Azure Fast Start for Mobile Application Development

Module 5: Azure Web Apps

Student Lab Manual

Instructor Edition (Book Title Hidden Style)

Version 1.0

**Conditions and Terms of Use**

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2015 Microsoft Corporation. All rights reserved.

**Copyright and Trademarks**

© 2015 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at  
<http://www.microsoft.com/en-us/legal/intellectualproperty/Permissions/default.aspx>

Azure, Microsoft, Internet Explorer, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Contents

[Lab 1: Website Front End 1](#_Toc433385518)

[Exercise 1: Create a WordPress Blog 2](#_Toc433385519)

[Task 1: Create a Web App in the Portal 2](#_Toc433385520)

[Exercise 2: Create a Landing Page 8](#_Toc433385521)

[Task 1: Set Up the Landing Site 8](#_Toc433385522)

[Task 2: Execute Website Locally 9](#_Toc433385523)

[Task 3: Deploy in the Cloud 10](#_Toc433385524)

[Exercise 3: Use the sites in the app settings 11](#_Toc433385525)

[Task 1: Add a Link in Client Application to Application Blog 13](#_Toc433385526)

[Lab 2: Contoso Products Admin Web Application 19](#_Toc433385527)

[Exercise 1: Create an Admin Website 20](#_Toc433385528)

[Task 1: Create a ASP.NET Web Application 20](#_Toc433385529)

[Task 2: Add Scaffolding 23](#_Toc433385530)

[Exercise 2: Set up the Website Environment 27](#_Toc433385531)

[Task 1: Add a Deployment Slot to a Web App 27](#_Toc433385532)

[Task 2: Configuration for Deployment Slots 29](#_Toc433385533)

[Task 3: Swap Deployment Slots 31](#_Toc433385534)

[Exercise 3: Enabling Continuous Integration and Continuous Deployment 32](#_Toc433385535)

[Task 1: Create Team Project 32](#_Toc433385536)

[Task 2: Set Up Continuous Deployment 34](#_Toc433385537)

[Task 3: Check In a Project to Source Control 38](#_Toc433385538)

[Task 4: Trigger a Rebuild and Redeploy Your Project (Optional) 42](#_Toc433385539)

[Lab 3: Azure Web Jobs 47](#_Toc433385540)

[Exercise 1: Set up and deploy 50](#_Toc433385541)

[Task 1: Create Azure Storage Account (Optional) 50](#_Toc433385542)

[Task 2: Download the application 52](#_Toc433385543)

[Task 3: Configure the Application to Use Your Storage Account 53](#_Toc433385544)

[Task 4: Run the application locally 56](#_Toc433385545)

[Task 5: Run the Application in the Cloud 61](#_Toc433385546)

[Exercise 2: Create the application from scratch 73](#_Toc433385547)

[Task 1: Create a Visual Studio solution with a web project and class library project 74](#_Toc433385548)

[Task 2: Add a Console Application Project having WebJobs Deployment Enabled 76](#_Toc433385549)

[Task 3: Add NuGet Packages 77](#_Toc433385550)

[Task 4: Set project references 77](#_Toc433385551)

[Task 5: Add Code and Configuration Files 78](#_Toc433385552)

[Task 6: Review the Application Code 79](#_Toc433385553)

# Lab 1: Website Front End

#### Introduction

In this lab, you will create websites, customize and deploy them on the most used computer service in Microsoft Azure.

#### Objectives

After completing this lab, you will be able to:

* Provision pre-existing websites from gallery.
* Create simple html/css/javascript website.
* Deploy and manage site in Azure.

#### Estimated Time to Complete This Lab

60 minutes

#### Prerequisites

All resources allocated in Azure will follow the following naming rule:

* Prefix: W10AZFS-PC: Windows 10 Azure Fast Start – Product Catalog.
* You can use your own or ignore it as long as you stay consistent.

#### Scenario

Azure Web Apps can be convenient when developing mobile applications. Indeed, it can help you in few ways to create online resource for mobile application:

* Create a blog – handle public communication, official release, etc.
* Create a landing page – dedicated online pages to pair up with mobile app. Application Store usually requires privacy policy, support contact, and about information.

## Exercise 1: Create a WordPress Blog

#### Objectives

In this exercise, you will learn:

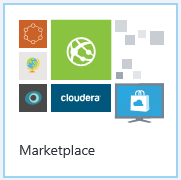
* How to create a new web app through the Azure Marketplace.
* How to deploy the web app through the Azure Preview portal.

#### Scenario

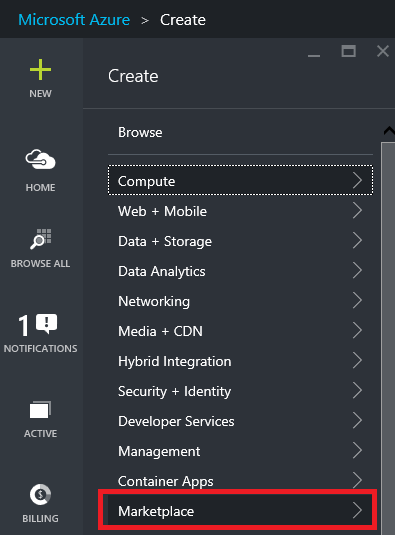
The Azure Marketplace makes available a wide range of popular web apps developed by Microsoft, third party companies, and open source software initiatives. Web apps created from the Azure Marketplace do not require installation of any software other than the browser used to connect to the [Azure Preview Portal](http://go.microsoft.com/fwlink/?linkid=529715&clcid=0x409).

### Task 1: Create a Web App in the Portal

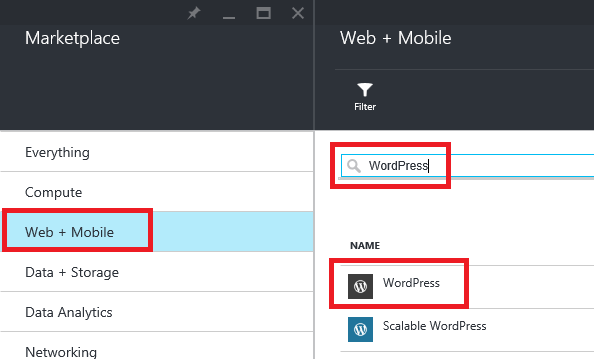
1. Log in to the [Azure Preview Portal.](http://portal.azure.com/)
2. To open the Azure Marketplace, click the **Marketplace** icon.



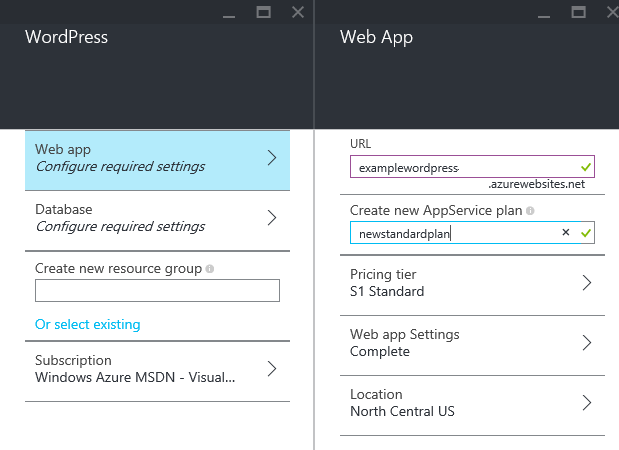
1. Alternatively, click the **New** icon on the upper-right corner of the dashboard, and select **Marketplace** at the bottom of the list.



1. Select **Web + Mobile**. In the search box, type **WordPress** and click the **WordPress** icon.

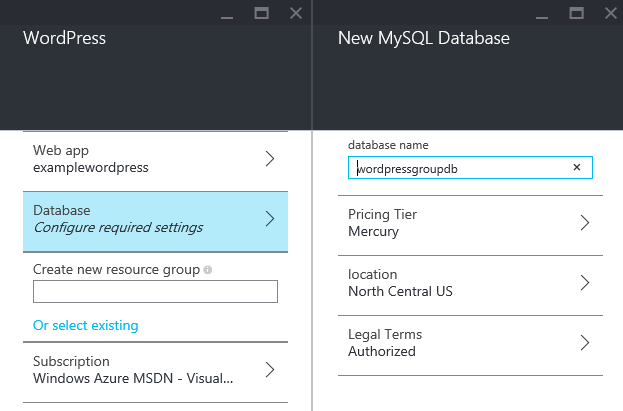


1. After reading the description of the WordPress app, click **Create**.
2. Click **Web app**, and provide the required values for configuring your web app.
3. In the **URL** box, type **w10azfs-pc-blog**.
4. In the Create new AppService plan box, type w10azfs-pc-front.



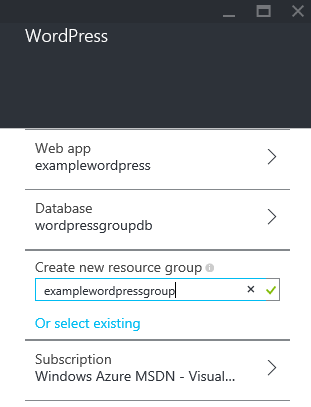
**Note**:  
**Resource name** The web application name has to be unique in Azure. However, you can always use custom domain name. For more details, see [Configure a custom domain name in Azure App Service](https://azure.microsoft.com/en-us/documentation/articles/web-sites-custom-domain-name/).  
**AppService plan** An App Service plan represents a set of features and capacity that you can share across multiple apps in Azure App Service, including web apps, mobile apps, logic apps, or API Apps. For more details, see [App Service plans](http://azure.microsoft.com/en-us/pricing/details/app-service/plans/).

1. **(Optional)** Change the web app location to match your user, in order to ensure the best user experience and performance (For example, for the main user in France, choose West Europe).
2. Click **Database**, and provide the required values for configuring your MySQL database. For example, in the **database name** box, type w10azfs\_pc\_wordpressdb.
3. **(Optional)** Change database location. Be careful of database location and ensure it matches the web apps to avoid additional charges.

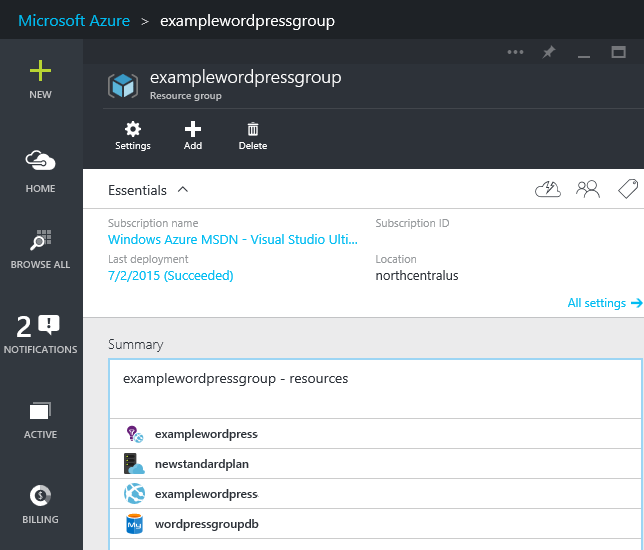


**Note**: MySQL database requires a credit card to be associated with the subscription. To add credit card to your subscription, [click here](http://account.windowsazure.com/subscriptions). Additionally, MySQL database is not available in certain regions and cannot be created by users who have an Enterprise Agreement or by co-admins. For more information, [click here](http://go.microsoft.com/fwlink/?LinkId=268693).

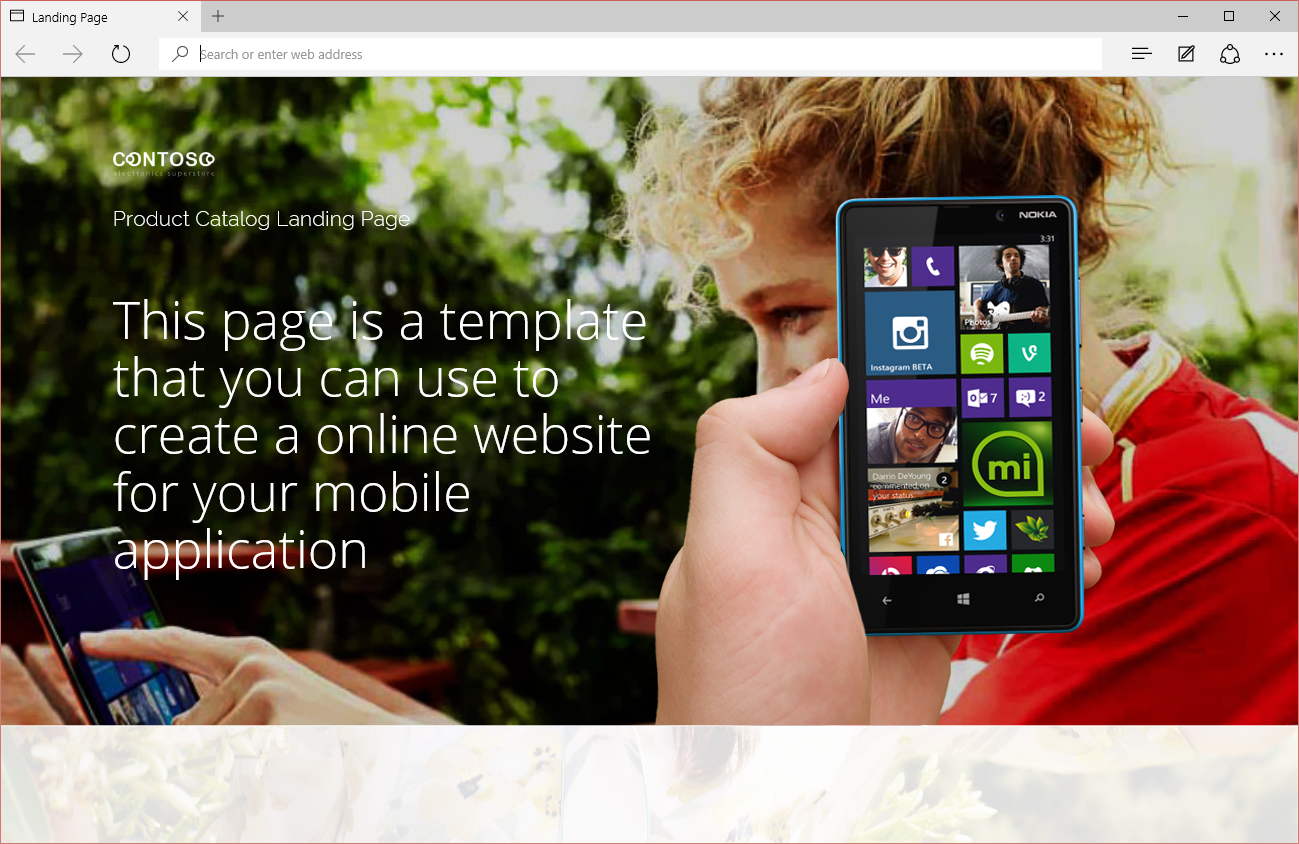
1. To provide a name for a new resource group, in the **Create new resource group** box, type a name, for example, W10AZFS-PC.



1. If necessary, click **SUBSCRIPTION**, and specify the subscription to use.
2. When you have finished defining the web app, click **Create**, and wait while the new web app is created.
3. When the app has been created, you will see the resource group containing web app and database.



1. Launch URL in the web browser and complete the WordPress online wizard.



## Exercise 2: Create a Landing Page

#### Objectives

In this exercise, you will learn:

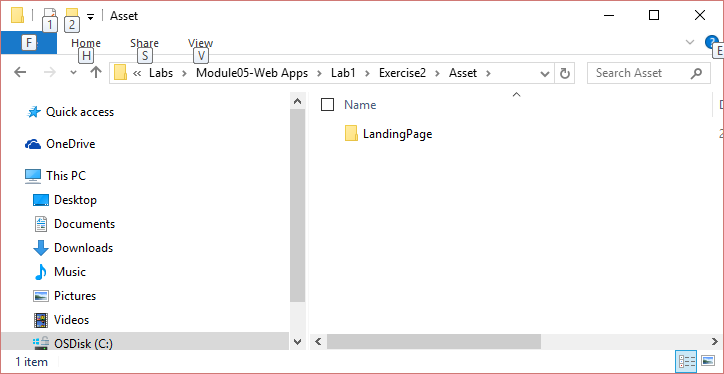
* How to enable your machine for Azure development by installing the Azure software development kit (SDK).
* How to create a Visual Studio ASP.NET web project and deploy it to an Azure web app.
* How to make a change to the web project and redeploy the application.
* How to use the [Azure Preview portal](https://azure.microsoft.com/overview/preview-portal/) to monitor and manage your web app.

#### Scenario

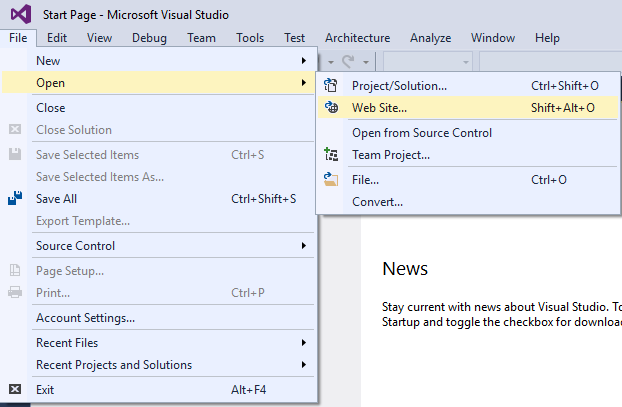
This tutorial shows how to create a ASP.NET web application and deploy it to [App Service Web Apps](https://azure.microsoft.com/en-us/documentation/articles/app-service-web-overview/) by using Visual Studio 2015 or Visual Studio 2013. The tutorial assumes that you have no prior experience in using Azure or ASP.NET. On completing the tutorial, you will have a simple web application up and running in the cloud.

### Task 1: Set Up the Landing Site

1. Copy the **LandingPage** folder (in the Asset folder- ***\Labs\Module05-Web Apps\Lab1\Exercise2\Asset***) to the ***\Labs\Module05-Web Apps\Lab1\Exercise2\*End** folder.



1. Launch Visual Studio.
2. In Visual Studio, click **File** > **Open** > **Web Site**.



1. Select the folder \Labs\Module05-Web Apps\Lab1\Exercise2\End\LandingPage.

### Task 2: Execute Website Locally

1. In Visual Studio, click **Debug** > **Start Without debugging** (or press Ctrl+F5).
2. A default browser will open the website.
3. Browse vertically and check out the html page (home page, privacy policy, contact, etc.)
4. **(Optional)** Customize content by opening the index.html file and updating text as required.
5. Change the following text to something more personal:

<div class="slogan">This page is a template that you can use to create a online website for your mobile application</div>

For example:

<div class="slogan">

The best app to keep to find information on electronic devices :

<ul>

<li>Audio</li>

<li>Cameras</li>

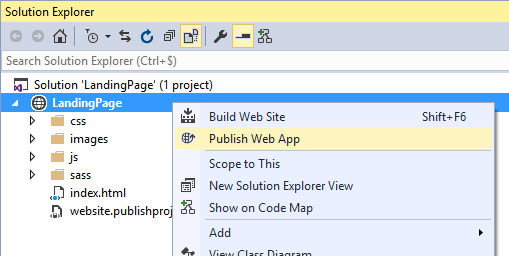
<li>Home Appliances</li>

</ul>

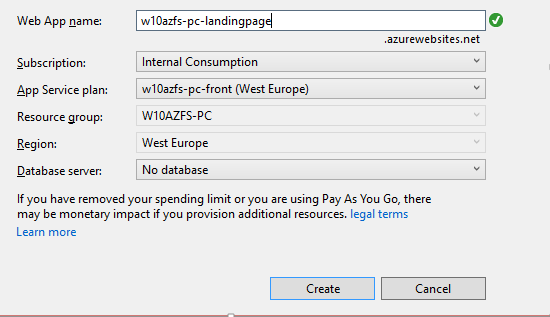
</div>

### Task 3: Deploy in the Cloud

1. In Solution Explorer of Visual Studio, right-click the project, and from the shortcut menu, select **Publish Web App**.



1. Create a new Azure Web App and follow the steps in the wizard. For example, in the Web App name box, type w10azfs-pc-lanidngpage.

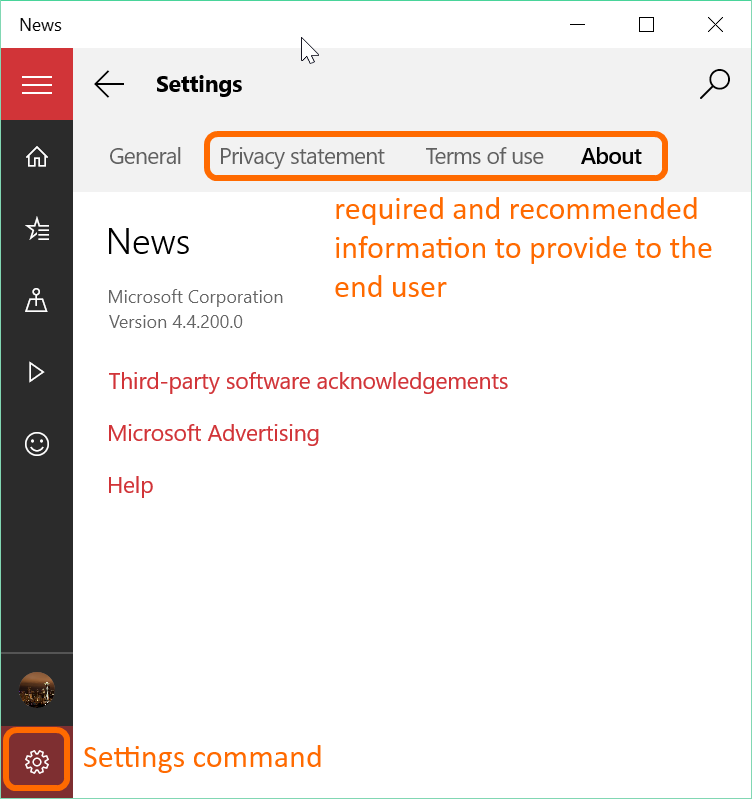


1. Once the Web App is created, Visual Studio will start the browser with the newly created landing page in the cloud. The next step would be to update the client application with all this information.

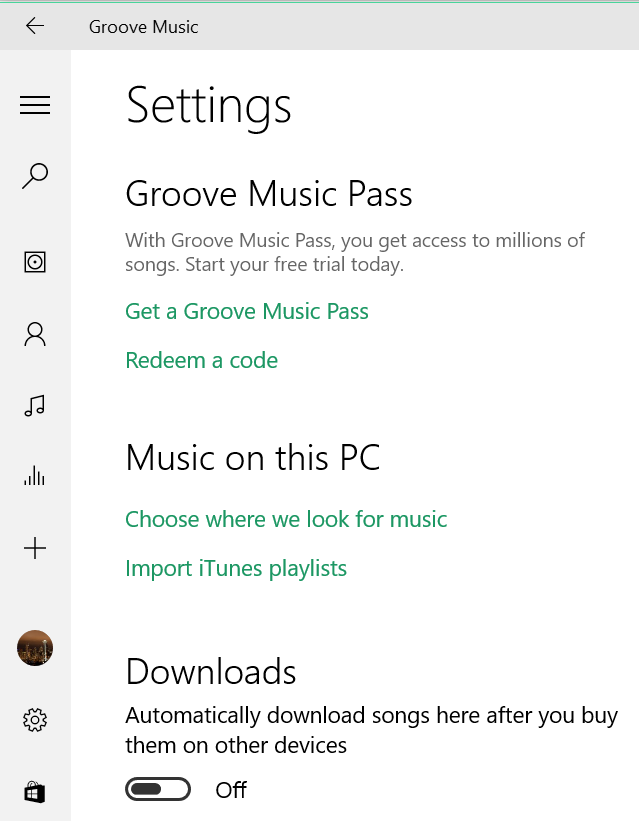
## Exercise 3: Use the sites in the app settings

#### Introduction

Most applications need to allow their users to customize its behavior and settings to run in the expected way. In addition, the Windows Store Policies require that all applications must provide an access to their privacy policy. As you can see from default Windows Apps such as News, this information is reachable through **Settings** window:



The Settings command is located at the bottom of the Hamburger menu in most of the applications. However, you are free to display the settings the way you want, like in News or in the Groove Music Pass case, for example.

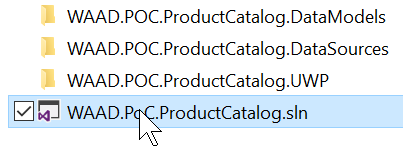


A new action should be available in the Hamburger menu of the app to provide access to the different settings sections.



### Task 1: Add a Link in Client Application to Application Blog

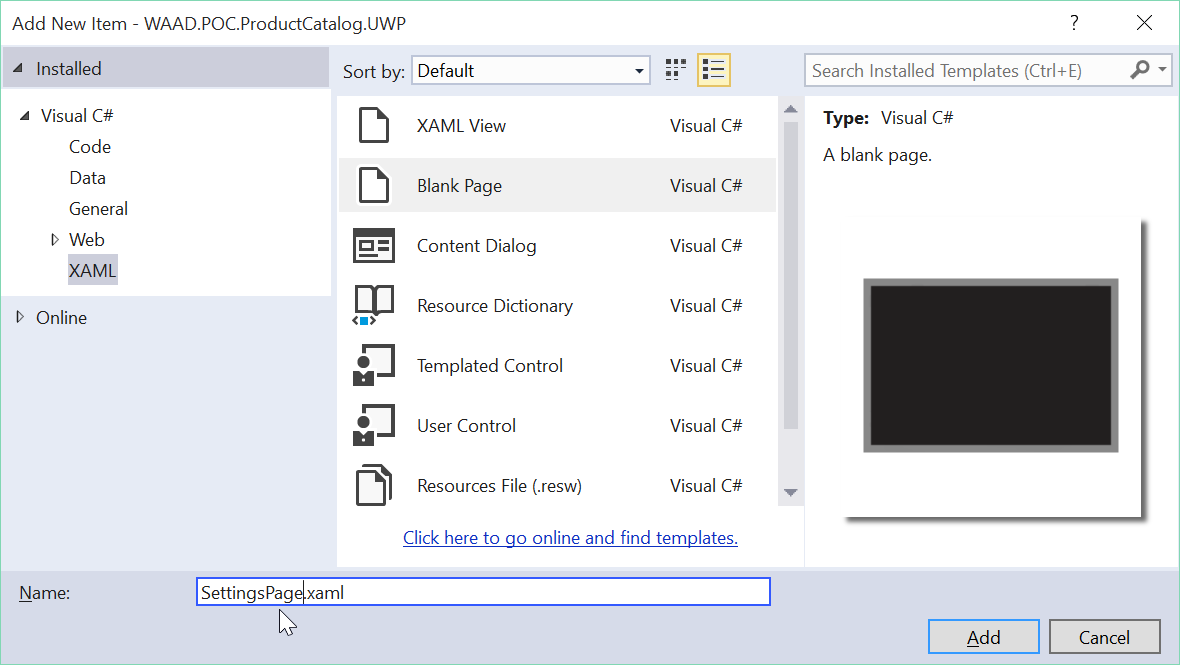
1. Using File Explorer, go to **\Module00-Apps\ProductCatalog\Source**, copy all content into the **\Module05-Web Apps\Lab1\Exercise3\Start** folder and double-click the **WAAD.PoC.ProductCatalog.sln** app to open it in Visual Studio.



1. Open the **AppShell.xaml** file and uncomment the settings radio button definition in the second row of the grid so it will appear at the bottom of the Hamburger menu.



1. Instead of MainCategory, replace the type of the view by SettingsPage. This is the name of the class that will be displayed when the user taps this command.   
   In Solution Explorer, right-click the **View** folder and from the shortcut menu, select **Add** > **New Item**.
2. In the Add New Item dialog box, in the left pane, expand Installed > Visual C# > Web and click XAML. Next, in the middle pane, click Blank Page, in the Name box, type SettingsPage.xaml, and then click Add.



1. In the page element at the top of the XAML, add the following namespace definitions:

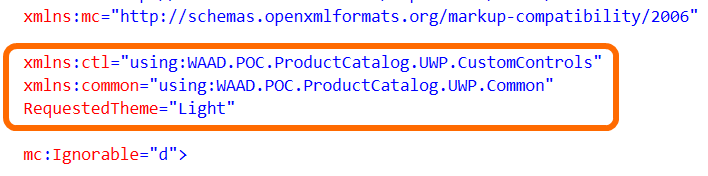
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:ctl="using:WAAD.POC.ProductCatalog.UWP.CustomControls"

xmlns:common="using:WAAD.POC.ProductCatalog.UWP.Common"

RequestedTheme="Light"

1. You now have:



1. The purpose is to be able to use the HeaderControl and DeviceFamilyAdaptiveTrigger:
2. Change the Grid element.



1. Add the definition of the VisualStateGroups of the VisualStateManager to change the user experience based on the device family (that is, set a margin of 0 for thin mobile devices).

<VisualStateManager.VisualStateGroups>

<VisualStateGroup x:Name="VisualStateGroup">

<VisualState x:Name="MobileDevice">

<VisualState.StateTriggers>

<common:DeviceFamilyAdaptiveTrigger DeviceFamily="Mobile"/>

</VisualState.StateTriggers>

<VisualState.Setters>

<Setter Target="SubCategoryPivot.Margin" Value="0,0,0,0"/>

</VisualState.Setters>

</VisualState>

<VisualState x:Name="DesktopDevice"/>

</VisualStateGroup>

</VisualStateManager.VisualStateGroups>

1. Like the other pages of the application, the user interface is split into two rows:

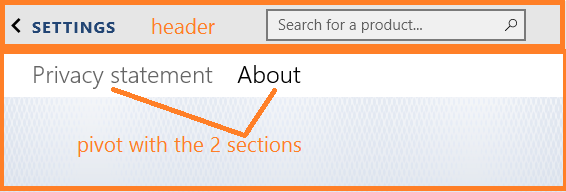
<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="\*"/>

</Grid.RowDefinitions>

1. The top row displays the header and the bottom row shows the pivot that provides different choices.



1. In the default row 0, First, define the HeaderControl.

<ctl:HeaderControl>

<ctl:HeaderControl.HeaderContent>

<TextBlock Text="SETTINGS" Style="{ThemeResource PageTitleTextBlockStyle}"

Foreground="{StaticResource HeroBackgroundThemeBrush}" />

</ctl:HeaderControl.HeaderContent>

</ctl:HeaderControl>

1. Next, add a white Rectangle below to make the list of sections more readable.

<Rectangle Fill="{StaticResource SolidWhiteBrush}" Height="48"

VerticalAlignment="Top" Margin="0,0,0,0" Grid.Row="1"/>

1. Finally, in row 1, define the Pivot.

<Pivot x:Name="SubCategoryPivot" Grid.Row="1"

Foreground="{StaticResource SolidBlackBrush}"

Margin="20,0,0,0" FontFamily="Segoe UI Light">

Each section (Privacy Statement and About) is defined by a PivotItem element which the Header attribute sets its title.

1. The first section provides the privacy policy. Instead of building an XAML-based user interface, the existing web pages such as the one built in Task 1 can easily be displayed by a WebView control.

<PivotItem Header="Privacy statement">

<WebView Source="http://w10azfs-pc-landingpage.azurewebsites.net/index.html#privacy\_policy" />

</PivotItem>

Note the #privacy\_policy anchor to jump to the privacy policy section of the page.

1. The last About section displays the application name, some version information.

<PivotItem Header="About">

<StackPanel Margin="0,20,0,0">

<StackPanel>

<TextBlock Text="Product Catalog" Style="{StaticResource TitleTextBlockStyle}"/>

<TextBlock x:Name="tbVersion" Style="{StaticResource BodyTextBlockStyle}" />

</StackPanel>

1. Next, add a link to send emails as a feedback mechanism. Read <https://msdn.microsoft.com/en-us/library/aa767737(v=vs.85).aspx> for more details about the mailto protocol.

<StackPanel Margin="0,20">

<TextBlock Text="Contact us" Style="{StaticResource TitleTextBlockStyle}"/>

<HyperlinkButton FontFamily="Segoe UI" FontWeight="SemiLight" FontSize="18"

Content="Feedbacks" Margin="20,-6,0,0"

NavigateUri="mailto:ProductCatalogSupport@Contoso.com?subject=Feedbacks about Product Catalog Application"

/>

1. Last, add a link to point to the blog site as About.

<HyperlinkButton FontFamily="Segoe UI" FontWeight="SemiLight" FontSize="18"

Content="About" Margin="20,-8,0,0"

NavigateUri="http://w10azfs-pc-blog.azurewebsites.net/"

/>

</StackPanel>

</StackPanel>

</PivotItem>

</Pivot>

1. Open the SettingsPage.xaml.cs file and in the constructor, write the code to set the version number to the tbVersion Text property.

public SettingsPage()

{

this.InitializeComponent();

Version v = GetAppAssemblyVersion();

tbVersion.Text =

string.Format(

"Version {0}.{1}.{2}.{3}",

v.Major, v.Minor, v.Build, v.Revision

);

}

1. Write the GetAppAssemblyVersion helper method to extract the version of the application.

private Version GetAppAssemblyVersion()

{

Type t = typeof(App);

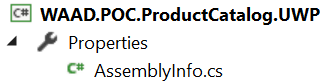
Assembly assembly = t.GetTypeInfo().Assembly;

Version version = assembly.GetName().Version;

return version;

}

Note that the version is set through attributes in the AssemblyInfo.cs file.

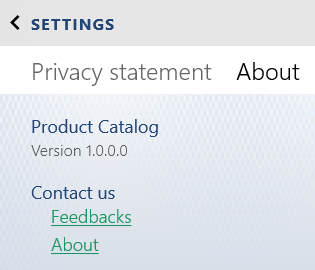


Only the AssemblyVersion attribute is taken into account by the .NET Framework and the Store:

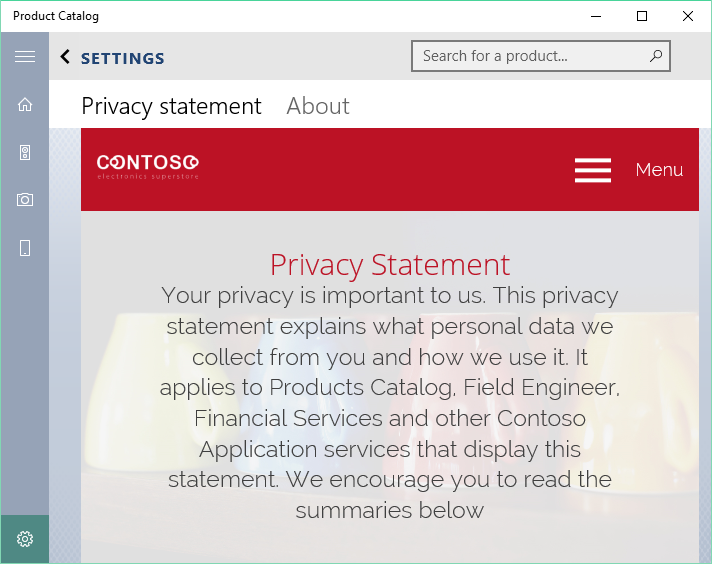


Each time you publish a new version of an application, you need to increment this value.

1. Run the application and the About settings should look like the following:



1. The Privacy statement page will look like the following:



# Lab 2: Contoso Products Admin Web Application

#### Introduction

This tutorial shows how to create an ASP.NET web application and deploy it to [App Service Web Apps](https://azure.microsoft.com/en-us/documentation/articles/app-service-web-overview/) by using Visual Studio 2015 or Visual Studio 2013. The tutorial assumes that you have no prior experience in using Microsoft Azure or ASP.NET. On completing the tutorial, you'll have a simple web application up and running in the cloud.

#### Objectives

After completing this lab, you will be able to leverage rapid application development and improve your application lifecycle management (ALM) through the following features:

* Quickly Create ASP.NET Web App in Azure.
* Auto-generate your basic back end code for your model.
* Save time by enabling continuous deployment.

#### Estimated Time to Complete This Lab

75 minutes

#### Scenario

In order to maximize the reach of your application, you could extend part of your solution as a web front. Indeed, you could define that your application on mobile devices would provide administrative tasks through a web application. An example is: the Centralize Model management (Add, Update, Remove, etc.)

## Exercise 1: Create an Admin Website

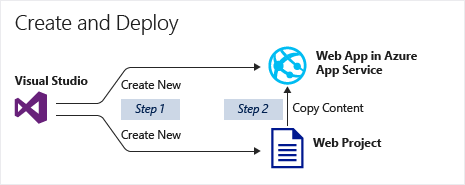
#### Objectives

In this exercise, you will learn how to:

* Create ASP.NET application.
* Deploy to Azure.
* Update and redeploy.

#### Scenario

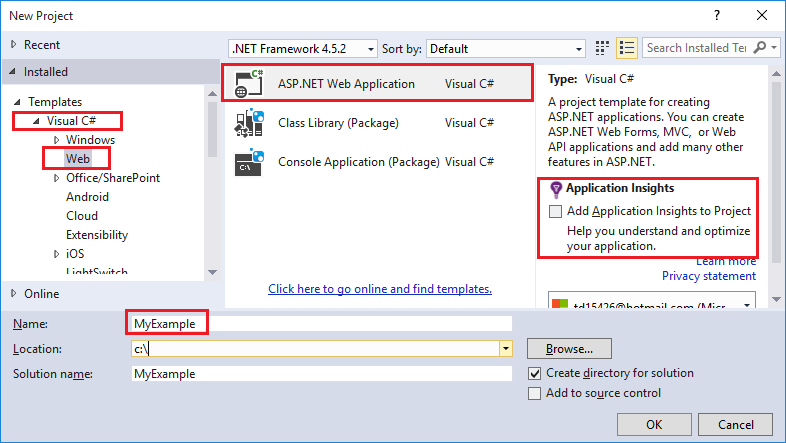
Your first step is to create a web project in Visual Studio and a Web App in Azure App Service. When that is done, you will deploy the project to the Web App to make it available on the Internet. The diagram illustrates the create and deploy steps.



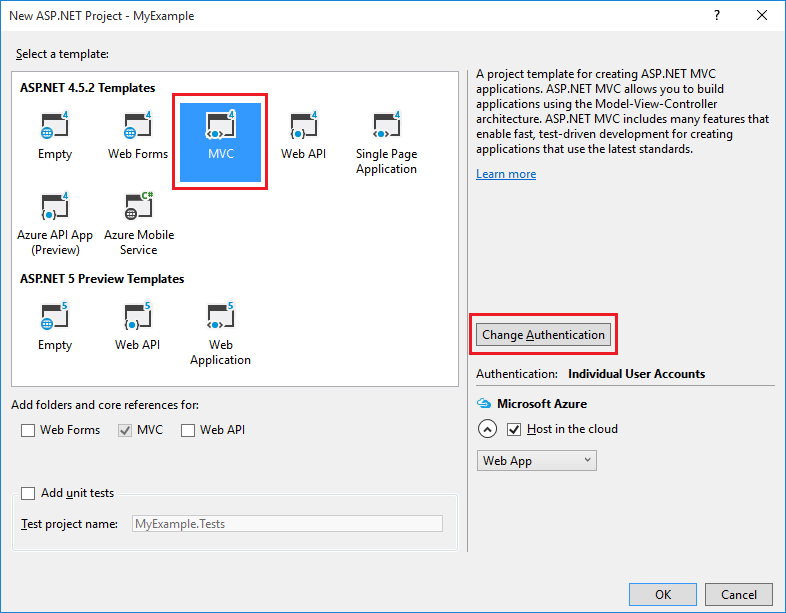
The application has been named **MyExample** in the guide. Feel free to customize and provide a unique name as it will be mandatory when creating a resource in Azure. It is recommended that you choose a more meaningful application name, for example, ProductsAdminWebsite.

### Task 1: Create a ASP.NET Web Application

1. Open Visual Studio 2015 and click **File** > **New** > **Project**.
2. In the New Project dialog box, in the left pane, expand **Installed** > **Templates** > **Visual** C# and then click **Web**. In the middle pane, click ASP.NET Web Application. If you prefer, you can choose Visual Basic.
3. In the **Name** box, type MyExample, and then click **OK**.

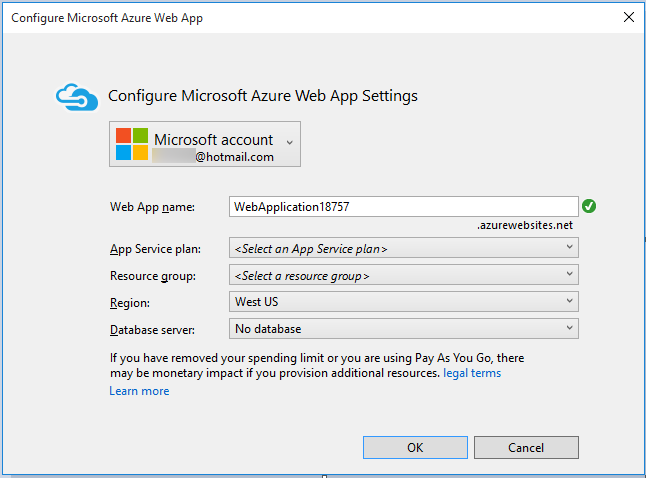


1. In the New ASP.NET Project dialog box, under **ASP.NET 4.5.2 Templates**, select MVC.



**Note**:[MVC](http://www.asp.net/mvc) is an ASP.NET framework for developing web applications. For more information, see: <http://www.asp.net/mvc>

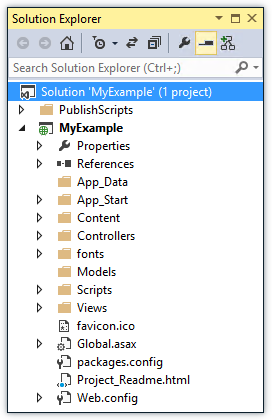
1. The Configure Microsoft Azure Web App dialog box asks you what resources you want to create. In the **Web App name** box, enter a Web App name that is unique in the azurewebsites.net domain. For example, you can name it **MyExample** with numbers to the right to make it unique, such as MyExample810. If a default web name has been created for you, it will be unique and you can use that.



**Note**: If someone else has already used the name you entered, you will see a red exclamation mark to the right instead of a green check mark, and you will need to enter a different Web App name .

Azure will use this name as the prefix for your application's URL. The complete URL will consist of this name plus.azurewebsites.net (as shown next to the **Web App name** box). For example, if the name is MyExample810, the URL will be MyExample810.azurewebsites.net. The URL has to be unique.

1. In the **Configure Microsoft Azure Web App** dialog box, in the App Service plan drop-down box, enter **MyExamplePlan** or another name if you prefer, for the plan name.
2. In the **Resource group** drop-down box, enter **MyExampleGroup**, or another name if you prefer, for the resource group name.
3. In the **Region** drop-down box, select the location that is closest to you.
4. Click **OK**. The allocation of the resources in the cloud and the creation of the process will begin. Once done, go to Solution Explorer, which shows the files and folders in the new project.



### Task 2: Add Scaffolding

We will now leverage an Entity Framework Code First and ASP.NET MVC scaffolding feature to auto-generate the code for the CRUD operation (Create, Read, Update, Delete).

**Note**: We will not go through all implementation details for those features, however, you can always refer to the following links for additional information:

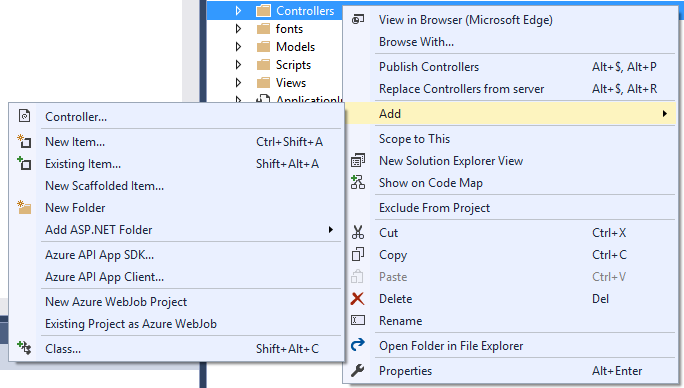
• [Create MVC controllers and views using scaffolding](http://www.asp.net/mvc/tutorials/mvc-5/introduction/getting-started)

• [How to write Entity Framework code that works with SQL Server databases](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc)

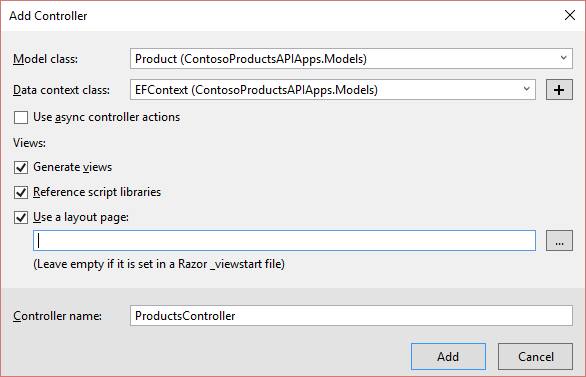
1. Copy content of ***\Labs\Module05-Web Apps\Lab2\Exercise1\Asset\Models*** to the project **Models** folder (two files DatataLayer.cs and DataModel.cs).



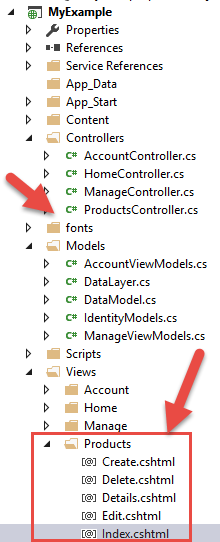
1. Build project.
2. Right-click the **Controllers** folder, and from the shortcut menu, select **Add**.



1. In the **Add** > **Controller** dialog box, **click Create MVC 5 Controller with views using Entity Framework**.
2. In the **Model** class drop-down list, select **Product** and in the **Data context class** drop-down list, select **EFContext**.



You should have a few file generated for you (controller and view).



1. In the **View\Products** folder, right-click **Index.cshtml** and click **Navigate**. The default browser should launch the page with the CRUD operations (Create, Read, Update, Delete).
2. You can repeat the process for all the classes in **DataModel.cs** (ProductSpecification, ProductCategory, ProductSubCategory).
3. Press F5 to run you application locally and test this newly added feature in your browser.

**Note**: Do not deploy yet, this will be done in Exercise 3.

## Exercise 2: Set up the Website Environment

#### Objectives

In this exercise, you will learn:

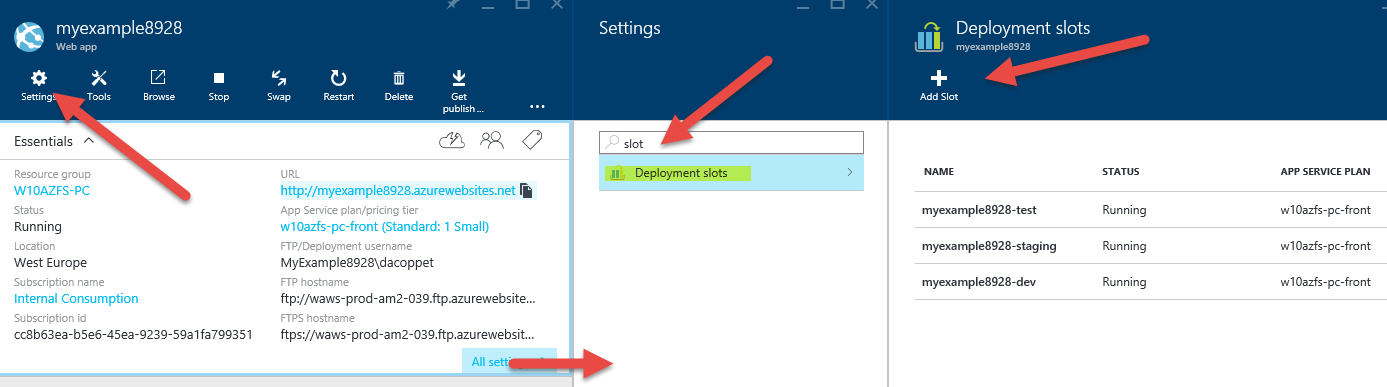
* How to create deployment slots for your Web App.
* How to configure settings for deployment slots.
* How to swap deployment slots.

#### Scenario

When you deploy your Web App to [App Service](http://go.microsoft.com/fwlink/?linkid=529714&clcid=0x409), you can deploy to a separate deployment slot instead of the default production slot when running in the Standard or Premium App Service plan mode. Deployment slots are actually live Web Apps with their own hostnames. Web App content and configurations elements can be swapped between two deployment slots, including the production slot.

### Task 1: Add a Deployment Slot to a Web App

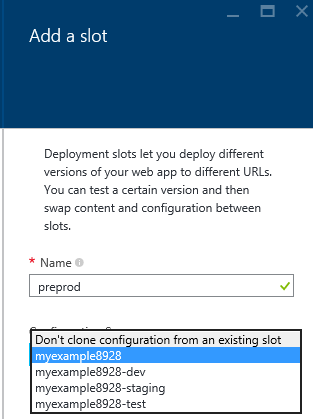
1. In the [Azure Preview portal](https://portal.azure.com/), open the Web App's blade.
2. In the **Settings** blade, in the search text box, type **slots** and then click **Deployment slots**.
3. In the **Deployment slots** blade, click **Add Slot**. We will create three slots matching the following environment: dev, test, and staging.



**Note**: The Web App must be running in the Standard or Premium mode in order for you to enable multiple deployment slots.

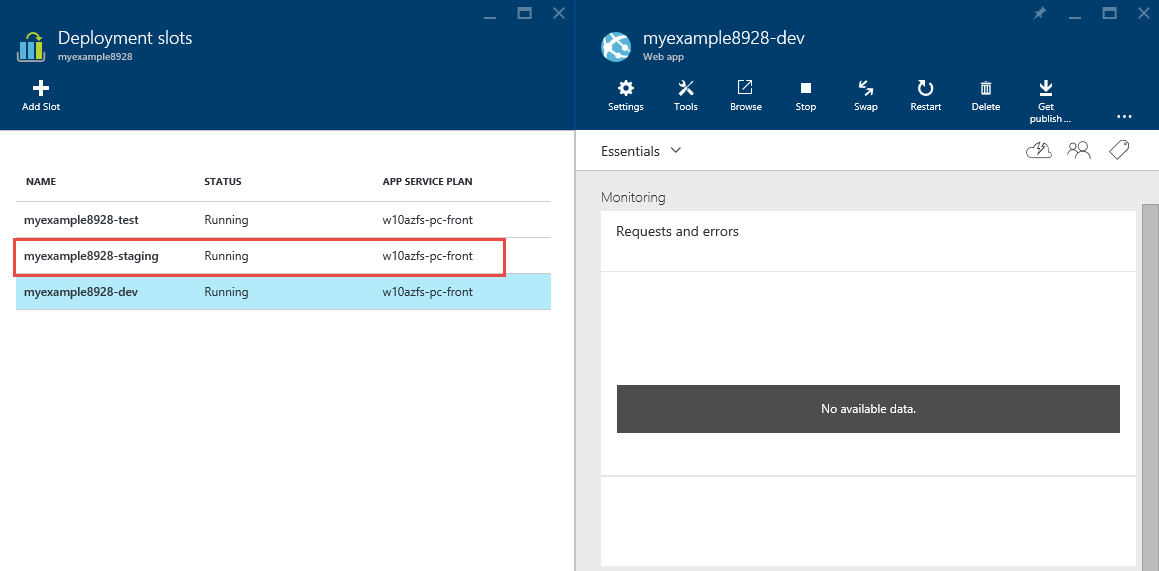
If the Web App is not already in the Standard or Premium mode, you will receive a message indicating the supported modes for enabling staged publishing. At this point, you have the option to select Upgrade and navigate to the Scale tab of your Web App before continuing.

1. In the **Add a slot** blade, in the Name box, enter a slot name, and select whether to clone Web App configuration from another existing deployment slot. Click the check mark to continue.



* The first time you add a slot, you will only have two choices: clone configuration from the default slot in production or not at all.
* After you have created several slots, you will be able to clone configuration from a slot other than the one in production.

1. In the **Deployment slots** blade, click the deployment slot to open a blade for the slot, with a set of metrics and configuration just like any other Web App. The **your-web-app-name-deployment-slot-name** will appear at the top of blade to remind you that you are viewing the deployment slot.



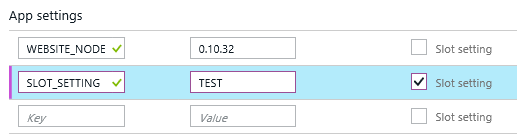
1. Click the app URL in the slot's blade. Notice the deployment slot has its own hostname and is also a live app.

**Note** There is no content after deployment slot creation. You can deploy to the slot from a different repository branch, or an altogether different repository. You can also change the slot's configuration. Use the publish profile or deployment credentials associated with the deployment slot for content updates. For example, you can publish to this slot with **git**. We will see an example with Visual Studio Online in the next exercise.

To limit the public access to the deployment slot, see [App Service Web App – block web access to non-production deployment slots](http://ruslany.net/2014/04/azure-web-sites-block-web-access-to-non-production-deployment-slots/).

### Task 2: Configuration for Deployment Slots

1. When you clone configuration from another deployment slot, the cloned configuration is editable. Furthermore, some configuration elements will follow the content across a swap (not slot specific) while other configuration elements will stay in the same slot after a swap (slot-specific).
2. To configure an app setting or connection string to stick to a slot (not swapped), access the **Application Settings** blade for a specific slot, then select the **Slot Setting** box for the configuration elements that should stick to the slot. Note that marking a configuration element as slot-specific has the effect of establishing that element as "not swappable" across all the deployment slots associated with the Web App.
3. Add to each slot the following setting "SLOT\_SETTING" with the following values: PROD, DEV, TEST, and STAGING.
4. This should look like the picture below:



**Note**: The following list shows the configuration that will change when you swap slots.

**Settings that are swapped:**

• General settings, such as framework version, 32- or 64-bit, web sockets

• App settings (can be configured to stick to a slot)

• Connection strings (can be configured to stick to a slot)

• Handler mappings

• Monitoring and diagnostic settings

• WebJobs content

**Settings that are not swapped:**

• Publishing endpoints

• Custom domain names

• Secure Sockets Layer (SSL) certificates and bindings

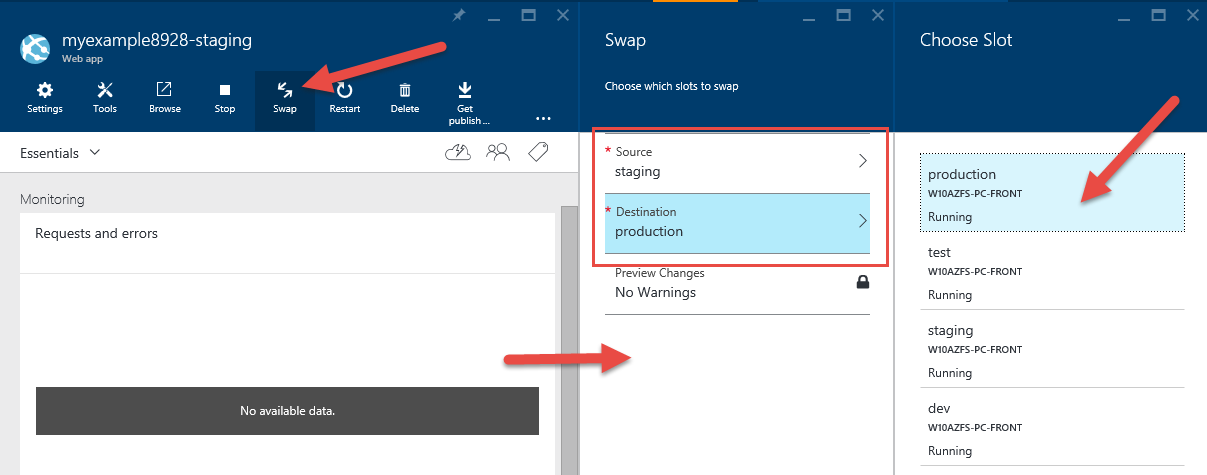
• Scale settings

• WebJobs schedulers



### Task 3: Swap Deployment Slots

1. To swap deployment slots, click the **Swap** button in the command bar of the Web App or in the command bar of a deployment slot. Make sure that the swap source and swap target are set properly. Usually, the swap target would be the production slot.



1. To complete the swap operation, click **OK**. When the operation finishes, the deployment slots have been swapped.

**Important**: Before you swap a Web App from a deployment slot into production, make sure that all non-slot specific settings are configured exactly as you want to have it in the swap target.

## Exercise 3: Enabling Continuous Integration and Continuous Deployment

#### Objectives

In this exercise, you will learn:

* How to extend classic continuous integration process up to the deployment phase.
* Connect Visual Studio Online and Azure Web Apps from the new preview portal.

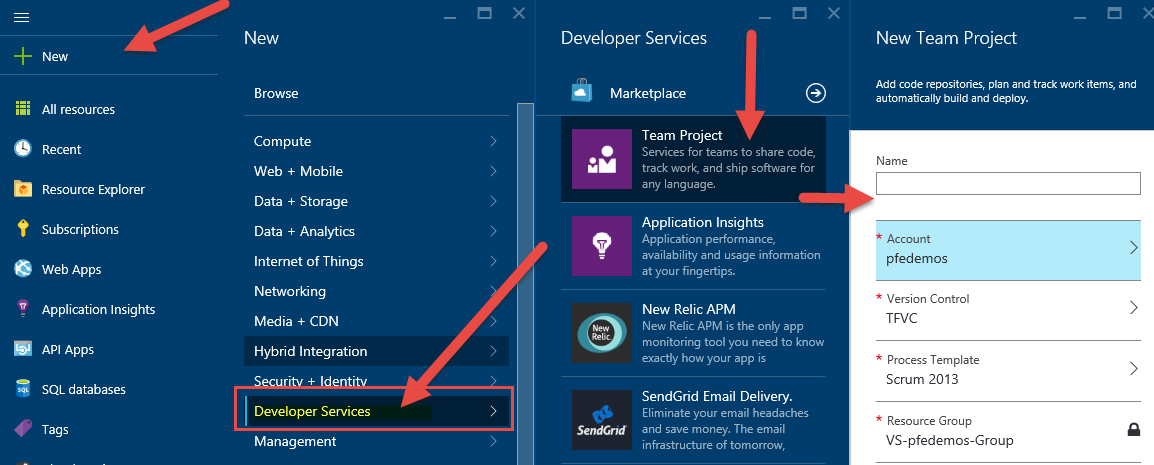
#### Scenario

Because time is money and reducing the time to release is a critical factor in modern application development driven by agility, companies always seek new ways of improving and optimizing the whole delivery cycle of an application. Web Apps align with modern development practices and needs. It offers a functionality called continuous deployment, which enable you to extend the application life cycle from coding, compiling, testing to delivery while conforming to the new DevOps trend.

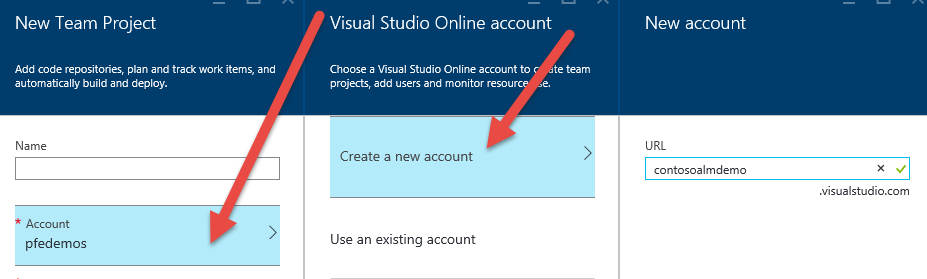
### Task 1: Create Team Project

You need to have a Visual Studio Online account for the exercise. The easiest way to create one is to use the new Azure portal integration with Visual Studio Online.

1. In the **General** menu, click the **New** button.
2. Go to **Developer Services**.
3. Click **Team Project**.



1. If you already have an account, just create a new team project. If not, follow these steps:
2. Go to the account.
3. Create a new account.
4. Enter a name.

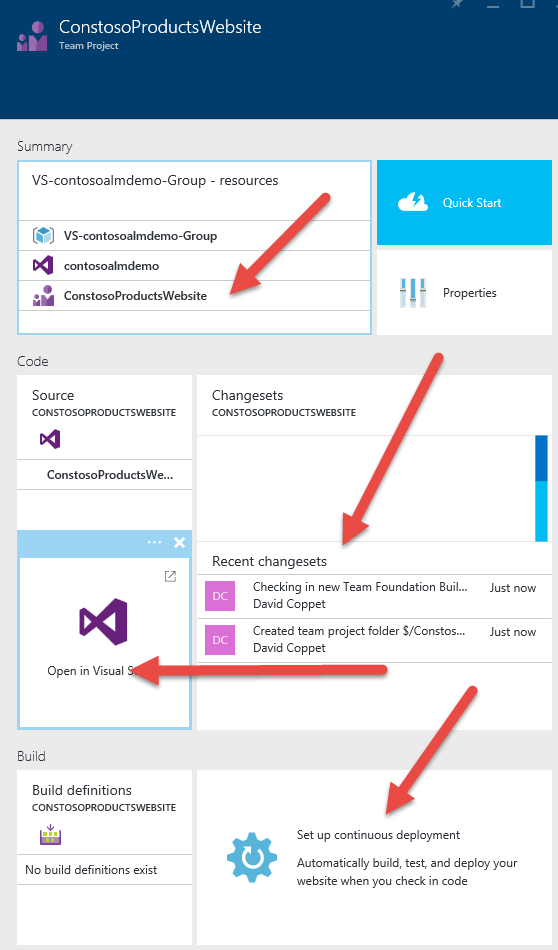


1. In the **Name** box, enter a name for the new project, for example, **ContosoProductsWebsite**.

**Note:** Remember to adjust settings like subscription (if you have several) and the location.



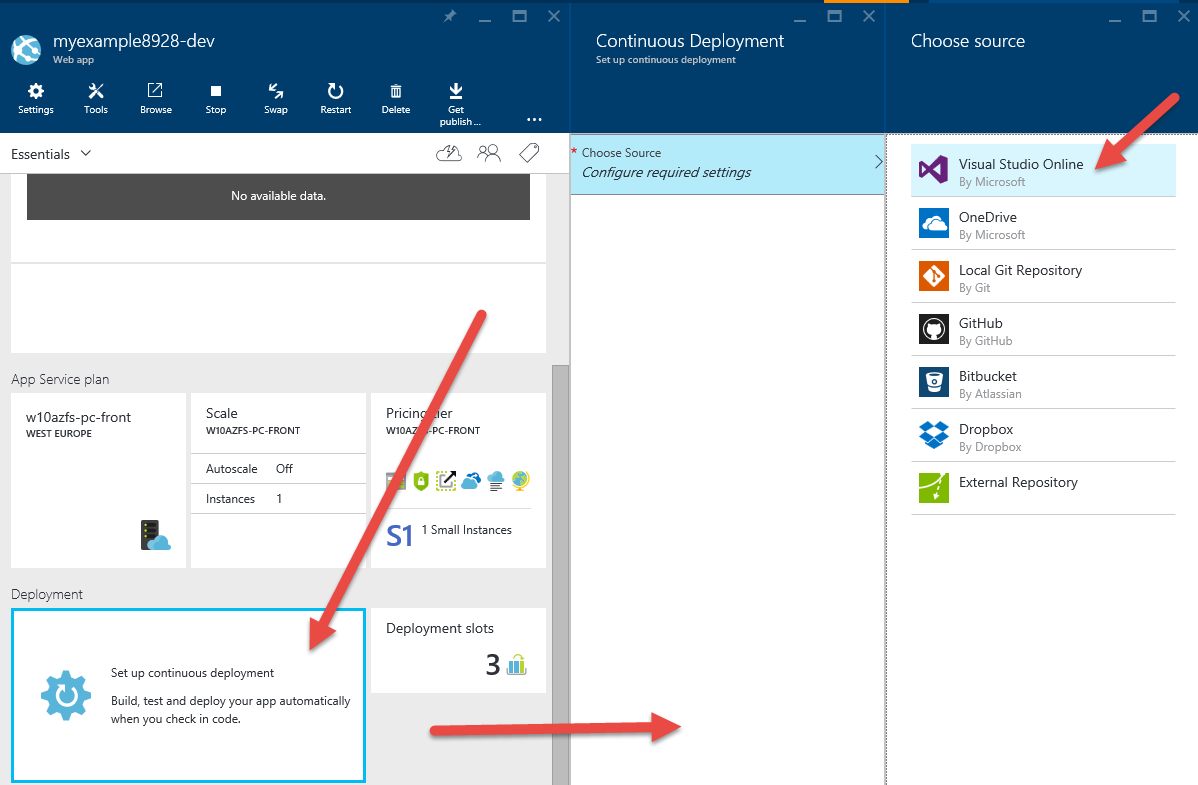
1. After a while, you will have your dedicated team project created in the portal.



**Note**: We could set up continuous deployment from there, but let's set up this feature directly from a specific slot.

### Task 2: Set Up Continuous Deployment

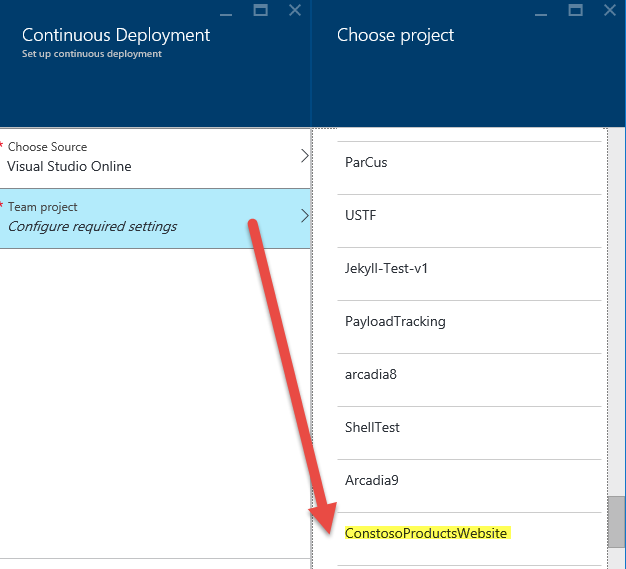
1. Go back to your Web App's dev slot.
2. Click **Setup continuous deployment**.
3. Select **Visual Studio Online**.



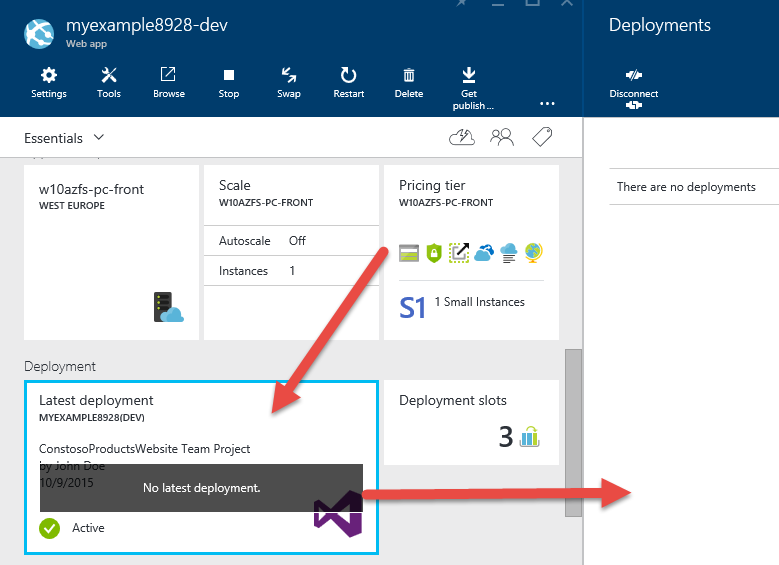
1. Select your **Team project**.
2. Browse the list of result and select your newly created team project, **ConsosoProductsWebsie**, in this case.

**Note**: It should appear at the bottom of the list if just created and never used or is at the top if recent.

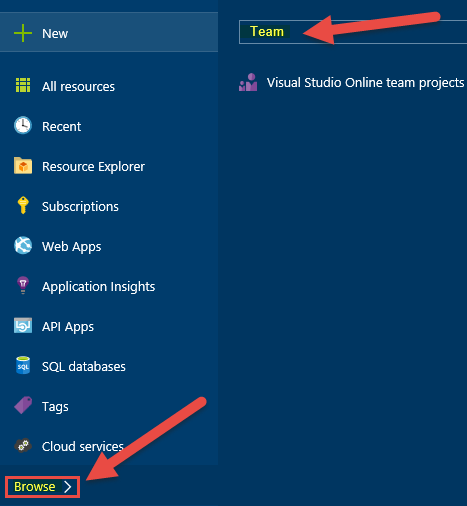
1. To finalize the process, click **OK**.



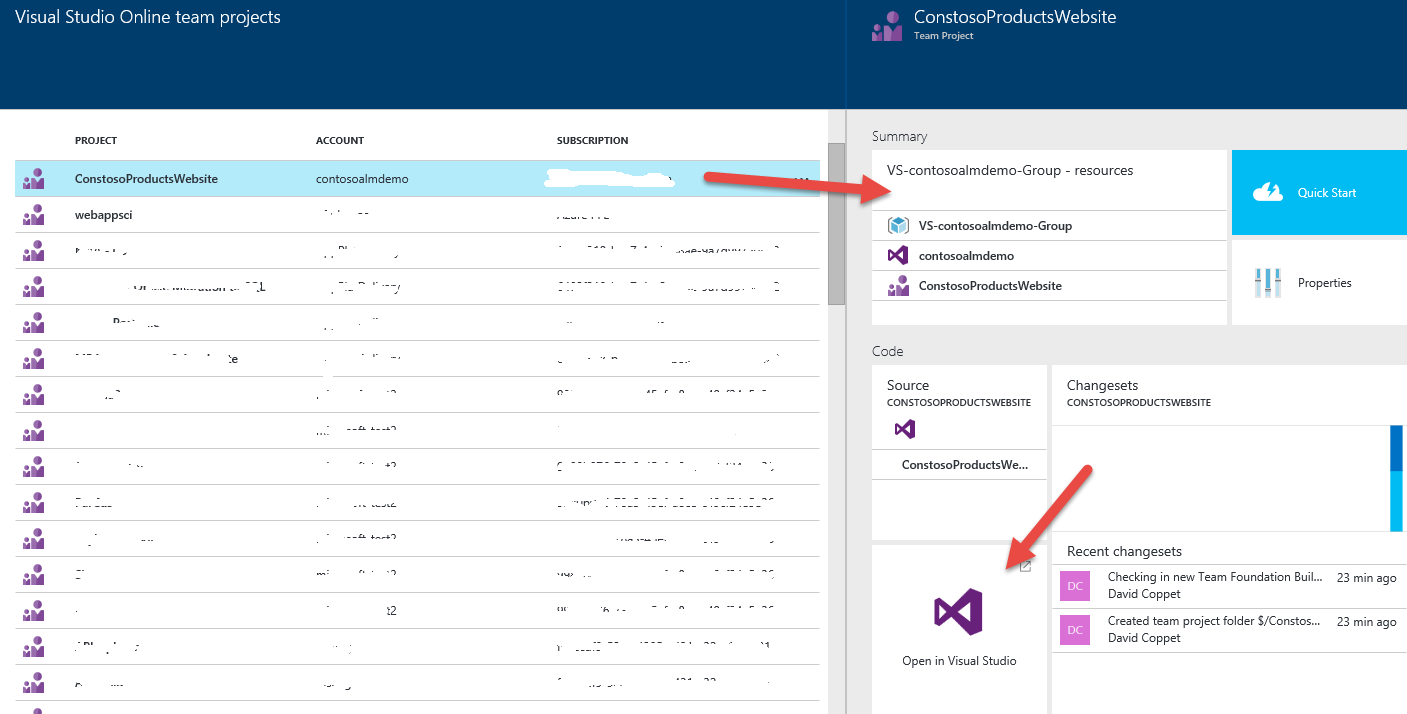
1. After few seconds, you should see the following state:



1. Go back to your **Team Project**.



1. This will open a blade with your team project instance.



1. Click **Open in Visual Studio**. It will open a URL and launch your favorite IDE.

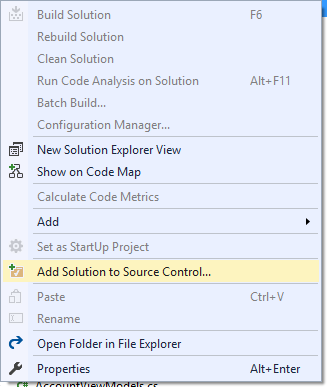
**Note**: You will probably be asked to log in order to access the VSO account. Enter the same Microsoft account you’ve used to create it. You should then see the “Team Explorer” window with the team project's information.

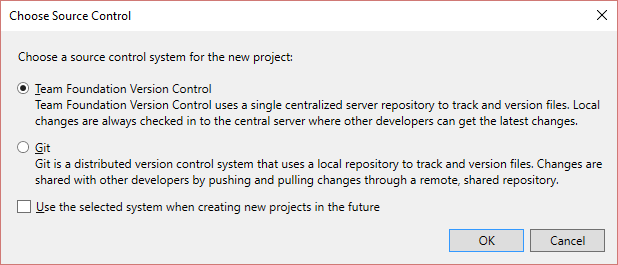
### Task 3: Check In a Project to Source Control

1. In Visual Studio, open the solution you want to deploy, or create a new one. You can deploy a Web App or a cloud service (Azure application) by following the steps in this walkthrough. If you want to create a new solution, create a new Azure Cloud Service project, or a new ASP.NET MVC project. Make sure that the project targets .NET Framework 4 or 4.5, and if you are creating a cloud service project, add a ASP.NET MVC web role and a worker role, and select Internet Application for the web role. When prompted, select **Internet Application**. If you want to create a Web App, choose the **ASP.NET Web Application** project template, and then choose **MVC**.

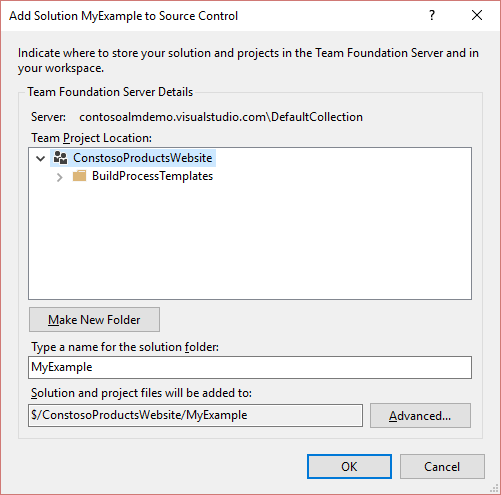
**Note**: Visual Studio Online only supports CI deployments of Visual Studio Web Applications at this time. Website projects are out of scope.

1. Right-click the solution, and in the shortcut menu, select **Add Solution to Source Control**.

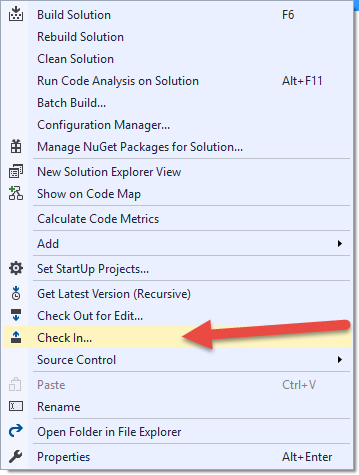




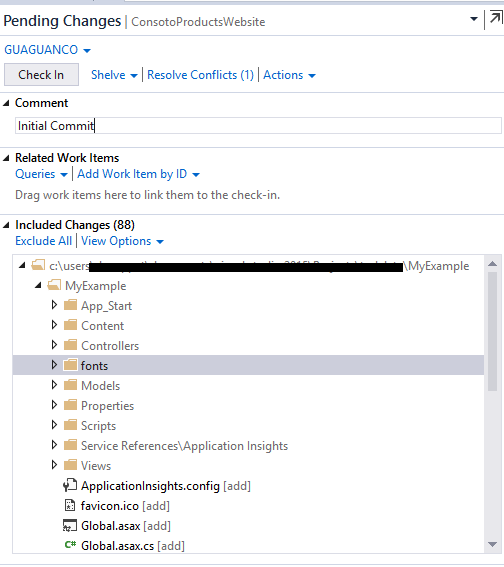
1. Accept or change the defaults and click **OK**. Once the process completes, the source control icons appears in Solution Explorer.



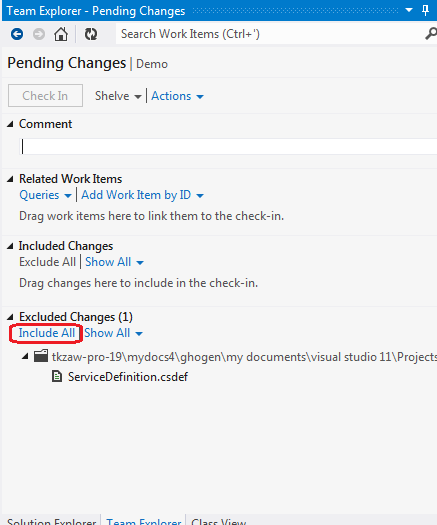
1. Open the shortcut menu for the solution, and select **Check In**.



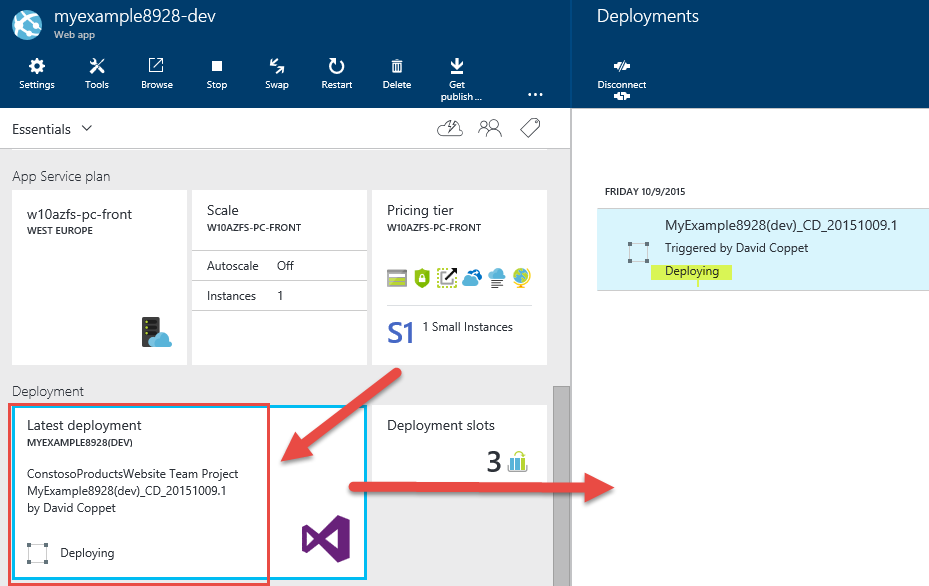
1. In the **Pending Changes** section of Team Explorer, type a comment for the check-in and click the **Check In** button.



**Note**: Check the options to include or exclude specific changes when you check in. If desired changes are excluded, click the **Include All** link.

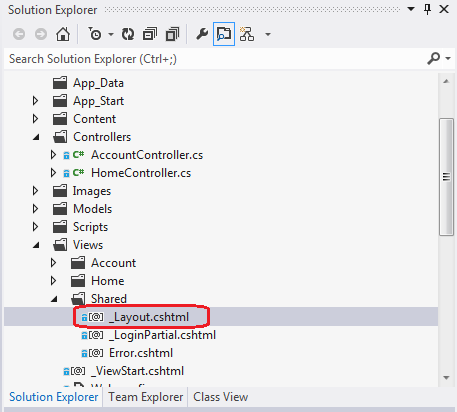


1. If you go back to your dev slot in the Azure portal, it will look like the screenshot given below:

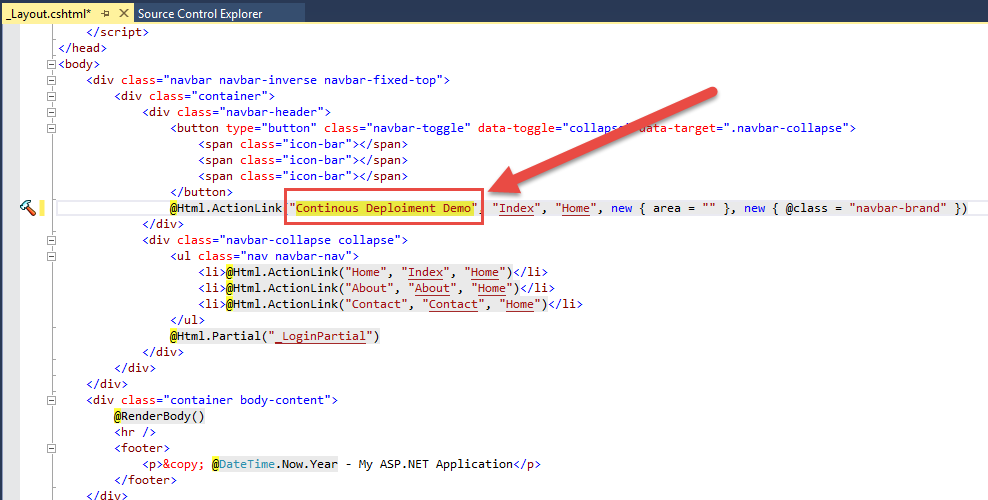


### Task 4: Trigger a Rebuild and Redeploy Your Project (Optional)

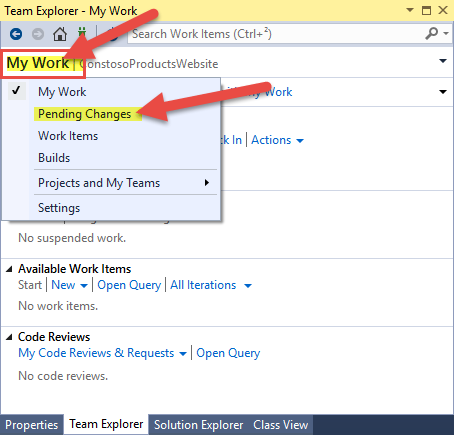
1. In Solution Explorer, open up a file and change it. For example, in the **Views\Shared** folder in an MVC web role, change the file **\_Layout.cshtml**.



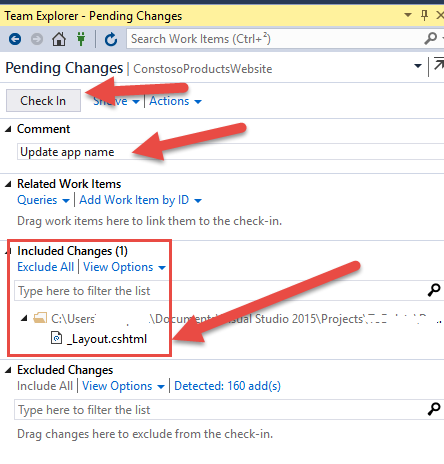
1. Edit the logo for the site and press Ctrl+S to save the file.



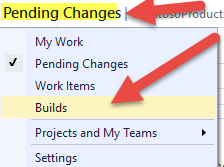
1. In **Team Explorer**, click the **Pending Changes** link.



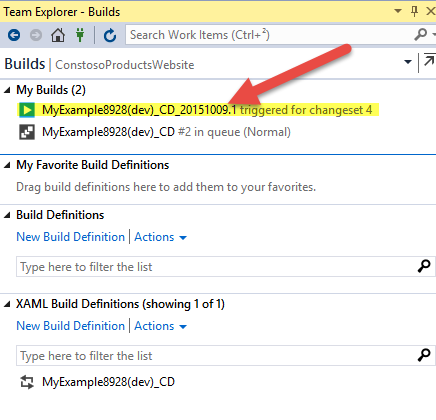
1. Type a comment and click the **Check In** button.



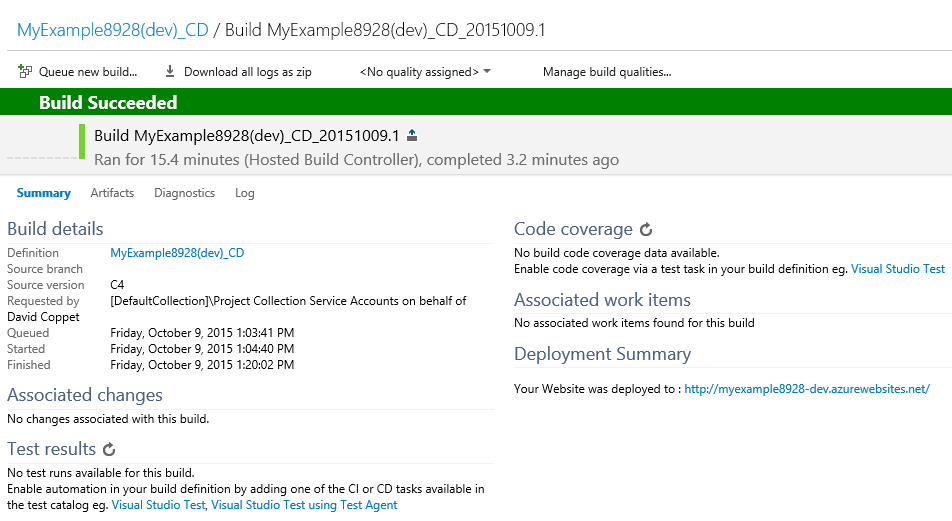
1. In **Team Explorer**, click **Pending Changes** and in the drop-down list, click **Builds**.



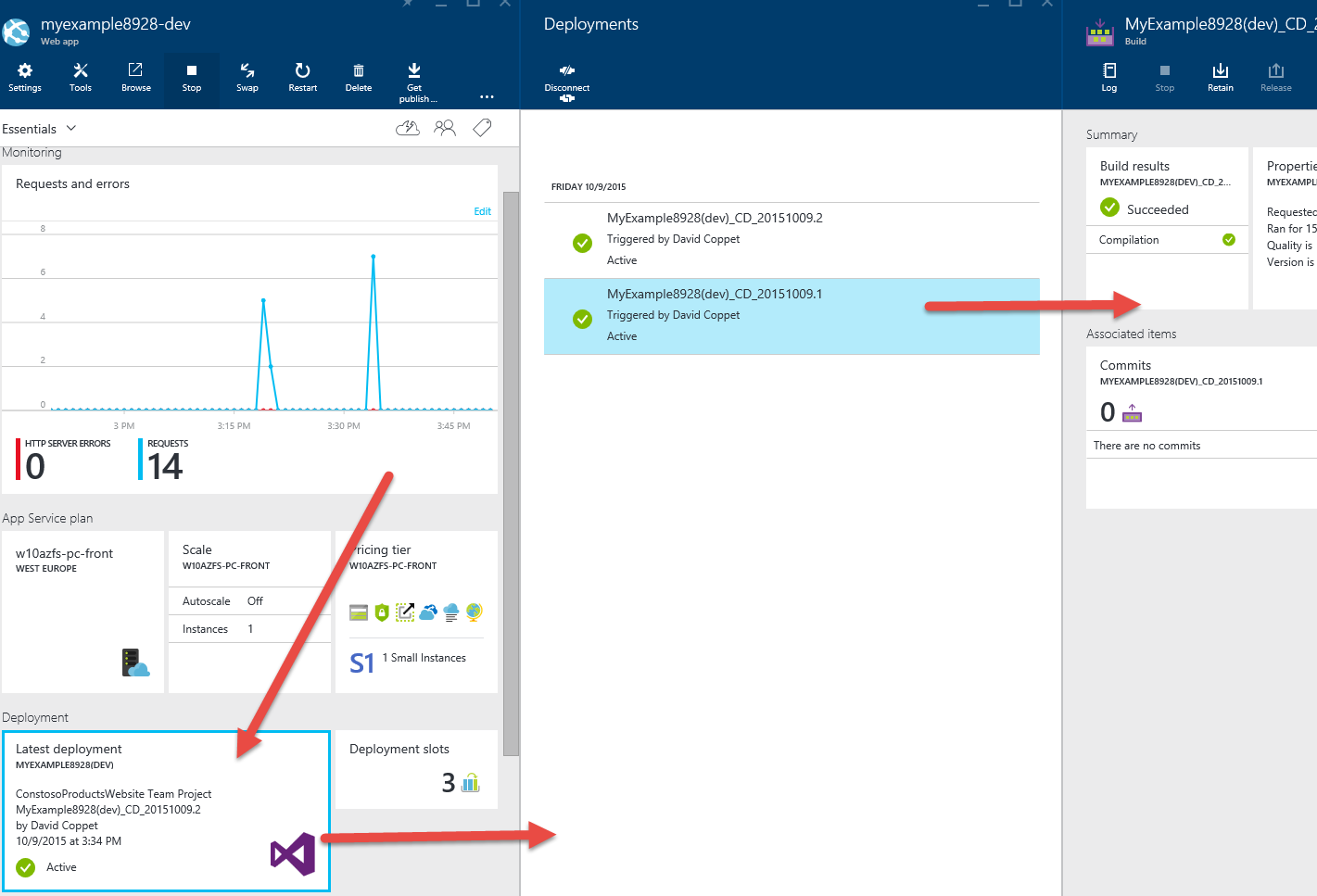
1. Click the builds link to view the builds in progress. The Team Explorer shows that a build has been triggered for your check-in.



1. Type a comment and click the **Check In** button.
2. To return to the Team Explorer home page, click the **Home** button.
3. Click the **Builds** link to view the builds in progress. The Team Explorer shows that a build has been triggered for your check-in.
4. Double-click the name of the build in progress to view a detailed log as the build progresses.



1. In the [Azure portal](http://manage.windowsazure.com/), on the **Deployments** tab, view the associated deployment when the staging environment is selected.



# Lab 3: Azure Web Jobs

#### Introduction

In this lab, you will create a web application that expose a front end that will serve pages to your users and will also run a background job on that front end server. This lab will demonstrate a classic scenario where users upload a picture and a process asynchronously generates a thumbnail. The goal is to show you how you can create online resources to support your mobile applications, and how easy it is to integrate other Azure service in order to complete your solution.

#### Objectives

After completing this lab, you will be able to:

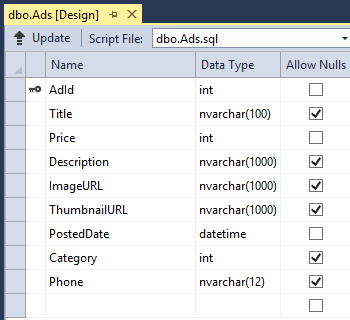
* Enable your machine for Azure development by installing the Azure software development kit (SDK).
* Create a Console Application project that automatically deploys as an Azure WebJob when you deploy the associated web project.
* Test a WebJobs SDK back end locally on the development computer.
* Publish an application with a WebJobs back end to a Web App in App Service.
* Upload files and store them in the Azure Blob service.
* Use the Azure WebJobs SDK to work with Azure Storage queues and blobs.

#### Estimated time to complete this lab

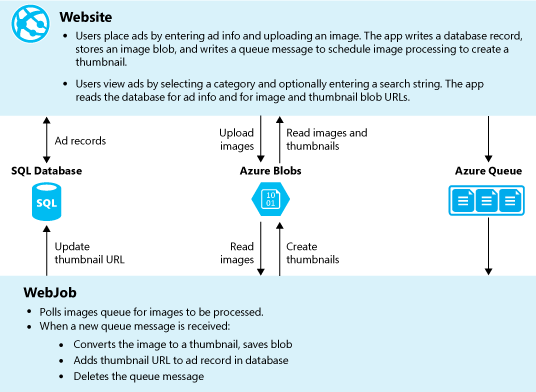
45 minutes

#### Scenario

The sample application uses the [queue-centric work pattern](http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/queue-centric-work-pattern) to off-load the CPU-intensive work of creating thumbnails to a back end process. The app stores ads in a Microsoft SQL Server database, using Entity Framework Code First to create the tables and access the data. For each ad, the database stores two URLs: one for the full-size image and one for the thumbnail.



When a user uploads an image, the front end of the Web App stores the image in an [Azure blob](http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/unstructured-blob-storage), and it stores the ad information in the database with a URL that points to the blob. At the same time, it writes a message to an Azure queue. A back end process running as an Azure WebJob uses the WebJobs SDK to poll the queue for new messages. When a new message appears, the WebJob creates a thumbnail for that image and updates the thumbnail URL database field for that ad. Here is a diagram that shows how the parts of the application interact.



## Exercise 1: Set up and deploy

#### Objectives

In this exercise, we will recreate the necessary Azure resources to execute the solution. Also, we will perform the following steps:

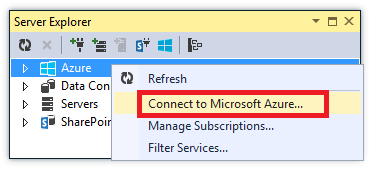
* Deploy a Web app and run locally.
* Deploy a Web app and run in the cloud.

### Task 1: Create Azure Storage Account (Optional)

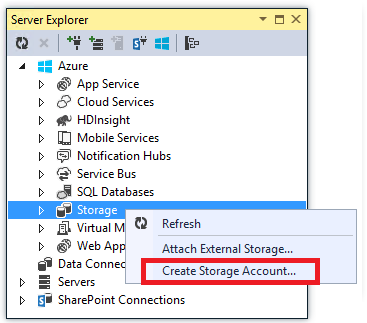
An Azure storage account provides resources for storing queue and blob data in the cloud. It is also used by the WebJobs SDK to store logging data for the dashboard.

In a real-world application, you typically create separate accounts for application data versus logging data, and separate accounts for test data versus production data. For this tutorial, you'll use just one account.

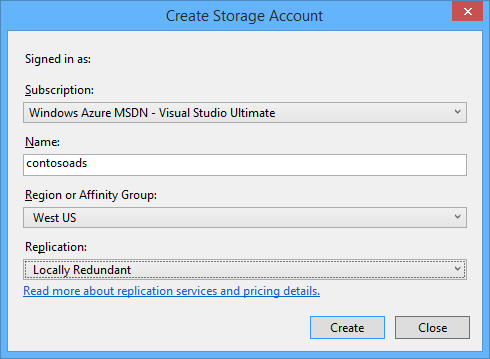
1. In Visual Studio, open the **Server Explorer** window.
2. Right-click the **Azure** node, and then select **Connect to Microsoft Azure**.



1. Sign in using your Azure credentials.
2. Under the **Azure** node, right-click **Storage**, and then select **Create Storage Account**.



1. In the **Create Storage Account** dialog box, in the **Name** box, enter a name for the storage account. The name must be must be unique (no other Azure storage accounts can have the same name). If the name you enter is already in use, then you will get a chance to change it. The URL to access your storage account will be {name}.core.windows.net.
2. In the **Region or Affinity Group** drop-down list, select the region closest to you. This setting specifies which Azure datacenter will host your storage account. For this tutorial, your choice won't make a noticeable difference. However, for a production Web App, you want your web server and your storage account to be in the same region to minimize latency and data egress charges. The Web App (which you'll create later) datacenter should be as close as possible to the browsers accessing the Web App in order to minimize latency.
3. In the **Replication** drop-down list, select **Locally Redundant**. When geo-replication is enabled for a storage account, the stored content is replicated to a secondary datacenter to enable failover to that location in case of a major disaster in the primary location. Geo-replication can incur additional costs. For test and development accounts, you generally don't want to pay for geo-replication. For more information, see [Create, manage, or delete a storage account](https://azure.microsoft.com/en-us/documentation/articles/storage-create-storage-account/#replication-options).
4. Click **Create**.



### Task 2: Download the application

1. Copy and unzip the (Labs\Module05-Web Apps\Lab3\Start).
2. Start Visual Studio. Click **File** > **Open** > **Project/Solution**, navigate to where you downloaded the solution, and then open the solution file.
3. To build the solution, press Ctrl+Shift+B.
4. By default, Visual Studio automatically restores the NuGet package content, which was not included in the .zip file. If the packages don't restore, install them manually by going to the **Manage NuGet Packages for Solution** dialog box and clicking the **Restore** button at the top right.
5. In Solution Explorer, make sure that **ContosoAdsWeb** is selected as the startup project.

### Task 3: Configure the Application to Use Your Storage Account

1. In the **ContosoAdsWeb** project, open the application **Web.config** file.

The file contains a SQL Server connection string and an Azure storage connection string for working with blobs and queues.

1. The SQL Server connection string points to a [SQL Server Express LocalDB](http://msdn.microsoft.com/library/hh510202.aspx) database. The storage connection string is an example that has placeholders for the storage account name and access key. Replace this with a connection string that has the name and key of your storage account.

<connectionStrings>

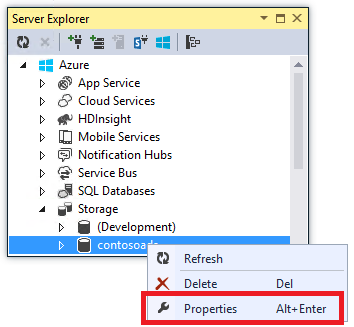
<add name="ContosoAdsContext" connectionString="Data Source=(localdb)\v11.0; Initial Catalog=ContosoAds; Integrated Security=True; MultipleActiveResultSets=True;" providerName="System.Data.SqlClient" />

<add name="AzureWebJobsStorage" connectionString="DefaultEndpointsProtocol=https;AccountName=[accountname];AccountKey=[accesskey]"/>

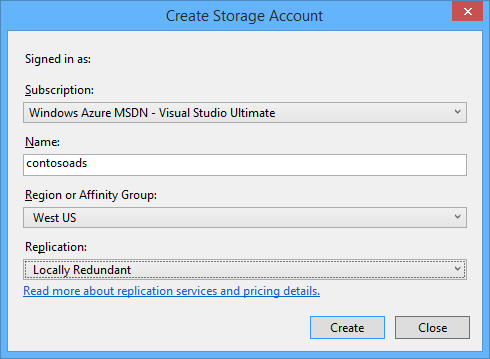
</connectionStrings>

The storage connection string is named AzureWebJobsStorage because that's the name the WebJobs SDK uses by default. This name is used here so you have to set only one connection string value in the Azure environment.

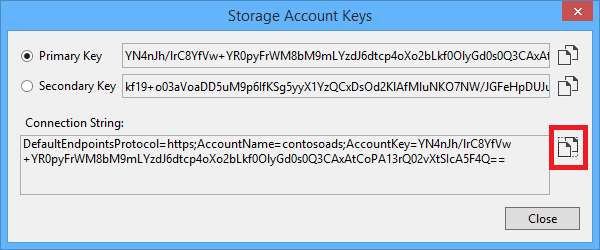
1. In **Server Explorer**, under the **Storage** node, right-click your storage account, and then click **Properties**.



1. In the **Properties** window, click **Storage Account Keys**, and then click the ellipsis.



1. Copy the **Connection String**.



1. Replace the storage connection string in the Web.config file with the connection string you just copied. Make sure you select everything inside the quotation marks but excluding the quotation marks before pasting.
2. Open the App.config file in the ContosoAdsWebJob project.

This file has two storage connection strings: one for application data and one for logging. For this tutorial, you will use the same account for both. The connection strings have placeholders for the storage account keys.

<configuration>

<connectionStrings>

<add name="AzureWebJobsDashboard" connectionString="DefaultEndpointsProtocol=https;AccountName=[accountname];AccountKey=[accesskey]"/>

<add name="AzureWebJobsStorage" connectionString="DefaultEndpointsProtocol=https;AccountName=[accountname];AccountKey=[accesskey]"/>

<add name="ContosoAdsContext" connectionString="Data Source=(localdb)\v11.0; Initial Catalog=ContosoAds; Integrated Security=True; MultipleActiveResultSets=True;"/>

</connectionStrings>

<startup>

<supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />

</startup>

</configuration>

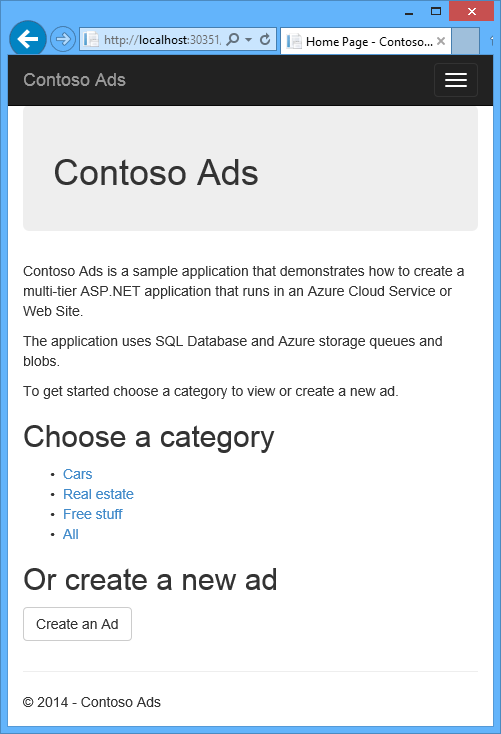
**Note**: By default, the WebJobs SDK looks for connection strings named AzureWebJobsStorage and AzureWebJobsDashboard. As an alternative, you can [store the connection string however you want and pass it in explicitly to the JobHost object](https://azure.microsoft.com/en-us/documentation/articles/websites-dotnet-webjobs-sdk-storage-queues-how-to/#config).

1. Replace both storage connection strings with the connection string you copied earlier.
2. Save your changes.

### Task 4: Run the application locally

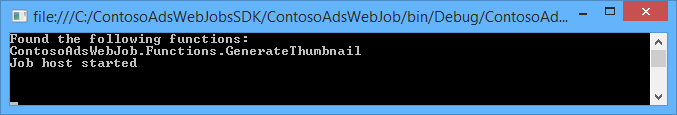
1. To start the web frontend of the application, press Ctrl+F5.

The default browser opens to the home page. (The web project runs because you've made it the startup project.)

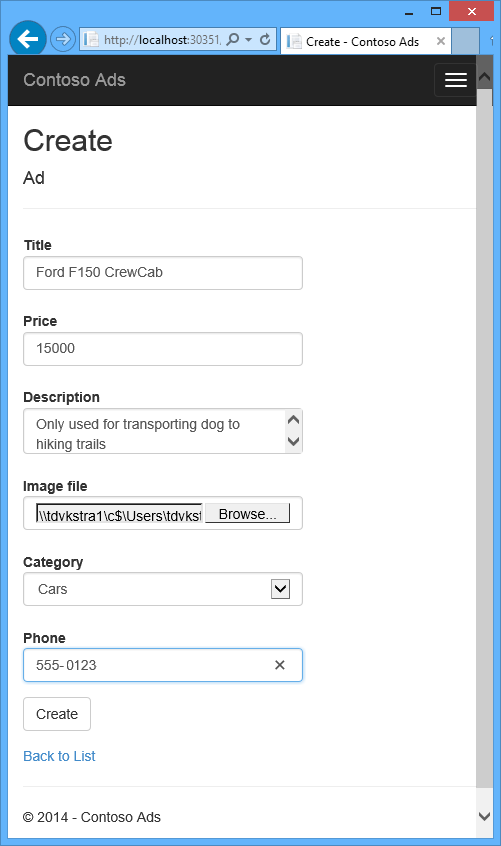


1. To start the WebJob back end of the application, in Solution Explorer, right-click the **ContosoAdsWebJob** project, and then select **Debug** > **Start new instance**.

A console application window opens and displays logging messages indicating the WebJobs SDK JobHost object has started to run.

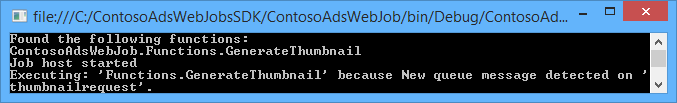


1. In your browser, click **Create an Ad**.
2. Enter some test data and select an image to upload, and then click **Create**.

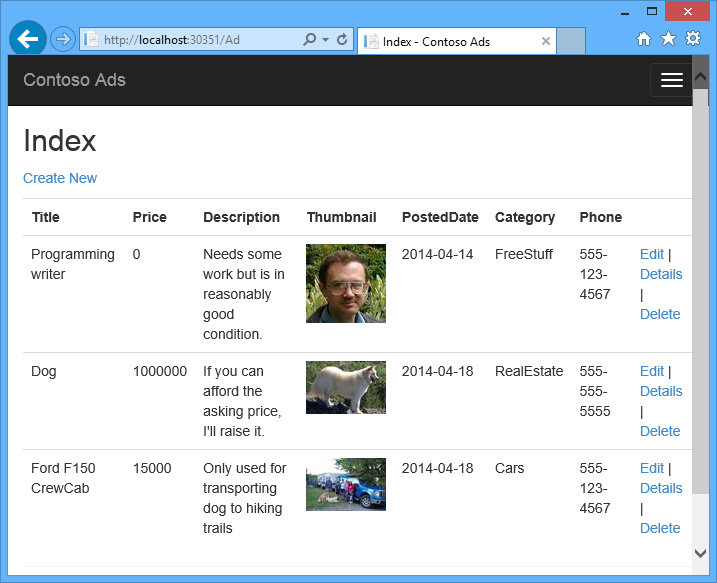


The app goes to the Index page, but it does not show a thumbnail for the new ad because that processing has not happened yet.

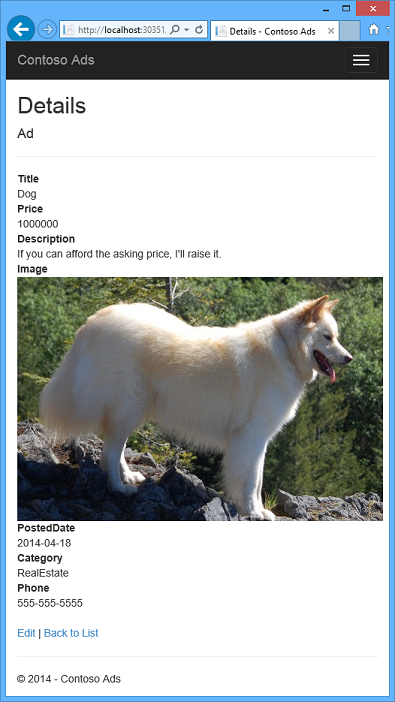
After a short wait, a logging message in the console application window shows that a queue message was received and has been processed.



1. After you see the logging messages in the console application window, refresh the Index page to see the thumbnail.



1. Click **Details** for your ad to see the full-size image.



**Note**: You've been running the application on your local computer, and it's using a SQL Server database located on your computer, but it is working with queues and blobs in the cloud. In the following section, you'll run the application in the cloud, using a cloud database as well as cloud blobs and queues.

### Task 5: Run the Application in the Cloud

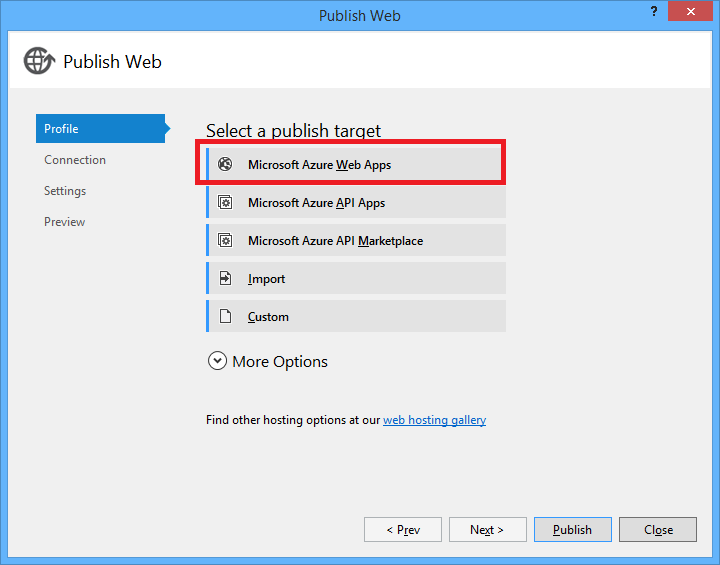
You'll do the following steps to run the application in the cloud:

* Deploy to Web Apps. Visual Studio will automatically create a new Web App in App Service and SQL Server database instance.
* Configure the Web App to use your Azure SQL Server database and storage account.

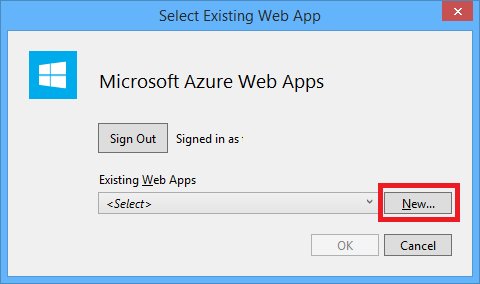
After you've created some ads while running in the cloud, you'll view the WebJobs SDK dashboard to see the rich monitoring features it has to offer.

**Deploy to Web Apps**

1. Close the browser and the console application window.
2. In Solution Explorer, right-click the **ContosoAdsWeb** project, and then click **Publish**.
3. In the **Publish Web** wizard, on the **Profile** page, click **Microsoft Azure Web Apps**.



1. In the **Select existing Web App** box, click **Sign In** and enter your credentials if you're not already signed in.
2. After you're signed in, click **New**.



1. In the **Create web app on Microsoft Azure** dialog box, in the **Web App name** box, enter a unique name.

**Note**: The complete URL will consist of what you enter here plus .azurewebsites.net (as shown next to the Web App name box). For example, if the Web App name is ContosoAds, the URL will be ContosoAds.azurewebsites.net.

1. In the [**App Service plan**](https://azure.microsoft.com/en-us/documentation/articles/azure-web-sites-web-hosting-plans-in-depth-overview/) drop-down list, select Create new App Service plan and then enter a name for the App Service plan, for example, ContosoAdsPlan.
2. In the [**Resource** group](https://azure.microsoft.com/en-us/documentation/articles/resource-group-overview/) drop-down list, select Create new resource group, and then enter a name for the resource group, for example, ContosoAdsGroup.
3. In the **Region** drop-down list, select the same region you chose for your storage account.

**Note**: This setting specifies which Azure datacenter your Web App will run in. Keeping the Web App and storage account in the same datacenter minimizes latency and data egress charges.

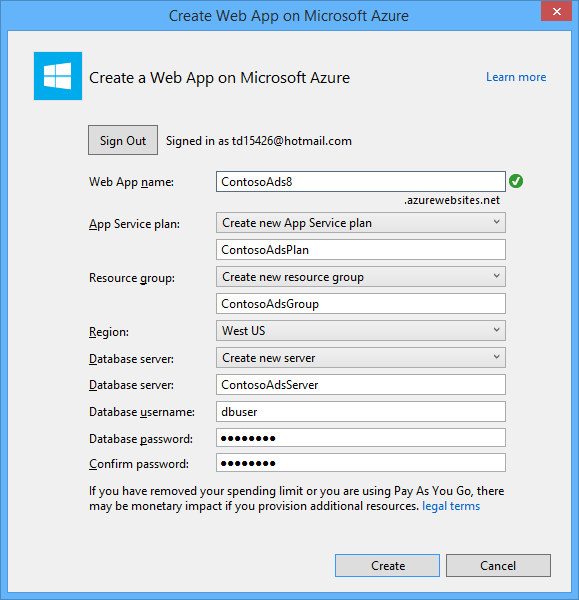
1. In the **Database server** drop-down list, select **Create new server**.

Alternatively, if your subscription already has a server, you can select that server from the drop-down list.

1. Enter a name for the database server, for example, ContosoAdsServer.
2. Enter an administrator **Database username** and **Database password**.

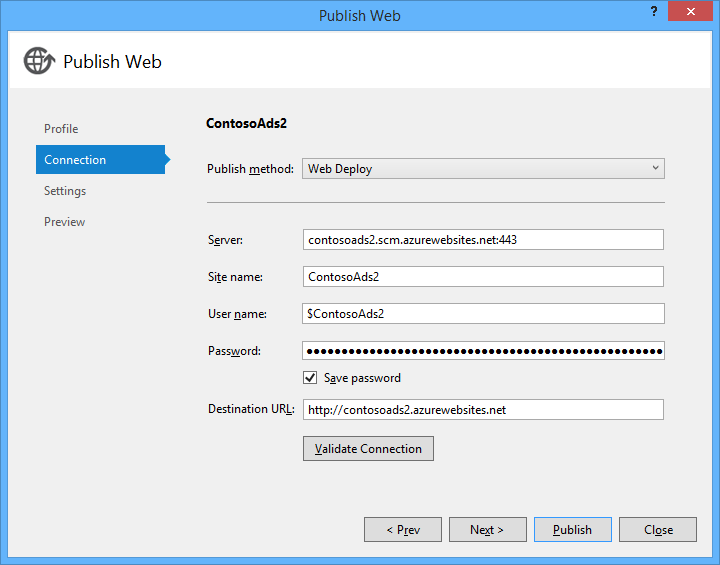
**Note**: If you selected New SQL Server Database server, you are not entering an existing name and password here; you are entering a new name and password that you have defined now to use later when you access the database. If you selected a server that you created previously, you will be prompted for the password to the administrative user account you already created.

1. Click **Create**.

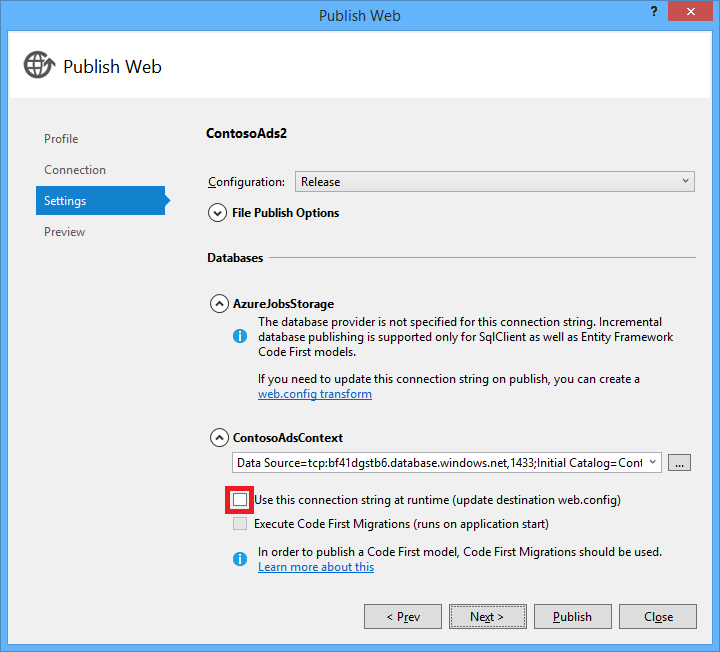


**Note**: Visual Studio creates the solution, the web project, the Web App in Azure, and the Azure SQL Server Database instance.

1. On the **Connection** page of the wizard, click **Next**.



1. On the **Settings** page of the wizard, clear the **Use this connection string** **at runtime** check box, and then click **Next**.



You don't need to use the Publish Web wizard to set the SQL Server connection string because you'll set that value in the Azure environment later.

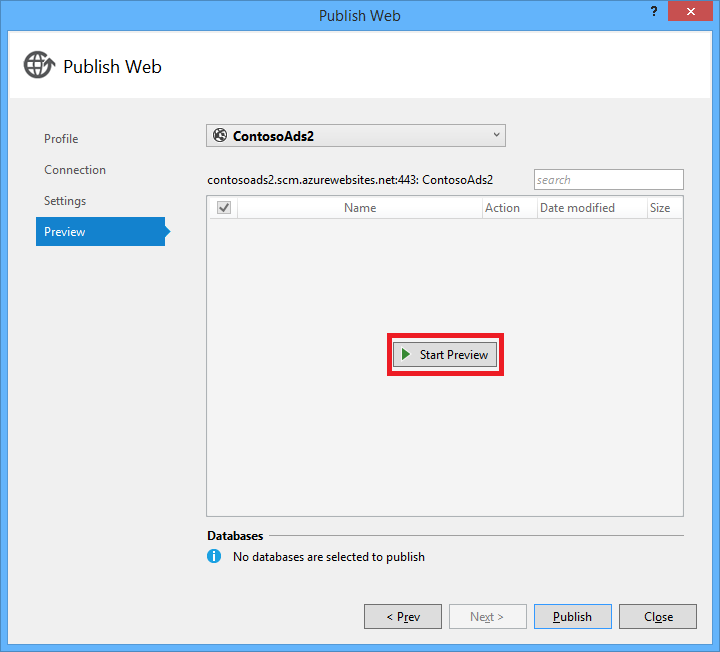
You can ignore the warnings on this page.

**Note**: Normally the storage account you use when running in Azure would be different from the one you use when running locally, but for this tutorial, you're using the same one in both environments. So the AzureWebJobsStorage connection string does not need to be transformed. Even if you want to use a different storage account in the cloud, you wouldn't need to transform the connection string because the app will use an Azure environment setting when it runs in Azure. You'll see this later in the tutorial.

For this tutorial, you aren't going to be making changes to the data model used for the ContosoAdsContext database, so there is no need to use Entity Framework Code First Migrations for deployment. Entity Framework Code First will automatically create a new database the first time the app tries to access SQL Server data.

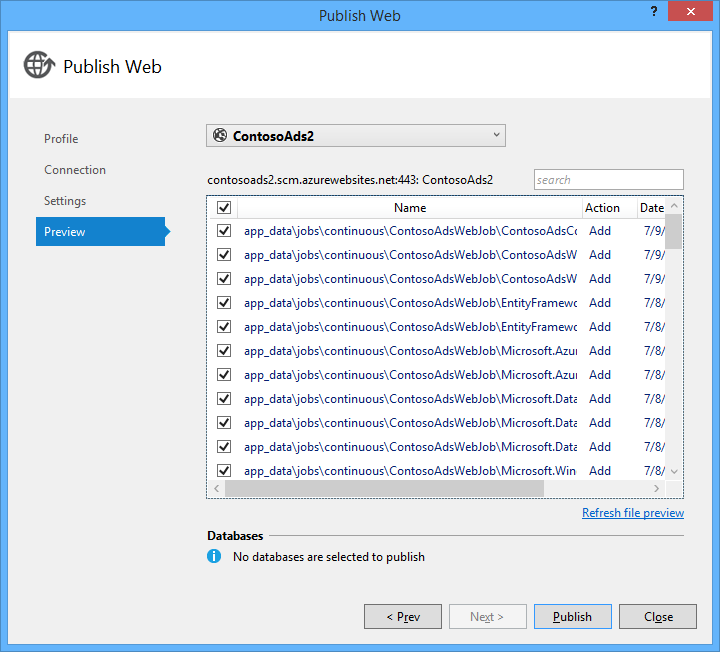
For this tutorial, the default values of the options under File Publish Options are fine.

1. On the Preview page of the wizard, click **Start Preview**.



**Note**: You can ignore the warning about no databases being published. Entity Framework Code First will create the database; it doesn't need to be published.

The preview window shows that binaries and configuration files from the WebJob project will be copied to the *app\_data\jobs\continuous* folder of the Web App.



1. Click **Publish**.

Visual Studio deploys the application and opens the home page URL in the browser.

You won't be able to use the Web App until you set connection strings in the Azure environment in the next section. You'll see either an error page or the home page depending on Web App and database creation options you chose earlier.

Configure the Web App to use your Azure SQL Server database and storage account.

It's a security best practice to [avoid putting sensitive information such as connection strings in files that are stored in source code repositories](http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/source-control#secrets). Azure provides a way to do that. You can set connection string and other setting values in the Azure environment, and ASP.NET configuration APIs automatically pick up these values when the app runs in Azure. You can set these values in Azure by using Server Explorer, the portal, Windows PowerShell, or the cross-platform command-line interface. For more information, see [How Application Strings and Connection Strings Work](https://azure.microsoft.com/blog/2013/07/17/windows-azure-web-sites-how-application-strings-and-connection-strings-work/).

In this section, you use Server Explorer to set connection string values in Azure.

1. In Server Explorer, under the Web Apps node, right-click your Web App, and then click View Settings.

The Azure Web App window opens on the Configuration tab.

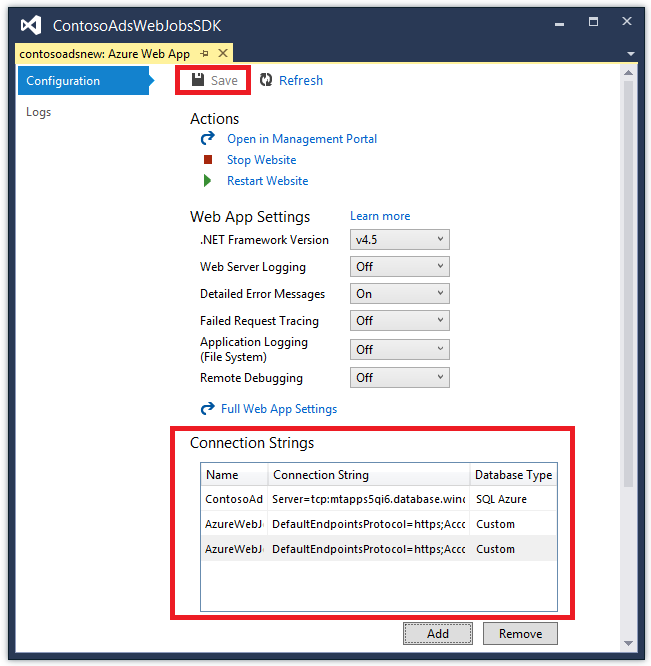
1. Change the name of the DefaultConnection connection string to ContosoAdsContext.

Azure automatically created this connection string when you created the Web App with an associated database, so it already has the right connection string value. You're changing just the name to what your code is looking for.

1. Add two new connection strings, named AzureWebJobsStorage and AzureWebJobsDashboard. Set the type to Custom, and set the connection string value to the same value that you used earlier for the Web.config and App.config files. (Make sure you include the entire connection string, not just the access key, and don't include the quotation marks.)

These connection strings are used by the WebJobs SDK, one for application data and one for logging. As you saw earlier, the one for application data is also used by the web front end code.

1. Click **Save**.



1. In **Server Explorer**, right-click **Web App**, and then select **Stop Web App**.
2. After the Web App stops, right-click the Web App again, and then click **Start web app**.

The WebJob automatically starts when you publish, but it stops when you make a configuration change. To restart it, you can either restart the Web App or restart the WebJob in the [Azure portal](http://go.microsoft.com/fwlink/?linkid=529715&clcid=0x409). It's generally recommended to restart the Web App after a configuration change.

1. Refresh the browser window that has the Web App URL in its address bar.

The home page appears.

1. Create an ad, as you did when you ran the application locally.

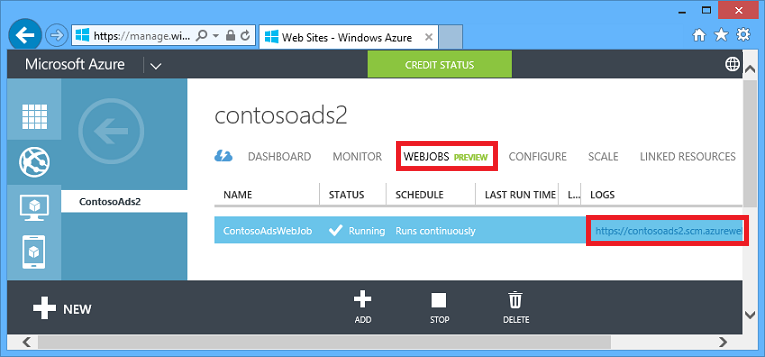
The Index page shows without a thumbnail at first.

1. Refresh the page after a few seconds, and the thumbnail appears.

If the thumbnail doesn't appear, the WebJob may not have started automatically. In that case, go to the WebJobs tab in the Azure Portal page for your web app, and then click **Start.**

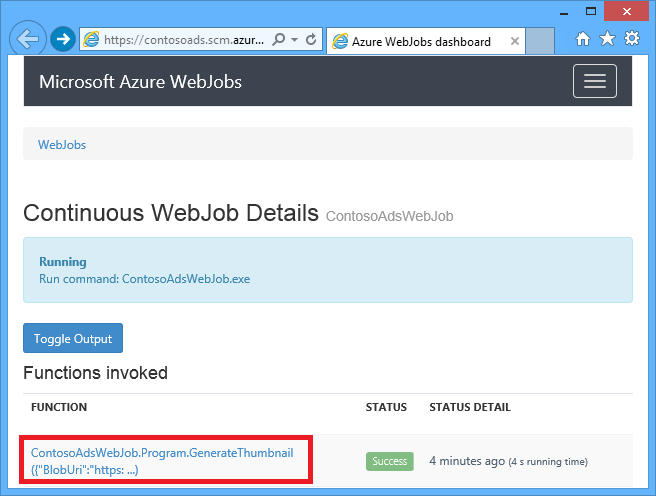
View the WebJobs SDK dashboard

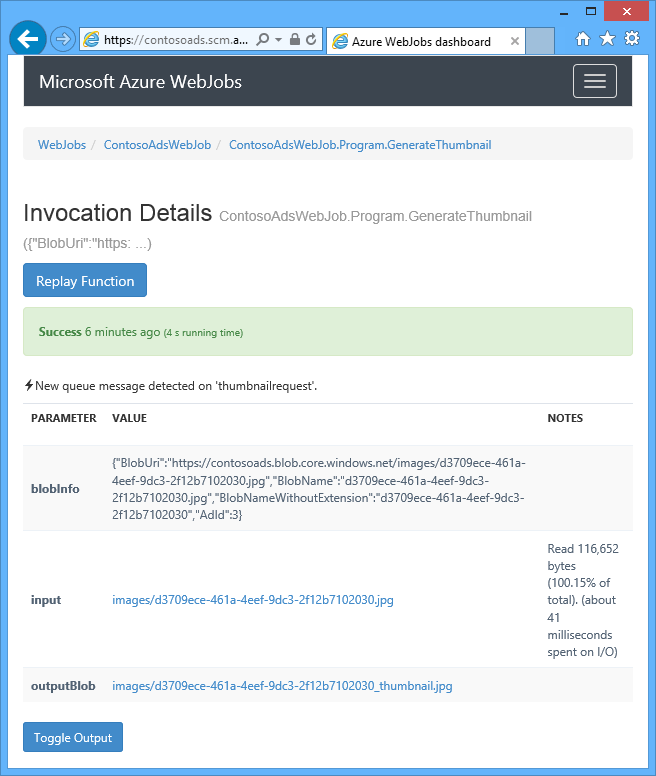
1. In the [Azure portal](https://manage.windowsazure.com/), select your Web App.
2. Click the **WEBJOBS** tab.
3. In the **LOGS** column of your WebJob, click the URL.



A new browser tab opens to the WebJobs SDK dashboard. The dashboard shows that the WebJob is running and shows a list of functions in your code that the WebJobs SDK triggered.

1. Click one of the functions to see details about its execution.





The **Replay Function** button on this page causes the WebJobs SDK framework to call the function again, and it gives you a chance to change the data passed to the function first.

**Note**:

When you're finished testing, delete the Web App and the SQL Server database instance. The Web App is free, but the SQL Server database instance and storage account accrue charges (minimal due to small size). Also, if you leave the Web App running, anyone who finds your URL can create and view ads. In the Azure portal, go to the **Dashboard** tab for your Web App, and then click the **Delete** button at the bottom of the page. You can then select a check box to delete the SQL Server database instance at the same time. If you just want to temporarily prevent others from accessing the Web App, click Stop instead. In that case, charges will continue to accrue for the SQL Server database and Storage account. You can follow a similar procedure to delete the SQL Server database and storage account when you no longer need them.

Enable AlwaysOn for long-running processes.

To make sure your WebJobs are always running, and running on all instances of your Web App you have to enabled the [AlwaysOn](http://weblogs.asp.net/scottgu/archive/2014/01/16/windows-azure-staging-publishing-support-for-web-sites-monitoring-improvements-hyper-v-recovery-manager-ga-and-pci-compliance.aspx) feature.

## Exercise 2: Create the application from scratch

To fully understand how we achieved the previous result, we will recreate everything and detail out all the steps in the following exercise.

#### Objectives

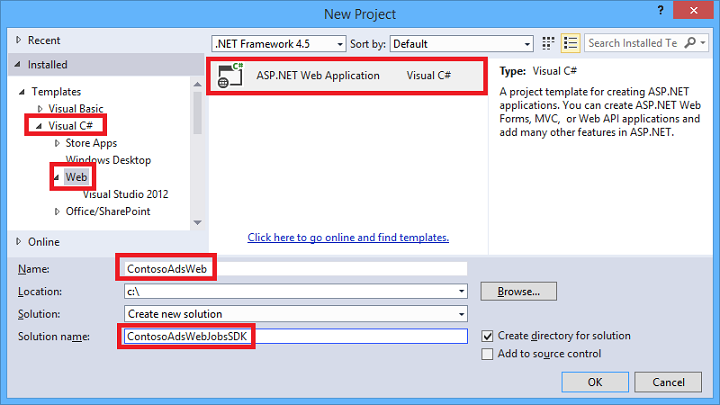
In this exercise, you will:

* Create a Visual Studio solution with a web project.
* Add a class library project for the data access layer that is shared between frontend and back end.
* Add a Console Application project for the back end, with WebJobs deployment enabled.
* Add NuGet packages.
* Set project references.
* Copy application code and configuration files from the downloaded application that you worked with in the previous section of the tutorial.
* Review the parts of the code that work with Azure blobs and queues and the WebJobs SDK.

### Task 1: Create a Visual Studio solution with a web project and class library project

1. In Visual Studio, click **File** > **Project** > **New**.
2. In the **New Project** dialog box, in the left pane, expand Installed > Templates > Visual C#, and then click Web.
3. In the middle pane of the New Project dialog box, click ASP.NET Web Application, in the Name box, enter the project name ContosoAdsWeb, in the Solution box, enter the solution name ContosoAdsWebJobsSDK, and then click OK.

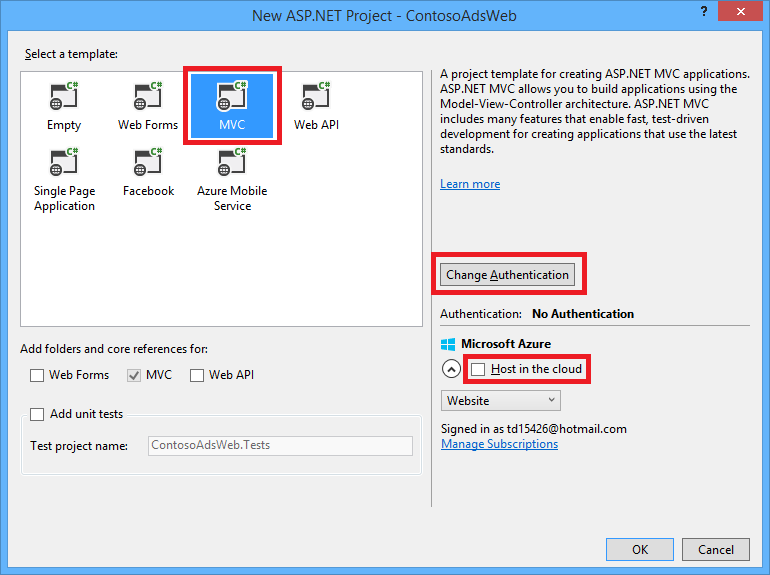
**Note**: You can change the solution name if you're putting it in the same folder as the downloaded solution.



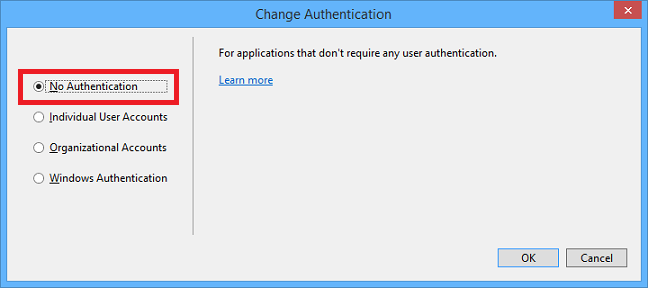
1. In the **New ASP.NET Project** dialog box, in the **Select a template** section, click **MVC**. Next, under **Microsoft Azure**, clear the **Host in the cloud** check box.

**Note**: Selecting Host in the cloud enables Visual Studio to automatically create a new Azure Web App and SQL Server database. Since you already created these earlier, you don't need to do so now while creating the project. If you want to create a new one, select the check box. You can then configure the new Web App and SQL Server database the same way you did earlier when you were deploying the application.

1. In the New ASP.NET Project dialog box, click Change Authentication.



1. In the **Change Authentication** dialog box, click **No Authentication**, and then click **OK**.



1. In the **New ASP.NET Project** dialog box, click **OK**.

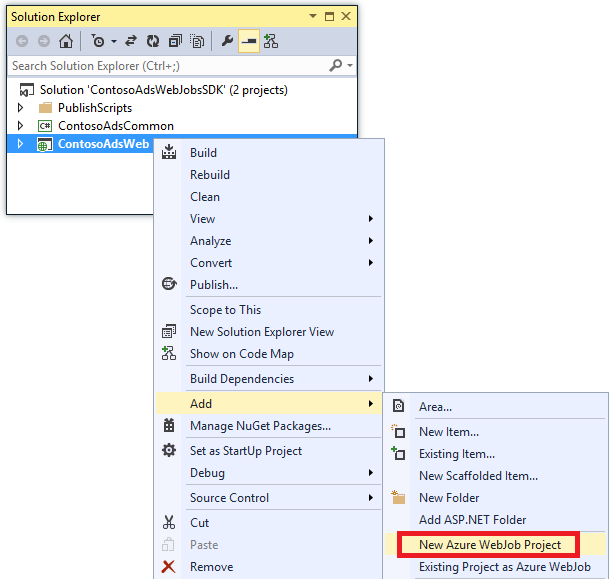
Visual Studio creates the solution and the web project.

1. In Solution Explorer, right-click the solution, and select **Add** > **New Project**.
2. In the Add New Project dialog box, expand Installed > Templates > Visual C# and then click Windows Desktop > Class Library.
3. Name the project **ContosoAdsCommon**, and then click **OK**.

**Note**: This project will contain the Entity Framework context and the data model which both the frontend and backend will use. As an alternative you could define the EF-related classes in the web project and reference that project from the WebJob project. But then your WebJob project would have a reference to web assemblies which it doesn't need.

### Task 2: Add a Console Application Project having WebJobs Deployment Enabled

1. Right-click the web project (not the solution or the class library project), and then select **Add** > **New Azure WebJob Project**.



1. In the **Add Azure WebJob** dialog box, enter **ContosoAdsWebJob** as both the Project name and the WebJob name. Leave WebJob run mode set to **Run Continuously**.
2. Click **OK**.

Visual Studio creates a console application that is configured to deploy as a WebJob whenever you deploy the web project. To do that, it performed the following tasks after creating the project:

• Added a *webjob-publish-settings.json* file in the WebJob project Properties folder.

• Added a *webjobs-list.json* file in the web project Properties folder.

• Installed the Microsoft.Web.WebJobs.Publish NuGet package in the WebJob project.

For more information about these changes, see [How to Deploy WebJobs by using Visual Studio](https://azure.microsoft.com/en-us/documentation/articles/websites-dotnet-deploy-webjobs/).

### Task 3: Add NuGet Packages

The new-project template for a WebJob project automatically installs the WebJobs SDK NuGet package [Microsoft.Azure.WebJobs](http://www.nuget.org/packages/Microsoft.Azure.WebJobs) and its dependencies.

One of the WebJobs SDK dependencies that is installed automatically in the WebJob project is the Azure Storage Client Library (SCL). However, you need to add it to the web project to work with blobs and queues.

1. Open the **Manage NuGet Packages** dialog box for the solution.
2. In the left pane, select **Installed packages**.
3. Find the **Azure Storage package**, and then click **Manage**.
4. In the **Select Projects** box, select the **ContosoAdsWeb** check box, and then click **OK**.

All three projects use the Entity Framework to work with data in SQL Database.

1. In the left pane, select **Online**.
2. Find the **EntityFramework NuGet package**, and install it in all three projects.

### Task 4: Set project references

Both web and WebJob projects will work with the SQL Server database, so both need a reference to the ContosoAdsCommon project.

1. In the **ContosoAdsWeb** project, set a reference to the ContosoAdsCommon project. (Right-click the **ContosoAdsWeb** project, and then select **Add** > **Reference**.)
2. In the **Reference Manager** dialog box, select **Solution** > **Projects** > **ContosoAdsCommon**, and then click **OK**.
3. In the **ContosoAdsWebJob** project, set a reference to the **ContosAdsCommon** project.
4. The WebJob project needs references for working with images and for accessing connection strings.
5. In the ContosoAdsWebJob project, set a reference to System.Drawing and System.Configuration.

### Task 5: Add Code and Configuration Files

This tutorial does not show how to [create MVC controllers and views using scaffolding](http://www.asp.net/mvc/tutorials/mvc-5/introduction/getting-started), how to [write Entity Framework code that works with SQL Server databases](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc), or [the basics of asynchronous programming in ASP.NET 4.5](http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/web-development-best-practices#async). So all that remains to do is copy code and configuration files from the downloaded solution into the new solution. After you do that, the following sections will show and explain key parts of the code.

1. To add files to a project or a folder, right-click the project or folder and select **Add** > **Existing Item**. Select the files you want and click **Add**. If asked whether you want to replace existing files, click **Yes**.
2. In the ContosoAdsCommon project, delete the *Class1.cs* file and add in its place the following files from the downloaded project.
   * *Ad.cs*
   * *ContosoAdscontext.cs*
   * *BlobInformation.cs*
3. In the ContosoAdsWeb project, add the following files from the downloaded project.
   * *Web.config*
   * *Global.asax.cs*
   * In the *Controllers* folder: *AdController.cs*
   * In the *Views\Shared* folder: *\_Layout.cshtml* file.
   * In the *Views\Home* folder: *Index.cshtml*.
   * In the *Views\Ad* folder (create the folder first): five *.cshtml* files.
4. In the ContosoAdsWebJob project, add the following files from the downloaded project.
   * *App.config* (change the file type filter to All Files)
   * *Program.cs*
   * *Functions.cs*

You can now build, run, and deploy the application as instructed earlier in the tutorial. Before you do that, however, stop the WebJob that is still running in the first Web App you deployed to. Otherwise that WebJob will process queue messages created locally or by the app running in a new Web App, since all are using the same storage account.

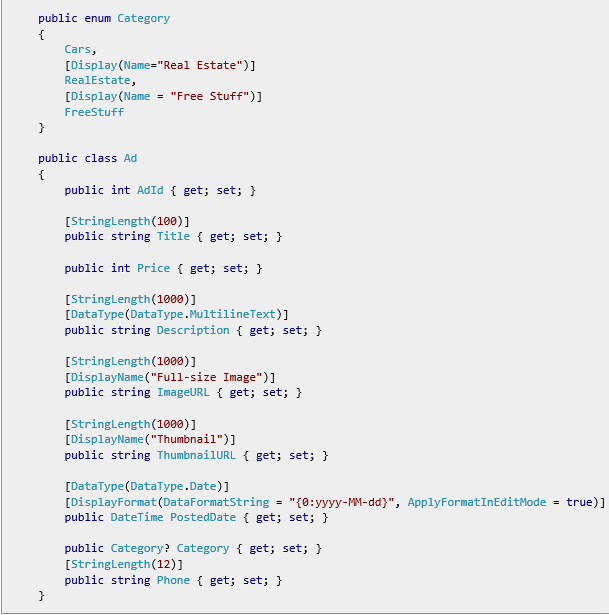
### Task 6: Review the Application Code

The following sections explain the code related to working with the WebJobs SDK and Azure Storage blobs and queues.

*Note: For the code specific to the WebJobs SDK, see* [*Program.cs and Functions.cs*](https://azure.microsoft.com/en-us/documentation/articles/websites-dotnet-webjobs-sdk-get-started/#programcs)*.*

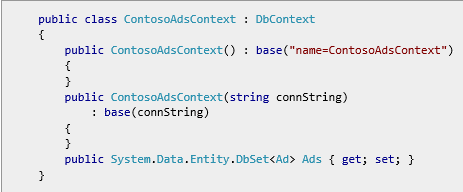
**ContosoAdsCommon - Ad.cs**

The Ad.cs file defines an enum for ad categories and a POCO entity class for ad information.



**ContosoAdsCommon - ContosoAdsContext.cs**

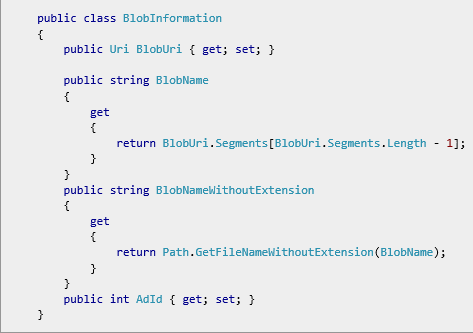
The ContosoAdsContext class specifies that the Ad class is used in a DbSet collection, which Entity Framework will store in a SQL Server database.



The class has two constructors. The first of them is used by the web project, and specifies the name of a connection string that is stored in the Web.config file or the Azure runtime environment. The second constructor enables you to pass in the actual connection string. That is needed by the WebJob project, since it doesn't have a Web.config file. You saw earlier where this connection string was stored, and you'll see later how the code retrieves the connection string when it instantiates the DbContext class.

**ContosoAdsCommon - BlobInformation.cs**

The BlobInformation class is used to store information about an image blob in a queue message.



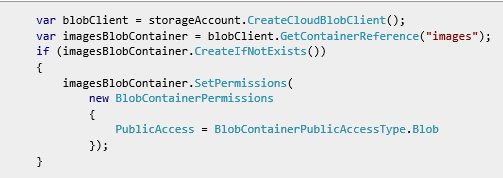
**ContosoAdsWeb - Global.asax.cs**

Code that is called from the Application\_Start method creates an *images* blob container and an *images* queue if they don't already exist. This ensures that whenever you start using a new storage account, the required blob container and queue will be created automatically.

The code gets access to the storage account by using the storage connection string from the *Web.config* file or Azure runtime environment.



Then it gets a reference to the *images* blob container, creates the container if it doesn't already exist, and sets access permissions on the new container. By default, new containers allow only clients with storage account credentials to access blobs. The Web App needs the blobs to be public so that it can display images using URLs that point to the image blobs.



Similar code gets a reference to the *blobnamerequest* queue and creates a new queue. In this case no permissions change is needed. The [ResolveBlobName](https://azure.microsoft.com/en-us/documentation/articles/websites-dotnet-webjobs-sdk-get-started/#resolveblobname) section later in the tutorial explains why the queue that the web application writes to is used just for getting blob names and not for generating thumbnails.



**ContosoAdsWeb - \_Layout.cshtml**

The *\_Layout.cshtml* file sets the app name in the header and footer, and creates an "Ads" menu entry.

**ContosoAdsWeb - Views\Home\Index.cshtml**

The *Views\Home\Index.cshtml* file displays category links on the home page. The links pass the integer value of the Category enum in a querystring variable to the Ads Index page.

<li>@Html.ActionLink("Cars", "Index", "Ad", new { category = (int)Category.Cars }, null)</li>

<li>@Html.ActionLink("Real estate", "Index", "Ad", new { category = (int)Category.RealEstate }, null)</li>

<li>@Html.ActionLink("Free stuff", "Index", "Ad", new { category = (int)Category.FreeStuff }, null)</li>

<li>@Html.ActionLink("All", "Index", "Ad", null, null)</li>

**ContosoAdsWeb - AdController.cs**

In the *AdController.cs* file, the constructor calls the InitializeStorage method to create Azure Storage Client Library objects that provide an API for working with blobs and queues.

Then the code gets a reference to the *images* blob container as you saw earlier in *Global.asax.cs*. While doing that, it sets a default [retry policy](http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/transient-fault-handling) appropriate for a Web App. The default exponential backoff retry policy could hang the Web App for longer than a minute on repeated retries for a transient fault. The retry policy specified here waits 3 seconds after each try for up to three tries.

var blobClient = storageAccount.CreateCloudBlobClient();

blobClient.DefaultRequestOptions.RetryPolicy = new LinearRetry(TimeSpan.FromSeconds(3), 3);

imagesBlobContainer = blobClient.GetContainerReference("images");

Similar code gets a reference to the *images* queue.

CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();

queueClient.DefaultRequestOptions.RetryPolicy = new LinearRetry(TimeSpan.FromSeconds(3), 3);

imagesQueue = queueClient.GetQueueReference("blobnamerequest");

Most of the controller code is typical for working with an Entity Framework data model using a DbContext class. An exception is the HttpPost Create method, which uploads a file and saves it in blob storage. The model binder provides an [HttpPostedFileBase](http://msdn.microsoft.com/library/system.web.httppostedfilebase.aspx) object to the method.

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Create(

[Bind(Include = "Title,Price,Description,Category,Phone")] Ad ad,

HttpPostedFileBase imageFile)

If the user selected a file to upload, the code uploads the file, saves it in a blob, and updates the Ad database record with a URL that points to the blob.

if (imageFile != null && imageFile.ContentLength != 0)

{

blob = await UploadAndSaveBlobAsync(imageFile);

ad.ImageURL = blob.Uri.ToString();

}

The code that does the upload is in the UploadAndSaveBlobAsync method. It creates a GUID name for the blob, uploads and saves the file, and returns a reference to the saved blob.

private async Task<CloudBlockBlob> UploadAndSaveBlobAsync(HttpPostedFileBase imageFile)

{

string blobName = Guid.NewGuid().ToString() + Path.GetExtension(imageFile.FileName);

CloudBlockBlob imageBlob = imagesBlobContainer.GetBlockBlobReference(blobName);

using (var fileStream = imageFile.InputStream)

{

await imageBlob.UploadFromStreamAsync(fileStream);

}

return imageBlob;

}

After the HttpPost Create method uploads a blob and updates the database, it creates a queue message to inform the back-end process that an image is ready for conversion to a thumbnail.

BlobInformation blobInfo = new BlobInformation() { AdId = ad.AdId, BlobUri = new Uri(ad.ImageURL) };

var queueMessage = new CloudQueueMessage(JsonConvert.SerializeObject(blobInfo));

await thumbnailRequestQueue.AddMessageAsync(queueMessage);

The code for the HttpPost Edit method is similar except that if the user selects a new image file any blobs that already exist for this ad must be deleted.

if (imageFile != null && imageFile.ContentLength != 0)

{

await DeleteAdBlobsAsync(ad);

imageBlob = await UploadAndSaveBlobAsync(imageFile);

ad.ImageURL = imageBlob.Uri.ToString();

}

Here is the code that deletes blobs when you delete an ad:

private async Task DeleteAdBlobsAsync(Ad ad)

{

if (!string.IsNullOrWhiteSpace(ad.ImageURL))

{

Uri blobUri = new Uri(ad.ImageURL);

await DeleteAdBlobAsync(blobUri);

}

if (!string.IsNullOrWhiteSpace(ad.ThumbnailURL))

{

Uri blobUri = new Uri(ad.ThumbnailURL);

await DeleteAdBlobAsync(blobUri);

}

}

private static async Task DeleteAdBlobAsync(Uri blobUri)

{

string blobName = blobUri.Segments[blobUri.Segments.Length - 1];

CloudBlockBlob blobToDelete = imagesBlobContainer.GetBlockBlobReference(blobName);

await blobToDelete.DeleteAsync();

}

**ContosoAdsWeb - Views\Ad\Index.cshtml and Details.cshtml**

The *Index.cshtml* file displays thumbnails with the other ad data.

<img src="@Html.Raw(item.ThumbnailURL)" />

The *Details.cshtml* file displays the full-size image.

<img src="@Html.Raw(Model.ImageURL)" />

**ContosoAdsWeb - Views\Ad\Create.cshtml and Edit.cshtml**

The *Create.cshtml* and *Edit.cshtml* files specify form encoding that enables the controller to get the HttpPostedFileBase object.

@using (Html.BeginForm("Create", "Ad", FormMethod.Post, new { enctype = "multipart/form-data" }))

An <input> element tells the browser to provide a file selection dialog.

<input type="file" name="imageFile" accept="image/\*" class="form-control fileupload" />

**ContosoAdsWebJob - Program.cs**

When the WebJob starts, the Main method calls the WebJobs SDK JobHost.RunAndBlock method to begin execution of triggered functions on the current thread.

static void Main(string[] args)

{

JobHost host = new JobHost();

host.RunAndBlock();

}

**ContosoAdsWebJob - Functions.cs - GenerateThumbnail method**

The WebJobs SDK calls this method when a queue message is received. The method creates a thumbnail and puts the thumbnail URL in the database.

public static void GenerateThumbnail(

[QueueTrigger("thumbnailrequest")] BlobInformation blobInfo,

[Blob("images/{BlobName}", FileAccess.Read)] Stream input,

[Blob("images/{BlobNameWithoutExtension}\_thumbnail.jpg")] CloudBlockBlob outputBlob)

{

using (Stream output = outputBlob.OpenWrite())

{

ConvertImageToThumbnailJPG(input, output);

outputBlob.Properties.ContentType = "image/jpeg";

}

// Entity Framework context class is not thread-safe, so it must

// be instantiated and disposed within the function.

using (ContosoAdsContext db = new ContosoAdsContext())

{

var id = blobInfo.AdId;

Ad ad = db.Ads.Find(id);

if (ad == null)

{

throw new Exception(String.Format("AdId {0} not found, can't create thumbnail", id.ToString()));

}

ad.ThumbnailURL = outputBlob.Uri.ToString();

db.SaveChanges();

}

}

The QueueTrigger attribute directs the WebJobs SDK to call this method when a new message is received on the thumbnailrequest queue.

[QueueTrigger("thumbnailrequest")] BlobInformation blobInfo,

The BlobInformation object in the queue message is automatically deserialized into the blobInfo parameter. When the method completes, the queue message is deleted. If the method fails before completing, the queue message is not deleted; after a 10-minute lease expires, the message is released to be picked up again and processed. This sequence won't be repeated indefinitely if a message always causes an exception. After five unsuccessful attempts to process a message, the message is moved to a queue named {queuename}-poison. The maximum number of attempts is configurable.

The two Blob attributes provide objects that are bound to blobs: one to the existing image blob and one to a new thumbnail blob that the method creates.

[Blob("images/{BlobName}", FileAccess.Read)] Stream input,

[Blob("images/{BlobNameWithoutExtension}\_thumbnail.jpg")] CloudBlockBlob outputBlob)

Blob names come from properties of the BlobInformation object received in the queue message (BlobName and BlobNameWithoutExtension). To get the full functionality of the Storage Client Library you can use the CloudBlockBlob class to work with blobs. If you want to reuse code that was written to work with Stream objects, you can use the Stream class.