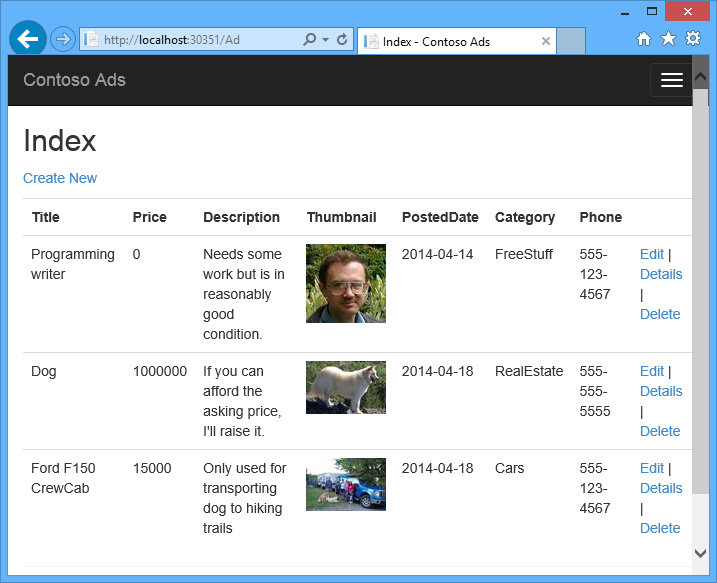
# Get Started with Azure Cloud Services and ASP.NET

This tutorial shows how to create a multi-tier .NET application with an ASP.NET MVC front-end, and deploy it to an [Azure cloud service](file:///C:\en-us\documentation\articles\fundamentals-application-models\#CloudServices). The application uses [Azure SQL Database](http://msdn.microsoft.com/library/azure/ee336279), the [Azure Blob service](http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/unstructured-blob-storage), and the [Azure Queue service](http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/queue-centric-work-pattern). You can [download the Visual Studio project](http://code.msdn.microsoft.com/todo) from the MSDN Code Gallery.

### The Contoso Ads application

The application is an advertising bulletin board. Users create an ad by entering text and uploading an image. They can see a list of ads with thumbnail images, and they can see the full size image when they select an ad to see the details. Here's a screenshot:



The 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.

### Alternative architecture: Web Sites and WebJobs

This tutorial shows how to run both front-end and back-end in an Azure cloud service. An alternative is to run the front-end in an [Azure web site](file:///C:\en-us\services\web-sites\) and use the [WebJobs](http://go.microsoft.com/fwlink/?LinkId=390226) feature (currently in preview) for the back-end. For information about how to choose the services that best fit your scenario, see [Azure Web Sites, Cloud Services, and Virtual Machines Comparison](http://azure.microsoft.com/en-us/documentation/articles/choose-web-site-cloud-service-vm/).

### What you'll learn

* How to enable your machine for Azure development by installing the Azure SDK.
* How to create a Visual Studio cloud service project with an ASP.NET MVC web role and a worker role.
* How to test the cloud service project locally, using the Azure storage emulator.
* How to publish the cloud project to an Azure cloud service and test using an Azure storage account.
* How to upload files and store them in the Azure Blob service.
* How to use the Azure Queue service for communication between tiers.

### Prerequisites

The tutorial assumes that you understand [basic concepts about Azure cloud services](http://azure.microsoft.com/en-us/documentation/articles/fundamentals-application-models/#CloudServices) such as web role and worker role terminology. It also assumes that you know how to work with [ASP.NET MVC](http://www.asp.net/mvc/tutorials/mvc-5/introduction/getting-started) or [Web Forms](http://www.asp.net/web-forms/tutorials/aspnet-45/getting-started-with-aspnet-45-web-forms/introduction-and-overview) projects in Visual Studio. The sample application uses MVC, but most of the tutorial also applies to Web Forms.

You can run the app locally without an Azure subscription, but you'll need one in order to deploy the application to the cloud. If you don't have an account, you can [activate your MSDN subscriber benefits](file:///C:\en-us\pricing\member-offers\msdn-benefits-details\?WT.mc_id=A261C142F) or [sign up for a free trial](file:///C:\en-us\pricing\free-trial\?WT.mc_id=A261C142F).

The tutorial instructions work with either of the following products:

* Visual Studio 2013
* Visual Studio 2013 Express for Web

If you don't have one of these, Visual Studio 2013 Express for web will be installed automatically when you install the Azure SDK.

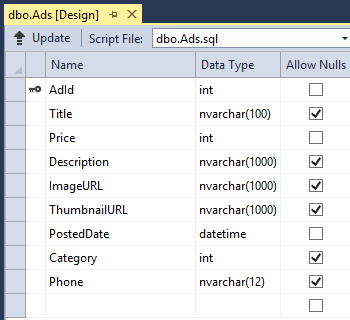
### Segments of this tutorial

The tutorial shows you how to build and run the application locally, how to deploy it to Azure and run in the cloud, and finally how to build it from scratch. You can start by building from scratch and then do the test and deploy steps afterward if you prefer.

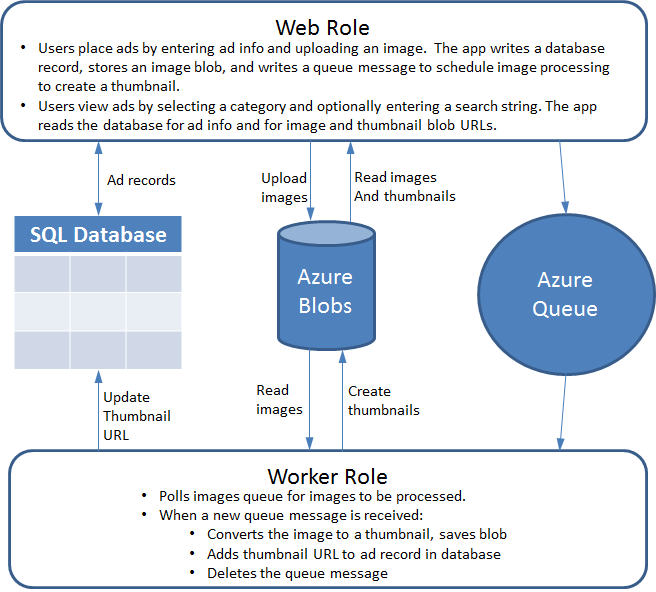
* [Application architecture](file:///C:\Users\tdykstra\SkyDrive%20(3)\Craigslist%20app\Untitled_1.html#application-architecture)
* [Set up the development environment](file:///C:\Users\tdykstra\SkyDrive%20(3)\Craigslist%20app\Untitled_1.html#set-up-the-development-environment)
* [Download and run the completed solution](file:///C:\Users\tdykstra\SkyDrive%20(3)\Craigslist%20app\Untitled_1.html#download-and-run-the-completed-solution)
* [Deploy the application to Azure](file:///C:\Users\tdykstra\SkyDrive%20(3)\Craigslist%20app\Untitled_1.html#deploy-the-application-to-azure)
* [Create the application from scratch](file:///C:\Users\tdykstra\SkyDrive%20(3)\Craigslist%20app\Untitled_1.html#create-the-application-from-scratch)
* [Troubleshooting](file:///C:\Users\tdykstra\SkyDrive%20(3)\Craigslist%20app\Untitled_1.html#troubleshooting) (Go here if you have problems running the sample.)
* [Next steps](file:///C:\Users\tdykstra\SkyDrive%20(3)\Craigslist%20app\Untitled_1.html#next-steps)

## Application architecture

The app stores ads in a SQL 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 running in a web role 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 in a worker role periodically polls the queue for new messages. When a new message appears, the worker role creates a thumbnail for that image and updates the thumbnail URL database field for that ad. Here's a diagram that shows how the parts of the application interact:



[WACOM.INCLUDE [install-sdk-2013-only](file:///C:\Users\tdykstra\SkyDrive%20(3)\includes\install-sdk-2013-only.md)]

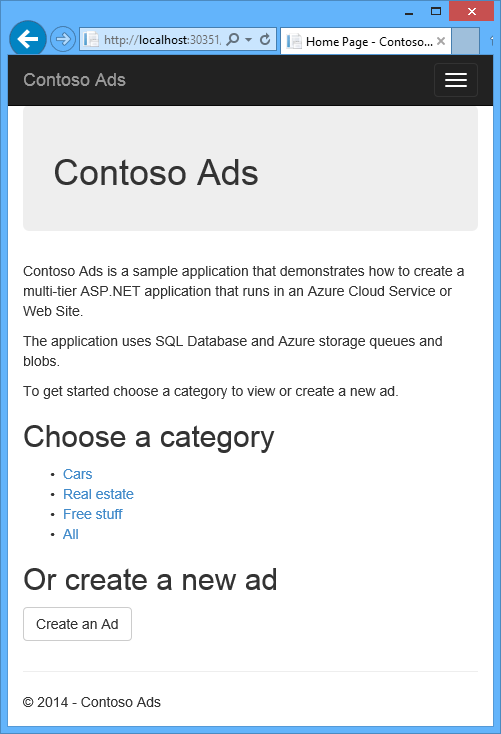
## Download and run the completed solution

1. Download and unzip the [completed solution](http://code.msdn.microsoft.com/todo).
2. Start Visual Studio.
3. From the **File** menu choose **Open Project**, navigate to where you downloaded the solution, and then open the solution file.
4. Press CTRL+SHIFT+B to build the solution.

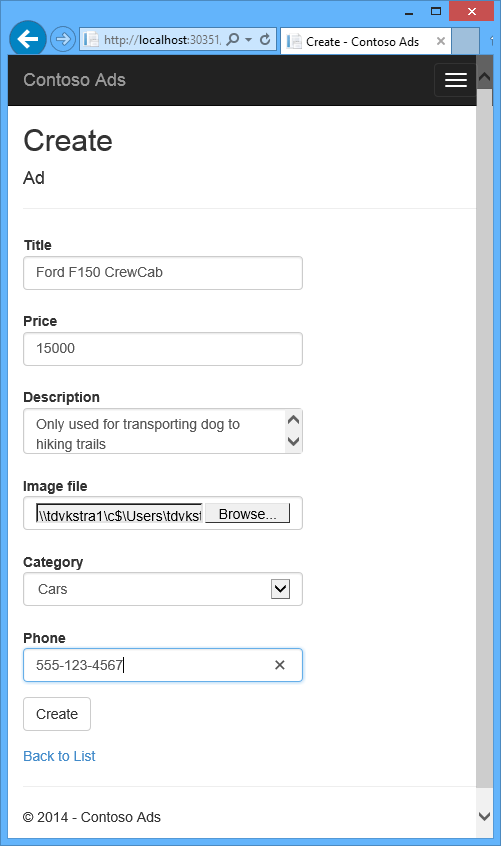
By default, Visual Studio automatically restores the NuGet package content, which was not included in the .zip file. If the packages don't restore, install them manually by going to the **Manage NuGet Packages for Solution** dialog and clicking the **Restore** button at the top right.

1. In **Solution Explorer**, make sure that **ContosoAdsCloudService** is selected as the startup project.
2. Press CTRL+F5 to run the application.

The application home page appears in your browser.

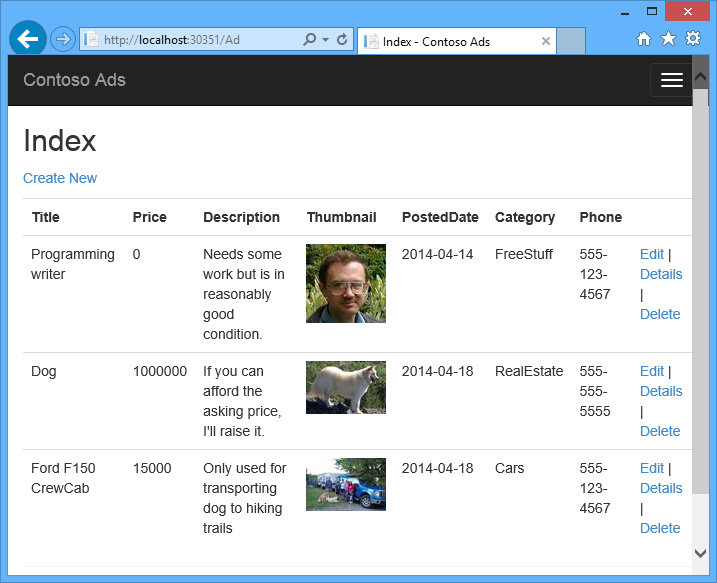


1. Click **Create an Ad**.
2. Enter some test data and select an image to upload, and then click **Create**.

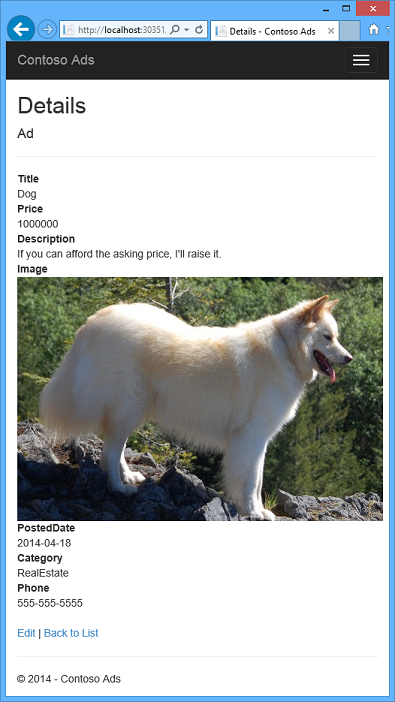


The app goes to the Index page, but it doesn't show a thumbnail for the new ad because that processing hasn't happened yet.

1. Wait a moment and then refresh the Index page to see the thumbnail.



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



You've been running a cloud app on your development machine without using any connection to the cloud. When you ran the cloud service project, Visual Studio automatically invoked the Azure compute emulator and Azure storage emulator. The compute emulator uses your computer's resources to simulate the web role and worker role environments. The storage emulator uses a [SQL Server Express LocalDB](http://msdn.microsoft.com/en-us/library/hh510202.aspx) database to simulate Azure cloud storage.

Ad data is also stored locally, in another SQL Server Express LocalDB database. Entity Framework Code First automatically created the database the first time the web app tried to access it.

In the following section of the tutorial you'll configure the solution so that when the app runs in the cloud, it will use Azure cloud resources for queues and blobs and the application database.

## Deploy the application to Azure

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

* Create an Azure cloud service
* Create an Azure SQL database
* Create an Azure storage account
* Configure the solution to use your Azure SQL database when it runs in Azure
* Configure the solution to use your Azure storage account when it runs in Azure
* Deploy the project to your Azure cloud service

### Create an Azure cloud service

An Azure cloud service is the environment the application will run in.

1. In your browser, open the [Azure Management Portal](http://manage.windowsazure.com/).
2. In Visual Studio, click **New** - **Compute** - **Cloud Service** - **Quick Create**.
3. In the URL input box, enter a URL prefix.

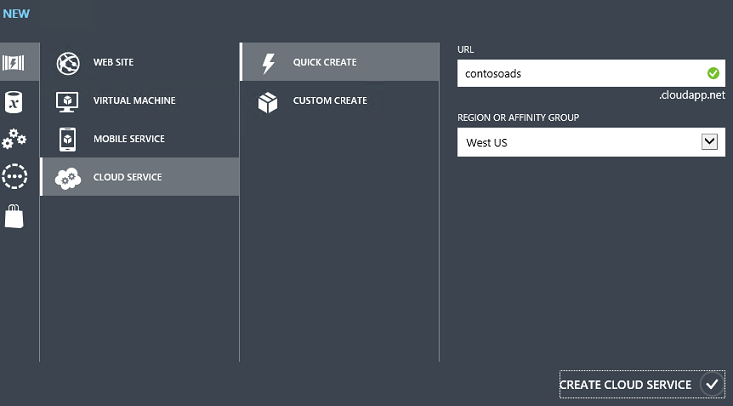
This URL has to be unique. You'll get an error message if the prefix you choose is already in use by someone else.

1. Choose the region where you want to deploy the application.

This field specifies which datacenter your cloud service will be hosted in. For a production application, you'd choose the region closest to your customers. For this tutorial, choose the region closest to you.

1. Click **Create Cloud Service**.

In the following image, a cloud service is created with the URL contosoads.cloudapp.net.



### Create an Azure SQL database

When the app runs in the cloud, it will use a cloud-based database.

1. In the Azure Management Portal, click **New** - **Data Services** - **SQL Database** - **Quick Create**.
2. In the **Database Name** box, enter contosoads.
3. From the **Server** drop-down list, choose **New SQL Database server**.

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

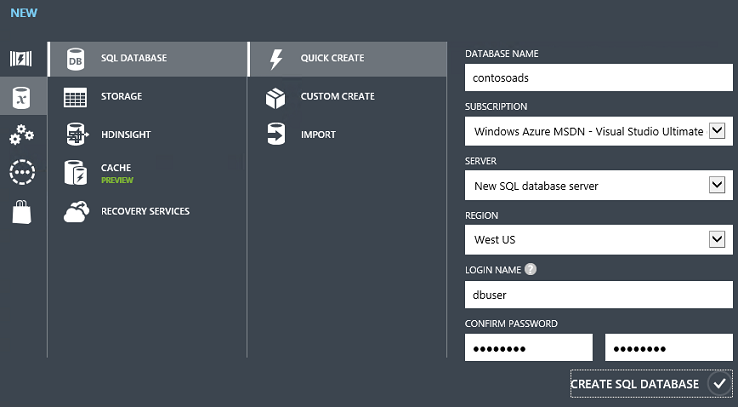
1. Choose the same **Region** that you chose for the cloud service.

When the cloud service and database are in different datacenters (different regions), latency will increase and you will be charged for bandwidth outside the data center. Bandwidth within a data center is free.

1. Enter an administrator **Login Name** and **Password**.

If you selected **New SQL Database server** you aren't entering an existing name and password here, you're entering a new name and password that you're defining now to use later when you access the database. If you selected a server that you created previously, you'll be prompted for the password to the administrative user account you already created.

1. Click **Create SQL Database**.



### Create an Azure storage account

An Azure storage account provides resources for storing queue and blob data in the cloud.

In a real-world application, you would 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 the Azure Management Portal, click **New** - **Data Services** - **Storage** - **Quick Create**.
2. In the **URL** box, enter a URL prefix.

This prefix plus the text you see under the box will be the unique URL to your storage account. If the prefix you enter has already been used by someone else, you'll have to choose a different prefix.

1. Set the **Region** drop-down list to the same region you chose for the cloud service.

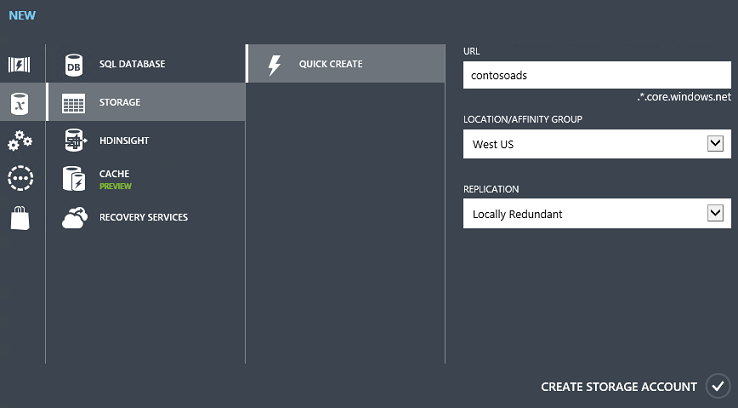
When the cloud service and storage account are in different datacenters (different regions), latency will increase and you will be charged for bandwidth outside the data center. Bandwidth within a data center is free.

Azure affinity groups provide a mechanism to minimize the distance between resources in a data center, which can reduce latency. This tutorial does not use affinity groups. For more information, see [How to Create an Affinity Group in Azure](http://msdn.microsoft.com/en-us/library/jj156209.aspx).

1. Set the **Replication** drop-down list to **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 [How To Manage Storage Accounts](file:///C:\en-us\documentation\articles\storage-manage-storage-account\).

1. Click **Create Storage Account**.



In the image, a storage account is created with the URL contosoads.core.windows.net.

### Configure the solution to use your Azure SQL database when it runs in Azure

The web project and the worker role project each has its own database connection string, and each needs to point to the Azure SQL database when the app runs in Azure.

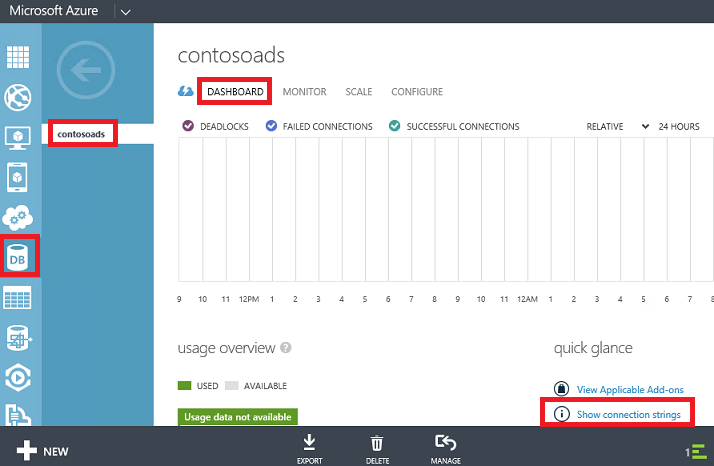
You'll use a [Web.config transform](http://www.asp.net/mvc/tutorials/deployment/visual-studio-web-deployment/web-config-transformations) for the web role and a cloud service environment setting for the worker role.

[WACOM.NOTE] In this section and the next section you store credentials in project files. [Don't store sensitive data in public 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).

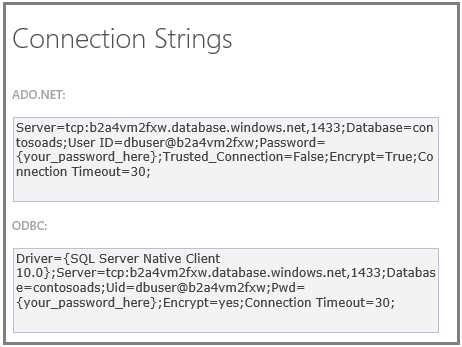
1. In the ContosoAdsWeb project, open the Web.Release.config transform file for the application Web.config file, delete the comment block that contains a <connectionStrings> element, and paste the following code in its place.
2. <connectionStrings>
3. <add name="ContosoAdsContext" connectionString="{connectionstring}"
4. providerName="System.Data.SqlClient" xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
5. </connectionStrings>

Leave the file open for editing.

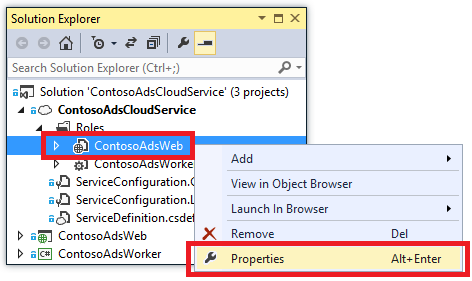
1. In the Azure Management Portal, click **SQL Databases** in the left pane, click the database you created for this tutorial, click the **Dashboard** tab, and then click **Show connection strings**.



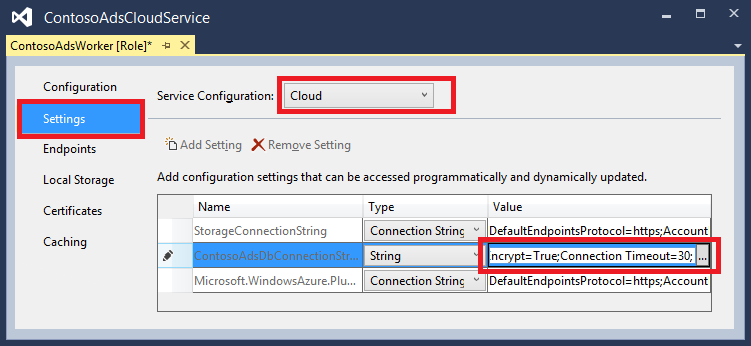
The portal displays connection strings, with a placeholder for the password.



1. In the Web.Release.config transform file, delete {connectionstring} and paste in its place the ADO.NET connection string from the management portal.
2. In the connection string that you pasted into the Web.Release.config transform file, replace {your\_password\_here} with the password you created for the new SQL database.
3. Save the file.
4. Select and copy the connection string (without the surrounding quotation marks) for use in the following steps for configuring the worker role project.
5. In **Solution Explorer**, under **Roles** in the cloud service project, right-click **ContosoAdsWorker** and click **Properties**.



1. Click the **Settings** tab.
2. Change **Service Configuration** to **Cloud**.
3. Select the text in the ContosoAdsDbConnectionString setting, and then paste the connection string that you copied from the previous section of the tutorial.

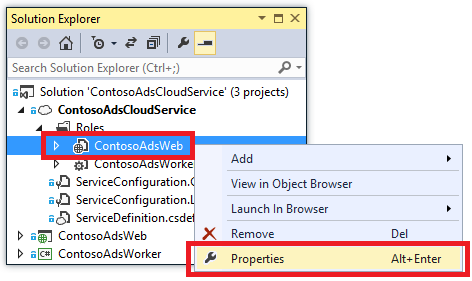


1. Save your changes.

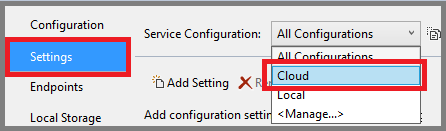
### Configure the solution to use your Azure storage account when you it runs in Azure

Azure storage account connection strings for both the web role project and the worker role project are stored in environment settings in the cloud service project. For each project there is a separate set of settings to be used when the application runs locally and when it runs in the cloud. You'll update the cloud environment settings for both web and worker role projects.

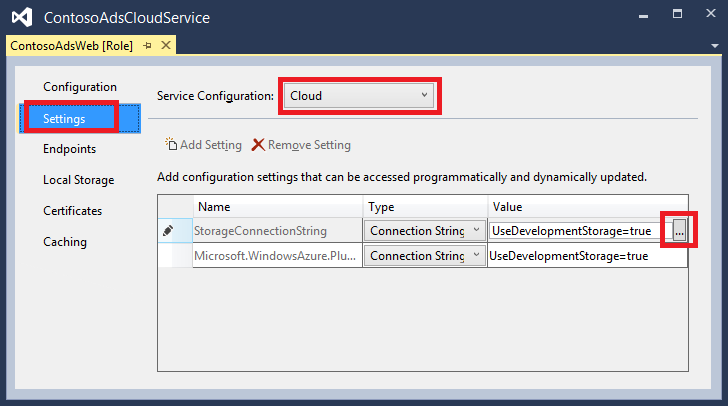
1. In **Solution Explorer**, right-click **MvcWebRole** under **Roles** in the **ContosoAdsCloudService** project, and click **Properties**.



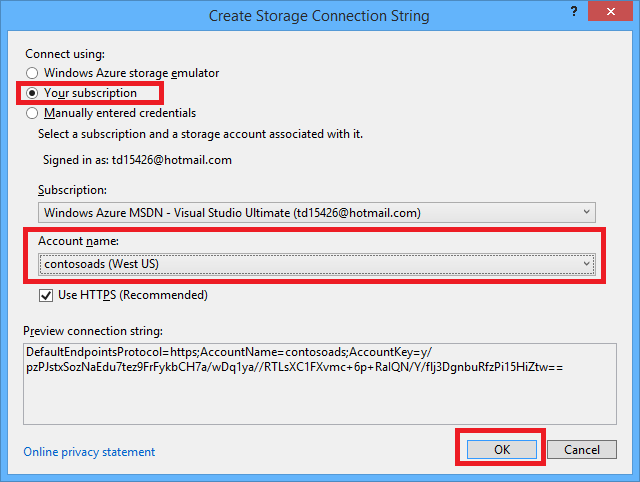
1. Click the **Settings** tab. In the **Service Configuration** drop-down box, choose **Cloud**.



1. Select the **StorageConnectionString** entry, and you'll see an ellipsis (**...**) button at the right end of the line. Click the ellipsis button to open the **Create Storage Account Connection String** dialog box.



1. In the **Create Storage Connection String** dialog, click **Connect using your subscription**, choose the storage account that you created earlier, and then click **OK**. If you're not already logged in, you'll be prompted for your Azure account credentials.



1. Save your changes.
2. Follow the same procedure that you used for the StorageConnectionString connection string to set the Microsoft.WindowsAzure.Plugins.Diagnostics.ConnectionString connection string.

This connection string is used for logging.

1. Follow the same procedure that you used for the **ContosoAdsWeb** role to set both connection strings for the **ContosoAdsWorker** role. Don't forget to set **Service Configuration** to **Cloud**.

The role environment settings that you have configured using the Visual Studio UI are stored in the following files in the ContosoAdsCloudService project:

* ServiceDefinition.csdef - Defines the setting names.
* ServiceConfiguration.Cloud.cscfg - Provides values for when the app runs in the cloud.
* ServiceConfiguration.Local.cscfg - Provides values for when the app runs locally.

For example, the ServiceDefinition.csdef includes these definitions:

<ConfigurationSettings>

<Setting name="StorageConnectionString" />

<Setting name="ContosoAdsDbConnectionString" />

</ConfigurationSettings>

And the ServiceConfiguration.Cloud.cscfg file includes the values you entered for those settings in Visual Studio:

<Role name="ContosoAdsWorker">

<Instances count="1" />

<ConfigurationSettings>

<Setting name="StorageConnectionString" value="{yourconnectionstring}" />

<Setting name="ContosoAdsDbConnectionString" value="{yourconnectionstring}" />

<!-- other settings not shown -->

</ConfigurationSettings>

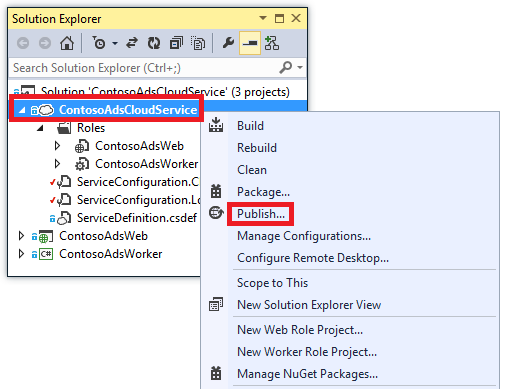
<!-- other settings not shown -->

</Role>

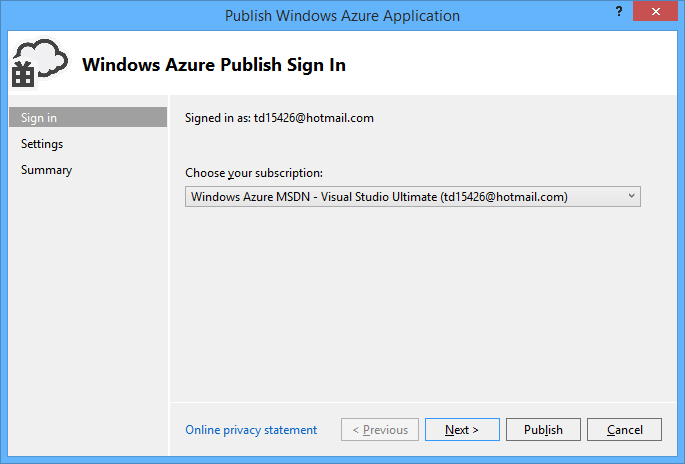
The <Instances> setting specifies the number of virtual machines that Azure will run the worker role code on. The [Next Steps](file:///C:\Users\tdykstra\SkyDrive%20(3)\Craigslist%20app\Untitled_1.html#next-steps) section includes links to more information about scaling out a cloud service,

### Deploy the project to Azure

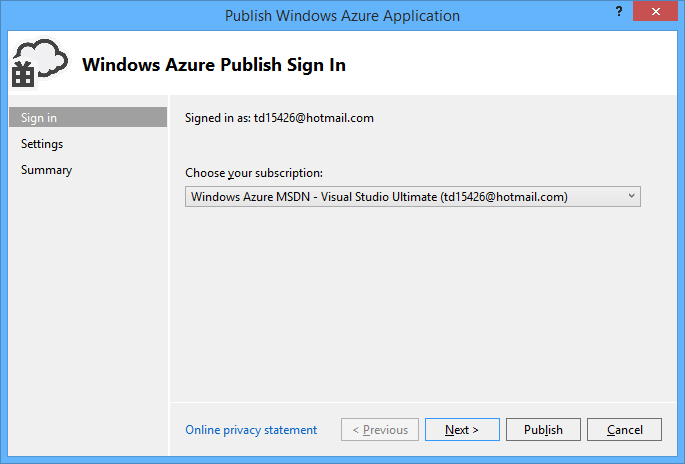
1. In **Solution Explorer**, right-click the **AzureEmailService** cloud project and select **Publish**.



1. In the **Sign in** step of the **Publish Windows Azure Application** wizard, click **Next**.

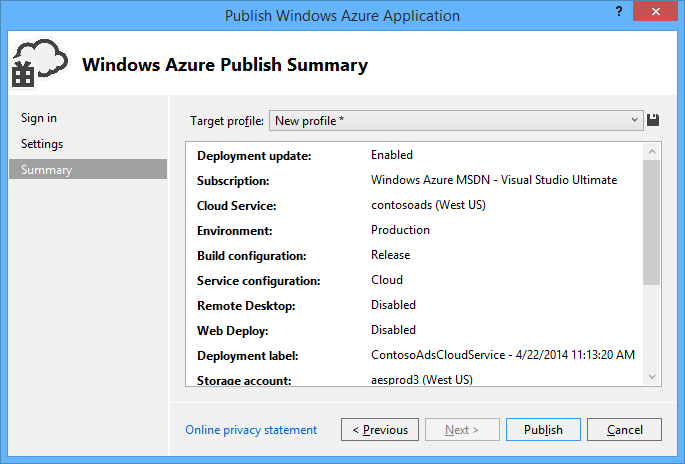


1. In the **Settings** step of the wizard, click **Next**.



The default settings in the **Advanced** tab are fine for this tutorial. For information about the advanced tab, see [Publish Azure Application Wizard](http://msdn.microsoft.com/library/windowsazure/hh535756.aspx).

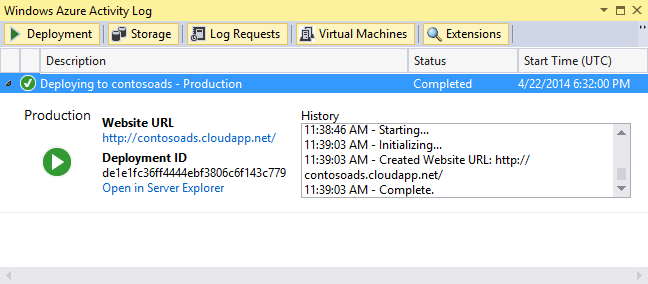
1. In the **Summary** step, click **Publish**.



The **Azure Activity Log** window opens in Visual Studio.

1. Click the right arrow icon to expand the deployment details.

The deployment can take up to 5 minutes or more to complete.



1. When the deployment status is complete, click the **Website URL** to launch the application.
2. You can now test the app by creating, viewing, and editing some ads, as you did when you ran the application locally.

[WACOM.NOTE] Anyone who finds your URL can create and view ads. To prevent other people from using the site, you can stop it when you're not using it. In the Azure management portal, click your cloud service and go to the **Dashboard** tab, then click the **Stop** button at the bottom of the page. With the cloud service stopped, you are still accruing charges for it, however. To avoid paying for resources that you aren't using, click **Delete** instead of **Stop**. You can follow a similar procedure to delete the SQL database and storage account when you no longer need them.

## Create the application from scratch

If you haven't already downloaded [the completed application](http://code.msdn.microsoft.com/todo), do that now. Instead of creating new files and copying and pasting code into them, you'll copy files from the downloaded project into the new project.

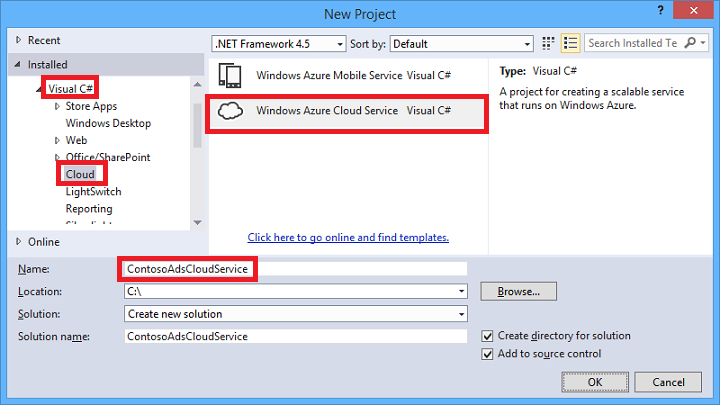
Creating the Contoso Ads application involves the following steps:

* Create a cloud service Visual Studio solution
* Update and add NuGet packages
* Set project references
* Configure connection strings
* Add code files

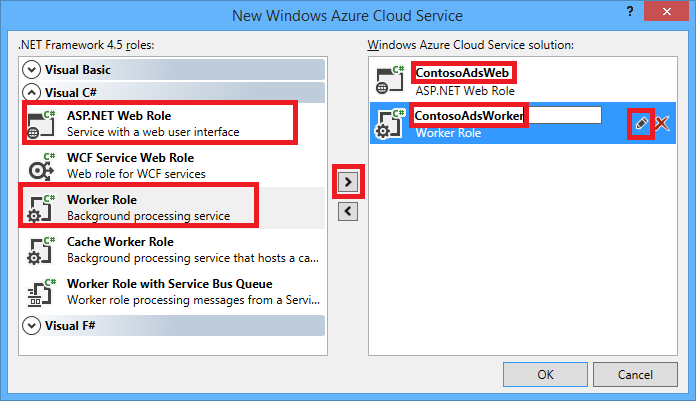
After the solution is created, you'll review the code and settings that are unique to cloud service projects and Azure blobs and queues.

### Create a cloud service Visual Studio solution

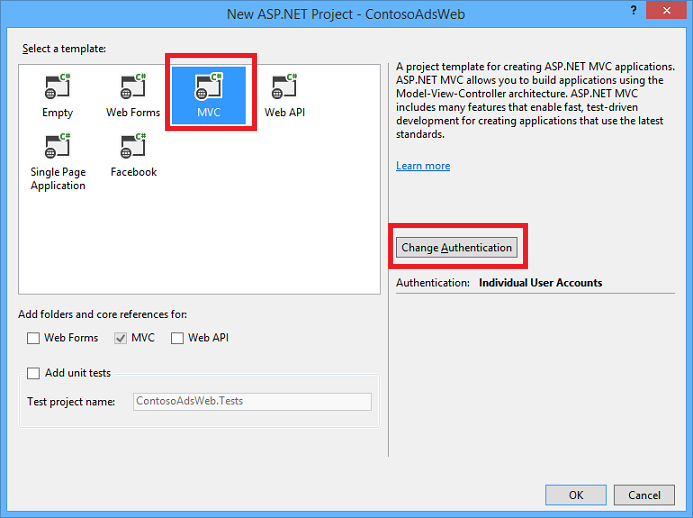
1. In Visual Studio, choose **New Project** from the **File** menu.
2. In the left pane of the **New Project** dialog, expand **Visual C#** and choose **Cloud** templates, and then choose the **Windows Azure Cloud Service** template.
3. Name the project and solution ContosoAdsCloudService, and then click **OK**.



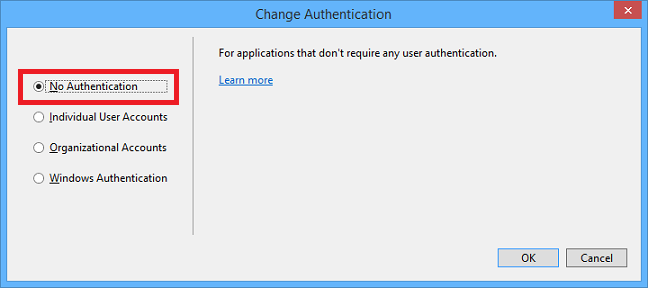
1. In the **New Azure Cloud Service** dialog, add a web role and a worker role. Name the web role ContosoAdsWeb, and name the worker role ContosoAdsWorker. (Use the pencil icon in the right-hand pane to change the default names of the roles.)



1. When you see the **New ASP.NET Project** dialog for the web role, choose the MVC template, and click **Change Authentication**.



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

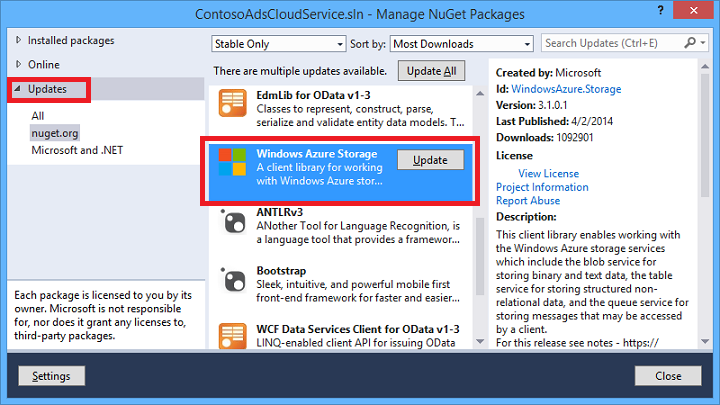


1. In the **New ASP.NET Project** dialog, click **OK**.
2. In **Solution Explorer**, right-click the solution (not one of the projects), and choose **Add - New Project**.
3. In the **Add New Project** dialog, choose **Windows Desktop** under **Visual C#** in the left pane, and then click the **Class Library** template.
4. Name the project ContosoAdsCommon, and then click **OK**.

You need to reference the Entity Framework context and the data model from both web and worker role projects. As an alternative you could define the EF-related classes in the web role project and reference that project from the worker role project. But in that , but then your worker role project would have a reference to web assemblies which it doesn't need.

### Update and add NuGet packages

1. Open the **Manage NuGet Packages** dialog for the solution.
2. In the left pane, select **Updates**.
3. Look for the Azure Storage package, and if it's in the list, click **Update** to get the latest version of the storage client library.



The storage client library is updated more frequently than Visual Studio project templates, so you'll often find that the version in a newly created projected needs to be updated.

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

### Set project references

1. In the ContosoAdsWeb project, set a reference to the ContosoAdsCommon project. Right-click the ContosoAdsWeb project, and then click **References** - **Add References**. In the **Reference Manager** dialog box, select **Solution – projects** in the left pane, select ContosoAdsCommon, and then click **OK**.
2. In the ContosoAdsWorker project, set a reference to the ContosAdsCommon project.

ContosoAdsCommon will contain the Entity Framework data model and context class, which will be used by both the front-end and back-end.

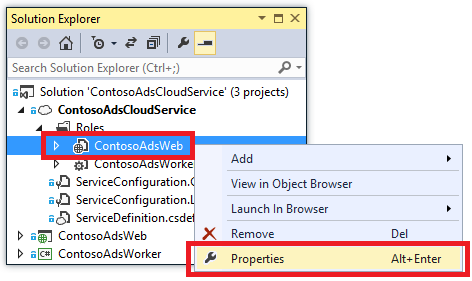
1. In the ContosoAdsWorker project, set a reference to System.Drawing.

This assembly is used by the back-end to convert images to thumbnails.

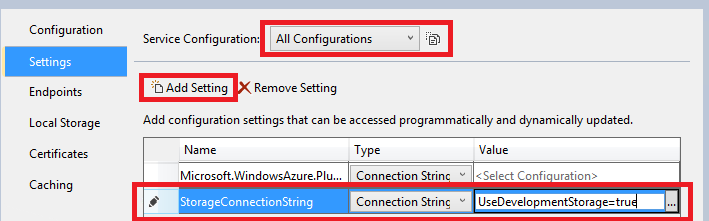
### Configure connection strings

In this section you configure Azure Storage and SQL connection strings for testing locally. The deployment instructions earlier in the tutorial explain how to set up the connection strings for when the app runs in the cloud.

1. In the ContosoAdsWeb project, open the application Web.config file, and insert the following connectionStrings element after the configSections element and before the system.diagnostics element:
2. <connectionStrings>
3. <add name="ContosoAdsContext" connectionString="Data Source=(localdb)\v11.0; Initial Catalog=ContosoAds; Integrated Security=True; MultipleActiveResultSets=True;" providerName="System.Data.SqlClient" />
4. </connectionStrings>
5. Save your changes.
6. In the ContosoAdsCloudService project, right-click ContosoAdsWeb under **Roles**, and then click **Properties**.



1. In the **ContosAdsWeb [Role]** properties window, click the **Settings** tab, and then click **Add Setting**.
2. Leave **Service Configuration** set to **All Configurations**.
3. Add a new setting named StorageConnectionString. Set **Type** to ConnectionString, and set **Value** to UseDevelopmentStorage=true.



1. Save your changes.
2. In the ContosoAdsWorker project, follow the same procedure to add a storage connection string that you followed for the ContosoAdsWeb project.
3. Still in the **ContosoAdsWorker [Role]** properties window, add another connection string:
   * Name: ContosoAdsDbConnectionString
   * Type: String
   * Value: Paste the same connection string you used for the web role project:
   * ContosoAdsContext" connectionString="Data Source=(localdb)\v11.0; Initial Catalog=ContosoAds; Integrated Security=True; MultipleActiveResultSets=True;

### Add code files

In this section you copy code files from the downloaded solution into the new solution. The following sections will show and explain key parts of this code.

To add files to a project or a folder, right-click the project or folder and click **Add** - **Existing Item**. Select the files you want and click **Add**. If asked whether you want to replace existing files, click **Yes**.

1. In the ContosoAdsCommon project, delete the Class1.cs file and add in its place the Add.cs and ContosoAdscontext.cs files from the downloaded project.
2. In the ContosoAdsWeb project, add the following files from the downloaded project.
   * Global.asax.cs
   * \_Layout.cshtml file in the Views\Shared folder
   * Index.cshtml in the Views\Home folder
   * AdController.cs in the Controllers folder
   * Five .cshtml files in the Views\Ad folder
3. In the ContosoAdsWorker project, add WorkerRole.cs from the downloaded project.

You can now build and run the application as instructed earlier in the tutorial, and the app will use local database and storage emulator resources.

The following sections explain the code related to working with the Azure environment, blobs, and queues. For help understanding the code related to working with Entity Framework Code First, see an [introductory MVC tutorial](http://www.asp.net/mvc/tutorials/mvc-5/introduction/getting-started) or [introductory MVC with EF tutorial](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc). For help understanding the asynchronous code (async and await keywords and Task class), see [an introduction to asynchronous programming in .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).

### ContosoAdsCommon - Ad.cs

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

public enum Category

{

Cars,

[Display(Name="Real Estate")]

RealEstate,

[Display(Name = "Free Stuff")]

FreeStuff

}

public class Ad

{

public int AdId { get; set; }

[StringLength(100)]

public string Title { get; set; }

public int Price { get; set; }

[StringLength(1000)]

[DataType(DataType.MultilineText)]

public string Description { get; set; }

[StringLength(1000)]

[DisplayName("Full-size Image")]

public string ImageURL { get; set; }

[StringLength(1000)]

[DisplayName("Thumbnail")]

public string ThumbnailURL { get; set; }

[DataType(DataType.Date)]

[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]

public DateTime PostedDate { get; set; }

public Category? Category { get; set; }

[StringLength(12)]

public string Phone { get; set; }

}

### 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 database.

public class ContosoAdsContext : DbContext

{

public ContosoAdsContext() : base("name=ContosoAdsContext")

{

}

public ContosoAdsContext(string connString)

: base(connString)

{

}

public System.Data.Entity.DbSet<Ad> Ads { get; set; }

}

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. The second constructor enables you to pass in the actual connection string. That is needed by the worker role 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.

### 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, or start using the storage emulator on a new computer, 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 .cscfg file.

var storageAccount = CloudStorageAccount.Parse

(RoleEnvironment.GetConfigurationSettingValue("StorageConnectionString"));

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 only allow clients with storage account credentials to access blobs. The web site needs the blobs to be public so that it can display images using URLs that point to the image blobs.

var blobClient = storageAccount.CreateCloudBlobClient();

var imagesBlobContainer = blobClient.GetContainerReference("images");

if (imagesBlobContainer.CreateIfNotExists())

{

imagesBlobContainer.SetPermissions(

new BlobContainerPermissions

{

PublicAccess =BlobContainerPublicAccessType.Blob

});

}

Similar code gets a reference to the images queue and creates a new queue. In this case no permissions change is needed.

CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();

var imagesQueue = queueClient.GetQueueReference("images");

imagesQueue.CreateIfNotExists();

### 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 3 tries.

var blobClient = storageAccount.CreateCloudBlobClient();

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

imagesQueue = queueClient.GetQueueReference("images");

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/en-us/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 that back-end process that an image is ready for conversion to a thumbnail.

string queueMessageString = ad.AdId.ToString();

var queueMessage = new CloudQueueMessage(queueMessageString);

queue.AddMessage(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 must be deleted.

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

{

await DeleteAdBlobsAsync(ad);

imageBlob = await UploadAndSaveBlobAsync(imageFile);

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

}

Here is the DeleteAdBlobsAsync method:

private async Task DeleteAdBlobsAsync(Ad ad)

{

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

{

CloudBlockBlob blobToDelete = imagesBlobContainer.GetBlockBlobReference(ad.ImageURL);

await blobToDelete.DeleteAsync();

}

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

{

CloudBlockBlob blobToDelete = imagesBlobContainer.GetBlockBlobReference(ad.ThumbnailURL);

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" />

### ContosoAdsWorker - WorkerRole.cs - OnStart method

The Azure worker role environment calls the OnStart method in the WorkerRole class when the worker role is getting started, and it calls the Run method when the OnStart method finishes.

The OnStart method gets the database connection string from the .cscfg file and passes it to the Entity Framework DbContext class. The SQLClient provider is used by default, so the provider does not have to be specified.

var dbConnString = CloudConfigurationManager.GetSetting("ContosoAdsDbConnectionString");

db = new ContosoAdsContext(dbConnString);

After that the method gets a reference to the storage account and creates the blob container and queue if they don't exist. The code for that is similar to what you already saw in the web role Application\_Start method.

### ContosoAdsWorker - WorkerRole.cs - Run method

The Run method is called when the OnStart method finishes its initialization work. The method executes an infinite loop that watches for new queue messages and processes them when they arrive.

public override void Run()

{

CloudQueueMessage msg = null;

while (true)

{

try

{

msg = this.imagesQueue.GetMessage();

if (msg != null)

{

ProcessQueueMessage(msg);

}

else

{

System.Threading.Thread.Sleep(1000);

}

}

catch (StorageException e)

{

if (msg != null && msg.DequeueCount > 5)

{

this.imagesQueue.DeleteMessage(msg);

}

System.Threading.Thread.Sleep(5000);

}

}

}

After each iteration of the loop, if no queue message was found, the program sleeps for a second. This prevents the worker role from incurring excessive CPU time and storage transaction costs. The Microsoft Customer Advisory Team tells a story about a developer who forgot to include this, deployed to production, and left for vacation. When he got back, his oversight cost more than the vacation.

Sometimes the content of a queue message causes an error in processing. This is called a poison message, and if you just logged an error and restarted the loop, you could endlessly try to process that message. Therefore the catch block includes an if statement that checks to see how many times the app has tried to process the current message, and if it has been more than 5 times, the message is deleted from the queue.

ProcessQueueMessage is called when a queue message is found.

private void ProcessQueueMessage(CloudQueueMessage msg)

{

var adId = int.Parse(msg.AsString);

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

if (ad == null)

{

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

}

CloudBlockBlob inputBlob = this.imagesBlobContainer.GetBlockBlobReference(ad.ImageURL);

string thumbnailName = Path.GetFileNameWithoutExtension(inputBlob.Name) + "thumb.jpg";

CloudBlockBlob outputBlob = this.imagesBlobContainer.GetBlockBlobReference(thumbnailName);

using (Stream input = inputBlob.OpenRead())

using (Stream output = outputBlob.OpenWrite())

{

ConvertImageToThumbnailJPG(input, output);

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

}

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

db.SaveChanges();

this.imagesQueue.DeleteMessage(msg);

}

This code reads the database to get the image URL, converts the image to a thumbnail, saves the thumbnail in a blob, updates the database with the thumbnail blob URL, and deletes the queue message.

[WACOM.NOTE] The code in the ConvertImageToThumbnailJPG method uses classes in the System.Drawing namespace for simplicity. However, the classes in this namespace were designed for use with Windows Forms. They are not supported for use in a Windows or ASP.NET service.

## Troubleshooting

In case something doesn't work while you're following the instructions in this tutorial, here are some common errors and how to resolve them.

### ServiceRuntime.RoleEnvironmentException

The RoleEnvironment object is provided by Azure when you run an application in Azure or when you run locally using the Azure compute emulator. If you get this error when you're running locally, make sure that you have set the ContosoAdsCloudService project as the startup project. This sets up the project to run using the Azure compute emulator.

One of the things the application uses the Azure RoleEnvironment for is to get the connection string values that are stored in the .cscfg files, so another cause of this exception is a missing connection string. Make sure that you created the StorageConnectionString setting for both Cloud and Local configurations in the ContosoAdsWeb project, and that you created both connection strings for both configurations in the ContosoAdsWorker project. If you do a **Find All** search for StorageConnectionString in the entire solution, you should see it 9 times in 6 files.

### Cannot override to port xxx. New port below minimum allowed value 8080 for protocol http

Try changing the port number used by the web project. Right-click the ContosoAdsWeb project, and then click **Properties**. Click the **Web** tab, and then change the port number in the **Project Url** setting.

For another alternative that might resolve the problem, see the following section.

### Other errors when running locally

By default new Cloud Service projects use the Azure compute emulator express to simulate the Azure environment. This is a lightweight version of the full compute emulator, and under some conditions the full emulator will work when the express version does not.

To change the project to use the full emulator, right-click the ContosoAdsCloudService project, and then click **Properties**. In the **Properties** window click the **Web** tab, and then click the **Use Full Emulator** radio button.

In order to run the application with the full emulator, you have to open Visual Studio with administrator privileges.

## Next steps

The Contoso Ads application has intentionally been kept simple for a getting-started tutorial. For example, it doesn't implement [dependency injection](http://www.asp.net/mvc/tutorials/hands-on-labs/aspnet-mvc-4-dependency-injection) or the [repository and unit of work patterns](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/advanced-entity-framework-scenarios-for-an-mvc-web-application#repo), it doesn't [use an interface for logging](http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/monitoring-and-telemetry#log), it doesn't use [EF Code First Migrations](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/migrations-and-deployment-with-the-entity-framework-in-an-asp-net-mvc-application) to manage data model changes or [EF Connection Resiliency](http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/connection-resiliency-and-command-interception-with-the-entity-framework-in-an-asp-net-mvc-application) to manage transient network errors, and so forth.

Here are some cloud service sample applications that demonstrate more real-world coding practices, listed from less complex to more complex:

* [PhluffyFotos](http://code.msdn.microsoft.com/PhluffyFotos-Sample-7ecffd31). Similar in concept to Contoso Ads but implements more features and more real-world coding practices.
* [Azure Cloud Service Multi-Tier Application with Tables, Queues, and Blobs](http://code.msdn.microsoft.com/windowsazure/Windows-Azure-Multi-Tier-eadceb36). Introduces Azure Storage Tables as well as blobs and queues, and comes with a [step-by-step tutorial series](http://azure.microsoft.com/en-us/documentation/articles/cloud-services-dotnet-multi-tier-app-storage-1-overview/).
* [Cloud Service Fundamentals in Windows Azure](http://code.msdn.microsoft.com/Cloud-Service-Fundamentals-4ca72649). A comprehensive sample demonstrating a wide range of best practices, produced by the Microsoft Patterns and Practices group.

For general information about developing for the cloud, see [Building Real-World Cloud Apps with Windows Azure](http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/introduction).

For a video introduction to Azure Storage best practices and patterns, see [Microsoft Azure Storage – What's New, Best Practices and Patterns](http://channel9.msdn.com/Events/Build/2014/3-628).

For more information, see the following resources:

* [Azure Cloud Services](file:///C:\en-us\documentation\services\cloud-services\)
* [Azure Storage](file:///C:\en-us\documentation\services\storage\)
* [How to use Blob Storage from .NET](http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-blobs/)
* [How to use Queue Storage from .NET](http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-queues/)