AWS .NET SDK

AWS Guide for .NET Developers

Table of Contents

AWS Guide for .NET Developers

Amazon Web Services is a secure cloud services platform, offering compute power, database storage, web hosting and other functionality to help .NET developers build sophisticated applications with increased flexibility, scalability and reliability.

This guide will help you better understand what AWS offers .NET developers and how to choose the right service for your .NET application scenario.

We Want Your Feedback

The best place to contribute is the AWS Guide for .NET Developers GitHub repository. You can do all of these wonderful things there:

- Report a problem
- Make a suggestion
- Contribute new content
- Enhance existing content
- Review issues and suggestions

You can also provide feedback using the feedback buttons at the bottom of the page.

Thank you.

Getting Started with the AWS Guide for .NET Developers

Find the information you need about cloud computing, cloud deployment models, how to sign up for AWS, and more to help you get started using AWS services.

Topics

AWS Tools for .NET Developers

Amazon provides the following tools to help .NET developers work with Amazon Web Services.

AWS Toolkit for Visual Studio

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

And you don't have to leave Visual Studio.

For details about how to download and install the toolkit, see Setting up the AWS Toolkit for Visual Studio.

AWS Tools for Microsoft Visual Studio Team Services

AWS Tools for Microsoft Visual Studio Team Services (VSTS) adds tasks to easily enable build and release pipelines in VSTS and Team Foundation Server (TFS) to work with AWS services. You can work with Amazon S3, AWS Elastic Beanstalk, AWS CodeDeploy, AWS Lambda, AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS), and Amazon Simple Notification Service (Amazon SNS). You can also run commands using the Windows PowerShell module and the AWS CLI.

To get started with AWS Tools for Microsoft Visual Studio Team Services, see AWS Tools for Microsoft Visual Studio Team Services.

AWS Tools for Windows PowerShell and PowerShell Core

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

See AWS Tools for Windows PowerShell to get started. You can download the tools, check out sample scenarios, and more. You can download AWS Tools for PowerShell Core from the PowerShell Gallery.

AWS SDK for .NET

The AWS SDK for .NET makes it easier for Windows developers to build .NET applications that tap in to the cost-effective, scalable, and reliable AWS infrastructure services such as Amazon Simple Storage Service, Amazon Elastic Compute Cloud, AWS Lambda, and more.

The AWS SDK for .NET supports development on any platform that supports the .NET Framework 3.5 or later.

The AWS SDK for .NET targets .NET Standard 1.3. You can use it with .NET Core 1.x or .NET Core 2.0.

See the AWS SDK for .NET Developer Guide to get started.

Types of Cloud Computing

Cloud computing provides a simple way to access servers, storage, databases, and a broad set of application services over the internet. A cloud services platform such as Amazon Web Services owns and maintains the network-connected hardware required for these application services. You provision and use what you need via a web application, command-line tool, or SDK.

There are three main models for cloud computing. Each model represents a different part of the cloud computing stack, as follows:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

Infrastructure as a Service

Infrastructure as a Service (IaaS) contains the basic building blocks for cloud IT. Typically, it provides access to networking features, computers (virtual or on dedicated hardware), and data storage space.

As a .NET developer, you might use IaaS to host an ASP.NET website, set up and tear down test or development environments, deploy high-performance computing apps, and take advantage of networking and security resources.

Platform as a Service

Platform as a Service (PaaS) removes the need for organizations to manage the underlying infrastructure (usually hardware and operating systems). PaaS enables you to focus on the deployment and management of your applications. This helps you be more efficient, because you don't need to worry about resource procurement, capacity planning, software maintenance, patching, or any of the other undifferentiated heavy lifting involved in running your application.

Software as a Service

Software as a Service (SaaS) provides you with a completed product that the service provider runs and manages. In most cases, people referring to SaaS are referring to end-user applications. As a .NET developer, you might write an application or integrate with an application provided as a service.

Cloud Deployment Models

There are three cloud deployment models : cloud, hybrid, and on-premises.

Cloud Deployment Model

A cloud-based application is fully deployed in the cloud, and all parts of the application run in the cloud. Applications in the cloud are either created in the cloud or are migrated from an existing infrastructure to benefit from cloud computing. Cloud-based applications can be built on low-level infrastructure pieces, or can use higher-level services that provide abstraction from the management, architecting, and scaling requirements of core infrastructure.

Hybrid Deployment Model

A hybrid deployment is a way to connect infrastructure and applications between cloud-based resources and existing resources that are not located in the cloud. The most common method of hybrid deployment is between the cloud and existing on-premises infrastructure to extend and grow an organization's infrastructure into the cloud, while connecting cloud resources to the internal system.

On-Premises Deployment Model

Deploying resources on-premises, using virtualization and resource management tools, is sometimes referred to as a "private cloud". On-premises deployment doesn't provide many of the benefits of cloud computing. However, it's sometimes sought for its ability to provide dedicated resources. In most cases this deployment model is the same as legacy IT infrastructure, while using application management and virtualization technologies to try and increase resource utilization.

Signing up for an AWS Account

To use the AWS SDK for .NET to access AWS, you need an AWS account. You also need credentials for individual services, such as Amazon S3 or Amazon EC2. You acquire those credentials in a profile for the AWS SDK for .NET when you create your first sample app [add link].

Sign up for an AWS Account

1. Open http://aws.amazon.com/ and choose **Create a Free Account**. This might be unavailable in your browser if you previously signed in to the AWS Management Console. In that case, choose **Sign in to a different account**, and then choose **Create a new AWS account**.
2. Follow the onscreen instructions.

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

After signing up for an AWS account, you can use AWS CloudFormation through the AWS Management Console, AWS CloudFormation API, or AWS CLI.

Pricing

AWS offers you a pay-as-you-go approach for pricing for over 70 cloud services. With AWS you pay only for the individual services you need, for as long as you use them, and without requiring long-term

contracts or complex licensing. You only pay for the services you consume, and once you stop using them, there are no additional costs or termination fees.

For more information about AWS pricing, go to the detail page for each product on http://aws.amazon.com. For example, see aws.amazon.com/s3/pricing for Amazon Simple Storage Service pricing.

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

Web Applications

Amazon Web Services offers cloud web hosting solutions for delivering websites and web applications. AWS provides development support for .NET and other popular platforms, has data centers worldwide, and can dynamically grow and shrink resources with fluctuating website traffic. With flexible pricing models, AWS only charges you for the resources you use.

Types of Websites

Static Website Hosting

Static websites deliver HTML, JavaScript, images, video, and other files to your website visitors. They contain no server-side application code, like ASP.NET or PHP. They typically are used to deliver personal or marketing sites. ASP.NET and ASP.NET Core web applications typically require simple or enterprise website hosting.

Consider using static website hosting when your website doesn't contain server-side scripting, changes infrequently, and needs to scale for intervals of high traffic. Static websites are best for customers who don't want to manage infrastructure.

Simple Website Hosting

Simple websites typically consist of a single Linux, Unix, or Windows web server with a development stack, such as ASP.NET. They provide a simple starting point for a website that might grow in the future. Simple websites require IT administration of the web server and aren't built to be highly available or scalable beyond a few servers.

Consider using simple website hosting when your website is unlikely to scale beyond five servers, and you want to manage your own web server and resources.

Enterprise Website Hosting

Enterprise websites use multiple servers and AWS services and often span multiple data centers (or Availability Zones). They dynamically scale resources based on demand and need to be highly available. There are often different development stacks used for different portions of the application.

Consider using enterprise website hosting when your website needs web servers across at least two data centers, needs to scale, or requires sustained high CPU utilization. Enterprise website hosting is great for those who need maximum control and flexibility configuring and administrating their web server.

Choosing Where to Deploy Your Web App

You have several deployment options on Amazon Web Services. You can choose the mostly automatic approach, answering a few initial configuration questions and letting AWS do the rest. Or you can choose to go hands on and fully configurable, adjusting DNS, automatic scalers, and other aspects of your deployment environment.

AWS Elastic Beanstalk

AWS Elastic Beanstalk is an easy-to-use service for deploying and scaling web applications developed with .NET.

Simply deploy your application, and Elastic Beanstalk automatically handles the details of deployment provisioning, load balancing, scaling, and application health monitoring. You retain full control over the AWS resources powering your application, and can access the underlying resources at any time.

Elastic Beanstalk supports several platforms for different versions of the .NET programming framework and Windows Server. It also supports Docker containers.

Amazon Elastic Container Service

Amazon Elastic Container Service is a highly scalable, high-performance container management service that makes it easy to run, manage, and stop Docker containers on a cluster of Amazon EC2 instances.

Amazon ECS is a good option if you have a containerized .NET Core application.

AWS Lambda

AWS Lambda enables you to run .NET Core functions or serverless applications without provisioning or managing servers. You get flexible scaling and high availability, and have no idle capacity because there is no charge when your code isn't running.

Lambda is a good option if you want to really benefit from serverless computing.

Deploying Web Applications from Visual Studio

Using the Toolkit for Visual Studio, you can deploy your web applications to AWS without leaving Visual Studio.

Topics

Deploying an ASP.NET App to Windows

You can use the **Publish to Elastic Beanstalk** wizard, provided as part of the Toolkit for Visual Studio, to deploy an application to Elastic Beanstalk. To practice, you can use an instance of a web application starter project that is built into Visual Studio or you can use your own project.

Create a Sample Web Application Starter Project

1. In Visual Studio, choose **File**, **New**, **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 Application (.NET Framework)**.

4. In the **Name** box, type `WebAppDemo`. Choose a **Location** and confirm that the **Create directory for solution** box is selected. Then choose **OK**.
5. In the **New ASP.NET Web Application** dialog box, choose **Web Forms** or **MVC**, and then choose **OK**.

   Visual Studio creates a solution and project based on the Application project template you selected. Visual Studio then displays the **Solution Explorer**, where the new solution and project appear.

Deploy an Application by Using the Publish to Elastic Beanstalk Wizard

1. In **Solution Explorer**, open the context (right-click) menu for the **WebAppDemo** project, or open the context menu for the project for your own application, and then choose **Publish to AWS Elastic Beanstalk**.

   The **Publish to Elastic Beanstalk** wizard appears.
2. In **Profile**, in the **Account profile to use for deployment** list, choose the AWS account profile to use for the deployment. This account profile is used only for deployment. You can specify your application credentials separately, if needed.

   Optionally, if you have an AWS account you want to use, but haven't yet created an AWS account profile for it, you can choose the plus symbol (**+**) button to add an AWS account profile.
3. In the **Region** list, choose the AWS Region to which you want Elastic Beanstalk to deploy the application.
4. In **Deployment Target**, choose **Create a new application environment**. If you wanted to redeploy a previously deployed application, you would choose **Redeploy to an existing environment**.
5. Choose **Next**.

   On the **Application Environment** page, in the **Application** area, the **Name** defaults to **WebAppDemo**.
6. In **Environment**, in the **Name** list, choose **WebAppDemo-dev**. In this context, the term *environment* refers to the infrastructure Elastic Beanstalk provisions for your application.
7. Choose **Check availability** to ensure the default URL domain **EBWebAppDemo-dev** for your web application isn't already in use. If it is in use, try other names until the requested URL is available.
8. Choose **Next**.
9. **In the Key pair list, choose an Amazon EC2 instance key pair to use to sign in to**

   the instances that will be used for your application. Select **<Create new key pair>** and type in a key name. We have used "MyKeyPair" in this example.

   We recommend you launch your instance with a key pair so that you can connect to it with SSH or RDP in the future.
10. Ensure **Use non-default VPC**, **Single instance environment**, and **Enable Rolling Deployments** are not selected. You can add these options later.

    Optionally, if you have an Amazon Relational Database Service database security group with a database you want your application to access, select it in the **Relational Database Access** list. It will be modified to permit access from the Amazon EC2 instances hosting your application.

    Choose **Next**.

1. On the **Permissions** page, choose **Next** to accept the defaults.
2. On the **Applications Options** page, choose **Next** to accept the defaults.
3. On the **Review** page, select **Open environment status window when wizard closes** and **Generate AWSDeploy configuration**. Click **Choose**, type in **WebAppDemo**, and then choose **Save**.

4. Choose **Deploy** to deploy to Elastic Beanstalk.

   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 **Events** window of the environment page. It might take several minutes to complete the deployment. When complete, you'll see a green INFO event indicating that the environment launch succeeded.

   Choose the **URL** to view the website.

Delete an AWS Elastic Beanstalk Deployment

Terminate an environment, delete the app.

You can use the Toolkit for Visual Studio to delete a deployment. In AWS Explorer, expand the **Elastic Beanstalk** node, open the context (right-click) menu for the subnode for the deployment, and then choose **Terminate Environment**. The termination process might take a few minutes. You can monitor termination status on the event tab of the environment view.

Once the deployment is terminated, expand the **Elastic Beanstalk** node in AWS Explorer, open the context (right-click) menu for the subnode for the deployment, and then choose **Delete**.

Deploying an ASP.NET Core 2.0 application to AWS Elastic Beanstalk

You can use the **Publish to AWS Elastic Beanstalk** wizard, provided as part of the Toolkit for Visual Studio, to deploy an ASP.NET Core 2.0 application targeting Windows to Elastic Beanstalk.

Create a Sample Web Application Starter Project

1. In Visual Studio, choose **File**, **New**, **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 **ASP.NET Core Web Application**.
4. In the **Name** box, type `WebAppDemo`, and then choose **OK** to go to the next screen.
5. Confirm **NET Core** and **ASP.NET Core 2.0** are selected, and then select the **Web Application** application template.
6. Confirm the **Create directory for solution** box is selected. In the **Solution** list, confirm **Create new solution** is selected, and then choose **OK** to go to the next page.
7. Confirm **NET Core** and **ASP.NET Core 2.0** are selected, and then select the **Web Application** application template.

Deploy an ASP.NET Core 2.0 Application Using the Publish to AWS Elastic Beanstalk Wizard

1. In **Solution Explorer**, open the context (right-click) menu for the **WebAppDemo** project, or open the context menu for the project for your own application. Then choose **Publish to AWS Elastic Beanstalk**.

   The **Publish to Elastic Beanstalk** wizard appears.

2. In **Profile**, from the **Account profile to use for deployment** list, choose the AWS account profile to use for the deployment. This account profile is used only for deployment. You specify other credentials separately in the wizard.

   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 plus symbol (**+**) button to add an AWS account profile.
3. In the **Region** list, choose the AWS Region to which you want Elastic Beanstalk to deploy the application.
4. In **Deployment Target**, choose **Create a new application environment**. If you wanted to redeploy a previously deployed application, you would choose **Redeploy to an existing environment**.
5. Choose **Next**.

   On the **Application Environment** page, in the **Application** area, the **Name** defaults to **WebAppDemo**.

1. In the **Environment** area, in the **Name** list, choose **WebAppDemo-dev**. In this context, the term *environment* refers to the infrastructure Elastic Beanstalk provisions for your application.
2. Choose **Check availability** to ensure the default URL domain **EBWebAppDemo-dev** for your web application isn't already in use. If it is in use, try other names until the requested URL is available.
3. Choose **Next**.
4. In the **Key pair** list, choose an Amazon EC2 instance key pair to use to sign in to the instances that will be used for your application. Select **<Create new key pair...>** and type in a key name. We have used "MyKeyPair" in this example.

   We recommend you launch your instance with a key pair so that you can connect to it with SSH or RDP in the future.
5. Ensure **Use non-default VPC**, **Single instance environment**, and **Enable Rolling Deployments** are not selected. You can add these options later.

   Optionally, if you have an Amazon Relational Database Service database security group with a database you want your application to access, select it in the **Relational Database Access** list. It will be modified to permit access from the Amazon EC2 instances hosting your application.

   Choose **Next**.
6. On the **Permissions** page, choose **Next** to accept the defaults.
7. On the **Applications Options** page, choose **Next** to accept the defaults.
8. On the **Review** page, review the options you configured. Choose **Open environment status window when wizard closes** and **Save settings to aws-beanstalk-tools-defaults.json and configure project for command line deployment**. You can use the settings file to make future deployments from the command line using the .NET CLI.
9. 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 **Events** window of the environment page. It might take several minutes to complete the deployment. When the complete, you'll see a green INFO event indicating that the environment launch succeeded.

   Choose the **URL** to view the website.

Expand Capacity

Your Elastic Beanstalk environment includes an Auto Scaling group that manages the Amazon EC2 instances in your environment. In a single-instance environment, the Auto Scaling group ensures that there is always one instance running. In a load-balanced environment, you configure the group with a range of instances to run, and Amazon EC2 Auto Scaling adds or removes instances as needed, based on load.

You can configure how Amazon EC2 Auto Scaling works by editing **Capacity** on the **Auto Scaling** page for your application in **AWS Explorer**.

1. In Visual Studio 2017, select **View**, and then choose **AWS Explorer** if it is not already visible.
2. In AWS Explorer, expand the **Elastic Beanstalk** node, and then double-click the node for your application environment. In this example, it's **EBWebAppDemo-dev**.
3. Choose **Auto Scaling**.
4. Configure automatic scaling settings. To ensure there are a minimum of two instances of your application running, adjust **Minimum Instance Count** to 2. You can also increase or decrease **Maximum Instance Count** to suit demand.

   For more information on auto scaling settings and triggers, see Auto Scaling Group.
5. Choose **Apply Changes**.

Delete an AWS Elastic Beanstalk Deployment

You can use the Toolkit for Visual Studio to delete an application. In **AWS Explorer**, expand the **Elastic Beanstalk** node, open the context (right-click) menu for the application you want to delete, and then choose **Delete**.

You can also terminate a deployment environment. In **AWS Explorer**, expand the **Elastic Beanstalk** node, expand the node for your application, open the context (right-click) menu for the environment you want to terminate, and then select **Terminate Environment**. The termination process might take a few minutes. You can monitor termination status on the event tab of the environment view.

Deploying an ASP.NET Core 2.0 App to Amazon Elastic Container Service

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 to Amazon ECS using the Fargate launch type. Because a web application is meant to run continuously, it will be deployed as a service.

Access the Publish Container to AWS Wizard

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

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

Create a Sample Web Application Starter Project

1. In Visual Studio, choose **File**, **New**, **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 **ASP.NET Core Web Application**.
4. In the **Name** box, type `WebAppDemo`, and then choose **OK** to go to the next page.
5. Confirm **NET Core** and **ASP.NET Core 2.0** are selected, then choose the **Web Application** application template.
6. Choose **Enable Docker Support**, choose **Linux** as the operating system, and then choose **OK**. Visual Studio 2017 generates the project.

Use the Publish Container to AWS Wizard

1. In **Solution Explorer**, open the context (right-click) menu for the **WebAppDemo** 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 then choose **Publish Container to AWS**.

   The **Publish Container to AWS...** wizard appears.
2. In **Profile**, in the **Account profile to use for deployment** list, choose the AWS account profile to use for the deployment. This account profile is used only for deployment. You can specify other credentials separately in the wizard.

   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 plus symbol (**+**) button to add an AWS account profile.
3. In the **Region** list, choose **US East (N. Virginia)** or another region offering Fargate.
4. Choose **Service on an ECS Cluster** as the **Deployment Target**. Ensure **Save settings to aws-ecs-tools-defaults.json and configure project for command line deployment** is selected. You can use the settings file

   to make future deployment from the command line using the .NET CLI.
5. On the **Launch Configuration** page, in the **ECS Cluster** list, choose **Create an empty cluster**, and then name the cluster **WebAppDemo**. Verify **Launch Type** is set to **FARGATE**.
6. In the **Network Configuration** area, choose **Create New** to create a new security group, and then choose **Next**.
7. On the **Service Configuration** page, in the **Service** list, choose **Create New**. The wizard provides a default service name.
8. Update the **Number of Tasks** to 2. Each task maps to an instance of your container. If one goes down, the other can be available. Choose **Next**
9. On the **Application Load Balancer Configuration** page, choose **Configure Application Load Balancer**. In the **Load Balancer** list, choose **Create New**. The wizard provides defaults for related fields. Choose **Next**.
10. On the **Application Load Balancer Configuration** page, in the **Task Definition** list, choose **Create New**. The **Container** list should also be set to **Create New**. Accept the default names and other values.
11. Choose **Publish**.

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

    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**, then expand **Clusters**, and then click on your cluster.

Deploying Web Applications from Team Services

The AWS Tools for Microsoft Visual Studio Team Services (VSTS) adds tasks to enable build and release pipelines in VSTS and Team Foundation Server (TFS) to work with AWS services.

Topics

Deploying an ASP.NET Application from Team Services to AWS Elastic Beanstalk

You can use use Elastic Beanstalk deployment tasks, provided as part of the AWS Tools for Microsoft Visual Studio Team Services, to deploy an ASP.NET application to a single existing Elastic Beanstalk application in one step.

1. In Team Services, choose the **Projects** tab. Select the project to deploy, and then choose **Build & Release**.
2. On the **Build Definitions** page, choose **+ New definition**.
3. On the **Select a template** page, choose **ASP.NET**, and then choose **Apply**.
4. Add the Elastic Beanstalk deployment task. Choose **Add Task**. In the **Add tasks** pane on the right, type `aws` into the search bar, scroll down to **AWS Elastic Beanstalk Deploy Application**, and then choose **Add**.
5. On the **Process** page, choose **Deploy to Elastic Beanstalk** to configure deployment details. Choose **AWS Credentials**, and then choose credentials that the build agent will use to access Elastic Beanstalk.
6. Choose the **AWS Region** for your Elastic Beanstalk deployment.
7. Provide an **Application Name** and **Environment Name** for the deployment. For example, `ContosoUniversity` and `ContosoUniversity-dev`.
8. In the **Deployment Bundle Type** list, choose **ASP.NET (Source: Web Deploy Archive)**, and then specify the **Web Deploy Archive** location. It's a .zip file named after your application. For example, `$(build.artifactstagingdirectory)\ContosoUniversity.zip`.

   To find the web deployment archive (the output package) folder, choose **Build Solution** in the **Process list**, and then look at `PackageLocation` in the **MSBuild Arguments** entry.
9. In the **Version Label** box, type **$(Build.BuildNumber)**. If you don't provide a version label, one based on date and time is automatically generated.
10. Select **Save & queue**. In the **Queue build** dialog box, choose **Queue**. You can see deployment progress in the build console.

Creating an AWS Elastic Beanstalk Version in Team Services

You can use AWS Elastic Beanstalk deployment tasks, provided as part of the AWS Tools for Microsoft Visual Studio Team Services, to build an ASP.NET application to deploy to multiple existing Elastic Beanstalk applications.

1. In Team Services, choose the **Projects** tab. Select the project to deploy, and then choose **Build & Release**.

The process is the same for Visual Studio Team Services.

2. On the **Build Definitions** page, choose **+ New definition**.
3. On the **Select a template** page, choose **ASP.NET**, and then choose **Apply**.
4. Add the Elastic Beanstalk deployment task. Choose **Add Task**. In the **Add tasks** pane on the right, type `aws` into the search bar, scroll down to **AWS Elastic Beanstalk Create Version**, and then choose **Add**.
5. On the **Process** page, choose **AWS Elastic Beanstalk Create Version** to configure deployment details. Choose **AWS Credentials**, and then choose credentials that the build agent will use to access Elastic Beanstalk.
6. Choose the **AWS Region** for your Elastic Beanstalk deployment.
7. Provide an **Application Name** for the deployment. It must be an existing application.
8. In the **Deployment Bundle Type** list, choose **ASP.NET (Source: Web Deploy Archive)**, and then specify the **Web Deploy Archive** location. It's a .zip file named after your application. For example `$(build.artifactstagingdirectory)\ContosoUniversity.zip`.

   To find the web deployment archive (the output package) folder, choose **Build Solution** in the **Process list**, and then look at `PackageLocation` in the **MSBuild Arguments** entry.
9. In the **Version Label** box, type **$(Build.BuildNumber)**. If you don't provide a version label, one based on date and time is automatically generated.
10. Choose **Save & queue**. In the **Queue build** dialog box, choose **Queue**. You can see task progress in the build console.

Connecting to a Deployed Instance

Learn how to connect to a Linux or Windows instance and transfer files between your local computer and your instance.

Topics

Connecting to Your Windows Instance

Amazon Elastic Compute CloudAmazon EC2 instances created from most Windows Amazon Machine Images (AMIs) enable you to connect using Remote Desktop. Remote Desktop uses the Remote Desktop Protocol (RDP) and enables you to connect to and use your instance in the same way you use a computer sitting in front of you.

1. In Visual Studio 2017, open **AWS Explorer**, expand **Amazon EC2**, and then double-click **Security Groups**.
2. Choose the **VPC Security Group** that corresponds to your deployed instance.
3. Choose **Add Permission**, choose **RDP** under **Protocol**, and then choose **OK**. The RDP protocol (port 3389) will be added to the inbound permissions.
4. In **AWS Explorer**, expand **AWS Elastic Beanstalk** and your app node, and then double-click the target environment.
5. In the Environment view, choose **Connect to Instance**. The **Open Remote Desktop** prompt opens. Choose **Use EC2 keypair to log on** and, optionally, whether to **Map local drives on remote desktop**, and then choose **OK**.

   Toolkit for Visual Studio will use the keypair created during the deployment.

6. You might be warned that the publisher of the remote connection cannot be verified. It was generated by Toolkit for Visual Studio to connect to your instance. To trust the connection, choose **Connect**.

   You might also be warned that the identify of the remote computer cannot be verified. To connect anyway, choose **Yes**.

   After a few moments, you will be connected.

See Connecting to Your Windows Instance to learn other ways to connect and transfer files between your local computer and your Windows instance.

Connecting to Your Linux instance

You can connect to your Linux instance within Visual Studio by using the Toolkit for Visual Studio.

The Toolkit for Visual Studio uses the SSH client Putty to connect to Amazon EC2 instances. It's a free SSH client available from https://www.chiark.greenend.org.uk/~sgtatham/putty/.

1. In **AWS Explorer**, expand **Instances** under **Amazon EC2**, right-click your Linux instance, and then choose **Open SSH Session**.
2. In the **Open SSH Session** dialog box, choose **Use EC2 keypair to log on** to use a keypair associated with the instance.

   If the keypair isn't stored in the Toolkit for Visual Studio, you can copy the private key for the instance and paste it in the **Private Key** box. Choose **Save Private Key** to save the private key to the Toolkit for Visual Studio. It will be associated with the instance and used for future connections.

   You can also use a password. Choose **Enter password** and type in the password for the instance. To save your credentials, choose **Save Credentials**. Your credentials will be used for future connections.
3. Verify the **Location of Putty**.
4. Choose **OK** to connect.

   A Putty window opens and you are connected.

   If you experience problems connecting, see Troubleshooting Connecting to Your Instance.

Working with SQL Server

Learn about web development working with SQL Server and AWS.

Topics

Connecting to a SQL Server Instance

You can connect to an existing SQL Server instance using **Server Explorer** and the Toolkit for Visual Studio in Visual Studio.

> **Note**
> If your machine is behind a proxy/firewall, you might need to contact your network administrator.

In this procedure you connect to your sample DB instance by using Toolkit for Visual Studio and Visual Studio **Server Explorer**.

1. In **AWS Explorer**, expand the **Amazon RDS** node and the **Instances** node. Right-click your DB instance, and then choose **Add to Server Explorer**.

   If the Toolkit for Visual Studio cannot connect, the **Unable to Connect** dialog box is displayed. The dialog box gives you the option of adding your current CIPR/IP to the DB instance security group. Choose **OK** to add or **Cancel** to exit.

   If you cannot connect, you might need to contact your network network administrator.
2. In the **Add Connection** dialog box, choose **SQL Server Authentication** in the **Authentication** list, and then type your **User name** and **Password**.
3. To connect to a specific database, in the **Select or enter a database name** list, choose a database.
4. Choose **OK** to connect.

Using Encrypted Connection Strings

AWS Systems Manager provides a centralized store to manage your configuration data, whether plain-text data such as database strings or secrets such as passwords. This enables you to separate your secrets and configuration data from your code. Parameters can be tagged and organized into hierarchies, helping you manage parameters more easily. For example, you can use the same parameter name, "constr", with a different hierarchical path, "/MyWebApp/Development/constr" or "/MyWebApp/Production/constr", to store different values. Values can be encrypted and you can control user and resource access.

If your IAM user account, group, or role is assigned administator permissions, you have access to Systems Manager. If you don't, an administrator must update your IAM account, group, or role.

Create an Encrypted Connection String

To create an encrypted SQL Server connection string using the AWS Management Console:

1. Open https://console.aws.amazon.com/systems-manager/, and under **Shared Reources** in the navigation pane, choose **Parameter Store**.
2. Choose **Create Parameter**.
3. In the **Name** box, type a hierarchy and a name. You can use the hierarchy to create a unique connection string for different deployment environments. For example, `/MyWebApp/Development/constr`, `/MyWebApp/Test/constr`, and `/MyWebApp/Production/constr`.
4. Provide a guilabel:*Description*. For example, `Dev environment SQL Server connection string.`
5. Select **Secure String**.
6. Type in the SQL Server connection string. At a minimum, you will usually specify `server`, `initial catalog`, `user id`, and `password` in the connection string. For example, `server=myserver.com;Initial Catalog=mydb;User ID=myid;Password=mypwd`.
7. Choose **Create parameter**.

You can also create a parameter and perform other operations using AWS Tools for PowerShell.

```
# Create a new connection string; returns parameter version
Write-SSMParameter -Name "/MyWebApp/Development/constr" -Value "server=<server>;initial
 catalog=<db>;user id=<id>;password=<pwd>" -Type SecureString -Overwrise $true
```

```
# Retrieve all the keys for this app
Get-SSMParametersByPath -Path "/MyWebApp" -Recursive $true

# Get latest version of a parameter
Get-SSMParameter -Name "/MyWebApp/Development/constr"

# Get version of a parameter
Get-SSMParameter -Name "/MyWebApp/Development/constr:1"

# Get parameter value with decryption
Get-SSMParameter -Name "/MyWebApp/Development/constr" -WithDecryption $true
```

To learn more about PowerShell Tools, see *AWS Tools for Windows PowerShell <https://docs.aws.amazon.com/powershell/latest/userguide/pstools-welcome.html>*.

Read the Encrypted Connection String from .NET

You can get the value from Parameter Store by using the AWS SDK for .NET.

```
// Add the AWSSDK.SimpleSystemsManagement NuGet package to your project
using Amazon.SimpleSystemsManagement;
using Amazon.SimpleSystemsManagement.Model;

class DbHelper
{
    public static string GetDBConnectionString()
    {
        // The parameter name is customized based on the ASPNETCORE_ENVIRONMENT
        //
        // You can change this to a fixed string or use a different mechanism
        // to customize.
        String parameterName = String.Format("/MyWebApp/{0}/constr",
 Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT"));

        // Using USEast1
        var ssmClient = new
 AmazonSimpleSystemsManagementClient(Amazon.RegionEndpoint.USEast1);
        var response = ssmClient.GetParameter(new GetParameterRequest
        {
            Name = parameterName,
            WithDecryption = true
        });
        return response.Parameter.Value;
    }
}
```

Working with Amazon DynamoDB

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It's a fully managed cloud database and supports both document and key-value store models. Its flexible data model, reliable performance, and automatic scaling of throughput capacity make it a great fit for web and other applications.

Topics

Selecting an Amazon DynamoDB Programming Model

The AWS SDK for .NET supports Amazon DynamoDB, which is a fast NoSQL database service offered by AWS. The SDK provides three programming models for communicating with DynamoDB: low-level, document, and object persistence.

The following information introduces these models and their APIs, provides examples for how and when to use them, and gives you links to additional DynamoDB programming resources in the AWS SDK for .NET.

Low-Level Model

The low-level programming model wraps direct calls to the DynamoDB service. You access this model through the Amazon.DynamoDBv2 namespace.

Of the three models, the low-level model requires you to write the most code. For example, you must convert .NET data types to their equivalents in DynamoDB. However, this model gives you access to the most features.

The following examples show you how to use the low-level model to create a table, modify a table, and insert items into a table in DynamoDB.

Create a Table

In the following example, you create a table by using the `CreateTable` method of the `AmazonDynamoDBClient` class. The `CreateTable` method uses an instance of the `CreateTableRequest` class that contains characteristics such as required item attribute names, primary key definition, and throughput capacity. The `CreateTable` method returns an instance of the `CreateTableResponse` class.

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

var client = new AmazonDynamoDBClient();

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

```
        KeySchema = new List<KeySchemaElement>
      {
        new KeySchemaElement
        {
          AttributeName = "Id",
          // "HASH" = hash key, "RANGE" = range key.
          KeyType = "HASH"
        },
        new KeySchemaElement
        {
          AttributeName = "Type",
          KeyType = "RANGE"
        },
      },
        ProvisionedThroughput = new ProvisionedThroughput
        {
            ReadCapacityUnits = 10,
            WriteCapacityUnits = 5
        },
    };

    var response = client.CreateTable(request);

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

Insert an Item into a Table

In the following example, you use the low-level model to insert two items into a table in DynamoDB. Each item is inserted through the `PutItem` method of the `AmazonDynamoDBClient` class, using an instance of the `PutItemRequest` class. Each of the two instances of the `PutItemRequest` class takes the name of the table that the items will be inserted in, with a series of item attribute values.

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

var client = new AmazonDynamoDBClient();

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

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

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

Reading an Item from a Table

The following example features the `Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` method and a set of expressions to get and then print the item that has an `Id` of `205`. Only the following attributes of the item are returned: `Id`, `Title`, `Description`, `Color`, `RelatedItems`, `Pictures`, and `ProductReviews`.

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

var client = new AmazonDynamoDBClient();
var request = new GetItemRequest
{
  TableName = "ProductCatalog",
  ProjectionExpression = "Id, Title, Description, Color, #ri, Pictures, #pr",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#ri", "RelatedItems" }
  },
  Key = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "205" } }
  },
};
var response = client.GetItem(request);
```

In the preceding example, the `ProjectionExpression` property specifies the attributes to return. The `ExpressionAttributeNames` property specifies the placeholder `#pr` to represent the `ProductReviews` attribute and the placeholder `#ri` to represent the `RelatedItems` attribute.

Document Model

The document programming model provides an easier way to work with data in DynamoDB. This model is specifically intended for accessing tables and items in tables. You access this model through the Amazon.DynamoDBv2.DocumentModel namespace.

Compared to the low-level programming model, the document model is easier to code against DynamoDB data. For example, you don't have to convert as many .NET data types to their equivalents in DynamoDB. However, this model doesn't provide access to as many features as the low-level programming model. For example, you can use this model to create, retrieve, update, and delete items in tables. However, to create the tables, you must use the low-level model. Compared to the object persistence model, this model requires you to write more code to store, load, and query .NET objects.

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

Insert an Item into a Table

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

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

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

```
var item = new Document();

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

table.PutItem(item);
```

Get an Item from a Table

In the following example, the item is retrieved through the `GetItem` method of the `Table` class. To determine the item to get, the `GetItem` method uses the hash-and-range primary key of the target item. To determine the table to get the item from, the `LoadTable` method of the `Table` class uses an instance of the `AmazonDynamoDBClient` class and the name of the target table in DynamoDB.

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

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

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

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

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

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

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

Object Persistence Model

The object persistence programming model is specifically designed for storing, loading, and querying .NET objects in DynamoDB. You access this model through the Amazon.DynamoDBv2.DataModel namespace.

Of the three models, the object persistence model is the easiest to code against whenever you are storing, loading, or querying DynamoDB data. For example, you work with DynamoDB data types directly. However, this model provides access only to operations that store, load, and query .NET objects in DynamoDB. For example, you can use this model to create, retrieve, update, and delete items in tables. However, you must first create your tables using the low-level model, and then use this model to map your .NET classes to the tables.

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

Define a .NET Class that Represents an Item in a Table

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

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

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

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

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

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

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

context.Save(item);
```

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

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

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

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

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

Managing ASP.NET Session State with Amazon DynamoDB

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

DynamoDB provides an effective solution for sharing session state across web servers without incurring any of these drawbacks.

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

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

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

Create the *ASP.NET_SessionState* Table

When your application starts, it looks for a DynamoDB table that's named, by default, `ASP.NET_SessionState`. We recommend you create this table before you run your application for the first time.

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

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

> **Note**
> If you don't create the table in advance, the session state provider creates the table during its initialization. See the following `web.config` options for a list of attributes that act as configuration parameters for the session state table. If the provider creates the table, it uses these parameters.

Configure the Session State Provider

1. Add references to both `AWSSDK.dll` and `AWS.SessionProvider.dll` to your Visual Studio ASP.NET project. These assemblies are available by installing the AWS SDK for .NET. You can also install them by using NuGet.

   In earlier versions of the SDK, the functionality for the session state provider was contained in `AWS.Extension.dll`. To improve usability, the functionality was moved to `AWS.SessionProvider.dll`. For more information, see the blog post <problematic>:aws-blogs-net:`AWS.Extension Renaming <Tx27RWMCNAVWZN9/AWS-Extensions-renaming>`</problematic>
   .
2. Edit your application's `Web.config` file. In the `system.web` element, replace the existing `sessionState` element with the following XML fragment.

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

```
</sessionState>
```

The profile represents the AWS credentials that are used to communicate with DynamoDB to store and retrieve the session state. If you are using the AWS SDK for .NET and are specifying a profile in the `appSettings` section of your application's `Web.config` file, you don't need to specify a profile in the `providers` section. The AWS .NET client code will discover it at run time. For more information, see net-dg-config.

If the web server is running on an Amazon EC2 instance configured to use IAM roles for EC2 instances, you don't need to specify any credentials in the `Web.config` file. In this case, the AWS .NET client will use the IAM role credentials. For more information, see net-dg-roles and net-dg-ddb-sess-security.

Web.config Options

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

AWSAccessKey

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

AWSSecretKey

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

AWSProfileName

The profile name associated with the credentials you want to use. For more information, see Configuring Your AWS SDK for .NET Application.

Region

Required `string` attribute. The AWS Region in which to use DynamoDB. For a list of AWS Regions, see Regions and Endpoints: DynamoDB.

Application

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

Table

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

ReadCapacityUnits

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

WriteCapacityUnits

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

CreateIfNotExist

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

Security Considerations

After the DynamoDB table is created and the application is configured, you can use sessions as you would with any other session provider.

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

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

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