choose **Check availability** to make sure the URL for your web application is okay to use.

8. Choose **Next**.
9. On the **AWS Options** page, configure the following.

- From the **Container type** drop-down list, choose a container type. The container type specifies an Amazon Machine Image (AMI) for your application and configurations for the Auto Scaling group, the load balancer, and other aspects of the environment in which your application will run.

- Optional. In the **Use custom AMI** field, you can specify a custom AMI. If you specify a custom AMI, it will override the AMI in **Container type**. For more information about how to create a custom AMI go to Using Custom AMIs in the AWS Elastic Beanstalk Developer Guide and Create an AMI from an Amazon EC2 Instance (p. 23).

- From the **Instance Type** drop-down list, choose an Amazon EC2 instance type. For this application, we recommend you use **Micro** because this will minimize the cost associated with running the instance. For more information about Amazon EC2 costs, go to the EC2 Pricing page.

- From the **Key pair** drop-down list, choose a key pair.

- The **IAM Role** drop-down list displays the roles available for your Elastic Beanstalk environment. If you do not have an IAM role, you can choose **Use the default role** from the list. In this case, Elastic Beanstalk creates a default IAM role and updates the Amazon S3 bucket policy to allow log rotation.

An IAM role provides applications and services access to AWS resources using temporary security credentials. For example, if your application requires access to DynamoDB, it must use AWS security credentials to make an API request. The application can use these temporary security credentials so you do not have to store long-term credentials on an Amazon EC2 instance or update the instance every time the credentials are rotated. Elastic Beanstalk requires an IAM role to rotate logs to Amazon S3.

If you choose not to use the IAM role, you need to grant permissions for Elastic Beanstalk to rotate logs. For instructions, see Using a Custom Instance Profile. For more information about log rotation,
The credentials you use for deployment must have permission to create the default IAM role. Choose Next.

The VPC Options page provides the option to launch your application to a VPC. The VPC must have already been created. You can use the Toolkit for Visual Studio or the AWS Management Console to create a VPC. If you created the VPC in the Toolkit, the Toolkit will populate this page for you. If you created the VPC in the console, type information about your VPC into this page.

Key considerations for deployment to a VPC

- Your VPC needs at least one public and one private subnet.
- In the ELB Subnet drop-down list, specify the public subnet. The Toolkit for Visual Studio deploys the Elastic Load Balancing load balancer for your application to the public subnet. The public subnet is associated with a routing table that has an entry that points to an Internet gateway. You can recognize an Internet gateway because it has an ID that begins with igw-`(for example, :code:`igw-83cddaee`). Public subnets that you create by using the Toolkit have tag values that identify them as public.
- In the Instances Subnet drop-down list, specify the private subnet. The Toolkit deploys the Amazon EC2 instances for your application to the private subnet.
- The Amazon EC2 instances for your application communicate from the private subnet to the Internet through an Amazon EC2 instance in the public subnet that performs network address translation (NAT). To enable this communication, you will need a VPC security group that allows traffic to flow
from the private subnet to the NAT instance. Specify this VPC security group in the Security Group drop-down list.

For more information about how to deploy an Elastic Beanstalk application to a VPC, go to the AWS Elastic Beanstalk Developer Guide.

1. On the Application Options page, configure the following.
   - Under Application Pool Options, in the Target framework drop-down list, choose the version of the .NET Framework required by your application (for example, .NET Framework 2.0, .NET Framework 3.0, .NET Framework 3.5, .NET Framework 4.0, .NET Framework 4.5).
     For this walkthrough, select Enable 32-bit applications.
   - Under Miscellaneous, in the Application health-check URL box, type a URL for Elastic Beanstalk to check to determine if your application is still responsive. This URL is relative to the root server URL. For example, if the full URL is, you would type /site-is-up.html. For this sample application, leave the default setting of a forward slash (/).
   - In Application Environment, use the parameter fields (PARAM1-5) to provide input data to your application. These values are made available to the deployed application through the appSettings element in the Web.config file. For more information, go to the Microsoft MSDN library.
   - In Application Credentials, choose the AWS credentials under which the application should run. These could be different from the credentials used to deploy to Elastic Beanstalk.
     - To use a different set of credentials, choose Use these credentials and type the access key and secret key in the fields provided.
• To use the same credentials as those used to deploy to Elastic Beanstalk, choose **Use credentials from profile '<account name>'** where {<account name>} is the account selected on the first page of the wizard.

• To use the credentials for an AWS Identity and Access Management (IAM) user, choose **Use an IAM user** and then specify the user.

To use an IAM user, you must have:

• created the IAM user in the Toolkit for Visual Studio.

• stored the secret key for the user with the Toolkit for Visual Studio.

For more information, see [Create and Configure an IAM User](p. 134) and [Generate Credentials for an IAM User](p. 137).

An IAM user could have more than one set of credentials stored with the Toolkit. If that is the case, you will need to choose the credentials to use. The root account could rotate the credentials for the IAM user, which would invalidate the credentials. In this scenario, you would need to redeploy the application and then manually enter new credentials for the IAM user.

Choose **Next**.

2. If you have deployed Amazon RDS instances, a page similar to the following will appear as part of the deployment wizard. You can use this page to add the Amazon EC2 instances for your deployment to one or more of the Amazon RDS security groups associated with your RDS instances. If your application needs to access your RDS instances, you will need to enable this access here or by setting the permissions on your RDS security groups. For more information, see [Amazon RDS Security Groups](p. 127).
If you are deploying to a VPC, this page will not appear because for VPCs, RDS instances are managed by Amazon EC2 security groups.

3. On the **Review** page, review the options you configured earlier, and select **Open environment status window when wizard closes**.

If everything looks correct, choose **Deploy**.

**Note**
When you deploy the application, the active account will incur charges for the AWS resources used by the application.

You can save the deployment configuration to a text file to use with standalone deployment tool. To save the configuration, select **Generate AWSDeploy configuration**. Choose **Choose File** and then specify a file to which to save the configuration. You can also save the deployment configuration after the deployment is complete. In AWS Explorer, open the context (right-click) menu for the deployment and choose **Save Configuration**.

**Note**
When you deploy the application, the active account will incur charges for the AWS resources used by the application.
4. A status page for the deployment will open. The deployment may take a few minutes.

When the deployment is complete, the Toolkit will display an alert. This is useful because it allows you to focus on other tasks while the deployment is in progress.

Choose the Application URL link to connect to the application.

5. To delete the deployment, in AWS Explorer, expand the Elastic Beanstalk node, open the context (right-click) menu for the subnode for the deployment, and choose Delete. Elastic Beanstalk will begin the deletion process, which might take a few minutes. If you specified a notification email address in the deployment, Elastic Beanstalk will send status notifications to this address.

To create a sample web application starter project

Follow these steps to create a sample application if you do not have a project ready to deploy.

1. In Visual Studio, from the File menu, choose New, and then choose Project.
2. In the New Project dialog box, in the navigation pane, expand Installed, expand Templates, expand Visual C#, and then choose Web.
3. In the list of available web project templates, choose any template containing the words Web and Application in its description. For this example, choose ASP.NET Web Forms Application.
4. In the **Name** box, type AEBWebAppDemo.

5. In the **Location** box, type the path to a solution folder on your development machine or choose **Browse**, and then browse to and choose a solution folder, and choose **Select Folder**.

6. Confirm the **Create directory for solution** box is selected. In the **Solution** drop-down list, confirm **Create new solution** is selected, and then choose **OK**. Visual Studio will create a solution and project based on the ASP.NET Web Forms Application project template.
Deploying to AWS CloudFormation (Legacy)

Note
The information in this topic refers to the Publish to Amazon Web Services wizard, which has been replaced by deploying through Elastic Beanstalk through the use of the Publish to Elastic Beanstalk wizard. The following information is provided for those who prefer to, or must, use the legacy wizard to deploy through AWS CloudFormation.
For information about using the preferred Publish to Elastic Beanstalk wizard, see Deploying to Elastic Beanstalk (p. 30).

AWS CloudFormation is a service that simplifies the process of provisioning AWS resources for your application. The AWS resources are described in a template file. The AWS CloudFormation service consumes this template and automatically provisions the required resources for you. For more information, go to AWS CloudFormation.

We'll deploy an application to AWS and use AWS CloudFormation to provision the resources for the application. To practice, you can use an instance of a web application starter project that is built in to Visual Studio or you can use your own project.

To create a sample web application starter project

Follow these steps if you do not have project ready to deploy.
1. In Visual Studio, from the **File** menu, choose **New**, and then choose **Project**.

2. In the navigation pane of the **New Project** dialog box, expand **Installed**, expand **Templates**, expand **Visual C#**, and then choose **Web**.

3. In the list of available web project templates, select any template containing the words **Web** and **Application** in its description. For this example, choose **ASP.NET Web Forms Application**.

4. In the **Name** box, type **AEBWebAppDemo**.

5. In the **Location** box, type the path to a solution folder on your development machine or choose **Browse**, and then browse to and choose a solution folder, and choose **Select Folder**.

6. Confirm the **Create directory for solution** box is selected. In the **Solution** drop-down list, confirm **Create new solution** is selected, and then choose **OK**. Visual Studio will create a solution and project based on the ASP.NET Web Forms Application project template.
To deploy an application by using the legacy Publish to Amazon Web Services wizard

1. In Solution Explorer, open the context (right-click) menu for the AEBWebAppDemo project folder (or your own project folder), and then choose Publish to AWS.

2. On the Publish to AWS Elastic Beanstalk page, choose Use legacy wizard.
3. On the Template page of the wizard, choose the profile you will use for the deployment. To add a new profile, choose Other. For more information about profiles, see creds.

4. There are options to deploy a new application or redeploy an application that was deployed previously through either the deployment wizard or the standalone deployment tool. If you choose a redeployment, there may be a delay while the wizard retrieves information from the previous deployment.

The Load Balanced Template and Single Instance Template are included with the Toolkit for Visual Studio. Load Balanced Template provisions an Amazon EC2 instance with an Elastic Load Balancing load balancer and an Auto Scaling group. Single Instance Template provisions just a single Amazon EC2 instance.

For this example, choose Load Balanced Template, and then choose Next.
5. On the **AWS Options** page, configure the following:
   - From the **Key pair** drop-down list, choose an Amazon EC2 key pair.
   - Leave **SNS Topic** blank. If you specify an SNS topic, AWS CloudFormation will send status notifications during the deployment.
   - Leave the **Custom AMI** field blank. The AWS CloudFormation template includes an AMI.
   - From the **Instance type** drop-down list, leave the default set to **Micro**. This will minimize the cost associated with running the instance. For more information about Amazon EC2 costs, go to the [EC2 Pricing](https://aws.amazon.com/ec2/pricing) page.
   - From the **Security group** drop-down list, choose a security group that has port 80 open. If you have already configured a security group with port 80 open, then choose it. The **default** selection in this drop-down list does not have port 80 open.

   Applications deployed to AWS CloudFormation must have port 80 open because AWS CloudFormation uses this port to relay information about the deployment. If the security group you choose does not have port 80 open, the wizard will ask if it should open it. If you say yes, port 80 will be open for any Amazon EC2 instances that use that security group. For more information about creating a security group, see [Managing Security Groups from AWS Explorer](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/admin-guide.html) (p. 20).

   Choose **Next**.
6. On the **Application Options** page, in the **Application Credentials** section, choose the profile under which the application (in this example, PetBoard) should run. It could be different from the profile used to deploy to AWS CloudFormation (that is, the profile you specified on the first page of the wizard).

To use a different set of credentials, choose **Use these credentials** and then type the access key and secret key in the fields provided.

To use the same credentials, choose **Use credentials from profile profile_name** where `{profile_name}` is the profile you specified on the first page of the wizard.

To use the credentials for an AWS Identity and Access Management (IAM) user, choose **Use an IAM user**, and then specify the user.

To use an IAM user, you must have:
- created the IAM user in the Toolkit for Visual Studio.
- stored the secret key for the user with the Toolkit for Visual Studio.

For more information, see Create and Configure an IAM User (p. 134) and Generate Credentials for an IAM User (p. 137).

An IAM user could have more than one set of credentials stored with the Toolkit. If that is the case, you will need to choose the credentials to use. The root account could rotate the credentials for the IAM user, which would invalidate the credentials. In this scenario, you would need to redeploy the application and then manually enter new credentials for the IAM user.
The following table describes other options available on the Application Options page. For PetBoard, you can leave the defaults.

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARAM1, PARAM2, PARAM3, PARAM4, PARAM5</td>
<td>These values are made available to the deployed application through the appSettings element in the Web.config file. For more information, go to the Microsoft MSDN library.</td>
</tr>
<tr>
<td>Target framework</td>
<td>Specifies the version of the .NET Framework targeted by the application. Possible values are: .NET Framework 2.0, .NET Framework 3.0, .NET Framework 3.5, .NET Framework 4.0, .NET Framework 4.5</td>
</tr>
<tr>
<td>Enable 32-bit applications</td>
<td>Select if the application is 32-bit. Otherwise, leave the box cleared.</td>
</tr>
<tr>
<td>Application health check URL</td>
<td>This URL is relative to the root server URL. For example, if the full path to the URL is example.com/site-is-up.html, you would type /site-is-up.html. This setting applies only when you use the Load Balanced template. It is ignored when you use the Single Instance template.</td>
</tr>
</tbody>
</table>

Choose Finish.
7. On the **Review** page, select **Open environment status window when wizard closes**.

You can save the deployment configuration to a text file to use with standalone deployment tool. To save the configuration, select **Generate AWSDeploy configuration**. Choose **Choose File** and then specify a file to which to save the configuration. You can also save the deployment configuration after the deployment is complete. In AWS Explorer, open the context (right-click) menu for the deployment and choose **Save Configuration**.

**Note**

Because the deployment configuration includes the credentials that were used for deployment, you should keep the configuration file in a secure location.

Choose **Deploy**.

**Note**

When you deploy the application, the active account will incur charges for the AWS resources used by the application.
8. A status page for the deployment will open. The deployment may take a few minutes.

When the deployment is complete, the Toolkit will display an alert. This is useful because it allows you to focus on other tasks while the deployment is in progress.

When the deployment is complete, the status displayed in the Toolkit for Visual Studio will be CREATE_COMPLETE.

Choose the Application URL link to connect to the application.

9. To delete the deployment, in AWS Explorer, expand the CloudFormation node and open the context (right-click) menu for the subnode for the deployment and choose Delete. AWS CloudFormation will
Deploying to Amazon EC2 Container Service

Amazon Elastic Container Service is a highly scalable, high performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances.

To deploy applications on Amazon Elastic Container Service, your application components must be developed to run in a Docker container. A Docker container is a standardized unit of software development, containing everything that your software application needs to run: code, runtime, system tools, system libraries, etc.

The Toolkit for Visual Studio provides a wizard that simplifies publishing applications through Amazon ECS. This wizard is described in the following sections.

For more information about Amazon ECS, go to the Elastic Container Service documentation. It includes an overview of Docker basics and creating a cluster.

Topics
- Specify AWS Credentials for Your ASP.NET Core 2 Application (p. 69)
- Deploying an ASP.NET Core 2.0 App to Amazon ECS (Fargate) (p. 70)
- Deploying an ASP.NET Core 2.0 App to Amazon ECS (EC2) (p. 77)

Specify AWS Credentials for Your ASP.NET Core 2 Application

There are two types of credentials in play when you deploy your application to a Docker container: deployment credentials and instance credentials.

Deployment credentials are used by the Publish Container to AWS wizard to create the environment in Amazon ECS. This includes things like tasks, services, IAM roles, a Docker container repository, and if you choose, a load balancer.

Instance credentials are used by the instance (including your application) to access different AWS services. For example, if your an ASP.NET Core 2.0 application reads and writes to Amazon S3 objects, it will need appropriate permissions. You can provide different credentials using different methods based on the environment. For example, your ASP.NET Core 2 application might target Development and Production environments. You could use a local Docker instance and credentials for development and a defined role in production.

Specifying deployment credentials

The AWS account you specify in the Publish Container to AWS wizard is the AWS account the wizard will use for deployment to Amazon ECS. The account profile must have permissions to Amazon Elastic Compute Cloud, Amazon Elastic Container Service, and AWS Identity and Access Management.

If you notice options missing from drop-down lists, it may be because you lack permissions. For example, if you created a cluster for your application but do not see it on the Publish Container to AWS wizard Cluster page. If this happens, add the missing permissions and try the wizard again.

Specifying development instance credentials

For non-production environments, you can configure your credentials in the appsettings.<environment>.json file. For example, to configure your credentials in the appsettings.Development.json file in Visual Studio 2017:
1. Add the AWSSDK.Extensions.NETCore.Setup NuGet package to your project.
2. Add AWS settings to appsettings.Development.json. The configuration below sets Profile and Region.

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

### Specifying production instance credentials

For production instances, we recommend you use an IAM role to control what your application (and the service) can access. For example, to configure an IAM role with Amazon ECS as the service principal with permissions to Amazon Simple Storage Service and Amazon DynamoDB from the AWS Management Console:

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane of the IAM console, choose Roles, and then choose Create role.
3. Choose the AWS Service role type, and then choose EC2 Container Service.
4. Choose the EC2 Container Service Task use case. Use cases are defined by the service to include the trust policy that the service requires. Then choose Next: Permissions.
5. Choose the AmazonS3FullAccess and AmazonDynamoDBFullAccess permissions policies. Check the box next to each policy, and then choose Next: Review,
6. For Role name, type a role name or role name suffix to help you identify the purpose of this role. Role names must be unique within your AWS account. They are not distinguished by case. For example, you cannot create roles named both PRODROLE and prodrole. Because various entities might reference the role, you cannot edit the name of the role after it has been created.
7. (Optional) For Role description, type a description for the new role.
8. Review the role and then choose Create role.

You can use this role as the task role on the ECS Task Definition page of the Publish Container to AWS wizard.

For more information, see Using Service-Based Roles.

### Deploying an ASP.NET Core 2.0 App to Amazon ECS (Fargate)

This section describes how to use the Publish Container to AWS wizard, provided as part of the Toolkit for Visual Studio, to deploy a containerized ASP.NET Core 2.0 application targeting Linux through Amazon ECS using the Fargate launch type. Because a web application is meant to run continuously, it will be deployed as a service.

#### Before you publish your container

Before using the Publish Container to AWS wizard to deploy your ASP.NET Core 2.0 application:

- Specify your AWS credentials (p. 69) and get setup with Amazon ECS.
- Install Docker. You have a few different installation options including Docker for Windows.
- In Visual Studio, create (or open) a project for an ASP.NET Core 2.0 containerized app targeting Linux.
Accessing the Publish Container to AWS wizard

To deploy an ASP.NET Core 2.0 containerized application targeting Linux, right-click the project in the Solution Explorer and select **Publish Container to AWS**.

You can also select **Publish Container to AWS** on the Visual Studio Build menu.
Account profile to use - Select an account profile to use.

Region - Choose the deployment region. Profile and region are used to set up your deployment environment resources and to select the default Docker registry.

Configuration - Select the Docker image build configuration.

Docker Repository - Choose an existing Docker repository or type in the name of a new repository and it will be created. This is the repository the build container is pushed to.

Tag - Select an existing tag or type in the name of a new tag. Tags can track important details like version, options or other unique configuration elements of the Docker container.

Deployment Target - Select Service on an ECS Cluster. Use this deployment option when your application is meant to be long-running (like an ASP.NET web application).

Save settings to aws-docker-tools-defaults.json and configure project for command line deployment - Check this option if you want the flexibility of deploying from the command line. Use dotnet ecs deploy from your project directory to deploy and dotnet ecs publish the container.
ECS Cluster - Pick the cluster that will run your Docker image. If you choose to create an empty cluster, provide a name for your new cluster.

Launch Type - Choose FARGATE.

CPU Maximum (vCPU) - Choose the maximum amount of compute capacity needed for your application. To see allowed ranges of CPU and Memory values, see task size.

Memory Maximum (GB) - Select the maximum amount of memory available to your application.

VPC Subnets - Choose one or more subnets under a single VPC. If you choose more than one subnet, your tasks will be distributed across them. This can improve availability. For more information, see default VPC and default subnets.

Security Groups - Choose a security group.

A security group acts as a firewall for associated Amazon EC2 instances, controlling both inbound and outbound traffic at the instance level.

Default security groups are configured to allow inbound traffic from instances assigned to the same security group and all outbound IPv4 traffic. You need outbound allowed so the service can reach the container repository.

Assign Public IP Address - Check this to make your task accessible from the internet.
Service Configuration page

Service - Select one of the services in the drop-down to deploy your container into an existing service. Or choose Create New to create a new service. Service names must be unique within a cluster, but you can have similarly named services in multiple clusters within a region or across multiple regions.

Number of Tasks - The number of tasks to deploy and keep running on your cluster. Each task is one instance of your container.

Minimum Healthy Percent - The percentage of tasks that must remain in RUNNING state during a deployment rounded up to the nearest integer.

Maximum Percent - The percentage of tasks that are allowed in the RUNNING or PENDING state during a deployment rounded down to the nearest integer.
**Application Load Balancer page**

*Configure Application Load Balancer* - Check to configure an application load balancer.

*Load Balancer* - Select an existing load balancer or choose **Create New** and type in the name for the new load balancer.

*Listener Port* - Select an existing listener port or choose **Create New** and type in a port number. The default, port 80, is appropriate for most web applications.

*Target Group* - Select the target group Amazon ECS will register the tasks to the service to.

*Path Pattern* - The load balancer will use path-based routing. Accept the default / or provide a different pattern. The path pattern is case-sensitive, can be up to 128 characters in length, and contains a select set of characters.

*Health Check Path* - The ping path that is the destination on the targets for health checks. By default, it is /. Enter a different path if needed. If the path you enter is invalid, the health check will fail and it will be considered unhealthy.

If you deploy multiple services, and each service will be deployed to a different path or location, you will need custom check paths.
### Task Definition page

**Task Definition** - Select an existing task definition or choose **Create New** and type in the new task definition name.

**Container** - Select an existing container or choose **Create New** and type in the new container name.

**Task Role** - Select an IAM role that has the credentials your app needs to access AWS Services. This is how credentials are passed in to your application. See how to specify AWS security credentials for your application.

**Task Execution Role** - Select a role with permissions to pull private images and publish logs. AWS Fargate will use it on your behalf.

**Port Mapping** - Choose the port number on the container that is bound to the automatically assigned host port.

**Environment Variables** - Add, modify, or delete environment variables for the container. You can modify it to suit your deployment.

When you are satisfied with the configuration, click **Publish** to begin the deployment process.
Events are displayed during deployment. The wizard is automatically closed on successful completion. You can override this by unchecking the box at the bottom of the page.

You can find the URL of your new instances in the AWS Explorer. Expand Amazon ECS and Clusters, then click on your cluster.

**Deploying an ASP.NET Core 2.0 App to Amazon ECS (EC2)**

This section describes how to use the Publish Container to AWS wizard, provided as part of the Toolkit for Visual Studio, to deploy a containerized ASP.NET Core 2.0 application targeting Linux through Amazon ECS using the EC2 launch type. Because a web application is meant run continuously, it will deployed as a service.

**Before you publish your container**

Before using the Publish Container to AWS to deploy your ASP.NET Core 2.0 application:

- Specify your AWS credentials (p. 69) and get setup with Amazon ECS.
- Install Docker. You have a few different installation options including Docker for Windows.
- Create an Amazon ECS cluster based on the needs of your web application. It only takes a few steps.
- In Visual Studio, create (or open) a project for an ASP.NET Core 2.0 containerized app targeting Linux.

**Accessing the Publish Container to AWS wizard**

To deploy an ASP.NET Core 2.0 containerized application targeting Linux, right-click the project in the Solution Explorer and select Publish Container to AWS.

You can also select Publish Container to AWS on the Visual Studio Build menu.
**Publish Container to AWS Wizard**

**Account profile to use** - Select an account profile to use.

**Region** - Choose a deployment region. Profile and region are used to set up your deployment environment resources and select the default Docker registry.

**Configuration** - Select the Docker image build configuration.

**Docker Repository** - Choose an existing Docker repository or type in the name of a new repository and it will be created. This is the repository the built container image is pushed to.

**Tag** - Select an existing tag or type in the name of a new tag. Tags can track important details like version, options or other unique configuration elements of the Docker container.

**Deployment** - Select Service on an ECS Cluster. Use this deployment option when your application is meant to be long-running (like an ASP.NET Core 2.0 web application).

**Save settings to aws-docker-tools-defaults.json and configure project for command line deployment** - Check this option if you want the flexibility of deploying from the command line. Use `dotnet ecs deploy` from your project directory to deploy and `dotnet ecs publish` the container.

**Launch Configuration page**

**ECS Cluster** - Pick the cluster that will run your Docker image. You can create an ECS cluster using the AWS Management Console.

**Launch Type** - Choose EC2. To use the Fargate launch type, see Deploying an ASP.NET Core 2.0 Application to Amazon ECS (Fargate).

**Service Configuration page**

**Service** - Select one of the services in the drop-down to deploy your container into an existing service. Or choose Create New to create a new service. Service names must be unique within a cluster, but you can have similarly named services in multiple clusters within a region or across multiple regions.

**Number of Tasks** - The number of tasks to deploy and keep running on your cluster. Each task is one instance of your container.

**Minimum Healthy Percent** - The percentage of tasks that must remain in **RUNNING** state during a deployment rounded up to the nearest integer.

**Maximum Percent** - The percentage of tasks that are allowed in the **RUNNING** or **PENDING** state during a deployment rounded down to the nearest integer.

**Placement Templates** - Select a task placement template.

When you launch a task into a cluster, Amazon ECS must determine where to place the task based on the requirements specified in the task definition, such as CPU and memory. Similarly, when you scale down the task count, Amazon ECS must determine which tasks to terminate.

The placement template controls how tasks are launched into a cluster:

- **AZ Balanced Spread** - distribute tasks across Availability Zones and across container instances in the Availability Zone.
- **AZ Balanced BinPack** - distribute tasks across Availability Zones and across container instances with the least available memory.
- **BinPack** - distribute tasks based on the least available amount of CPU or memory.
- **One Task Per Host** - place, at most, one task from the service on each container instance.
For more information, see Amazon ECS Task Placement.

**Application Load Balancer page**

**Configure Application Load Balancer** - Check to configure an application load balancer.

**Select IAM role for service** - Select an existing role or choose Create New and a new role will be created.

**Load Balancer** - Select an existing load balancer or choose Create New and type in the name for the new load balancer.

**Listener Port** - Select an existing listener port or choose Create New and type in a port number. The default, port 80, is appropriate for most web applications.

**Target Group** - By default, the load balancer sends requests to registered targets using the port and protocol that you specified for the target group. You can override this port when you register each target with the target group.

**Path Pattern** - The load balancer will use path-based routing. Accept the default / or provide a different pattern. The path pattern is case-sensitive, can be up to 128 characters in length, and contains a select set of characters.

**Health Check Path** - The ping path that is the destination on the targets for health checks. By default, it is / and is appropriate for web applications. Enter a different path if needed. If the path you enter is invalid, the health check will fail and it will be considered unhealthy.

If you deploy multiple services, and each service will be deployed to a different path or location, you might need custom check paths.

**ECS Task Definition page**

**Task Definition** - Select an existing task definition or choose Create New and type in the new task definition name.

**Container** - Select an existing container or choose Create New and type in the new container name.

**Memory (MiB)** - Provide values for Soft Limit or Hard Limit or both.

The soft limit (in MiB) of memory to reserve for the container. Docker attempts to keep the container memory under the soft limit. The container can consume more memory, up to either the hard limit specified with the memory parameter (if applicable), or all of the available memory on the container instance, whichever comes first.

The hard limit (in MiB) of memory to present to the container. If your container attempts to exceed the memory specified here, the container is killed.

**Task Role** - Select a task role for an IAM role that allows the container permission to call the AWS APIs that are specified in its associated policies on your behalf. This is how credentials are passed in to your application. See how to specify AWS security credentials for your application.

**Port Mapping** - Add, modify or delete port mappings for the container. If a load balancer is on, the host port will be default to 0 and port assignment will be dynamic.

**Environment Variables** - Add, modify, or delete environment variables for the container.

When you are satisfied with the configuration, click Publish to begin the deployment process.

**Publishing Container to AWS**

Events are displayed during deployment. The wizard is automatically closed on successful completion. You can override this by unchecking the box at the bottom of the page.
You can find the URL of your new instances in the AWS Explorer. Expand Amazon ECS and Clusters, then click on your cluster.

**Standalone Deployment Tool**

**Note**

Standalone Deployment Tool options related to AWS CloudFormation deployments and incremental deployments to Elastic Beanstalk are obsolete in the current version and should not be used. For information about using the preferred Publish to Elastic Beanstalk wizard, see Deploying to Elastic Beanstalk (p. 30).

The Toolkit for Visual Studio includes a command line tool that provides the same functionality as the deployment wizard. You can use the standalone deployment tool in your build pipeline or in other scripts to automate deployments to Elastic Beanstalk.

The deployment tool supports both initial deployments and redeployments. If you used the deployment tool to deploy your application, you can use the deployment wizard in Visual Studio to redeploy it, and vice versa.

The deployment tool consumes a configuration file that specifies parameter values for the deployment. If you used the deployment wizard in Visual Studio to deploy your application, you can generate a configuration file either from AWS Explorer or the last step in the wizard.

**Note**

Because the deployment configuration includes the credentials that were used for deployment, you should keep the configuration file in a secure location.

To deploy your web application with the deployment tool, package the application in a .zip file. For more information about how to package your application for deployment, go to How to: Create a Web Deployment Package in Visual Studio on MSDN.
Deployment Tool Installation and Invocation

The deployment tool is typically installed in the following directory:

```
C:\Program Files\AWS Tools\Deployment Tool\awsdeploy.exe
```

Or, on Microsoft Windows 64-bit system, in the following directory:

```
C:\Program Files (x86)\AWS Tools\Deployment Tool\awsdeploy.exe
```

Invocation Syntax

```
awsdeploy [options] configFile
```

The configuration file must be the last item specified on the command line.

Command line options can be specified using a forward slash (/) or hyphen (-).

Except for the `D` option, each command line option has a long form and a single letter abbreviation. For example, you can specify silent mode in any of the following ways.

```
/s
-s
/silent
-silent
```

Other command line options follow a similar form.

The following table shows the available command line options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/s, /silent, -s, -silent</code></td>
<td>Do not output messages to the console.</td>
</tr>
<tr>
<td><code>/v, /verbose, -v, -verbose</code></td>
<td>Send more detailed information about the deployment to the console.</td>
</tr>
<tr>
<td><code>/r, /redeploy, -r, -redeploy</code></td>
<td>Do not create stack. Deploy to existing stack. This option does not change the AWS CloudFormation configuration.</td>
</tr>
<tr>
<td><code>/u, /updateStack, -u, -updateStack</code></td>
<td>Update the AWS CloudFormation configuration for an existing deployment. Do not redeploy the application. ** (Obsolete. Do not use.) **</td>
</tr>
<tr>
<td><code>/w, /wait, -w, -wait</code></td>
<td>Block until deployment is complete. This option is useful for scripts that need to take some action after the deployment is complete.</td>
</tr>
<tr>
<td><code>/l &lt;logfile&gt;, /log &lt;logfile&gt;, -l &lt;logfile&gt;, -log &lt;logfile&gt;</code></td>
<td>Log debugging information to the specified log file.</td>
</tr>
<tr>
<td><code>/D&lt;key&gt;=&lt;value&gt;, -D&lt;key&gt;=&lt;value&gt;</code></td>
<td>Override a configuration setting from the command line. For more information, see the section of the configuration file.</td>
</tr>
</tbody>
</table>
Output and Exit Codes

Warnings and errors are output to the console. If the log option is specified, additional logging output is sent to the log file.

The deployment tool uses the following exit codes.

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Success</td>
</tr>
<tr>
<td>1</td>
<td>Invalid argument</td>
</tr>
<tr>
<td>3</td>
<td>Failed deployment</td>
</tr>
</tbody>
</table>

If the deployment is successful, the deployment tool will output the URL for the deployed application.

Configuration File Samples

You use a configuration file to specify the action of the deployment tool. The Toolkit for Visual Studio includes three sample configuration files:

- Elastic Beanstalk deployment
- AWS CloudFormation single instance deployment
- AWS CloudFormation load-balanced deployment

Sample Web App

A sample web app (in a .zip file archive) that you can deploy using the deployment tool is also included in the Toolkit for Visual Studio. You can find these files in the Samples subdirectory of the directory where the deployment tool is installed.

You can use the \D command line option to override settings in the configuration file:

/D<key>=<value>

or

-D<key>=<value>

You can specify the \D option multiple times to override multiple configuration file settings. If you repeat the same key with different values on the command line, the deployment tool will use the last value specified.

Deployment Tool Configuration File Format

The configuration files provide the same information you would specify in the deployment wizard. The formatting of the configuration files divides the configuration into sections that correspond to the pages in the deployment wizard.

Elastic Beanstalk Deployment Configuration File

The following configuration parameters are for deployments using Elastic Beanstalk.
For a walkthrough of the use of the standalone deployment tool to deploy to Elastic Beanstalk, go to the [Developer Guide](#).

**General Settings**

/\Daws:autoscaling:launchconfiguration.SecurityGroups=RDPOnly,HTTPOnly

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeploymentPackage = archive.zip</td>
<td>Relative path to the web deployment archive. This path is relative to your working directory (that is, the directory from which you invoke the deployment tool).</td>
</tr>
<tr>
<td>IncrementalPushLocation</td>
<td>(Obsolete: Do not use) If specified, incremental deployment is enabled. The value specifies a location (for example, C:\Temp\VS2008App1) where a Git repository will be created to store the versioned contents of the deployment package.</td>
</tr>
<tr>
<td>Template = ElasticBeanstalk</td>
<td>Can be Elastic Beanstalk or ElasticBeanstalk.</td>
</tr>
<tr>
<td>Application.Name</td>
<td>Specifies a name for the application. This value is required.</td>
</tr>
<tr>
<td>Application.Description</td>
<td>Specifies an optional description for the application.</td>
</tr>
<tr>
<td>Application.Version</td>
<td>Specifies a version string for the application. If you are using incremental deployment, this value is ignored. Elastic Beanstalk uses the Git commit ID for the version string.</td>
</tr>
<tr>
<td>Region = us-east-1</td>
<td>Target <a href="#">Regions and Endpoints</a>.</td>
</tr>
<tr>
<td>UploadBucket = awsdeployment-us-east-1-samples</td>
<td>Amazon S3 bucket where the deployment materials will be stored. If this bucket doesn't exist, it will be created. If you use the deployment wizard, it generates the bucket name for you.</td>
</tr>
<tr>
<td>KeyPair = default</td>
<td>Amazon EC2 key pair for signing in to the instance. The key pair must exist before deployment. (The deployment wizard allows you to create the key pair during deployment.)</td>
</tr>
<tr>
<td>AWSAccessKey = DEPLOYMENT_CREDENTIALS_HERE</td>
<td>AWS access key and secret key used to create the stack and deploy the application to Elastic Beanstalk. We do not recommend using these parameters to specify credentials. Instead, create a profile for the credentials and use AWSProfileName to reference the profile. For more information, see creds.</td>
</tr>
<tr>
<td>AWSSecretKey = DEPLOYMENT_CREDENTIALS_HERE</td>
<td></td>
</tr>
<tr>
<td>AWSProfileName = {profile_name}</td>
<td>The profile used to create the stack and deploy the application to Elastic Beanstalk.</td>
</tr>
</tbody>
</table>
### Key and Value

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws:autoscaling:launchconfiguration.SecurityGroups = default</td>
<td>The names of the security groups for the Amazon EC2 instance. If you specify multiple security groups, separate them with commas.</td>
</tr>
<tr>
<td>/Daws:autoscaling:launchconfiguration.SecurityGroups=RDPOnly,HTTPOnly</td>
<td>The security groups must already exist and must allow ingress on port 80 (HTTP). For information about how to create security groups, see tkv-sg</td>
</tr>
</tbody>
</table>

### Environment Settings

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment.Name</td>
<td>Specifies a name for your Elastic Beanstalk environment. This value is required.</td>
</tr>
<tr>
<td>Environment.Description</td>
<td>Optional. Specifies a description for your environment.</td>
</tr>
<tr>
<td>Environment.CNAME</td>
<td>Optional. Specifies the URL prefix for your application. If you do not specify this value, Elastic Beanstalk will derive the prefix from your environment name.</td>
</tr>
</tbody>
</table>

### Container Settings

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container.TargetRuntime = 4.0</td>
<td>Specifies the target runtime for the .NET Framework. Possible values are 2.0 or 4.0. The following .NET Framework versions are mapped to a target runtime of 2.0:</td>
</tr>
<tr>
<td>• .NET Framework 2.0</td>
<td></td>
</tr>
<tr>
<td>• .NET Framework 3.0</td>
<td></td>
</tr>
<tr>
<td>• .NET Framework 3.5</td>
<td></td>
</tr>
<tr>
<td>The following .NET Framework versions are mapped to a target runtime of 4.0:</td>
<td></td>
</tr>
<tr>
<td>• .NET Framework 4.0</td>
<td></td>
</tr>
<tr>
<td>• .NET Framework 4.5</td>
<td></td>
</tr>
</tbody>
</table>

The deployment wizard (p. 30) in the Toolkit for Visual Studio allows you to specify the .NET Framework version. The wizard then maps the .NET Framework version to the appropriate target runtime version.
### AWS Toolkit for Visual Studio User Guide

**Standalone Deployment Tool**

#### Key and Value

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Container.Enable32BitApplications = false</td>
<td>If the application is 32-bit, specify true. If the application is 64-bit, specify false.</td>
</tr>
<tr>
<td>Container.ApplicationHealthcheckPath = /</td>
<td>This URL is relative to the root server URL. For example, if the full URL is example.com/site-is-up.html, you would type /site-is-up.html. The setting applies only when you use the load balanced template. It is ignored when you use the single instance template. The responsiveness of the application at this URL affects into the actions taken by the load balancer and auto scaler. If the application is unresponsive or responds slowly, the load balancer will direct incoming network traffic to other Amazon EC2 instances, and the auto scaler may add additional Amazon EC2 instances.</td>
</tr>
<tr>
<td>Container.InstanceType = t1.micro</td>
<td>The type of Amazon EC2 instance to use. The Micro instance shown here is the EC2 Pricing type of instance.</td>
</tr>
<tr>
<td>Container.AmiID</td>
<td>Specifies a custom Amazon Machine Image (AMI). For more information about how to create a custom AMI, go to Using Custom AMIs in the AWS Elastic Beanstalk Developer Guide and Create an AMI from an Amazon EC2 Instance (p. 23).</td>
</tr>
<tr>
<td>Container.NotificationEmail</td>
<td>Optional. Specifies an email address for deployment status notifications.</td>
</tr>
</tbody>
</table>

#### AWS CloudFormation Deployment Configuration File

**Note**

Deployments to AWS CloudFormation using the Standalone Deployment Tool are deprecated.

The following configuration parameters are taken from the load balanced template.

**General Settings**

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeploymentPackage = archive.zip</td>
<td>Relative path to the web deployment archive. This path is relative to your working directory (that is, the directory from which you invoke the deployment tool). If you are updating a deployment (/updateStack switch), this parameter is ignored.</td>
</tr>
<tr>
<td>Region = us-east-1</td>
<td>Target region.</td>
</tr>
<tr>
<td>Template = LoadBalanced</td>
<td>The value for Template can be SingleInstance or LoadBalanced or a file path to a custom AWS CloudFormation template. For more information, see Customizing</td>
</tr>
</tbody>
</table>
**Key and Value** | **Description**
--- | ---
UploadBucket = awsdeployment-us-east-1-samples | Amazon Simple Storage Service (Amazon S3) bucket where the deployment materials will be stored. If the bucket doesn't exist, it will be created. If you use the deployment wizard, it generates this bucket name for you. If you used the wizard for a deployment and are redeploying, this parameter will be ignored. The deployment tool automatically uses the bucket that was used in the original deployment from the wizard.

KeyPair = default | Amazon Elastic Compute Cloud (Amazon EC2) key pair for signing in to the instance. The key pair must exist before deployment. (The deployment wizard allows you to create the key pair during deployment.)

AWSAccessKey = DEPLOYMENT_CREDENTIALS_HERE
AWSSecretKey = DEPLOYMENT_CREDENTIALS_HERE | The AWS access key and secret key used to create the stack and deploy the application to AWS CloudFormation. We do not recommend using these parameters to specify credentials. Instead, create a profile for the credentials and use AWSProfileName to reference the profile. For more information, see creds.

AWSProfileName = {profile_name} | The profile used to create the stack and deploy the application to AWS CloudFormation.

### Template Parameters

In addition to the following parameters, the load balanced template supports numerous other parameters to customize load balancing and Auto Scaling behavior.

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template.InstanceType = t1.micro</td>
<td>The type of Amazon EC2 instance to use. The Micro instance shown here is the least expensive type of instance.</td>
</tr>
<tr>
<td>Template.SecurityGroup = default</td>
<td>The security group for the Amazon EC2 instance. This security group must have already been created and must allow ingress on port 80 (HTTP). For information about how to create a security groups, see tkv-sg.</td>
</tr>
<tr>
<td>Environment.PARAM1 = Environment.PARAM2 = Environment.PARAM3 = Environment.PARAM4 = Environment.PARAM5 =</td>
<td>These values are made available to the deployed application through the appSettings in the Web.config file. For more information, go to the MSDN library.</td>
</tr>
</tbody>
</table>
| Environment.AWSAccessKey = APP_CREDENTIALS_HERE
Environment.AWSSecretKey = APP_CREDENTIALS_HERE | The access key and secret key used by the deployed application to access AWS services. We do not recommend using these parameters to specify credentials. Instead, create a profile for |
Key and Value | Description
--- | ---
| AWSProfileName = {profile_name} | The profile used by the deployed application to access AWS services.

**Container Settings**

| SolutionStack="64bit Windows Server 2008 R2 running IIS 7.5" |

| SolutionStack="64bit Windows Server 2012 running IIS 8" |

Key and Value | Description
--- | ---
| SolutionStack="64bit Windows Server 2012 running IIS 8" | Specifies the version of Windows Server and Internet Information Services (IIS) to which to deploy. Valid values are: SolutionStack="64bit Windows Server 2008 R2 running IIS 7.5" or SolutionStack="64bit Windows Server 2012 running IIS 8" If not specified, the default is 64bit Windows Server 2012 running IIS 8.0. You can use Container.Type as an alias for SolutionStack.

| Container.TargetRuntime = 4.0 | Specifies the target runtime for the .NET Framework. Possible values are 2.0 or 4.0.

**The following .NET Framework versions are mapped to a target runtime of 2.0:**

- .NET Framework 2.0
- .NET Framework 3.0
- .NET Framework 3.5

**The following .NET Framework versions are mapped to a target runtime of 4.0:**

- .NET Framework 4.0
- .NET Framework 4.5

The deployment wizard (p. 60) in the Toolkit for Visual Studio allows you to specify the .NET Framework version. The wizard then maps the .NET Framework version to the appropriate target runtime version.

| Container.Enable32BitApplications = false | If the application is 32-bit, specify true. If the application is 64-bit, specify false.

| Container.ApplicationHealthcheckPath = / | This URL is relative to the root server URL. For example, if the full URL is example.com/site-is-up.html, you would type /site-is-up.html. |
### AWS Toolkit for Visual Studio User Guide

**Standalone Deployment Tool**

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Settings.SNSTopic</td>
<td>SNS topic to use for deployment messages.</td>
</tr>
<tr>
<td>Settings.CreationTimeout = 0</td>
<td>The amount of time to allow for the creation of the stack. A value of zero means there is no time limit.</td>
</tr>
<tr>
<td>Settings.RollbackOnFailure = false</td>
<td>If this value is true, the deployment tool tears down the stack if the deployment fails.</td>
</tr>
</tbody>
</table>

### Stack Creation Settings

<table>
<thead>
<tr>
<th>Key and Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Settings.SNSTopic</td>
<td>SNS topic to use for deployment messages.</td>
</tr>
<tr>
<td>Settings.CreationTimeout = 0</td>
<td>The amount of time to allow for the creation of the stack. A value of zero means there is no time limit.</td>
</tr>
<tr>
<td>Settings.RollbackOnFailure = false</td>
<td>If this value is true, the deployment tool tears down the stack if the deployment fails.</td>
</tr>
</tbody>
</table>

### How to Update the Configuration for an Existing Deployment

You can use the `updateStack` feature of the deployment tool to modify the AWS CloudFormation configuration of an existing deployment. This configuration—the application's environment—includes the cloud resources your application runs on and has access to. The `updateStack` feature does not change or redeploy the application; it only updates the application's environment. In this way, the `updateStack` feature complements the redeployment feature. Redeployment provides a way to update your application without changing the environment.

There are various scenarios in which you might use `updateStack`. For example, if you develop your application using the single instance template, as the application nears production readiness, you could update its configuration to use a load balanced template, either for public beta testing or live release deployment. In a related scenario, a deployment using a load-balanced configuration could be optimized by modifying some of the configuration parameters—for example, by increasing the maximum number of supporting EC2 instances or changing the size of the instances, say from micro to large. You can use the `updateStack` feature of the deployment tool to implement either of these scenarios.

There are scenarios in which you might use both the `/updateStack` option and the `/redeploy` option, effectively modifying both the application itself and the environment in which the application is running. In some cases, this approach is more efficient than just performing a regular deployment. For example, you might change your environment to add an Amazon S3 bucket and then update your application to use that bucket. With a combination of `/updateStack` and `/redeploy`, you could implement both changes, but leave any already provisioned Amazon EC2 instances up and running. A regular deployment would result in all of the environment being taken down and rebuilt.

The `updateStack` feature is available only through the deployment tool. It is not available through the deployment wizard in Visual Studio. You can use `updateStack` to update a deployment that was initially deployed through the deployment wizard, but not vice versa.

The invocation syntax for updating a deployment is similar to the syntax for a new deployment.
Keep the following in mind when you attempt to update a deployment:

- You cannot update a deployment that is in the process of being created or taken down.
- The specified config file must use the same value for the `StackName` parameter as the original deployment.
- You cannot use `updateStack` to change the region for your deployment. However, you can change the Availability Zones for your deployment.
- If you use `updateStack` to transition your deployment from single instance to load balanced, the endpoint for your deployment will necessarily change. In the single instance case, the endpoint refers to an Amazon EC2 instance. In the load balanced template, the endpoint refers to the Elastic Load Balancing load balancer, a computer that distributes computing load across all EC2 instances. Therefore, if you are using a CNAME record to associate a domain name with your deployment, you should update the CNAME record so that it points to the load balancer of the load balanced template.

The deployment tool implements the `updateStack` feature by calling the AWS CloudFormation `UpdateStack` API. For more information about AWS CloudFormation, go to the AWS CloudFormation User Guide.

**Customizing the AWS CloudFormation Template Used for Deployment**

In addition to modifying a deployment by specifying parameters in the deployment wizard—or in the configuration file for the standalone deployment tool—you can also modify the deployment by providing your own custom AWS CloudFormation template. By default, the deployment automatically uses one of a set of templates that are stored in Amazon Simple Storage Service (Amazon S3). This default set of templates includes two templates for each AWS region. One of these two is for deployment to a single Amazon Elastic Compute Cloud (Amazon EC2) instance; the other is for deployment to a load-balanced set of Amazon EC2 instances. You can use these templates as a starting point for creating your own.

**To create your own custom template**

1. Copy the template that corresponds to your region and the type of deployment that you want to do. Links to each of the templates is provided below.

   **Note**
   Templates are available only for the regions listed below.

   **US East (N. Virginia)**

<table>
<thead>
<tr>
<th>SingleInstance.template</th>
<th>LoadBalanced.template</th>
</tr>
</thead>
</table>

   **US West (Oregon)**

<table>
<thead>
<tr>
<th>SingleInstance-us-west-2.template</th>
<th>LoadBalanced-us-west-2.template</th>
</tr>
</thead>
</table>

   **US West (N. California)**
If you need to create your own links to the templates, the format for each link is as follows:

http://vstoolkit.amazonwebservices.com/CloudFormationTemplates/{template-name}

For example, for the single instance template for the US West (N. California) region, the link would be:

http://vstoolkit.amazonwebservices.com/CloudFormationTemplates/SingleInstance-us-west-1.template

The links in the table show the HTTP protocol. The HTTPS protocol is also supported.

1. Edit the template to modify it for your specific needs. The templates are text files, so you can edit them with any standard text editor. The deployment information in the templates is represented in JavaScript Object Notation format. After editing the file, it's wise to revalidate the JSON formatting using a tool such as JSONLint.

The template file has three sections: Parameters, Resources, and Outputs.

To add resources to your deployment, add them to the Resources section of the template. For example, you could add an Amazon RDS database or an Amazon SNS topic. To configure these resources at deployment time, add parameters to the Parameters section of the template. When you add new parameters to the template, the AWS Toolkit adds them to the parameters that are displayed.
in the deployment wizard. You can specify values for these parameters either in the deployment wizard or in the config file for the standalone deployment tool.

Similarly, data that you specify in the Output section of the template is also displayed in the deployment wizard—as well as in the AWS Management Console. You can use the Output section to display post-deployment information about your resources. For example, if you add an Amazon S3 bucket to the Resources section of the template, you can use the Outputs section to display the autogenerated name for the bucket.

For more information about editing AWS CloudFormation templates, go to the CloudFormation User Guide.

2. Set the Template parameter in the deployment configuration file to the path to your customized template. The Template parameter is located under General Settings in the config file. The path that you specify could be the path to the file on your local hard drive or it could be a URL that points to the location of the configuration file on a remote server. When you next run a deployment, the tool will use your template.

Required Data in the Template File

The deployment process requires that certain data be specified in the template file. While editing your version of the template, you must ensure that it continues to provide this data. The required data is located only in the Parameters and Outputs sections of the template.

Parameters Section of Template

The following table shows the required parameters in the Parameters section of the template.

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstanceType</td>
<td>The &quot;API name&quot; for the type of the Amazon EC2 instances to use for the deployment. Examples are t1.micro for Micro instances or m1.xlarge for Extra Large instances. For a list of instance types and corresponding API names, see the Amazon EC2 detail page.</td>
</tr>
<tr>
<td>KeyPair</td>
<td>Which of your key pairs to use for the Amazon EC2 instances.</td>
</tr>
<tr>
<td>Security Group</td>
<td>The security group to use for the Amazon EC2 instances.</td>
</tr>
<tr>
<td>BucketName</td>
<td>Amazon S3 bucket where the deployment files are uploaded.</td>
</tr>
<tr>
<td>ConfigFile</td>
<td>Name of the config file that the deployment uses.</td>
</tr>
<tr>
<td>AmazonMachineImage</td>
<td>The Amazon Machine Image (AMI) that is used for the deployment. For more information about how to create a custom AMI, go to Using Custom AMIs in the Elastic Beanstalk Developer Guide and Create an AMI from an Amazon EC2 Instance (p. 23). Note that the Host Manager software that is installed on AMIs that are used in CloudFormation deployments is now auto-updating. Therefore, if you derive a custom AMI from one of the CloudFormation AMIs, you do</td>
</tr>
</tbody>
</table>
### AWS Toolkit for Visual Studio User Guide

#### Using the AWS CloudFormation Template Editor for Visual Studio

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>not need to maintain the Host Manager software. However, you still need to keep the operating system and application software up to date.</td>
<td></td>
</tr>
<tr>
<td>UserData</td>
<td>The user data that the deployment provides to the deployed application.</td>
</tr>
</tbody>
</table>

#### Outputs Section of Template

The following table shows the required outputs in the Outputs section of the template.

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bucket</td>
<td>The Amazon S3 bucket to which the deployment files were uploaded.</td>
</tr>
<tr>
<td>ConfigFile</td>
<td>The name of the configuration file that was used for the deployment.</td>
</tr>
<tr>
<td>VSToolkitDeployed</td>
<td>Boolean flag set to true, which indicates that this stack was created as part of a deployment from the AWS Toolkit for Visual Studio. This flag is also set to true if the deployment is done from the standalone deployment tool.</td>
</tr>
<tr>
<td>URL</td>
<td>The URL for the deployed application.</td>
</tr>
</tbody>
</table>

### Using the AWS CloudFormation Template Editor for Visual Studio

The Toolkit for Visual Studio includes an AWS CloudFormation template editor and AWS CloudFormation template projects for Visual Studio. The supported features include:

- Creating new templates (either empty or copied from an existing stack or sample template) using the supplied AWS CloudFormation template project type.
- Editing templates with automatic JSON validation, auto-completion, code folding, and syntax highlighting.
- Automatic suggestion of intrinsic functions and resource reference parameters for the field values in your template.
- Menu items to perform common actions for your template from Visual Studio: deploying the template, estimating the cost of your template, and formatting your template.

#### Topics

- Creating an AWS CloudFormation Template Project in Visual Studio (p. 93)
- Deploying a AWS CloudFormation Template in Visual Studio (p. 95)
- Estimating the Cost of Your AWS CloudFormation Template Project in Visual Studio (p. 97)
- Formatting a AWS CloudFormation Template in Visual Studio (p. 99)
Creating an AWS CloudFormation Template Project in Visual Studio

To create a template project

1. In Visual Studio, choose File, choose New, and then choose Project.
2. For Visual Studio 2017:

   In the New Project dialog box, expand Installed and select AWS.

   ![New Project dialog box](image)

For Visual Studio 2019:

   In the New Project dialog box, ensure that the Language, Platform, and Project type drop-down boxes are set to "All ..." and type aws in the Search field.
3. Select the **AWS CloudFormation Project** template.

4. **For Visual Studio 2017:**

   Enter the desired **Name**, **Location**, etc., for your template project, and then click **OK**.

   **For Visual Studio 2019:**

   Click **Next**. In the next dialog, enter the desired **Name**, **Location**, etc., for your template project, and then click **Create**.

5. On the **Select Project Source** page, choose the source of the template you will create:
   
   - **Create with empty template** generates a new, empty AWS CloudFormation template.
   - **Create from existing AWS [CFN] stack** generates a template from an existing stack in your AWS account. (The stack doesn't need to have a status of **CREATE_COMPLETE**.)
   - **Select sample template** generates a template from one of the AWS CloudFormation sample templates.
Deploying a AWS CloudFormation Template in Visual Studio

**To deploy an CFN template**

1. In Solution Explorer, open the context (right-click) menu for the template you want to deploy, and choose **Deploy to AWS CloudFormation**.
Alternatively, to deploy the template you’re currently editing, from the **Template** menu, choose **Deploy to AWS CloudFormation**.

2. On the **Deploy Template** page, choose the AWS account to use to launch the stack and the region where it will be launched.
3. Choose Create New Stack and type a name for your stack.

4. Choose any (or none) of the following options:
   - To receive notifications about the stack's progress, from the SNS Topic drop-down list, choose an SNS topic. You can also create an SNS topic by choosing Create New Topic and typing an email address in the box.
   - Use Creation Timeout to specify how long AWS CloudFormation should allow for the stack to be created before it is declared failed (and rolled back, unless the Rollback on failure option is cleared).
   - Use Rollback on failure if you want the stack to roll back (that is, delete itself) on failure. Leave this option cleared if you would like the stack to remain active for debugging purposes, even if it has failed to complete the launch.

5. Choose Finish to launch the stack.

Estimating the Cost of Your AWS CloudFormation Template Project in Visual Studio

With the Toolkit for Visual Studio, you can easily estimate the cost of the AWS CloudFormation stack you are working on before you deploy it. This way you'll have an idea of the monthly operating costs for the resources include in your template.

To estimate the cost of your CFN stack

1. In Solution Explorer, open the context (right-click) menu for the template and choose Estimate Cost.
Alternatively, to estimate the cost of the template you’re currently editing, from the **Template** menu, choose **Estimate Cost**.

2. Provide values for parameters you have defined for your stack, and choose **Finish**.

3. The **AWS Simple Monthly Calculator** will be displayed. The values for the form data will be filled in with information pulled from the template you’re editing. You can adjust the values, if needed.

   The **Estimate of Your Monthly Bill** tab will display an itemized view of the estimated monthly costs of running your stack.
Formatting a AWS CloudFormation Template in Visual Studio

- In Solution Explorer, open the context (right-click) menu for the template and choose Format Template.

Alternatively, to format the template you're currently editing, from the Template menu, choose Format Template.

Your JSON code will be formatted so that its structure is clearly presented.
Amazon Simple Storage Service (Amazon S3) enables you to store and retrieve data from any connection to the Internet. All data you store on Amazon S3 is associated with your account and, by default, can only be accessed by you. The Toolkit for Visual Studio enables you to store data on Amazon S3 and to view, manage, retrieve, and distribute that data.

Amazon S3 uses the concept of buckets, which you can think of as being similar to file systems or logical drives. Buckets can contain folders, which are similar to directories, and objects, which are similar to files. In this section, we’ll be using these concepts as we walk through the Amazon S3 functionality exposed by the Toolkit for Visual Studio.

**Note**
To use this tool, your IAM policy must grant permissions for the `s3:GetBucketAcl`, `s3:GetBucket`, and `s3:ListBucket` actions. For more information, see [Overview of AWS IAM Policies](#).

### Creating an Amazon S3 Bucket

The bucket is most fundamental unit of storage in Amazon S3.

To create an S3 bucket
1. In AWS Explorer, open the context (right-click) menu for the Amazon S3 node, and then choose Create Bucket.
2. In the Create Bucket dialog box, type a name for the bucket. Bucket names must be unique across AWS. For information about other constraints, go to the Amazon S3 documentation.
3. Choose OK.

Managing Amazon S3 Buckets from AWS Explorer

In AWS Explorer, the following operations are available when you open a context (right-click) menu for an Amazon S3 bucket.

Browse

Displays a view of the objects contained in the bucket. From here, you can create folders or upload files or entire directories and folders from your local computer. The lower pane displays status messages about the upload process. To clear these messages, choose the Clear icon. You can also access this view of the bucket by double-clicking the bucket name in AWS Explorer.

Properties

Displays a dialog box where you can do the following:

- Set Amazon S3 permissions that scope to:
  - you as the bucket owner.
  - all users who have been authenticated on AWS.
  - everyone with Internet access.
• Turn on logging for the bucket.
• Set up a notification using the Amazon Simple Notification Service (Amazon SNS) so that if you are using Reduced Redundancy Storage (RRS), you are notified if data loss occurs. RRS is an Amazon S3 storage option that provides less durability than standard storage, but at reduced cost. For more information, see S3 FAQs.
• Create a static website using the data in the bucket.

Policy

Enables you to set up AWS Identity and Access Management (IAM) policies for your bucket. For more information, go to the IAM documentation and the use cases for IAM and S3.

Create Pre-Signed URL

Enables you to generate a time-limited URL you can distribute to provide access to the contents of the bucket. For more information, see How to Create a Pre-Signed URL (p. 106).

View Multi-Part Uploads

Enables you to view multipart uploads. Amazon S3 supports breaking large object uploads into parts to make the upload process more efficient. For more information, go to the discussion of multipart uploads in the S3 documentation.

Delete

Enables you to delete the bucket. You can only delete empty buckets.

Uploading Files and Folders to Amazon S3

You can use AWS Explorer to transfer files or entire folders from your local computer to any of your buckets.

Note
If you upload files or folders that have the same name as files or folders that already exist in the Amazon S3 bucket, your uploaded files will overwrite the existing files without warning.

To upload a file to S3

1. In AWS Explorer, expand the Amazon S3 node, and double-click a bucket or open the context (right-click) menu for the bucket and choose Browse.
2. In the Browse view of your bucket, choose Upload File or Upload Folder.
3. In the File-Open dialog box, navigate to the files to upload, choose them, and then choose Open. If you are uploading a folder, navigate to and choose that folder, and then choose Open.

The Upload Settings dialog box enables you to set metadata and permissions on the files or folder you are uploading. Selecting the Make everything public check box is equivalent to setting Open/Download permissions to Everyone. You can select the option to use Reduced Redundancy Storage for the uploaded files.
Amazon S3 File Operations from AWS Toolkit for Visual Studio

If you choose a file in the Amazon S3 view and open the context (right-click) menu, you can perform various operations on the file.
Create Folder

Enables you to create a folder in the current bucket. (Equivalent to choosing the Create Folder link.)

Upload

Enables you to upload files or folders. (Equivalent to choosing the Upload File or Upload Folder links.)

Open

Attempts to open the selected file in your default browser. Depending on the type of file and your default browser's capabilities, the file might not be displayed. It might simply be downloaded by your browser instead.

Download

Opens a Folder-Tree dialog box to enable you to download the selected file.
**Make Public**

Sets permissions on the selected file to **Open/Download** and **Everyone**. (Equivalent to selecting the **Make everything public** check box on the **Upload Settings** dialog box.)

**Delete**

Deletes the selected files or folders. You can also delete files or folders by choosing them and pressing Delete.

**Change Storage Class**

Sets the storage class to either **Standard** or **Reduced Redundancy Storage (RRS)**. To view the current storage class setting, choose **Properties**.

**Change Encryption**

Enables you to set server-side encryption on the file. To view the current encryption setting, choose **Properties**.

**Rename**

Enables you to rename a file. You cannot rename a folder.

**Cut | Copy | Paste**

Enables you to cut, copy, and paste files or folders between folders or between buckets.

**Properties**

Displays a dialog box that enables you to set metadata and permissions for the file, as well as toggle storage for the file between Reduced Redundancy Storage (RRS) and Standard, and set server-side encryption for the file. This dialog box also displays an https link to the file. If you choose this link, the Toolkit for Visual Studio opens the file in your default browser. If you have permissions on the file set to **Open/Download** and **Everyone**, other people will be able to access the file through this link. Rather than distributing this link, we recommend you create and distribute pre-signed URLs.
Create Pre-Signed URL

Enables you to create a time-limited pre-signed URL that you can distribute to enable other people to access the content you have stored on Amazon S3.

How to Create a Pre-Signed URL

You can create a pre-signed URL for a bucket or files in a bucket. Other people can then use this URL to access the bucket or file. The URL will expire after a period of time that you specify when you create the URL.

To create a pre-signed URL

1. In the Create Pre-Signed URL dialog box, set the expiration date and time for the URL. The default setting is one hour from the current time.
2. Choose the Generate button.
3. To copy the URL to the clipboard, choose Copy.
Amazon DynamoDB is a fast, highly scalable, highly available, cost-effective, non-relational database service. DynamoDB removes traditional scalability limitations on data storage while maintaining low latency and predictable performance. The Toolkit for Visual Studio provides functionality for working with DynamoDB in a development context. For more information about DynamoDB, see DynamoDB on the AWS website.

In the Toolkit for Visual Studio, AWS Explorer displays all of the DynamoDB tables associated with the active AWS account.
Creating an DynamoDB Table

You can use the Toolkit for Visual Studio to create a DynamoDB table.

To create a table in AWS Explorer

1. In AWS Explorer, open the context (right-click) menu for Amazon DynamoDB, and then choose Create Table.

2. In the Create Table wizard, in Table Name, type a name for the table.

3. In the Hash Key Name field, type a primary hash key attribute and from the Hash Key Type buttons, choose the hash key type. DynamoDB builds an unordered hash index using the primary key attribute and an optional sorted range index using the range primary key attribute. For more information about the primary hash key attribute, go to the Primary Key section in the Amazon DynamoDB Developer Guide.

4. (Optional) Select Enable Range Key. In the Range Key Name field, type a range key attribute, and then from the Range Key Type buttons, choose a range key type.

5. In the Read Capacity field, type the number of read capacity units. In the Write Capacity field, type the number of write capacity units. You must specify a minimum of three read capacity units and five write capacity units. For more information about read and write capacity units, go to Provisioned Throughput in DynamoDB.

6. (Optional) Select Enable Basic Alarm to alert you when your table's request rates are too high. Choose the percentage of provisioned throughput per 60 minutes that must be exceeded before the alert is sent. In Send Notifications To, type an email address.

7. Click OK to create the table.
Viewing an DynamoDB Table as a Grid

To open a grid view of one of your DynamoDB tables, in AWS Explorer, double-click the subnode that corresponds to the table. From the grid view, you can view the items, attributes, and values stored in the table. Each row corresponds to an item in the table. The table columns correspond to attributes. Each cell of the table holds the values associated with that attribute for that item.

An attribute can have a value that is a string or a number. Some attributes have a value that consists of a set of strings or numbers. Set values are displayed as a comma-separated list enclosed by square brackets.

For more information about DynamoDB tables, go to Data Model Concepts - Tables, Items, and Attributes.
Editing and Adding Attributes and Values

By double-clicking a cell, you can edit the values for the item’s corresponding attribute. For set-value attributes, you can also add or delete individual values from the set.

In addition to changing the value of an attribute, you can also, with some limitations, change the format of the value for an attribute. For example, any number value can be converted into a string value. If you have a string value, the content of which is a number, such as 125, the cell editor enables you to convert the format of the value from string to number. You can also convert a single-value to a set-value. However, you cannot generally convert from a set-value to a single-value; an exception is when the set-value has, in fact, only one element in the set.

After editing the attribute value, choose the green check mark to confirm your changes. If you want to discard your changes, choose the red X.
After you have confirmed your changes, the attribute value will be displayed in red. This indicates the attribute has been updated, but that the new value has not been written back to the DynamoDB database. To write your changes back to DynamoDB, choose **Commit Changes**. To discard your changes, choose **Scan Table** and when the Toolkit asks if you would like to commit your changes before the Scan, choose **No**.

### Adding an Attribute

From the grid view, you can also add attributes to the table. To add a new attribute, choose **Add Attribute**.

In the **Add Attribute** dialog box, type a name for your attribute, and then choose **OK**.

To make the new attribute become part of the table, you must add a value to it for at least one item and then choose the **Commit Changes** button. To discard the new attribute, just close the grid view of the table without choosing **Commit Changes**.

### Scanning an DynamoDB Table

---

111
You can perform Scans on your DynamoDB tables from the Toolkit. In a Scan, you define a set of criteria and the Scan returns all items from the table that match your criteria. Scans are expensive operations and should be used with care to avoid disrupting higher priority production traffic on the table. For more information about using the Scan operation, go to the Amazon DynamoDB Developer Guide.

To perform a Scan on a DynamoDB table from AWS Explorer

1. In the grid view, choose the scan conditions: add button.
2. In the Scan clause editor, choose the attribute to match against, how the value of the attribute should be interpreted (string, number, set value), how it should be matched (for example Begins With or Contains), and the literal value it should match.
3. Add more Scan clauses, as needed, for your search. The Scan will return only those items that match the criteria from all of your Scan clauses. The Scan will perform a case-sensitive comparison when matching against string values.
4. On the button bar at the top of the grid view, choose Scan Table.

To remove a Scan clause, choose the red button with the white line to the right of each clause.

Paginating Scan Results

At the bottom of the view are three buttons.

The first two blue buttons provide pagination for Scan results. The first button will display an additional page of results. The second button will display an additional ten pages of results. In this context, a page is equal to 1 MB of content.

Export Scan Result to CSV

The third button exports the results from the current Scan to a CSV file.
Using AWS CodeCommit with Visual Studio Team Explorer

You can use AWS Identity and Access Management (IAM) user accounts to create Git credentials and use them to create and clone repositories from within Team Explorer.

**Credential Types for AWS CodeCommit**

Most AWS Toolkit for Visual Studio users are aware of setting up AWS credential profiles that contain their access and secret keys. These credential profiles are used in the Toolkit for Visual Studio to enable the calls to service APIs, for example, to list Amazon S3 buckets in AWS Explorer or to launch an Amazon EC2 instance. The integration of AWS CodeCommit with Team Explorer also uses these credential profiles. However, to work with Git itself you need additional credentials, specifically, Git credentials for HTTPS connections. You can read about these credentials (a user name and password) at Setup for HTTPS Users Using Git Credentials in the AWS CodeCommit User Guide.

You can create the Git credentials for AWS CodeCommit only for IAM user accounts. You cannot create them for a root account. You can create up to two sets of these credentials for the service and, although you can mark a set of credentials as inactive, inactive sets still count toward your limit of two sets. Note that you can delete and recreate credentials at any time. When you use AWS CodeCommit from within Visual Studio, your traditional AWS credentials are used for working with the service itself, for example, when you're creating and listing repositories. When working with the actual Git repositories hosted in AWS CodeCommit, you use the Git credentials.

As part of the support for AWS CodeCommit, the Toolkit for Visual Studio automatically creates and manages these Git credentials for you and associates them with your AWS credential profile. You don't need to be concerned about having the right set of credentials at hand to perform Git operations within Team Explorer. Once you connect to Team Explorer with your AWS credential profile, the associated Git credentials are used automatically whenever you work with a Git remote.

**Connecting to AWS CodeCommit**

When you open the Team Explorer window in Visual Studio 2015 or later, you'll see an AWS CodeCommit entry in the Hosted Service Providers section of Manage Connections.

Choosing **Sign up** opens the AWS home page in a browser window. What happens when you choose **Connect** depends on whether the Toolkit for Visual Studio can find a credential profile with AWS access and secret keys to enable it to make calls to AWS on your behalf. You might have set up a credential profile by using the new Getting Started page that displays in the IDE when the Toolkit for Visual Studio cannot find any locally stored credentials. Or you might have been using the Toolkit for Visual Studio, the AWS Tools for Windows PowerShell, or the AWS CLI and already have AWS credential profiles available for the Toolkit for Visual Studio to use.

When you choose **Connect**, the Toolkit for Visual Studio starts the process to find a credential profile to use in the connection. If the Toolkit for Visual Studio can't find a credential profile, it opens a dialog
box that invites you to enter the access and secret keys for your AWS account. We strongly recommend that you use an IAM user account, and not your root credentials. In addition, as noted earlier, the Git credentials you eventually need can only be created for IAM users. Once the access and secret keys are provided and the credential profile is created, the connection between Team Explorer and AWS CodeCommit is ready for use.

If the Toolkit for Visual Studio finds more than one AWS credential profile, you’re prompted to select the account you want to use within Team Explorer.

If you have only one credential profile, the Toolkit for Visual Studio bypasses the profile selection dialog box and you’re connected immediately:

When a connection is established between Team Explorer and AWS CodeCommit via your credential profiles, the invitation dialog box closes and the connection panel is displayed.

Because you have no repositories cloned locally, the panel shows just the operations you can perform: Clone, Create, and Sign out. Like other providers, AWS CodeCommit in Team Explorer can be bound to only a single AWS credential profile at any given time. To switch accounts, you use Sign out to remove the connection so you can start a new connection using a different account.

Now that you have established a connection, you can create a repository by clicking the Create link.

**Creating a Repository**

When you click the Create link, the Create a New AWS CodeCommit Repository dialog box opens.
AWS CodeCommit repositories are organized by region, so in **Region** you can select the region in which to host the repository. The list has all the regions in which AWS CodeCommit is supported. You provide the Name (required) and Description (optional) for our new repository.

The default behavior of the dialog box is to suffix the folder location for the new repository with the repository name (as you enter the name, the folder location also updates). To use a different folder name, edit the **Clone into** folder path after you finish entering the repository name.

You can also choose to automatically create an initial `.gitignore` file for the repository. The AWS Toolkit for Visual Studio provides a built-in default for Visual Studio file types. You can also choose to have no file or to use a custom existing file that you would like to reuse across repositories. Simply select **Use custom** in the list and navigate to the custom file to use.

Once you have a repository name and location, you are ready to click **OK** and start creating the repository. The Toolkit for Visual Studio requests that the service create the repository and then clone the new repository locally, adding an initial commit for the `.gitignore` file, if you're using one. It's at this point that you start working with the Git remote, so the Toolkit for Visual Studio now needs access to the Git credentials described earlier.

**Setting up Git Credentials**

To this point you've been using AWS access and secret keys to request that the service create your repository. Now you need to work with Git itself to do the actual clone operation, and Git doesn't understand AWS access and secret keys. Instead, you need to supply the user name and password credentials to Git to use on an HTTPS connection with the remote.

As noted in Setting up Git credentials (p. 115), the Git credentials you're going to use must be associated with an IAM user. You cannot generate them for root credentials. You should always set up your AWS credential profiles to contain IAM user access and secret keys, and not root keys. The Toolkit for Visual Studio can attempt to set up Git credentials for AWS CodeCommit for you, and associate them with the AWS credential profile that you used to connect in Team Explorer earlier.

When you choose **OK** in the **Create a New AWS CodeCommit Repository** dialog box and successfully create the repository, the Toolkit for Visual Studio checks the AWS credential profile that is connected in Team Explorer to determine if Git credentials for AWS CodeCommit exist and are associated locally with the profile. If so, the Toolkit for Visual Studio instructs Team Explorer to commence the clone operation on the new repository. If Git credentials are not available locally, the Toolkit for Visual Studio checks the
type of account credentials that were used in the connection in Team Explorer. If the credentials are for an IAM user, as we recommend, the following message is shown.

![Auto-create Git Credentials](image)

If the credentials are root credentials, the following message is shown instead.

![Auto-create Git Credentials](image)

In both cases, the Toolkit for Visual Studio offers to attempt to do the work to create the necessary Git credentials for you. In the first scenario, all it needs to create are a set of Git credentials for the IAM user. When a root account is in use, the Toolkit for Visual Studio first attempts to create an IAM user and then proceeds to create Git credentials for that new user. If the Toolkit for Visual Studio has to create a new user, it applies the AWS CodeCommit Power User managed policy to that new user account. This policy allows access only to AWS CodeCommit and enables all operations to be performed with AWS CodeCommit except for repository deletion.

When you're creating credentials, you can only view them once. Therefore, the Toolkit for Visual Studio prompts you to save the newly created credentials as a .csv file before continuing.
This is something we also strongly recommend, and be sure to save them to a secure location!

There might be cases where the Toolkit for Visual Studio can’t automatically create credentials. For example, you may already have created the maximum number of sets of Git credentials for AWS CodeCommit (two), or you might not have sufficient programmatic rights for the Toolkit for Visual Studio to do the work for you (if you’re signed in as an IAM user). In these cases, you can log into the AWS Management Console to manage the credentials or obtain them from your administrator. You can then enter them in the **Git Credentials for AWS CodeCommit** dialog box, which the Toolkit for Visual Studio displays.

Now that the credentials for Git are available, the clone operation for the new repository proceeds (see progress indication for the operation inside Team Explorer). If you elected to have a default `.gitignore` file applied, it is committed to the repository with a comment of ‘Initial Commit’.

That’s all there is to setting up credentials and creating a repository within Team Explorer. Once the required credentials are in place, all you see when creating new repositories in the future is the **Create a New AWS CodeCommit Repository** dialog box itself.
Cloning a Repository

To clone an existing repository, return to the connection panel for AWS CodeCommit in Team Explorer. Click the **Clone** link to open the **Clone AWS CodeCommit Repository** dialog box, and then select the repository to clone and the location on disk where you want to place it.

Once you choose the region, the Toolkit for Visual Studio queries the service to discover the repositories that are available in that region and displays them in the central list portion of the dialog box. The name and optional description of each repository are also displayed. You can reorder the list to sort it by either repository name or the last modified date, and to sort each in ascending or descending order.

Once you select the repository you can choose the location to clone to. This defaults to the same repository location used in other plugins to Team Explorer, but you can browse to or enter any other location. By default, the repository name is suffixed onto the selected path. However, if you want a specific path, simply edit the text box after you select the folder. Whatever text is in the box when you click **OK** will be the folder in which you will find the cloned repository.

Having selected the repository and a folder location, you then click **OK** to proceed with the clone operation. Just as with creating a repository, you can see the progress of the clone operation reported in Team Explorer.

Working with Repositories

When you clone or create repositories, notice that the local repositories for the connection are listed in the connection panel in Team Explorer under the operation links. These entries give you a convenient way to access the repository to browse content. Simply right-click the repository and choose **Browse in Console**.
You can also use **Update Git Credentials** to update the stored Git credentials associated with the credential profile. This is useful if you've rotated the credentials. The command opens the **Git Credentials for AWS CodeCommit** dialog box where you can enter or import the new credentials.

Git operations on the repositories work as you'd expect. You can make local commits and, when you are ready to share, you use the Sync option in Team Explorer. Because the Git credentials are already stored locally and associated with our connected AWS credential profile, we won't be prompted to supply them again for operations against the AWS CodeCommit remote.

### Amazon RDS from AWS Explorer

Amazon Relational Database Service (Amazon RDS) is a service that enables you to provision and manage SQL relational database systems in the cloud. Amazon RDS supports three types of database systems:

- MySQL Community Edition
- Oracle Database Enterprise Edition
- Microsoft SQL Server (Express, Standard, or Web Editions)

For more information, see the [Amazon RDS User Guide](#).

A lot of the functionality discussed here is also available through the [AWS Management Console](#) for Amazon RDS.

**Topics**

- Launch an Amazon RDS Database Instance (p. 119)
- Create a Microsoft SQL Server Database in an RDS Instance (p. 125)
- Amazon RDS Security Groups (p. 127)

### Launch an Amazon RDS Database Instance

With AWS Explorer, you can launch an instance of any of the database engines supported by Amazon RDS. The following walkthrough shows the user experience for launching an instance of Microsoft SQL Server Standard Edition, but the user experience is similar for all supported engines.

**To launch an Amazon RDS instance**

1. In AWS Explorer, open the context (right-click) menu for the **Amazon RDS** node and choose **Launch DB Instance**.
Alternatively, on the DB Instances tab, choose Launch DB Instance.

2. In the DB Engine Selection dialog box, choose the type of database engine to launch. For this walkthrough, choose Microsoft SQL Server Standard Edition (sqlserver-se), and then choose Next.
3. In the **DB Engine Instance Options** dialog box, choose configuration options.

   In the **DB Engine Instance Options and Class** section, you can specify the following settings.

   **License Model**

<table>
<thead>
<tr>
<th>Engine Type</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft SQL Server</td>
<td>license-included</td>
</tr>
<tr>
<td>MySql</td>
<td>general-public-license</td>
</tr>
<tr>
<td>Oracle</td>
<td>bring-your-own-license</td>
</tr>
</tbody>
</table>

   The license model varies, depending on the type of database engine. Engine Type License Microsoft SQL Server license-included MySql general-public-license Oracle bring-your-own-license

   **DB Instance Version**

   Choose the version of the database engine you would like to use. If only one version is supported, it is selected for you.

   **DB Instance Class**

   Choose the instance class for the database engine. Pricing for instance classes varies. For more information, see Amazon RDS Pricing.
**Perform a multi AZ deployment**

Select this option to create a multi-AZ deployment for enhanced data durability and availability. Amazon RDS provisions and maintains a standby copy of your database in a different Availability Zone for automatic failover in the event of a scheduled or unplanned outage. For information about pricing for multi-AZ deployments, see the pricing section of the Amazon RDS detail page. This option is not supported for Microsoft SQL Server.

**Upgrade minor versions automatically**

Select this option to have AWS automatically perform minor version updates on your RDS instances for you.

In the RDS Database Instance section, you can specify the following settings.

**Allocated Storage**

<table>
<thead>
<tr>
<th>Engine</th>
<th>Minimum (GB)</th>
<th>Maximum (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL</td>
<td>5</td>
<td>1024</td>
</tr>
<tr>
<td>Oracle Enterprise Edition</td>
<td>10</td>
<td>1024</td>
</tr>
<tr>
<td>Microsoft SQL Server Express Edition</td>
<td>30</td>
<td>1024</td>
</tr>
<tr>
<td>Microsoft SQL Server Standard Edition</td>
<td>250</td>
<td>1024</td>
</tr>
<tr>
<td>Microsoft SQL Server Web Edition</td>
<td>30</td>
<td>1024</td>
</tr>
</tbody>
</table>


**DB Instance Identifier**

Specify a name for the database instance. This name is not case-sensitive. It will be displayed in lowercase form in AWS Explorer.

**Master User Name**

Type a name for the administrator of the database instance.

**Master User Password**

Type a password for the administrator of the database instance.

**Confirm Password**

Type the password again to verify it is correct.
1. In the **Additional Options** dialog box, you can specify the following settings.

**Database Port**

This is the TCP port the instance will use to communicate on the network. If your computer accesses the Internet through a firewall, set this value to a port through which your firewall allows traffic.

**Availability Zone**

Use this option if you want the instance to be launched in a particular Availability Zone in your region. The database instance you have specified might not be available in all Availability Zones in a given region.

**RDS Security Group**

Select an RDS security group (or groups) to associate with your instance. RDS security groups specify the IP address, Amazon EC2 instances, and AWS accounts that are allowed to access your instance. For more information about RDS security groups, see Amazon RDS Security Groups (p. 127). The Toolkit for Visual Studio attempts to determine your current IP address and provides the option to add this address to the security groups associated with your instance. However, if your computer accesses the Internet through a firewall, the IP address the Toolkit generates for your computer may not be accurate. To determine which IP address to use, consult your system administrator.
**DB Parameter Group**

(Optional) From this drop-down list, choose a DB parameter group to associate with your instance. DB parameter groups enable you to change the default configuration for the instance. For more information, go to the Amazon Relational Database Service User Guide and this article.

When you have specified settings on this dialog box, choose **Next**.

2. The **Backup and Maintenance** dialog box enables you to specify whether Amazon RDS should back up your instance and if so, for how long the backup should be retained. You can also specify a window of time during which the backups should occur.

This dialog box also enables you to specify if you would like Amazon RDS to perform system maintenance on your instance. Maintenance includes routine patches and minor version upgrades.

The window of time you specify for system maintenance cannot overlap with the window specified for backups.

Choose **Next**.
Create a Microsoft SQL Server Database in an RDS Instance

Microsoft SQL Server is designed in such a way that, after launching an Amazon RDS instance, you need to create an SQL Server database in the RDS instance.

For information about how to create an Amazon RDS instance, see Launch an Amazon RDS Database Instance (p. 119).

To create a Microsoft SQL Server database

1. In AWS Explorer, open the context (right-click) menu for the node that corresponds to your RDS instance for Microsoft SQL Server, and choose Create SQL Server Database.
2. In the **Create SQL Server Database** dialog box, type the password you specified when you created the RDS instance, type a name for the Microsoft SQL Server database, and then choose **OK**.
3. The Toolkit for Visual Studio creates the Microsoft SQL Server database and adds it to the Visual Studio Server Explorer.

![Server Explorer](image)

Amazon RDS Security Groups

Amazon RDS security groups enable you to manage network access to your Amazon RDS instances. With security groups, you specify sets of IP addresses using CIDR notation, and only network traffic originating from these addresses is recognized by your Amazon RDS instance.

Although they function in a similar way, Amazon RDS security groups are different from Amazon EC2 security groups. It is possible to add an EC2 security group to your RDS security group. Any EC2 instances that are members of the EC2 security group are then able to access the RDS instances that are members of the RDS security group.

For more information about Amazon RDS security groups, go to the RDS Security Groups. For more information about Amazon EC2 security groups, go to the EC2 User Guide.

Create an Amazon RDS Security Group

You can use the Toolkit for Visual Studio to create an RDS security group. If you use the AWS Toolkit to launch an RDS instance, the wizard will allow you to specify an RDS security group to use with your instance. You can use the following procedure to create that security group before you start the wizard.

To create an Amazon RDS security group

1. In AWS Explorer, expand the Amazon RDS node, open the context (right-click) menu for the DB Security Groups subnode and choose Create.
Alternatively, on the Security Groups tab, choose Create Security Group. If this tab isn’t displayed, open the context (right-click) menu for the DB Security Groups subnode and choose View.

2. In the Create Security Group dialog box, type a name and description for the security group, and then choose OK.

Set Access Permissions for an Amazon RDS Security Group

By default, a new Amazon RDS security group provides no network access. To enable access to Amazon RDS instances that use the security group, use the following procedure to set its access permissions.
To set access for an Amazon RDS security group

1. On the Security Groups tab, choose the security group from the list view. If your security group does not appear in the list, choose Refresh. If your security group still does not appear in the list, verify you are viewing the list for the correct AWS region. Security Group tabs in the AWS Toolkit are region-specific.

   If no Security Group tabs appear, in AWS Explorer, open the context (right-click) menu for the DB Security Groups subnode and choose View.

2. Choose Add Permission.

   ![Add Permissions button on the Security Groups tab](image)

   Add Permissions button on the Security Groups tab

3. In the Add Permission dialog box, you can use CIDR notation to specify which IP addresses can access your RDS instance, or you can specify which EC2 security groups can access your RDS instance. When you choose EC2 Security Group, you can specify access for all EC2 instances associated with an AWS account have access, or you can choose a EC2 security group from the drop-down list.

   ![Add Permission dialog box](image)

   ![Add Permission dialog box](image)
The AWS Toolkit attempts to determine your IP address and auto-populate the dialog box with the appropriate CIDR specification. However, if your computer accesses the Internet through a firewall, the CIDR determined by the Toolkit may not be accurate.

Using Amazon SimpleDB from AWS Explorer

AWS Explorer displays all of the Amazon SimpleDB domains associated with the active AWS account. From AWS Explorer, you can create or delete Amazon SimpleDB domains.

Create, delete, or open Amazon SimpleDB domains associated with your account

Executing Queries and Editing the Results

AWS Explorer can also display a grid view of a Amazon SimpleDB domain from which you can view the items, attributes, and values in that domain. You can execute queries so that only a subset of the domain's items is displayed. By double-clicking a cell, you can edit the values for that item's corresponding attribute. You can also add new attributes to the domain.

The domain displayed here is from the Amazon SimpleDB sample included with the AWS SDK for .NET.

Amazon SimpleDB grid view

To execute a query, edit the query in the text box at the top of the grid view, and then choose Execute. The view is filtered to show only the items that match the query.

Execute query from AWS Explorer

To edit the values associated with an attribute, double-click the corresponding cell, edit the values, and then choose Commit Changes.

Adding an Attribute
To add an attribute, at the top of the view, choose **Add Attribute**.

**Add Attribute** dialog box

To make the attribute part of the domain, you must add a value for it to at least one item and then choose **Commit Changes**.

Commit changes for a new attribute

**Paginating Query Results**

There are three buttons at the bottom of the view.

Paginate and export buttons

The first two buttons provide pagination for query results. To display an additional page of results, choose the first button. To display an additional ten pages of results, choose the second button. In this context, a page is equal to 100 rows or the number of results specified by the LIMIT value, if it is included in the query.

**Export to CSV**

The last button exports the current results to a CSV file.

### Using Amazon SQS from AWS Explorer

Amazon Simple Queue Service (Amazon SQS) is a flexible queue service that enables message passing between different processes of execution in a software application. Amazon SQS queues are located in the AWS infrastructure, but the processes that are passing messages can be located locally, on Amazon EC2 instances, or on some combination of these. Amazon SQS is ideal for coordinating the distribution of work across multiple computers.

The Toolkit for Visual Studio enables you to view Amazon SQS queues associated with the active account, create and delete queues, and send messages through queues. (By active account, we mean the account selected in AWS Explorer.)
For more information about Amazon SQS, go to Introduction to SQS in the AWS documentation.

Creating a Queue

You can create an Amazon SQS queue from AWS Explorer. The ARN and URL for the queue will be based on the account number for the active account and the queue name you specify at creation.

To create a queue

1. In AWS Explorer, open the context (right-click) menu for the Amazon SQS node, and then choose Create Queue.
2. In the Create Queue dialog box, specify the queue name, the default visibility timeout, and the default delivery delay. The default visibility timeout and the default delivery delay are specified in seconds. The default visibility timeout is the amount of time that a message will be invisible to potential receiving processes after a given process has acquired the message. The default delivery delay is the amount of time from the moment the message is sent to the moment it first becomes visible to potential receiving processes.
3. Choose OK. The new queue will appear as a subnode under the Amazon SQS node.

Deleting a Queue

You can delete existing queues from AWS Explorer. If you delete a queue, any messages associated with the queue are no longer available.

To delete a queue

1. In AWS Explorer, open the context (right-click) menus for the queue you want to delete, and then choose Delete.

Managing Queue Properties

You can view and edit the properties for any of the queues displayed in AWS Explorer. You can also send messages to the queue from this properties view.

To manage queue properties

- In AWS Explorer, open the context (right-click) menu for the queue whose properties you want to manage, and then choose View Queue.

From the queue properties view, you can edit the visibility timeout, the maximum message size, message retention period, and default delivery delay. The default delivery delay can be overridden when you send a message. In the following screenshot, the obscured text is the account number component of the queue ARN and URL.
Sending a Message to a Queue

From the queue properties view, you can send a message to the queue.

To send a message

1. At the top of the queue properties view, choose the Send button.
2. Type the message. (Optional) Enter a delivery delay that will override the default delivery delay for the queue. In the following example, we have overridden the delay with a value of 240 seconds. Choose OK.
3. Wait for approximately 240 seconds (four minutes). The message will appear in the Message Sampling section of the queue properties view.
Identity and Access Management

AWS Identity and Access Management (IAM) enables you to more securely manage access to your AWS accounts and resources. With IAM, you can create multiple users in your primary (root) AWS account. These users can have their own credentials: password, access key ID, and secret key, but all IAM users share a single account number.

You can manage each IAM user’s level of resource access by attaching IAM policies to the user. For example, you can attach a policy to an IAM user that gives the user access to the Amazon S3 service and related resources in your account, but which doesn't provide access to any other services or resources.

For more efficient access management, you can create IAM groups, which are collections of users. When you attach a policy to the group, it will affect all users who are members of that group.

In addition to managing permissions at the user and group level, IAM also supports the concept of IAM roles. Like users and groups, you can attach policies to IAM roles. You can then associate the IAM role with an Amazon EC2 instance. Applications that run on the EC2 instance are able to access AWS using the permissions provided by the IAM role. For more information about using IAM roles with the Toolkit, see Create an IAM Role (p. 139). For more information about IAM, go to the IAM User Guide.

Create and Configure an IAM User

IAM users enable you to grant others access to your AWS account. Because you are able to attach policies to IAM users, you can precisely limit the resources an IAM user can access and the operations they can perform on those resources.

As a best practice, all users who access an AWS account should do so as IAM users—even the owner of the account. This ensures that if the credentials for one of the IAM users are compromised, just those credentials can be deactivated. There is no need to deactivate or change the root credentials for the account.
From the Toolkit for Visual Studio, you can assign permissions to an IAM user either by attaching an IAM policy to the user or by assigning the user to a group. IAM users who are assigned to a group derive their permissions from the policies attached to the group. For more information, see Create an IAM Group (p. 135) and Add an IAM User to an IAM Group (p. 136).

From the Toolkit for Visual Studio, you can also generate AWS credentials (access key ID and secret key) for the IAM user. For more information, see Generate Credentials for an IAM User (p. 137).

The Toolkit for Visual Studio supports specifying IAM user credentials for accessing services through AWS Explorer. Because IAM users typically do not have full access to all AWS services, some of the functionality in AWS Explorer might not be available. If you use AWS Explorer to change resources while the active account is an IAM user and then switch the active account to the root account, the changes might not be visible until you refresh the view in AWS Explorer. To refresh the view, choose the refresh () button.

For information about how to configure IAM users from the AWS Management Console, go to Working with Users and Groups in the IAM User Guide.

To create an IAM user

1. In AWS Explorer, expand the AWS Identity and Access Management node, open the context (right-click) menu for Users and then choose Create User.
2. In the Create User dialog box, type a name for the IAM user and choose OK. This is the IAM friendly name. For information about constraints on names for IAM users, go to the IAM User Guide.

The new user will appear as a subnode under Users under the AWS Identity and Access Management node.

For information about how to create a policy and attach it to the user, see Create an IAM Policy (p. 140).

Create an IAM Group

Groups provide a way of applying IAM policies to a collection of users. For information about how to manage IAM users and groups, go to Working with Users and Groups in the IAM User Guide.

To create an IAM group

1. In AWS Explorer, under Identity and Access Management, open the context (right-click) menu for Groups and choose Create Group.
2. In the Create Group dialog box, type a name for the IAM group and choose OK.
Add an IAM User to an IAM Group

IAM users who are members of an IAM group derive access permissions from the policies attached to the group. The purpose of an IAM group is to make it easier to manage permissions across a collection of IAM users.

For information about how the policies attached to an IAM group interact with the policies attached to IAM users who are members of that IAM group, go to Managing IAM Policies in the IAM User Guide.

In AWS Explorer, you add IAM users to IAM groups from the Users subnode, not the Groups subnode.

To add an IAM user to a IAM group

1. In AWS Explorer, under Identity and Access Management, open the context (right-click) menu for Users and choose Edit.
Assign an IAM user to a IAM group

2. The left pane of the **Groups** tab displays the available IAM groups. The right pane displays the groups of which the specified IAM user is already a member.

To add the IAM user to a group, in the left pane, choose the IAM group and then choose the > button.

To remove the IAM user from a group, in the right pane, choose the IAM group and then choose the < button.

To add the IAM user to all of the IAM groups, choose the >> button. Similarly, to remove the IAM user from all of the groups, choose the << button.

To choose multiple groups, choose them in sequence. You do not need to hold down the Control key. To clear a group from your selection, simply choose it a second time.

3. When you have finished assigning the IAM user to IAM groups, choose **Save**.

**Generate Credentials for an IAM User**

With Toolkit for Visual Studio, you can generate the access key ID and secret key used to make API calls to AWS. These keys can also be specified to access AWS services through the Toolkit. For more information about how to specify credentials for use with the Toolkit, see creds. For more information about how to safely handle credentials, see Best Practices for Managing AWS Access Keys.
The Toolkit cannot be used to generate a password for an IAM user.

To generate credentials for an IAM user

1. In AWS Explorer, open the context (right-click) menu for an IAM user and choose **Edit**.

2. To generate credentials, on the **Access Keys** tab, choose **Create**.

You can generate only two sets of credentials per IAM user. If you already have two sets of credentials and need to create an additional set, you must delete one of the existing sets.
If you want the Toolkit to save an encrypted copy of your secret access key to your local drive, select **Save the secret access key locally. AWS only returns the secret access key when created.** You can also copy the secret access key from the dialog box and save it in a secure location.

3. Choose **OK**.

After you generate the credentials, you can view them from the **Access Keys** tab. If you selected the option to have the Toolkit save the secret key locally, it will be displayed here.

Create credentials for IAM user

If you saved the secret key yourself and would also like the Toolkit to save it, in the **Secret Access Key** box, type the secret access key, and then select **Save the secret access key locally.**

To deactivate the credentials, choose **Make Inactive.** (You might do this if you suspect the credentials have been compromised. You can reactivate the credentials if you receive an assurance they are secure.)

## Create an IAM Role

The Toolkit for Visual Studio supports the creation and configuration of IAM roles. Just as with users and groups, you can attach policies to IAM roles. You can then associate the IAM role with an Amazon EC2 instance. The association with the EC2 instance is handled through an **instance profile**, which is a logical container for the role. Applications that run on the EC2 instance are automatically granted the level of access specified by the policy associated with the IAM role. This is true even when the application hasn't specified other AWS credentials.
For example, you can create a role and attach a policy to that role that limits access to Amazon S3 only. After associating this role with an EC2 instance, you can then run an application on that instance and the application will have access to Amazon S3, but not to any other services or resources. The advantage of this approach is that you don’t need to be concerned with securely transferring and storing AWS credentials on the EC2 instance.

For more information about IAM roles, go to Working with IAM Roles in the IAM User Guide. For examples of programs accessing AWS using the IAM role associated with an Amazon EC2 instance, go to the AWS developer guides for Java, .NET, PHP, and Ruby.

**Create an IAM role**

1. In AWS Explorer, under Identity and Access Management, open the context (right-click) menu for Roles and then choose Create Roles.
2. In the Create Role dialog box, type a name for the IAM role and choose OK.

The new IAM role will appear under Roles in Identity and Access Management.

For information about how to create a policy and attach it to the role, see Create an IAM Policy (p. 140).

**Create an IAM Policy**

Policies are fundamental to IAM. Policies can be associated with IAM entities such as users, groups, or roles. Policies specify the level of access enabled for a user, group, or role.

**To create an IAM policy**

In AWS Explorer, expand the AWS Identity and Access Management node, then expand the node for the type of entity (Groups, Roles, or Users) to which you will attach the policy. For example, open a context menu for an IAM role and choose Edit.

A tab associated with the role will appear in the AWS Explorer. Choose the Add Policy link.

In the New Policy Name dialog box, type a name for the policy (for example, s3-access).
New Policy Name dialog box

In the policy editor, add policy statements to specify the level of access to provide to the role (in this example, winapp-instance-role-2 associated with the policy. In this example, a policy provides full access to Amazon S3, but no access to any other resources.

Specify IAM policy

For more precise access control, you can expand the subnodes in the policy editor to allow or disallow actions associated with AWS services.

When you have edited the policy, choose the **Save** link.
Using the AWS Lambda Templates in the AWS Toolkit for Visual Studio

The AWS Toolkit for Visual Studio includes AWS Lambda .NET Core project templates for Visual Studio. Use the templates to quickly develop and deploy .NET Core-based C# Lambda functions. .NET Core is cross-platform, supporting Windows, macOS, and Linux, and can be used to develop device, cloud, and embedded applications.

For more information, see the following:

- For Microsoft .NET Core, see .NET Core.
- For .NET Core prerequisites and installation instructions for Windows, macOS, and Linux platforms, see .NET Core Downloads.
- For information about AWS Lambda functions, see What Is AWS Lambda?

Prerequisites

To do the following tutorials, you must first:

- Install the AWS Toolkit for Visual Studio and specify your credentials. See Setting Up the AWS Toolkit for Visual Studio (p. 3).

Topics

- Tutorial: Using the AWS Lambda Project in the AWS Toolkit for Visual Studio (p. 142)
- Tutorial: Build and Test a Serverless Application with AWS Lambda (p. 151)
- Tutorial: Creating an Amazon Rekognition Lambda Application (p. 160)
- Tutorial: Using Amazon Logging Frameworks with AWS Lambda to Create Application Logs (p. 167)

Tutorial: Using the AWS Lambda Project in the AWS Toolkit for Visual Studio

Using the AWS Lambda .NET Core project templates for Visual Studio you can easily create a AWS Lambda Function using Microsoft .NET Core.

For prerequisites and information about setting up the AWS Toolkit for Visual Studio, see Using the AWS Lambda Templates in the AWS Toolkit for Visual Studio (p. 142).

Create a Visual Studio .NET Core Lambda Project

2. For Visual Studio 2017:

   In the New Project dialog box, expand Installed, expand Visual C#, and select AWS Lambda.
For Visual Studio 2019:

In the **New Project** dialog box, ensure that the **Language**, **Platform**, and **Project type** drop-down boxes are set to "All ..." and type *aws lambda* in the **Search** field.
There are two types of project to choose from:

- AWS Lambda projects for creating a project to develop and deploy an individual Lambda function.
- AWS Serverless Applications projects for creating Lambda functions with a serverless AWS CloudFormation template. AWS serverless applications enable you to define more than just the function. For example, you can simultaneously create a database, add IAM roles, etc., with serverless deployment. AWS serverless applications also enable you to deploy multiple functions at one time.

3. Select the **AWS Lambda Project (.NET Core - C#)** template.

4. **For Visual Studio 2017:**

   Enter the desired **Name**, **Location**, etc., for your template project, then click **OK**.

   **For Visual Studio 2019:**

   Click **Next**. In the next dialog, enter the desired **Name**, **Location**, etc., for your template project, then click **Create**.

5. After you select the project type, choose a blueprint. For **AWS Lambda Project (.NET Core)**, the **Select Blueprint** page shows several Lambda function templates.
6. Choose the type of Lambda function you want to develop, and then choose Finish to create the Visual Studio project. You can now review the project's structure and code.

Review the Project Files

Examine the aws-lambda-tools-defaults.json file, which is created as part of your project. You can set the options in this file, which is read by the Lambda tooling by default. The project templates created in Visual Studio set many of these fields with default values. This is where the function handler is specified which is why you don't have to set it in the wizard. But if you rename the Function, Class or Assembly then you will need to update the field in the aws-lambda-tools-defaults.json file.

```json
{
    "profile":"default",
    "region": "us-east-2",
    "configuration": "Release",
    "framework": "netcoreapp1.0",
    "function-runtime": "dotnetcore1.0",
    "function-memory-size": 256,
    "function-timeout": 30,
    "function-handler": "BlogExample::BlogExample.Function::FunctionHandler"
}
```

When you use this aws-lambda-tools-defaults.json file, the only things left that the Lambda tooling needs to deploy the function are the name of the Lambda function and the IAM role.

Your project is now ready to publish to Lambda.

Publish to Lambda

To publish your function to Lambda:
1. In **Solution Explorer**, right-click the project, and then choose **Publish to AWS Lambda**.

2. On the **Upload Lambda Function** page, in **Function Name**, type a name for the function or select a previously published function to republish. Then choose **Next**.

3. In the **Advanced Function Details** page, set the fields as follows:
   
   - **Required**: Provide a **Role Name** for a role associated with your account. Choose an existing role or a new role based on an AWS managed policy or your own managed policy. The role is used to provide credentials for any AWS service calls made by the code in the function. Your account must have
permission to run the IAM ListPolicies action, or the **Role Name** list will be empty and you will be unable to continue.

- **Optional:** If your Lambda function accesses resources on an Amazon VPC, select the subnets and security groups.

- **Optional:** Set any environment variables that your Lambda function needs. The keys are automatically encrypted by the default service key (which is free) or you can specify an AWS KMS key (for which there is a charge). KMS is a managed service you can use to create and control the encryption keys used to encrypt your data. If you have an AWS KMS key, you can select it from the list.

4. Choose **Upload**.

5. The **Uploading Function** page is shown while the function is uploading, and automatically closes when the upload completes. To keep the wizard open so you can view the report, clear **Automatically close wizard on successful completion** at the bottom of the form before the upload completes. Close the page when you finish viewing the report.
6. After the function is uploaded, the **Function** page opens. Use the tabs on the left side of the page to test the function, add event sources, and view the log. Use the **Configuration** tab to add VPC subnets and security groups, memory, timeout, and environment variables.

7. To add event sources to establish a connection between an AWS resource (such as an Amazon S3 bucket, Amazon SNS topic, or Amazon Kinesis Data Streams streams) and a Lambda function, choose **Event Sources**. This will display the **Add Event Source** page.

On the **Add Event Source** page, from **Source Type**, choose the appropriate event source and choose **OK** to add the event source.
8. To test the function, in **Example Requests**, choose an example request.

9. To run the test, choose **Invoke**.
10. View the output from the test in **Log output**.

After your Lambda function is published, it's ready to use. For use cases, see **Examples of How to Use AWS Lambda**.

Lambda automatically monitors Lambda functions for you, reporting metrics through Amazon CloudWatch. To monitor and troubleshoot your function, see **Troubleshooting and Monitoring AWS Lambda Functions with Amazon CloudWatch**.
Tutorial: Build and Test a Serverless Application with AWS Lambda

You can build a serverless Lambda application by using an AWS Toolkit for Visual Studio template. The Lambda project templates include one for an **AWS Serverless Application**, which is the AWS Toolkit for Visual Studio implementation of the **AWS Serverless Application Model (AWS SAM)**. Using this project type you can develop a collection of AWS Lambda functions and deploy them with any necessary AWS resources as a whole application, using AWS CloudFormation to orchestrate the deployment.

For prerequisites and information about setting up the AWS Toolkit for Visual Studio, see Using the AWS Lambda Templates in the AWS Toolkit for Visual Studio (p. 142).

Topics
- Create a New AWS Serverless Application Project (p. 151)
- Examine the Files in the Serverless Application (p. 153)
- Deploy the Serverless Application (p. 156)

Create a New AWS Serverless Application Project

1. Open Visual Studio, and on the **File** menu, choose **New, Project**.
2. **For Visual Studio 2017**: In the **New Project** dialog box, expand **Installed**, expand **Visual C#**, and select **AWS Lambda**.

   ![New Project dialog box](image)

   For Visual Studio 2019:
In the **New Project** dialog box, ensure that the **Language**, **Platform**, and **Project type** drop-down boxes are set to "All ..." and type *aws lambda* in the **Search** field.

There are two types of project to choose from:
- AWS Lambda projects for creating a project to develop and deploy an individual Lambda function.
- AWS Serverless Applications projects for creating Lambda functions with a serverless AWS CloudFormation template. AWS serverless applications enable you to define more than just the function. For example, you can simultaneously create a database, add IAM roles, etc., with serverless deployment. AWS serverless applications also enable you to deploy multiple functions at one time.

3. Select the **AWS Serverless Application with Tests (.NET Core - C#)** template.

4. **For Visual Studio 2017:**
   
   Enter "Blogger" for the **Name**, enter the desired **Location**, etc., and then click **OK**.

   **For Visual Studio 2019:**
   
   Click **Next**. In the next dialog, enter "Blogger" for the **Name**, enter the desired **Location**, etc., and then click **Create**.

5. The **Select Blueprint** page shows several Lambda function templates.
6. Choose the Blog API using DynamoDB blueprint, and then choose Finish to create the Visual Studio project.

Examine the Files in the Serverless Application

Blog.cs

Blog.cs is a simple class used to represent the blog items that are stored in Amazon DynamoDB.

Functions.cs

Functions.cs defines the C# functions to expose as Lambda functions. There are four functions defined to manage a blog platform:

- GetBlogsAsync: gets a list of all the blogs.
- GetBlogAsync: gets a single blog identified by the query parameter ID or by the ID added to the URL resource path.
- AddBlogAsync: adds a blog to DynamoDB table.
- RemoveBlogAsync: removes a blog from the DynamoDB table.

Each of these functions accepts an APIGatewayProxyRequest object and returns an APIGatewayProxyResponse.

You expose these Lambda functions as HTTP APIs by using Amazon API Gateway. The APIGatewayProxyRequest contains all the information representing the HTTP request. The GetBlogAsync task finds the blog ID in the resource path or query string.

```csharp
public async Task GetBlogAsync(APIGatewayProxyRequest request, ILambdaContext context)
```
The default constructor for this class passes the name of the DynamoDB table storing the blogs as an environment variable. This environment variable is set when Lambda deploys the function.

```csharp
public Functions()
{
    // Check if a table name was passed in through environment variables and, if so, // add the table mapping
    var tableName = System.Environment.GetEnvironmentVariable(TABLENAMESPACE_ENVIRONMENT_VARIABLE_LOOKUP);
    if(!string.IsNullOrEmpty(tableName))
    {
        AWSConfigsDynamoDB.Context.TypeMappings[typeof(Blog)] = new Amazon.Util.TypeMapping(typeof(Blog), tableName);
    }

    var config = new DynamoDBContextConfig { Conversion = DynamoDBEntryConversion.V2 }; // Add the context configuration
    this.DDBContext = new DynamoDBContext(new AmazonDynamoDBClient(), config);
}
```

The `GetBlogs` declaration is similar to the function declarations.

```json
"GetBlogs": {
    "Type": "AWS::Serverless::Function",
    "Properties": {
        "Handler": "Blogger::Blogger.Functions::GetBlogsAsync",
        "Runtime": "dotnetcore1.0",
        "CodeUri": "",
        "Description": "Function to get a list of blogs",
        "MemorySize": 256,
    }
}
```
"Timeout": 30,
"Role": null,
"Policies": ["AWSLambdaFullAccess"],
"Environment": {
  "Variables": {
    "BlogTable": {
      "Fn::If": [
        "CreateBlogTable",
        {
          "Ref": "BlogTable"
        },
        {
          "Ref": "BlogTableName"
        }
      ]
    }
  },
  "Events": {
    "PutResource": {
      "Type": "Api",
      "Properties": {
        "Path": "/",
        "Method": "GET"
      }
    }
  }
}

Many of the fields are similar to those of a Lambda project deployment. In the `Environment` property, the name of the DynamoDB table is passed in as an environment variable. The `CodeUri` property tells DynamoDB where your application bundle is stored in Amazon S3. Leave this property blank. The toolkit fills it in during deployment, after it uploads the application bundle to S3 (it won't change the template file on disk when it does so). The `Events` section is where the HTTP bindings are defined for your Lambda function. This is all the API Gateway setup you need for your function. You can also set up other types of event sources in this section.

One of the benefits of using AWS CloudFormation to manage the deployment is you can also add and configure any other AWS resources necessary for your application in the template, and let DynamoDB take care of creating and deleting the resources.
Deploy the Serverless Application

Deploy the serverless application by right-clicking the project and choosing Publish to AWS Lambda.

This launches the deployment wizard, and because all the Lambda configuration was done in the serverless.template file, all you need to supply are the following:

- The name of the CloudFormation stack, which will be the container for all the resources declared in the template.
- The S3 bucket to upload your application bundle to.

These must exist in the same AWS Region.
Because the serverless template has parameters, an additional page is displayed in the wizard so you can specify the values for the parameters. You can leave the BlogTableName property blank and let CloudFormation generate a unique name for the table. You do need to set ShouldCreateTable to true so that DynamoDB will create the table. To use an existing table, enter the table name and set the ShouldCreateTable parameter to false. You can leave the other fields at their default values and choose Publish.
Once the publish step is complete, the CloudFormation stack view is displayed in AWS Explorer. This view shows the progress of the creation of all the resources declared in your serverless template.

When the stack creation is complete, the root URL for the API Gateway is displayed on the page. If you click that link, it returns an empty JSON array because you haven't added any blogs to the table. To get blogs in the table, you need to make an HTTP PUT method to this URL, passing in a JSON document that represents the blog. You can do that in code or in any number of tools. This example uses the Postman tool, which is a Chrome browser extension, but you can use any tool you like. In this tool, you set the URL
and change the method to PUT. In the **Body** tab, you put in some sample content. When you make the HTTP call, you can see the blog ID is returned.

Go back to the browser with the link to the AWS Serverless URL and you can see you are getting back the blog you just posted.
Tutorial: Creating an Amazon Rekognition Lambda Application

This tutorial shows you how to create a Lambda application that uses Amazon Rekognition to tag Amazon S3 objects with detected labels.

For prerequisites and information about setting up the AWS Toolkit for Visual Studio, see Using the AWS Lambda Templates in the AWS Toolkit for Visual Studio (p. 142).

Create a Visual Studio .NET Core Lambda Image Rekognition Project


2. For Visual Studio 2017:

   In the New Project dialog box, expand Installed, expand Visual C#, and select AWS Lambda.

   ![New Project dialog box with AWS Lambda selected]

For Visual Studio 2019:

   In the New Project dialog box, ensure that the Language, Platform, and Project type drop-down boxes are set to "All ..." and type aws lambda in the Search field.
3. Select the AWS Lambda Project with Tests (.NET Core - C#) template.

4. For Visual Studio 2017:

   Name the project "ImageRekognition", enter the desired Location, etc., and then click OK.

   For Visual Studio 2019:

   Click Next. In the next dialog, enter "ImageRekognition" for the Name, enter the desired Location, etc., and then click Create.

5. Choose a blueprint. Blueprints provide starting code to help you write your Lambda functions. For this example, choose the Detect Image Labels blueprint.

   This blueprint provides code for listening to Amazon S3 events and uses Amazon Rekognition to detect labels and add them to the S3 object as tags.
6. Choose the type of Lambda function you want to develop, and then choose **Finish** to create the Visual Studio project.

When the project is complete, you have a solution with two projects, as shown: the source project that contains your Lambda function code to deploy to Lambda, and a test project using xUnit for testing your function locally.

You might notice when you first create your projects that Visual Studio doesn't find all the NuGet references. This happens because these blueprints require dependencies that must be retrieved from
NuGet. When new projects are created, Visual Studio only pulls in local references and not remote references from NuGet. You can fix this easily by right-clicking your references and choosing **Restore Packages**.

**Examine the Files**

1. Open the `1Function.cs` file and look at the code that came with the blueprint. The first segment of code is the assembly attribute that is added to the top of the file.

    ```csharp
    // Assembly attribute to enable the Lambda function's JSON input to be converted into a .NET class.
    [assembly:
    
    By default, Lambda accepts only input parameters and return types of type `System.IO.Stream`. To use typed classes for input parameters and return types, you have to register a serializer. This assembly attribute is registering the Lambda JSON serializer, which uses `Newtonsoft.Json` to convert the streams to typed classes. You can set the serializer at the assembly or method level.

    The class has two constructors. The first is a default constructor that is used when Lambda invokes your function. This constructor creates the S3 and Rekognition service clients, and gets the AWS credentials for these clients from the IAM role you assign to the function when you deploy it. The AWS Region for the clients is set to the region your Lambda function is running in. In this blueprint, you only want to add tags to the S3 object if the Rekognition service has a minimum level of confidence about the label. This constructor checks the environment variable `MinConfidence` to determine the acceptable confidence level. You can set this environment variable when you deploy the Lambda function.

    ```csharp
    public Function()
    {
        this.S3Client = new AmazonS3Client();
        this.RekognitionClient = new AmazonRekognitionClient();

        var environmentMinConfidence = 
        System.Environment.GetEnvironmentVariable(MIN_CONFIDENCE_ENVIRONMENT_VARIABLE_NAME);
        if(!string.IsNullOrWhiteSpace(environmentMinConfidence))
        {
            float value;
            if(float.TryParse(environmentMinConfidence, out value))
            {
                this.MinConfidence = value;
                Console.WriteLine($"Setting minimum confidence to {this.MinConfidence}");
            }
            else
            {
                Console.WriteLine($"Failed to parse value {environmentMinConfidence} for minimum confidence. Reverting back to default of {this.MinConfidence}");
            }
        }
        else
        {
            Console.WriteLine($"Using default minimum confidence of {this.MinConfidence}");
        }
    }
    
    You can use the second constructor for testing. The test project configures its own S3 and Rekognition clients and passes them in.
public Function(IAmazonS3 s3Client, IAmazonRekognition rekognitionClient, float minConfidence)
{
    this.S3Client = s3Client;
    this.RekognitionClient = rekognitionClient;
    this.MinConfidence = minConfidence;
}

FunctionHandler is the method Lambda calls after it constructs the instance. Notice that the input parameter is of type S3Event and not a Stream. You can do this because of the registered Lambda JSON serializer. The S3Event contains all the information about the event triggered in Amazon S3. The function loops through all the S3 objects that were part of the event and tells Rekognition to detect labels. After the labels are detected, they are added as tags to the S3 object.

public async Task FunctionHandler(S3Event input, ILambdaContext context)
{
    foreach(var record in input.Records)
    {
        if(!SupportedImageTypes.Contains(Path.GetExtension(record.S3.Object.Key)))
        {
            Console.WriteLine("Object {record.S3.Bucket.Name}:{record.S3.Object.Key} is not a supported image type");
            continue;
        }
        Console.WriteLine("Looking for labels in image {record.S3.Bucket.Name}:{record.S3.Object.Key} is not a supported image type");
        var detectResponses = await this.RekognitionClient.DetectLabelsAsync(new DetectLabelsRequest
        {
            MinConfidence = MinConfidence,
            Image = new Image
            {
                S3Object = new Amazon.Rekognition.Model.S3Object
                {
                    Bucket = record.S3.Bucket.Name,
                    Name = record.S3.Object.Key
                }
            }
        });
        var tags = new List();
        foreach(var label in detectResponses.Labels)
        {
            if(tags.Count < 10)
            {
                Console.WriteLine("Found Label {label.Name} with confidence {label.Confidence}"
                {label.Confidence});
                tags.Add(new Tag { Key = label.Name, Value = label.Confidence.ToString() });
            }
            else
            {
                Console.WriteLine("Skipped label {label.Name} with confidence {label.Confidence} because maximum number of tags reached");
            }
        }
        await this.S3Client.PutObjectTaggingAsync(new PutObjectTaggingRequest
        {
            BucketName = record.S3.Bucket.Name,
            Key = record.S3.Object.Key,
        });
    }
}
Tagging = new Tagging
{
    TagSet = tags
}
);
} return;
}

Notice that the code contains calls to `Console.WriteLine()`. When the function is running in Lambda, all calls to `Console.WriteLine()` redirect to Amazon CloudWatch Logs.

2. Open the `aws-lambda-tools-defaults.json` file that the blueprint created. This file contains default values that the blueprint has set to help prepopulate some of the fields in the deployment wizard. It’s also helpful in setting command line options with our integration with the new .NET Core CLI. To use it, navigate to the function’s project directory and type `dotnet lambda help`.

An important field is the function handler. This indicates to Lambda the method to call in the code in response to the function we’re invoking. The format of this field is `<assembly-name>::<full-type-name>::<method-name>`. Be sure to include the namespace with the type name.

**Deploy the Function**

1. Right-click the Lambda project, and then choose **Publish to AWS Lambda**. This starts the deployment wizard. Notice that many of the fields are already set. These values come from the `aws-lambda-tools-defaults.json` file described earlier.

2. Enter a function name. For this example, use `ImageRekognition`, and then choose **Next**.

3. On the **Advanced Function Details** page, select an IAM role that gives permission for your code to access S3 and Rekognition. To keep this post short, select the **Power User managed policy**. The tools create a role based on this policy.

4. Finally, set the environment variable `MinConfidence` to 60, and then choose **Upload**.
This launches the deployment process, which builds and packages the Lambda project and then creates the Lambda function. Once publishing is complete, the Function view in the AWS Explorer window is displayed. From here, you can invoke a test function, view CloudWatch Logs for the function, and configure event sources.

5. With your function deployed, you need to configure Amazon S3 to send its events to your new function. On the Event Sources tab, choose Add. Then choose Amazon S3 and the bucket you want
to connect to your Lambda function. The bucket must be in the same region as the region where the Lambda function is deployed.

## Test the Function

Now that the function is deployed and an S3 bucket is configured as an event source for it, open the S3 bucket browser from the **AWS Explorer** for the bucket you selected. Then upload some images.

When the upload is complete, you can confirm that your function ran by looking at the logs from your function view. Or, right-click the images in the bucket browser and choose **Properties**. On the **Tags** tab, you can view the tags that were applied to your object.

![Properties: sample-pic.jpg](https://norm-images.s3.amazonaws.com/sample-pic.jpg)

### Tags Tab

<table>
<thead>
<tr>
<th>Tag Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dirt Road</td>
<td>97.90181</td>
</tr>
<tr>
<td>Road</td>
<td>97.90181</td>
</tr>
<tr>
<td>Gravel</td>
<td>97.90181</td>
</tr>
<tr>
<td>Pant</td>
<td>72.31149</td>
</tr>
<tr>
<td>Reed</td>
<td>72.31149</td>
</tr>
<tr>
<td>Grass</td>
<td>72.31149</td>
</tr>
<tr>
<td>Conifer</td>
<td>71.97598</td>
</tr>
<tr>
<td>Tree</td>
<td>71.97598</td>
</tr>
<tr>
<td>Fir</td>
<td>71.97598</td>
</tr>
<tr>
<td>Spruce</td>
<td>66.26569</td>
</tr>
</tbody>
</table>

---

## Tutorial: Using Amazon Logging Frameworks with AWS Lambda to Create Application Logs

You can use Amazon CloudWatch Logs to monitor, store, and access your application’s logs. To get log data into CloudWatch Logs, you can use an AWS SDK or install the CloudWatch Logs agent to monitor certain log folders. Today, we’ve made it even easier to use CloudWatch Logs with .NET applications by integrating CloudWatch Logs with several popular .NET logging frameworks.

The supported .NET logging frameworks are **NLog**, **Log4net**, and the new built-in **ASP.NET Core logging Framework**. For each framework, all you need to do is add the appropriate NuGet package, add CloudWatch Logs as an output source, and then use your logging library as you normally would.
For example to use CloudWatch Logs with a .NET application using NLog, add the AWS.Logger.NLog NuGet package, and then add the AWS target into your NLog.config file. Here is an example of an NLog.config file that enables both CloudWatch Logs and the console as output for the log messages.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<nlog xmlns="http://www.nlog-project.org/schemas/NLog.xsd"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     throwExceptions="true">
    <targets>
        <target name="aws" type="AWSTarget" logGroup="NLog.ConfigExample" region="us-east-1"/>
        <target name="logfile" xsi:type="Console" layout="${callsite} ${message}" />
    </targets>
    <rules>
        <logger name="*" minlevel="Info" writeTo="logfile,aws" />
    </rules>
</nlog>
```

After performing these steps, when you run your application the log messages written with NLog are sent to CloudWatch Logs. Then you can view your application's log messages in near real time from the CloudWatch Logs console. You can also set up metrics and alarms from the CloudWatch Logs console, based on your application's log messages.

These logging plugins are all built on top of the AWS SDK for .NET, and use the same behavior used by the SDK to find AWS credentials. The credentials used by the logging plugins must have the following permissions to access CloudWatch Logs.

```json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents",
                "logs:DescribeLogGroups"
            ],
            "Resource": ["arn:aws:logs:*:*:*"]
        }
    ]
}
```

The AWS .NET logging plugins are a new open source project on GitHub. All of the plugins are there, including samples and instructions on how to configure CloudWatch Logs for each of the supported .NET logging frameworks.

---

**Deploying an AWS Lambda Project with the .NET Core CLI**


**Note**
For information about creating Lambda functions in Visual Studio, see lambda.
Listing the Lambda Commands Available through the CLI

There are a variety of Lambda commands available through the .NET Core CLI.

1. Open a command prompt and navigate to the folder containing a Visual Studio .NET Core Lambda project.
2. Type `dotnet lambda --help`.

C:\Lambda\AWSLambda1\AWSLambda1>dotnet lambda --help
AWS Lambda Tools for .NET Core functions
Project Home: https://github.com/aws/aws-lambda-dotnet

Commands to deploy and manage |LAM| functions:

- deploy-function Deploy the project to |LAM|
- invoke-function Invoke the function in |LAM| with an optional input
- list-functions List all of your |LAM| functions
- delete-function Delete a |LAM| function
- get-function-config Get the current runtime configuration for a |LAM|

function
- update-function-config Update the runtime configuration for a |LAM| function

Commands to deploy and manage AWS Serverless applications using |CFNlong|:

- deploy-serverless Deploy an AWS Serverless application
- list-serverless List all of your AWS Serverless applications
- delete-serverless Delete an AWS Serverless application

Other Commands:

- package Package a |LAM| project into a .zip file ready for deployment

To get help on individual commands execute:

- dotnet lambda help <command>

Publishing a .NET Core Lambda Project from the .NET Core CLI

The following instructions assume you’ve created an AWS Lambda .NET Core function in Visual Studio.

1. Open a command prompt and navigate to the folder containing your Visual Studio .NET Core Lambda project.
2. Type `dotnet lambda deploy-function`. 
When prompted, type the name of the function to deploy. It can be a new name or the name of an existing function.

1. When prompted, enter the AWS Region (the region to which your Lambda function will be deployed).
2. When prompted, select or create the IAM role that Lambda will assume when executing the function.
3. On successful completion, the message **New Lambda function created** is displayed.

If you deploy an existing function, the deploy function asks only for the AWS Region.

After your Lambda function is deployed, it is ready to use. See [Examples of How to Use AWS Lambda](https://docs.aws.amazon.com/lambda/latest/dg/).
Lambda automatically monitors Lambda functions for you, reporting metrics through Amazon CloudWatch. To monitor and troubleshoot your Lambda function, see Troubleshooting and Monitoring AWS Lambda Functions with Amazon CloudWatch.

Last documentation update: March 28, 2019

Document History

The following table describes the important recent changes of the AWS Toolkit for Visual Studio User Guide. For notification about updates to this documentation, you can subscribe to an RSS feed.

<table>
<thead>
<tr>
<th>update-history-change</th>
<th>update-history-description</th>
<th>update-history-date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for Visual Studio 2019 (p. 3)</td>
<td>Support for Visual Studio 2019 was added to the AWS Toolkit for Visual Studio.</td>
<td>March 28, 2019</td>
</tr>
<tr>
<td>Support for Visual Studio 2019 (p. 70)</td>
<td>Support for Visual Studio 2019 was added to the AWS Toolkit for Visual Studio.</td>
<td>March 28, 2019</td>
</tr>
<tr>
<td>Support for Visual Studio 2019 (p. 77)</td>
<td>Support for Visual Studio 2019 was added to the AWS Toolkit for Visual Studio.</td>
<td>March 28, 2019</td>
</tr>
<tr>
<td>Detailed Views of Container Service (p. 19)</td>
<td>Added information about the detailed views of Amazon Elastic Container Service clusters and container repositories that are provided by AWS Explorer.</td>
<td>February 16, 2018</td>
</tr>
<tr>
<td>Deploying to Amazon EC2 Container Service (p. 69)</td>
<td>Added information about deploying to Amazon EC2 container service.</td>
<td>February 16, 2018</td>
</tr>
</tbody>
</table>
### Earlier Updates

The following table describes the important earlier changes of the AWS Toolkit for Visual Studio User Guide.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Release Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploying Container Service using Fargate</td>
<td>Added information about how to deploy a containerized ASP.NET Core 2.0 application targeting Linux through Amazon ECS using the Fargate launch type.</td>
<td>February 16, 2018</td>
</tr>
<tr>
<td>Deploying Container Service using EC2</td>
<td>Added information about how to deploy a containerized ASP.NET Core 2.0 application targeting Linux through Amazon ECS using the EC2 launch type.</td>
<td>February 16, 2018</td>
</tr>
<tr>
<td>Credentials for Deploying to Amazon EC2</td>
<td>Added information about how to specify credentials when deploying to Amazon EC2 container service.</td>
<td>February 16, 2018</td>
</tr>
<tr>
<td>Added ASP.NET Core Details</td>
<td>AWS Elastic Beanstalk deployment wizard now supports ASP.NET Core applications. See Deploying an ASP.NET Core Application to Elastic Beanstalk (p. 39) for details.</td>
<td>July 25, 2016</td>
</tr>
<tr>
<td>Revised deployment wizards</td>
<td>This release introduces a new Publish to Elastic Beanstalk wizard. For more information, see Deploying to Elastic Beanstalk (p. 30). With the introduction of this new wizard, the Publish to Amazon Web Services wizard has been moved to legacy status. For more information, see Deploying to Elastic Beanstalk (Legacy) (p. 49) and Deploying to AWS CloudFormation (Legacy) (p. 60).</td>
<td>December 17, 2014</td>
</tr>
<tr>
<td>Support for Amazon VPC</td>
<td>This release adds support for Amazon Virtual Private Cloud.</td>
<td>April 4, 2013</td>
</tr>
<tr>
<td>New release</td>
<td>This is version 3.0 of the AWS Toolkit for Visual Studio User Guide.</td>
<td>June 8, 2012</td>
</tr>
</tbody>
</table>