**Deployment and Azure**

**Overview**

Microsoft Azure offers secure and flexible development, deployment and scaling options for any size web application. Leverage your existing tools to create and deploy applications without the hassle of managing infrastructure.

Provision a production web application on your own in minutes by easily deploying content created using your favorite development tool. You can deploy an existing site directly from source control with support for **Git**, **GitHub**, **Bitbucket**, **CodePlex**, **TFS**, and even **Dropbox**. Deploy directly from your favorite IDE or from scripts using **PowerShell** in Windows or **CLI** tools running on any OS. Once deployed, keep your sites constantly up-to-date with support for continuous deployment.

Microsoft Azure provides scalable, durable cloud storage, backup, and recovery solutions for any data, big or small. When deploying applications to a production environment, storage services such as Tables, Blobs and SQL Databases help you scale your application in the cloud.

This module will show you the different topics you could encounter when deploying your site to production environments in Microsoft Azure. **Objectives**  
  
In this module, you'll see how to:

* Create and deploy a Web Application to a Microsoft Azure Web App using Visual Studio
* Change behavior of your app based on different deployment environments
* Work with Multiple Deployments Slots in Azure
* Add Docker support and deploy on Linux on Azure

**Note:** You can take advantage of the [Visual Studio Dev Essentials](https://www.visualstudio.com/en-us/products/visual-studio-dev-essentials-vs.aspx) subscription in order to get everything you need to build and deploy your app on any platform.  
  
**Exercises**  
  
This module includes the following exercises:

1. Build and Deploy from Visual Studio
2. Working with Environments in Visual Studio
3. Working with Multiple Deployment Slots (optional)
4. Add Docker support and deploy on Linux on Azure

Estimated time to complete this module: **60 minutes**

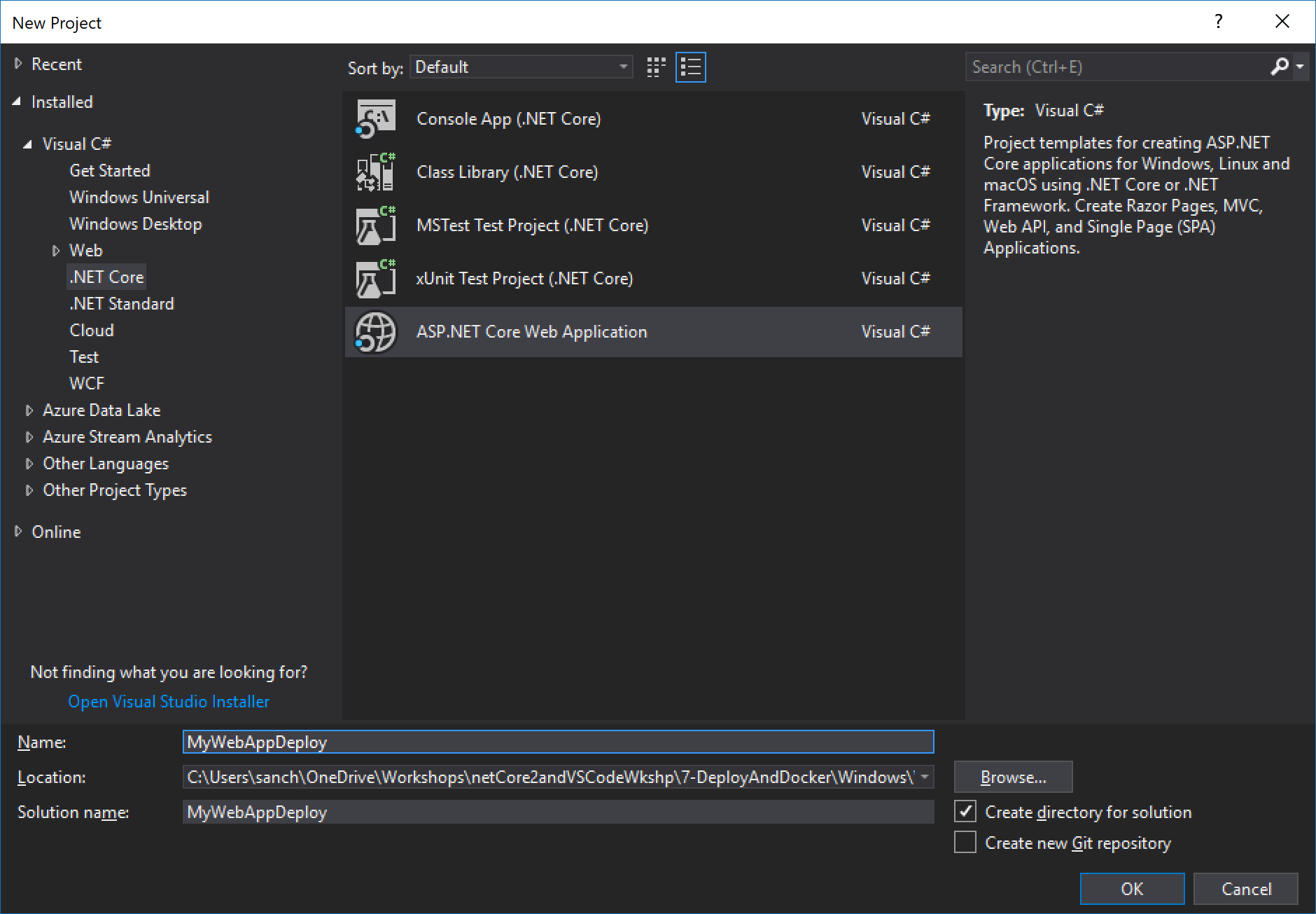
**Exercise 1: Build and Deploy from Visual Studio**

**Azure App Service Web Apps** is a fully managed platform that enables you to build, deploy and scale enterprise-grade web apps in seconds. Focus on your application code, and let Azure take care of the infrastructure to scale and securely run it for you.

In this exercise, you'll create a new Web site in Visual Studio based on the **ASP.NET Core 2.0 Web Application** project template and then publish the application to an **Azure App Service Web App**.

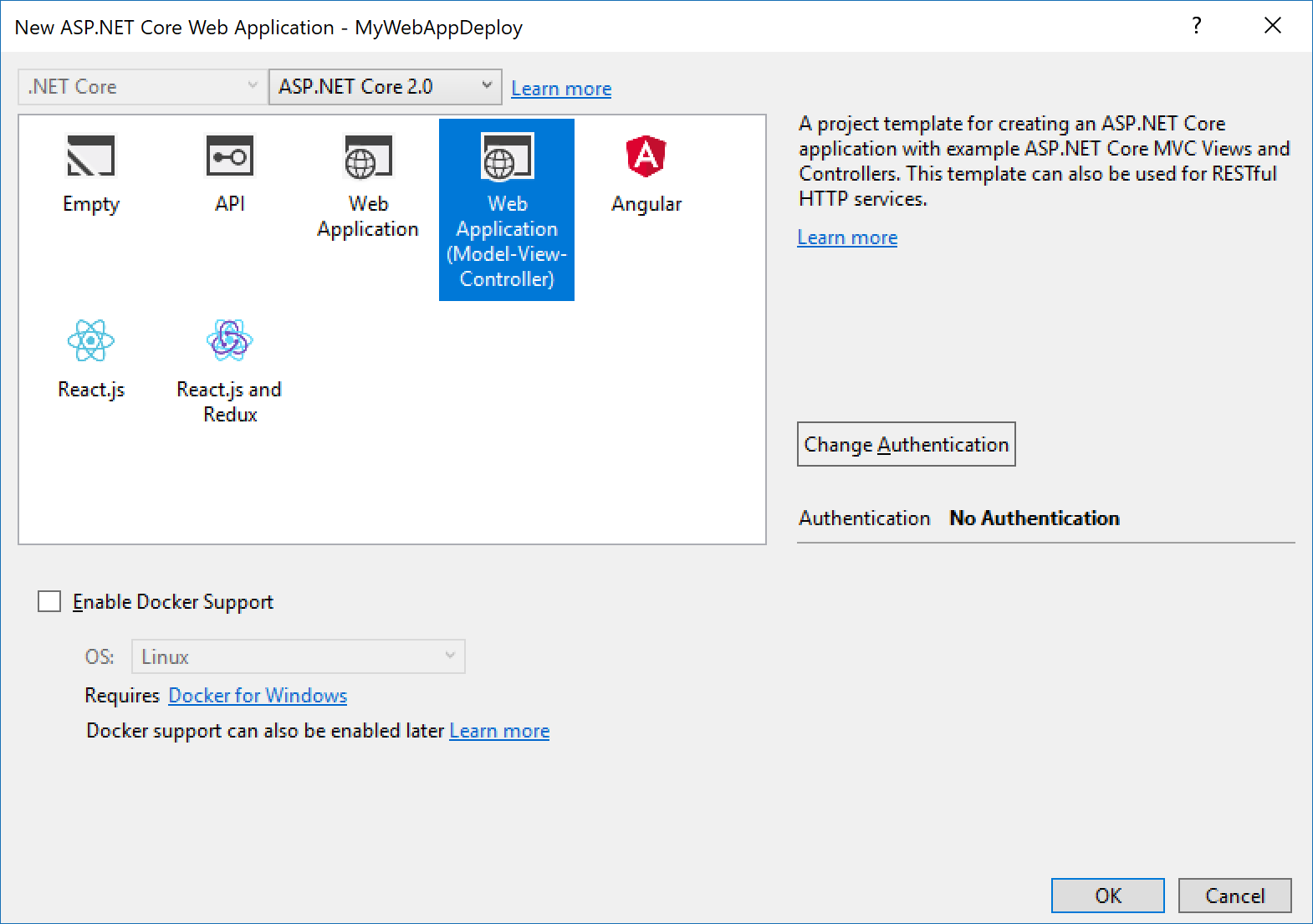
**Task 1 - Deploying a ASP.NET Core application to Azure**

1. Open **Visual Studio** and select **File | New | Project...** to create a new solution.
2. In the **New Project** dialog box, select **ASP.NET Core Web Application** under the **Visual C# | Web | .NET Core** tab. Name the project *MyWebAppDeploy*, choose a **Location** and click **OK**.



*New ASP.NET Web Application project*

1. In the **New ASP.NET Project** dialog box, select the **Web Application** **(Model-View-Controller)** template under **ASP.NET Core Templates**. Also, make sure that the **Authentication** option is set to **No Authentication**. Make sure the "**Enable Docker Support**" option is not checked (you will run this manually). Click **OK** to continue.

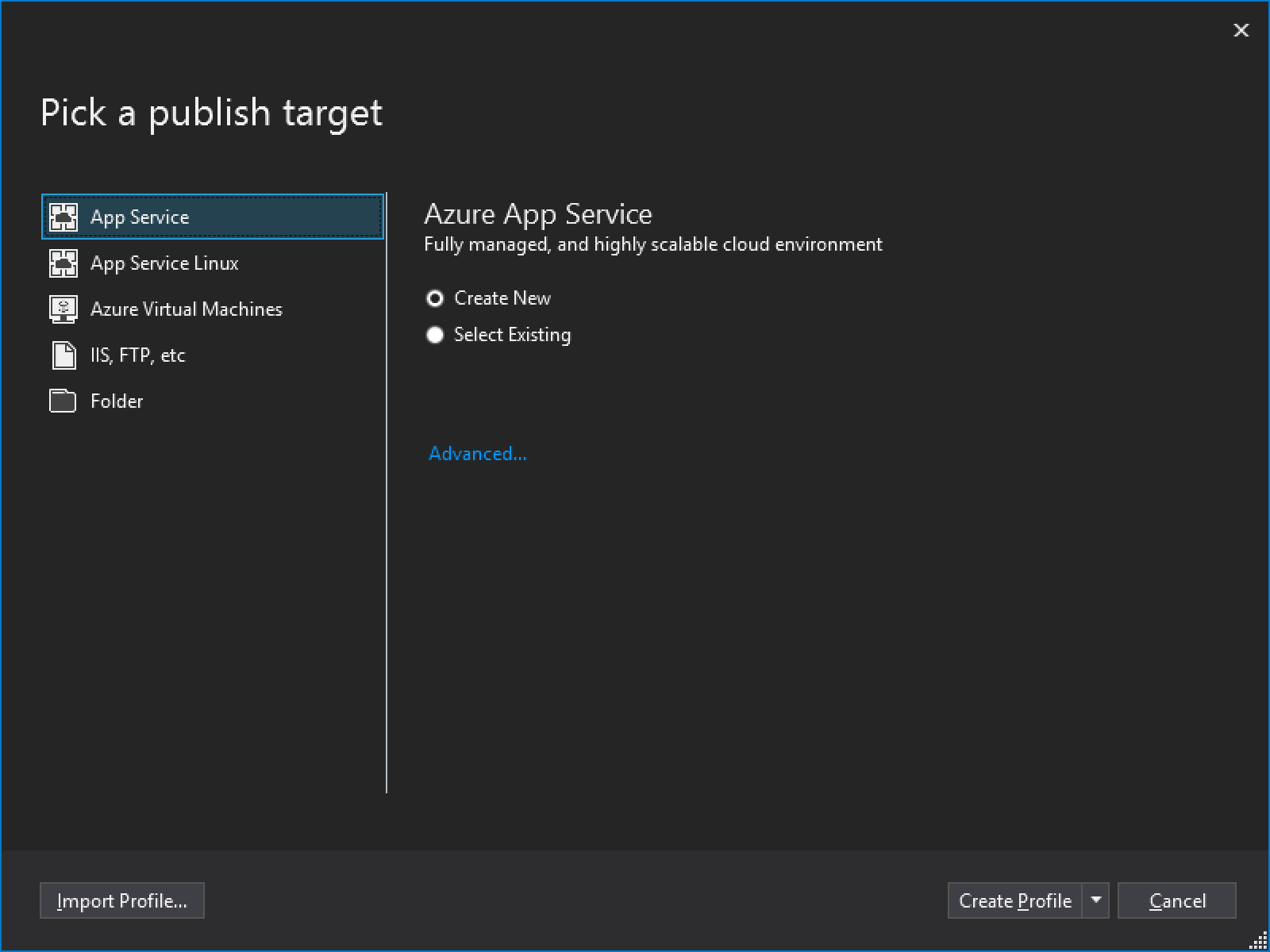


*Creating a new project with the Web Application template*

1. We are going to show the operative system version on the Web App. To do that add the following code line to the /Views/Shared/\_Layout.cstml before **@RenderBody()** file:

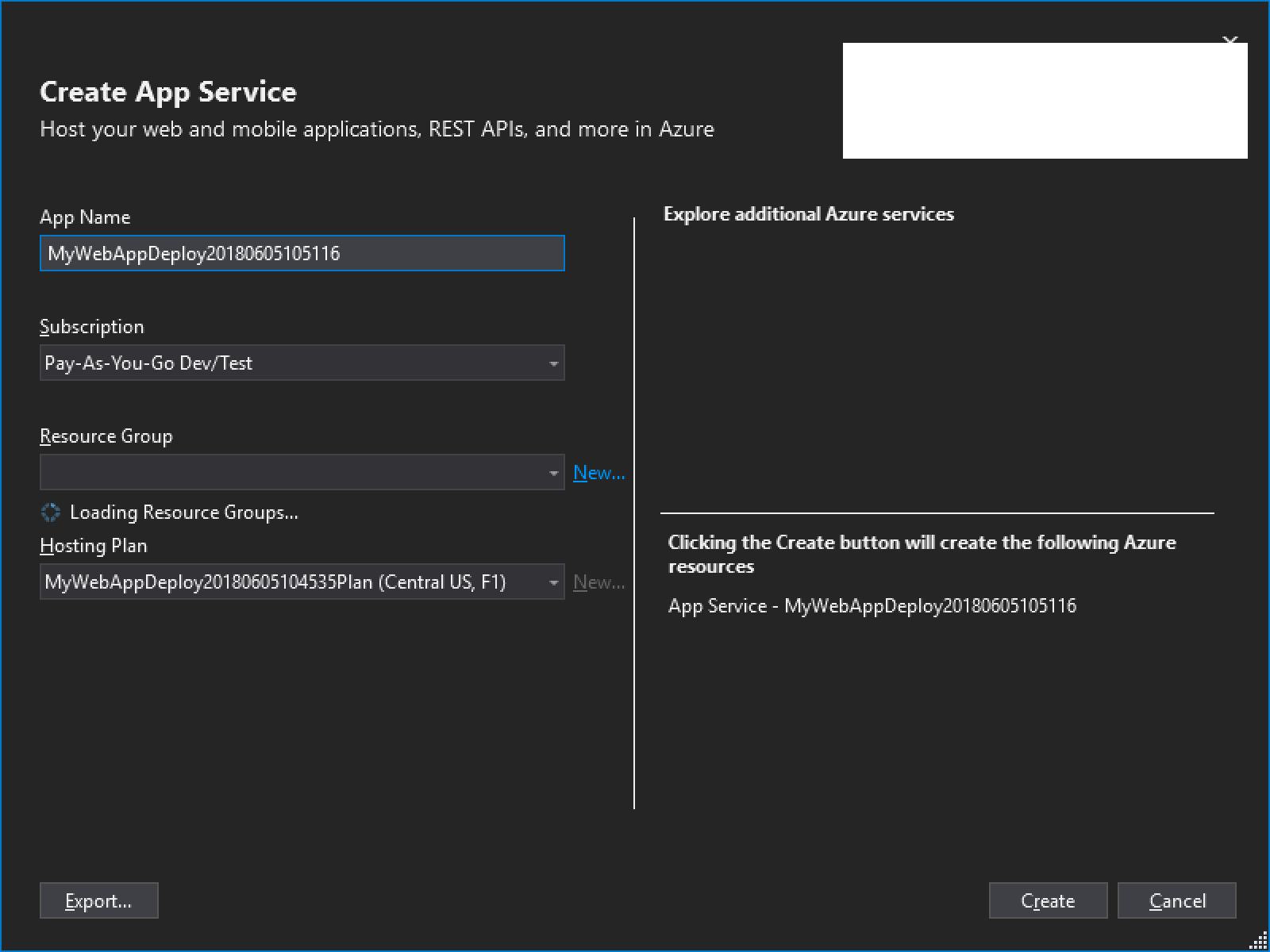
@System.Runtime.InteropServices.RuntimeInformation.OSDescription

1. Right-click the **MyWebAppDeploy** project and select **Publish...**. In the **Publish Web** dialog, click **Microsoft Azure App Service**, and then click **Create Profile**. If you don’t have and Acccount you will be prompted to add one.



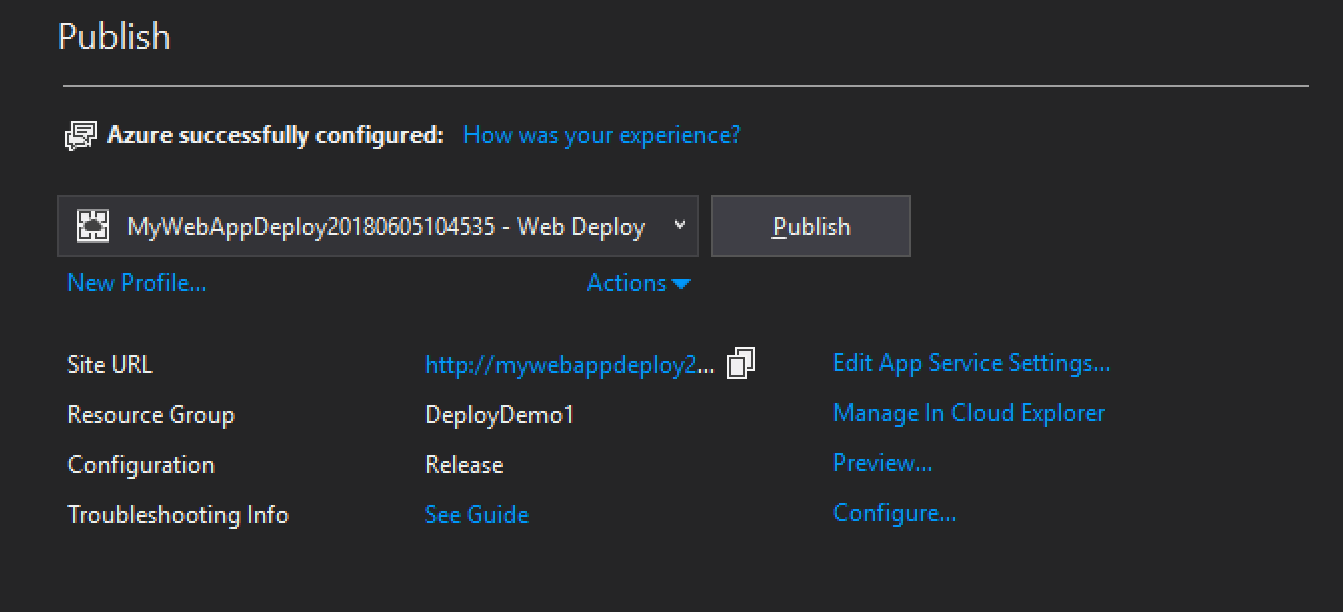
*Microsoft Azure App Service*

1. Then, the *Create App Service* dialog box will appear. Fill the **App Name, Subscription** and **Resource Group** fields, and choose a **Hosting Plan**. Then click the **Create** button.



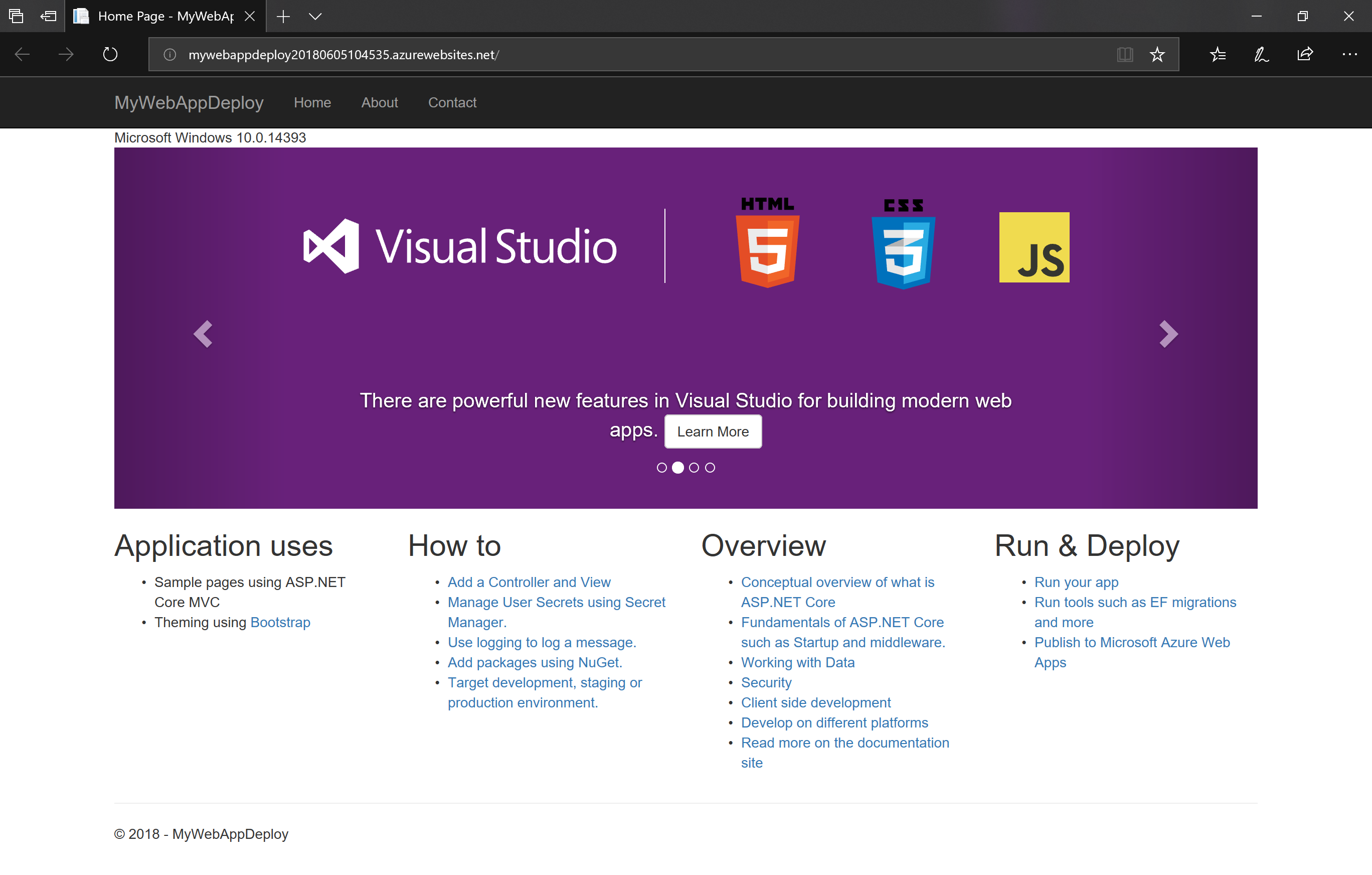
*Create App Service dialog box*

1. After the **App Service** was created, press **Publish** to publish the App .



*Publish to the App Service*

1. Once publishing completes, the web app will be automatically launched in your browser (at **http://{yourwebappname}.azurewebsites.net**).



*Web app published to Azure*

**Exercise 2: Working with Environments in Visual Studio**

ASP.NET Core introduces improved support for controlling application behavior across multiple environments, such as development, staging, and production. Environment variables are used to indicate which environment the application is running in, allowing the app to be configured appropriately.

In this exercise, you'll add code to this application to change its behavior based on the active environment. You will use Visual Studio launch profiles to test different environments locally.

ASP.NET Core uses a particular environment variable, ASPNET\_ENV (or Hosting:Environment), to describe the environment the application is currently running in. This variable can be set to any value you like, but three values are used by convention: Development, Staging, and Production.

**Task 1 - Add Environment Specific Code**

1. Open the Views\Shared\\_Layout.cshtml file in your editor. Observe the Environment TagHelper which renders its content only if the active environment matches one of the values in the names attribute.

<environment include="Development">

<link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.css" />

<link rel="stylesheet" href="~/css/site.css" />

</environment>

<environment exclude=" Development ">

<link rel="stylesheet" href="https://ajax.aspnetcdn.com/ajax/bootstrap/3.3.6/css/bootstrap.min.css"

asp-fallback-href="~/lib/bootstrap/dist/css/bootstrap.min.css"

asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-test-value="absolute" />

<link rel="stylesheet" href="~/css/site.min.css" asp-append-version="true" />

</environment>

1. At the top of the layout file, add the following line to inject the Hosting Environment service into your view.

@inject Microsoft.AspNetCore.Hosting.IHostingEnvironment HostingEnvironment

1. Find the footer element in layout page and modify it to display the active environment.

<footer>

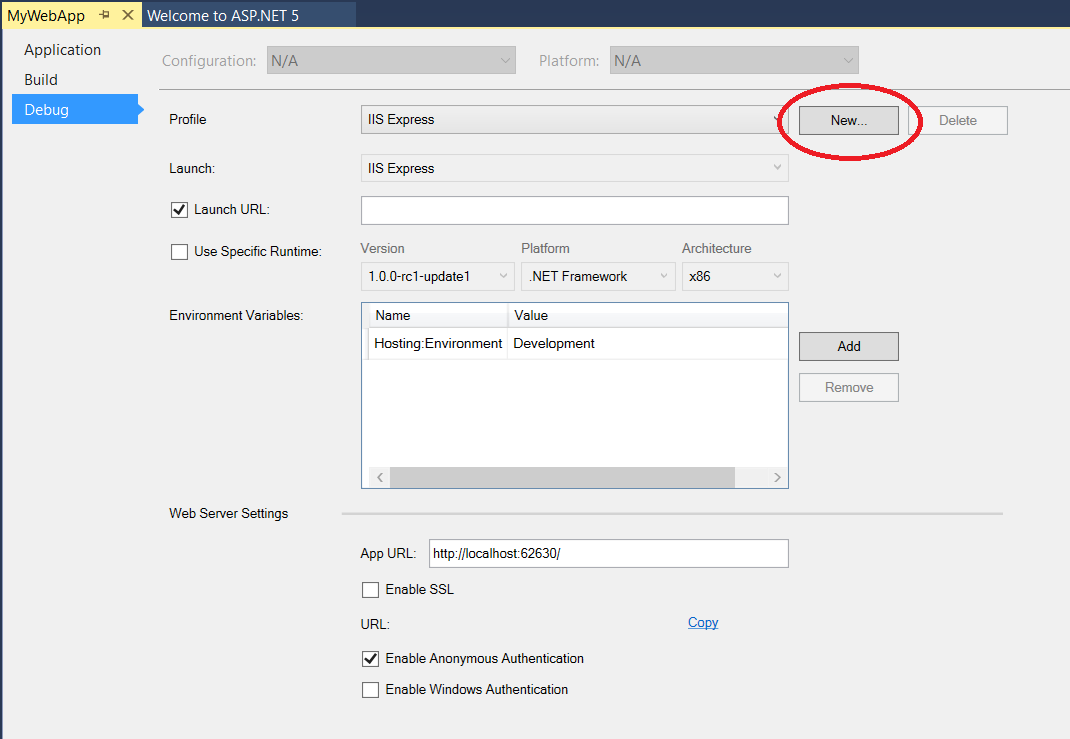
<p>&copy; 2018 - MyWebApp</p>

<p> Env: @HostingEnvironment.EnvironmentName </p>

</footer>

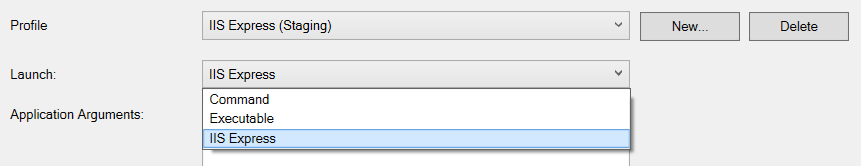
**Task 2 - Create a Visual Studio Launch Profile**

1. Right-click on the project in solution explorer and click on properties.
2. Select the **Debug** tab in the project properties window and click **New** button.

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/05-CreateNewLaunchProfile.png)

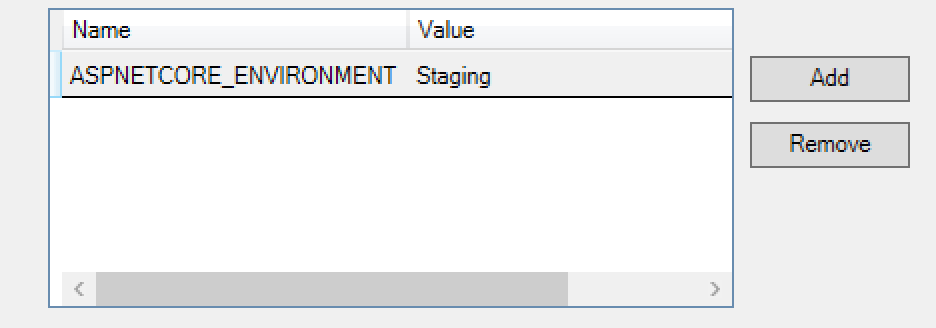
*Create a new launch profile*

1. Name the new launch profile **IIS Express (Staging)**
2. In the **Launch** dropdown chose **IIS Express**

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/06a-SetLaunchApp.png)

*Launch IIS Express*

1. Add an **Environment Variable** named ASPNETCORE\_ENVIRONMENT with a value of Staging and save the changes to your project properties.



*Set Environment Variable*

1. In the **Debug** Dropdown, select **IIS Express (Staging)** and launch your application. You should now see the environment name displayed in the footer of your web application.

**Exercise 3: Working with Multiple Deployment Slots (Optional)**

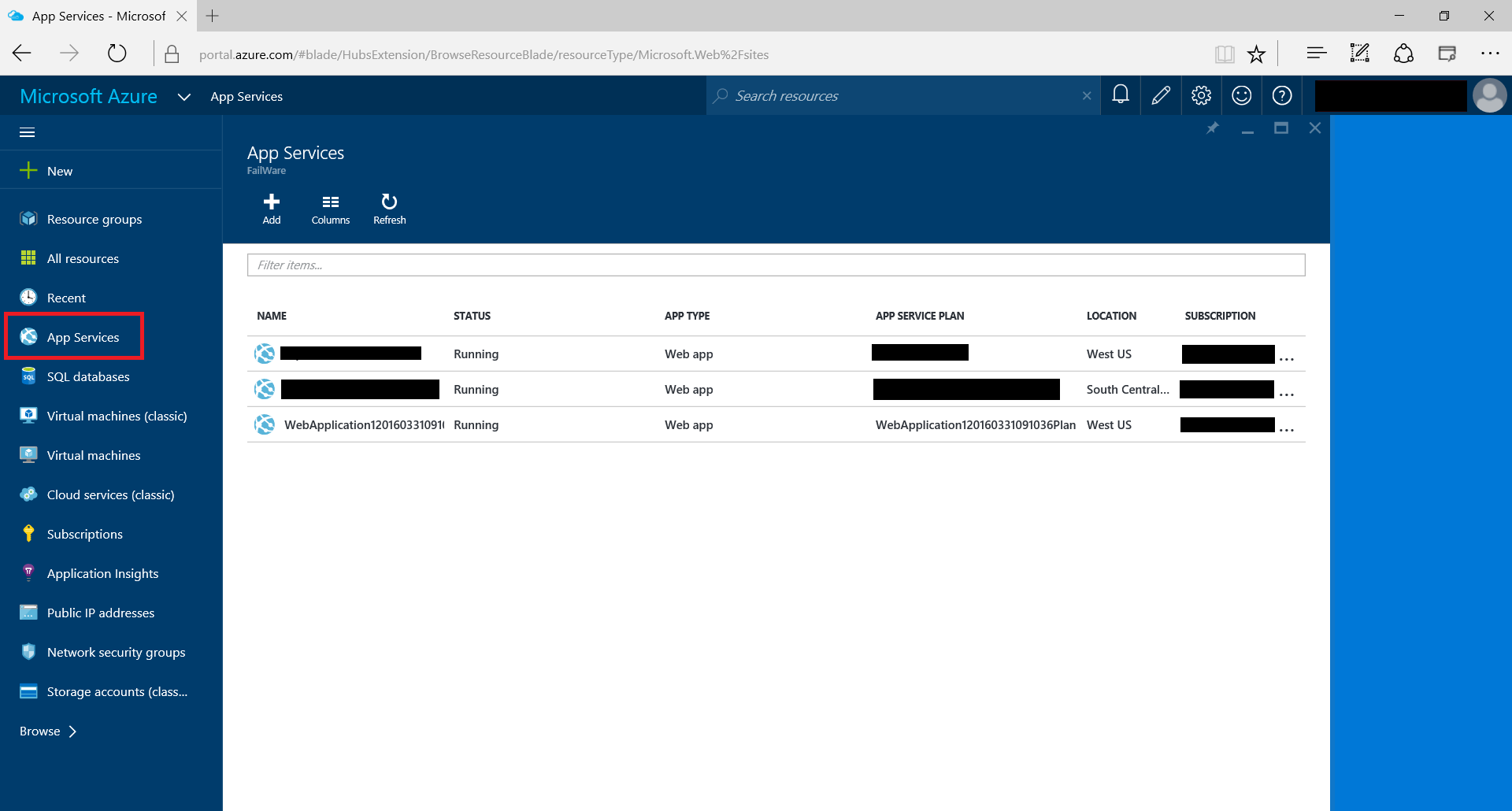
**Azure App Service Web Apps** enables you to perform staged publishing. When you deploy your site, you can choose to deploy it to a separate deployment slot instead of the default production slot. And then swap the deployments in these two slots with no down time. This is really useful for validating changes before releasing to the public, incrementally integrating site content, and rolling back if changes are not working as expected.

In this exercise, you'll deploy an application to the staging environment of your **Azure App Service Web Apps**. To do this, you'll create the Web App and provision the required components at the management portal, download a publish profile, and deploy from Visual Studio. You will then execute the application in this staging environment to verify its operation. Once you're satisfied that it's working according to your expectations, you'll promote the application to production.

**Note:** To enable staged publishing, the Web App must be on one of the Standard plans. Note that additional charges will be incurred if you upgrade your Web App to a Standard plan. For more information about pricing, see [App Service Pricing](http://azure.microsoft.com/en-us/pricing/details/app-service/).

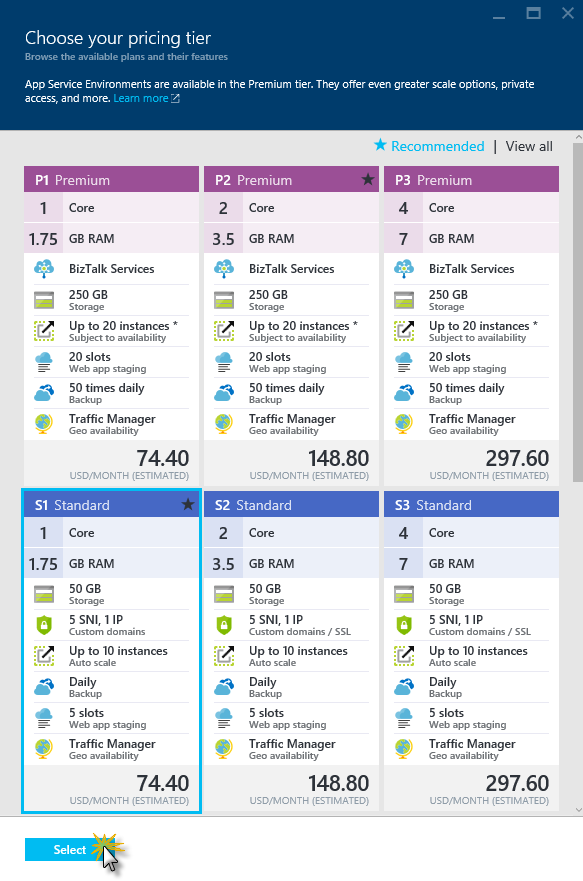
**Task 1 - Scaling up your Azure Web App**

1. Go to the [Azure Portal](https://portal.azure.com/) and sign in using the Microsoft account associated with your subscription.
2. Select **App Services** from the nav bar on portal.azure.com and select your previously created Web App.

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/09-SelectAppServiceInPortal.png)

*Select App Service in Portal*

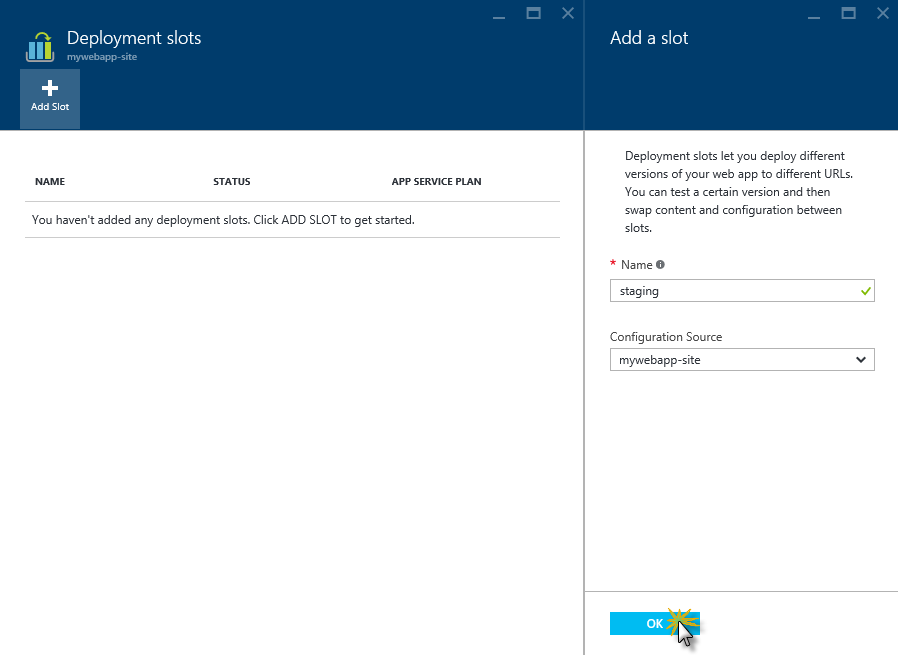
1. Select **Scale Up (App Service Plan)** in the **Settings** blade of your web app. If your Web App is not on a **Standard** plan, select one by clicking the **Pricing tier** tile. For instance, select the **S1 Standard** plan.

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/web-app-pricing-tiers.png)

*Web App Pricing tier*

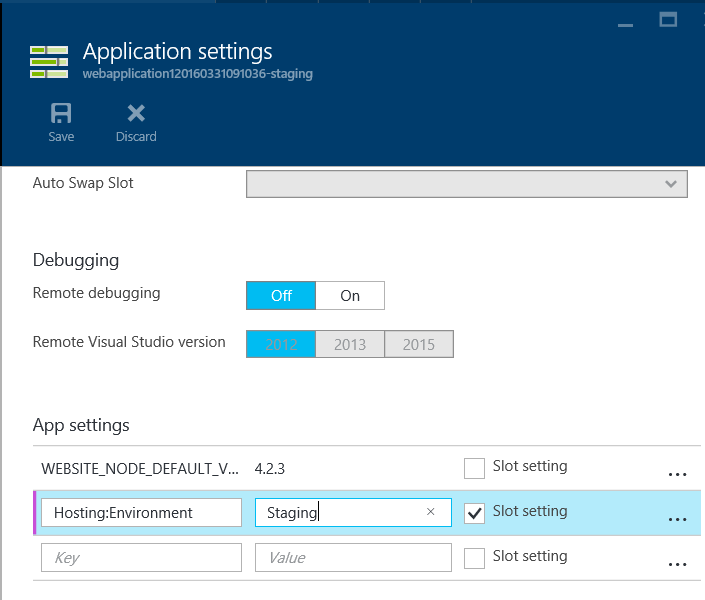
Microsoft Azure offers 5 plans for users to run their Web Apps - Free, Shared, Basic, Standard and Premium. In Free and Shared, all Web Apps run in a multi-tenant environment and have quotas for CPU, Memory, and Network usage. You can mix and match which sites are Free (strict quotas) vs. Shared (more flexible quotas). The maximum number of free Web Apps may vary with your plan. In Standard, you choose which Web Apps run on dedicated virtual machines that correspond to the standard Azure compute resources. You can change the mode of your Web App by clicking the **Pricing tier** tile in the **Usage** section of the corresponding App Service plan blade.

1. Back in the **Settings** blade, select **Deployment slots**. Click the **Add Slot** command at the top and create a new slot named **staging**. Set your Web App as **Configuration Source** and then click **OK**.

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/creating-deployment-slot.png)

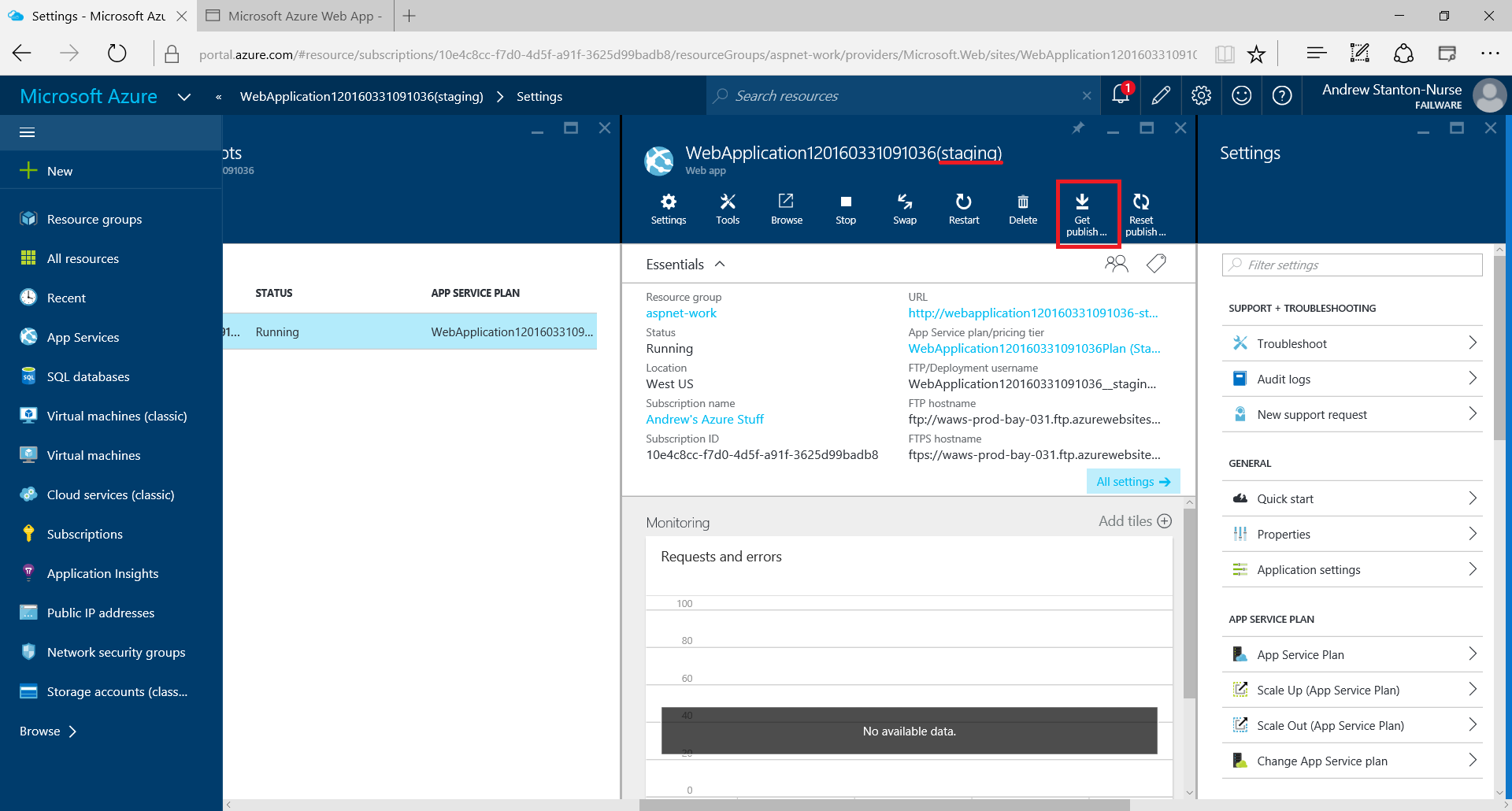
*Creating the staging deployment slot*

1. After a few seconds you'll see a new slot with the name of your Web App followed by ***-staging***. Select it to navigate to the *staging Web App* settings.
2. Select **Application Settings** in the **Settings** blade and add an **App Setting** named Hosting:Environment with a value of Staging

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/14-SettingStagingEnvironment.png)

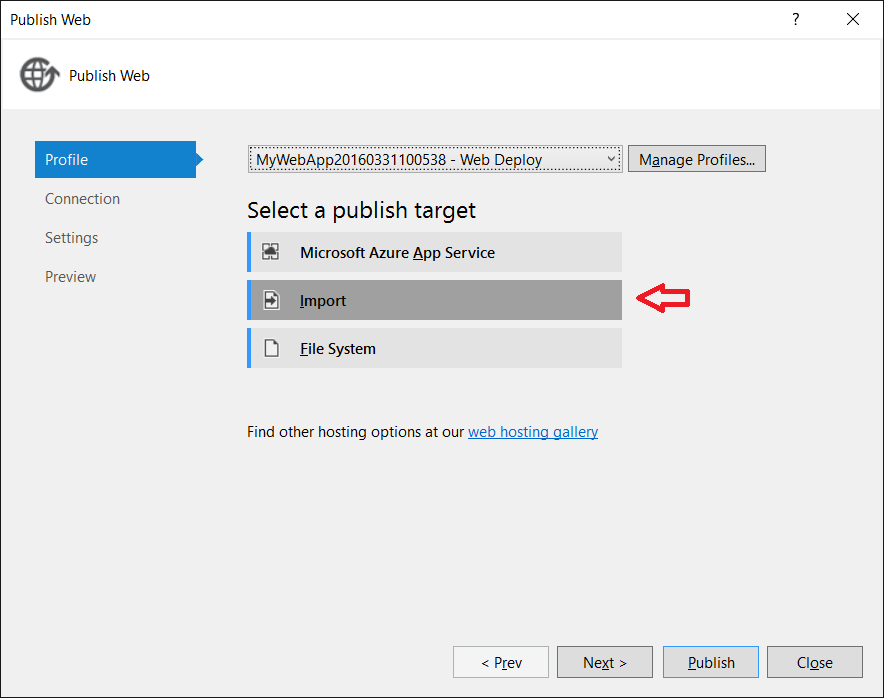
*Setting Staging Environment Variable*

1. Download the **Publish Profile** for the staging slot from the Web app blade.

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/11-GetPublishSettings.png)

*Download publish profile*

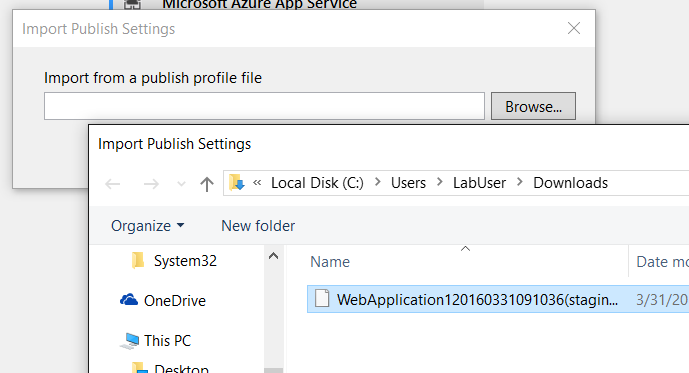
1. In Visual Studio, right-click on the Project in **Solution Explorer** and select **Publish...**.
2. In the **Publish Web** dialog, click the **Profile** tab and choose **Import**.

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/12-ImportPublishSettings.png)

*Import Publish Settings*

1. Import the downloaded **Publish Profile**

**Note**: Ensure you have downloaded the profile for your staging slot. The filename should end with *..(staging).publishsettings*

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/13-SelectPublishSettings.png)

*Select Publish Settings*

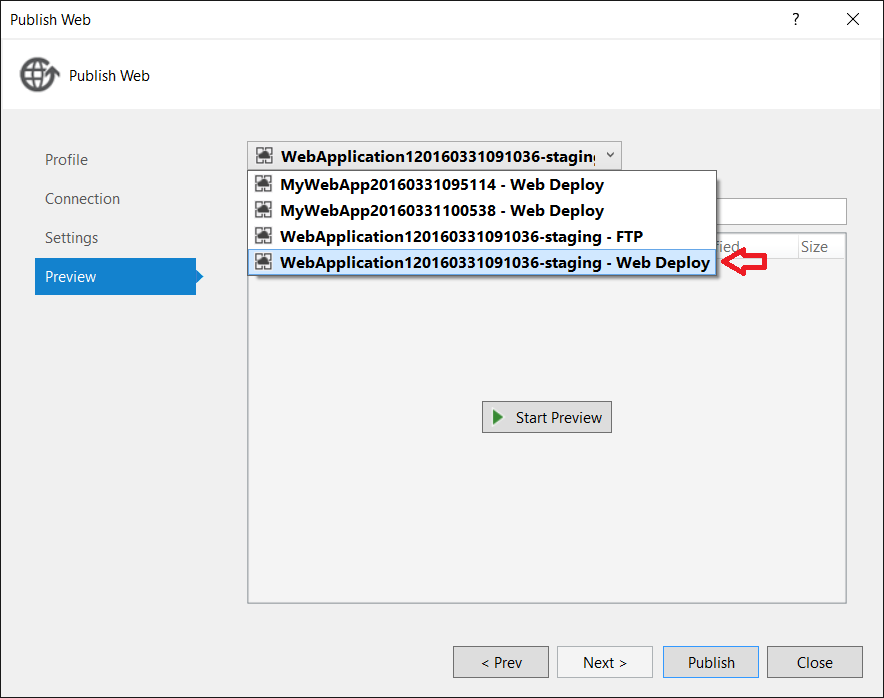
1. In the **Preview** tab on the **Publish Web** dialog, ensure you have selected the Web Deploy publish profile for your staging slot and then click Publish.

**Task 2 - Publish a change to the staging slot**

1. In Visual Studio, use the **Solution Explorer** to open the Views\Shared\\_Layout.cshtml file.
2. Change the content of the **About** link to **About Us**

<li><a asp-controller="Home" asp-action="About">About</a></li>

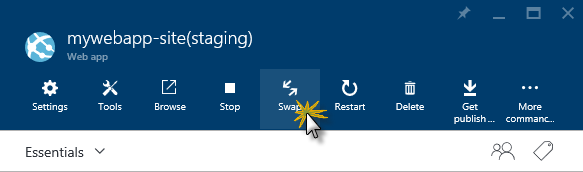
1. Right-click the **MyWebAppDeploy** project and select **Publish...**. In the **Publish Web** dialog, select your staging slot and click Publish.

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/15-PublishToStagingSlot.png)

*Publish to Staging*

**Task 3 - Promote your staging slot to production**

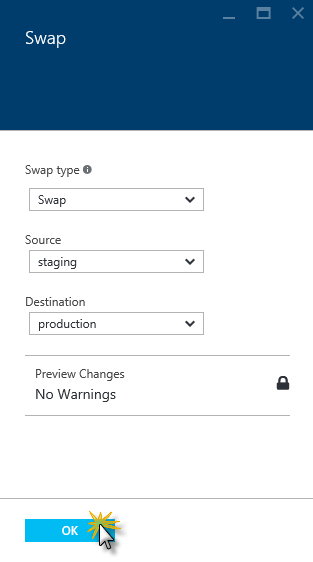
1. Go back to the Azure Portal and navigate to the *staging Web App*.
2. Click the **Swap** command at the top.

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/swap-to-production.png)

*Swap to production*

1. Verify that the **Source** targets the staging slot and the **Destination** targets production, and then click **OK** to proceed with the swap operation. Azure will immediately swap the content of the production site with the content of the staging site.

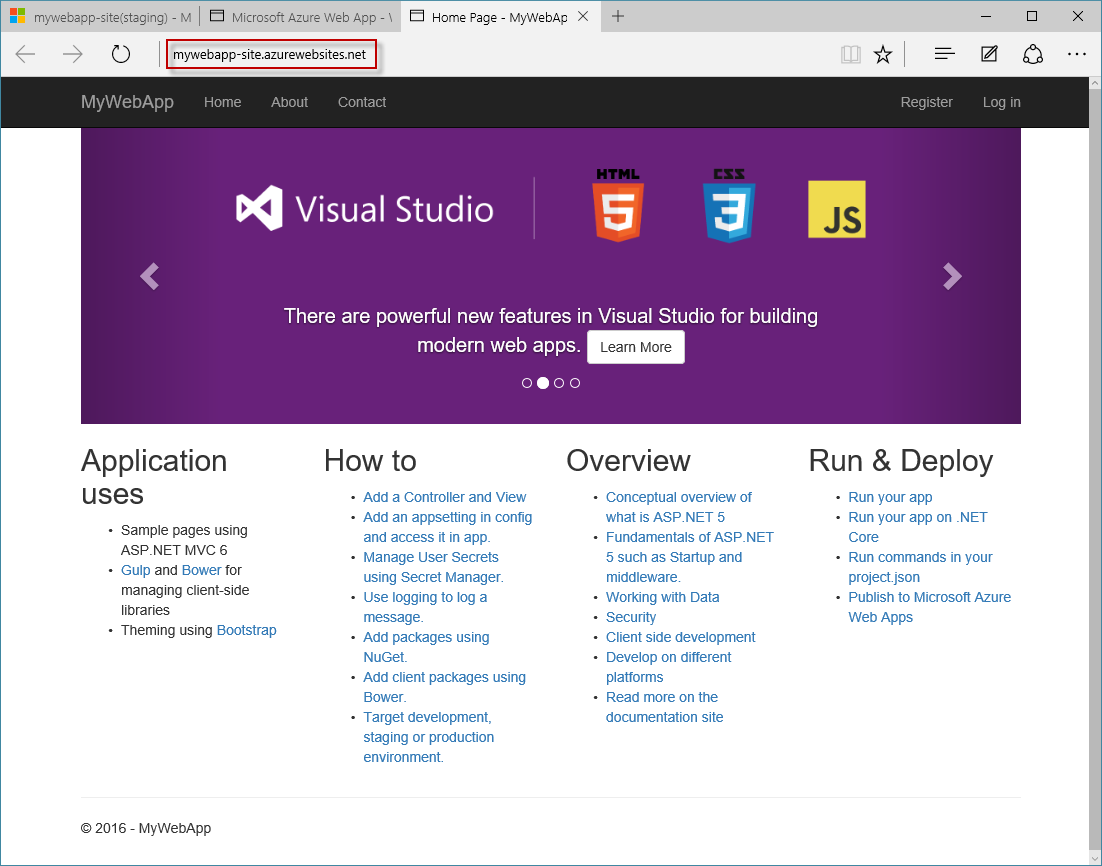
**Note:** Some settings from the staged version will automatically be copied to the production version (e.g. connection string overrides, handler mappings, etc.) but others will stay the same (e.g. DNS endpoints, SSL bindings, etc.).

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/confirm-swap-operation.png)

*Confirming swap operation*

1. Once the swap is complete, browse to your Web App in both slots. You can verify that the production site is now the one with the deployed application.

**Note:** You might need to refresh your browser to clear the cache. In Microsoft Edge, you can do this by pressing **CTRL+F5**.

[](https://github.com/Microsoft-Build-2016/CodeLabs-WebDev/blob/master/Module3-DeploymentAndAzure/Images/web-app-running-in-prod.png)

*Web App running in the production environment*

**Exercise 4: Working with Docker integration**

**Visual Studio** provides a mechanism to integrate our web app with [Docker](https://www.docker.com/).

Docker is the world’s leading software container platform. Developers use Docker to eliminate “works on my machine” problems when collaborating on code with co-workers. Operators use Docker to run and manage apps side-by-side in isolated containers to get better compute density. Enterprises use Docker to build agile software delivery pipelines to ship new features faster, more securely and with confidence for both Linux and Windows Server apps.

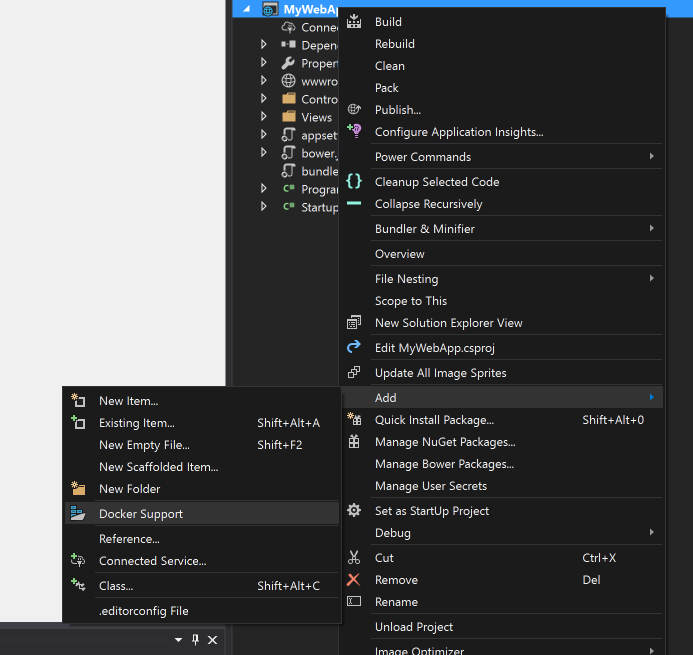
Using containers, everything required to make a piece of software run is packaged into isolated containers. Unlike VMs, containers do not bundle a full operating system - only libraries and settings required to make the software work are needed. This makes for efficient, lightweight, self-contained systems and guarantees that software will always run the same, regardless of where it’s deployed.

Docker automates the repetitive tasks of setting up and configuring development environments so that developers can focus on what matters: building great software.

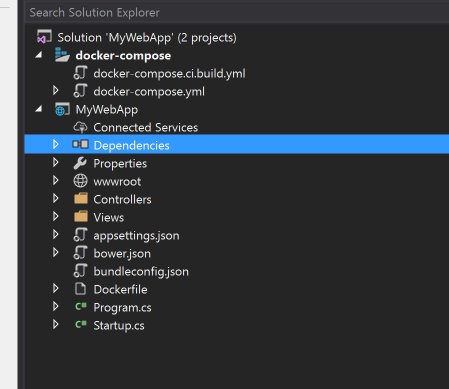
Developers using Docker don’t have to install and configure complex databases nor worry about switching between incompatible language toolchain versions. When an app is dockerized, that complexity is pushed into containers that are easily built, shared and run. Onboarding a co-worker to a new codebase no longer means hours spent installing software and explaining setup procedures. Code that ships with Dockerfiles is simpler to work on: Dependencies are pulled as neatly packaged Docker images and anyone with Docker and an editor installed can build and debug the app in minutes.

**Task 1 – Enable Docker integration for the Web App**

1. In Visual Studio, right-click on the Project in **Solution Explorer**, go to the **Add** command and select **Docker Support**.



*Adding Docker support*

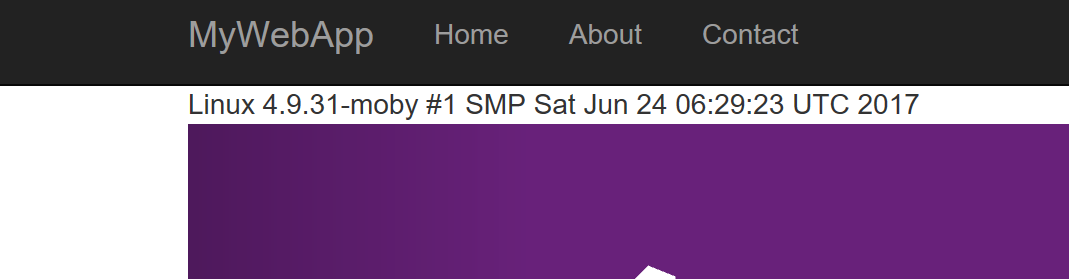
1. This will add a **docker-compose** project and a **Dockerfile** file on the root of the Web App  
     
   

*Docker support files*

1. In Visual Studio, right-click on the **docker-compose** project in Solution Explorer, and select **Set as Startup Project**
2. Before running the project, make sure that the **FROM** image configuration on **Dockerfile** (the first line), is set to the following, to use the latest image version:

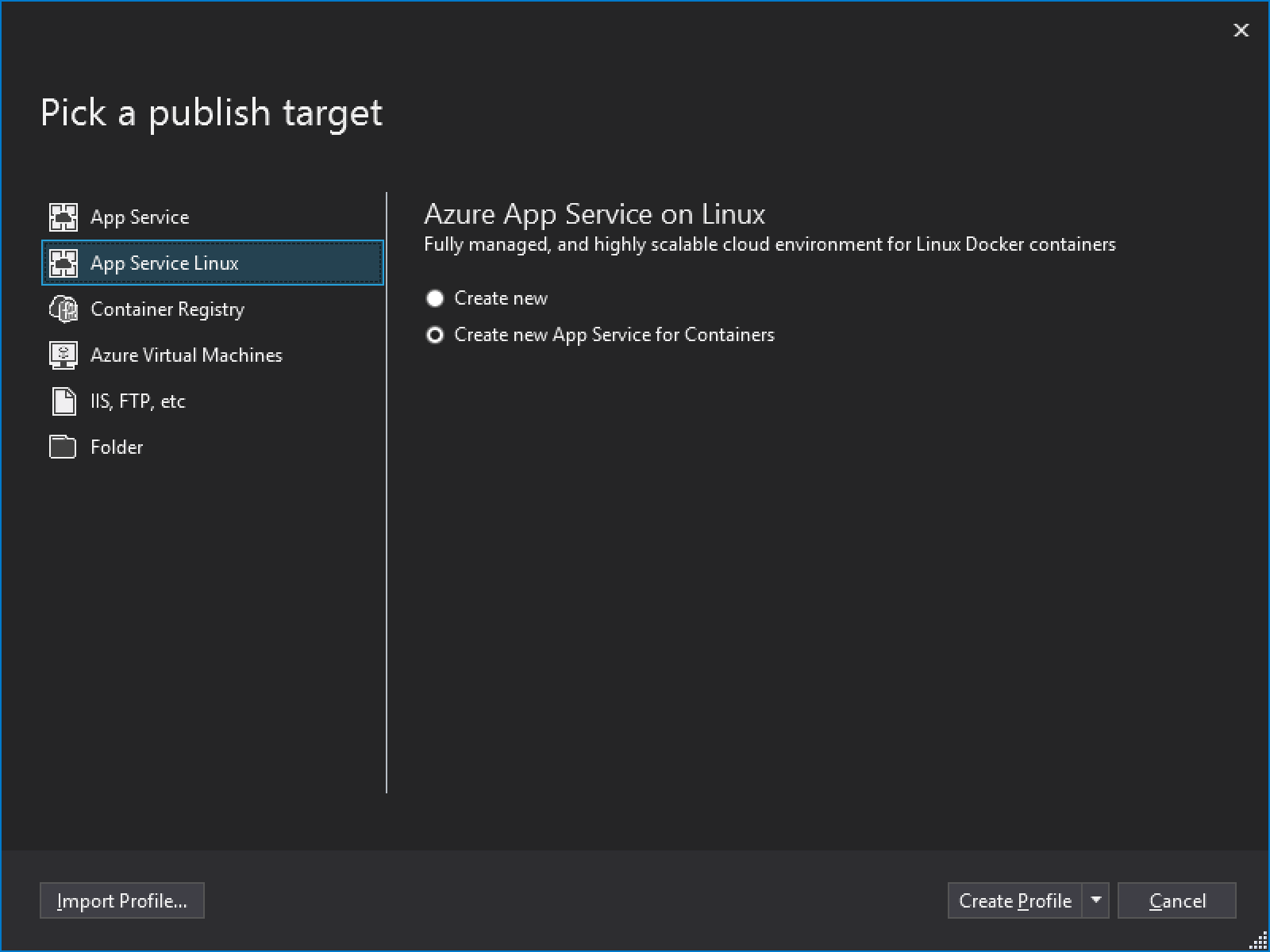
FROM microsoft/aspnetcore:2.0

1. Run the App using **F5**. You can check the **Output** window to see how **Visual Studio** create the **docker image,** using the **docker cli** commands.
2. After the application start, check the **Linux** OS version on the top left. This is the **Linux** version that the container is offering and is the same where the **.NET Core** Multiplatform Web App is running



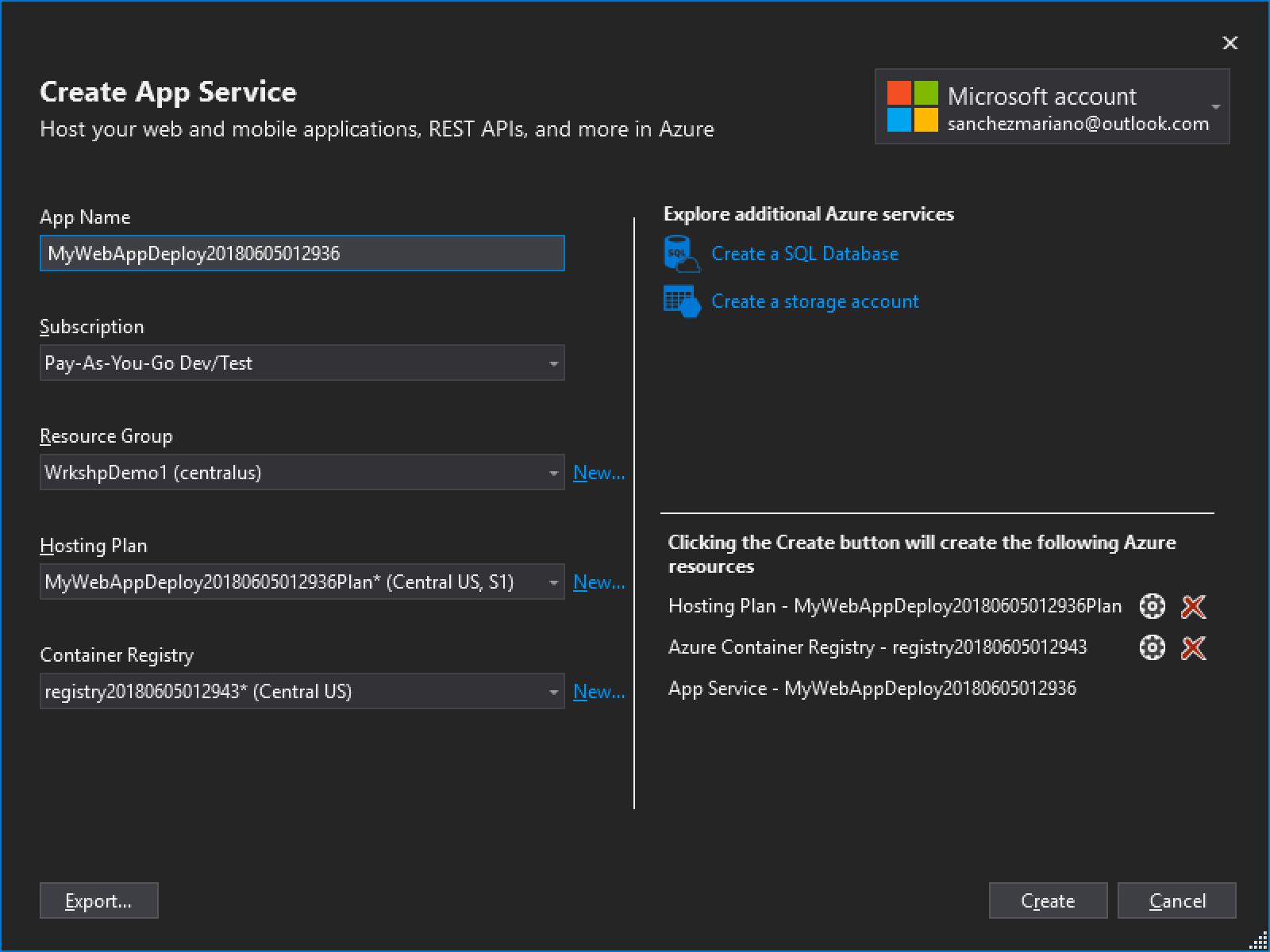
*Deployed Web App on Azure*

1. Under this configuration, you can debug your Web App running on **Linux** using **Docker**. You can open the **HomeController** and add some breakpoints to test de debugging experience.
2. Additionally, you can deploy this containerized Web App on Azure. For this, we need to create a new publish profile
3. Right-click the **MyWebAppDeploy** project and create a new **Publish** **profile**. In the **Publish Profile** dialog, click **Azure App Service Linux**, and then click **Create Profile**.



*Publish target for Azure App Service Linux*

1. Configure the App Service, you can reuse the Resource Group and the App Service Plan. You will need to create a Container Registry using the Publish UI. After the creation of the Container Registry, press Create.



1. After the resources are created, press the **Publish** button. You will see a **docker** console showing the progress. This operation will take some time, because we are uploading a full **docker** container.
2. After the **Publish** is completed, you can check the OS version of the App running on **Linux** on **Azure** using **Docker**. Additionally, you can compare with the previously deployed Web App to the **Azure App Service** using the first created publish profile (Web Deploy), and check the OS Version differences.

**Summary**

By completing this module, you should have:

* Created and deployed a Web Application to a Microsoft Azure Web App using Visual Studio
* Changed behavior of your app based on different deployment environments
* Worked with Multiple Deployments Slots in Azure
* Added Docker support and deployed on Linux on Azure