

Create a .NET WebJob in Azure App Service

TAGS: [APP SERVICE \(/EN-US/DOCUMENTATION/ARTICLES/?SERVICE=APP-SERVICE\)](/EN-US/DOCUMENTATION/ARTICLES/?SERVICE=APP-SERVICE) ,
[.NET \(/EN-US/DOCUMENTATION/ARTICLES/?PLATFORM=.NET\)](/EN-US/DOCUMENTATION/ARTICLES/?PLATFORM=.NET)

[By Tom Dykstra \(https://github.com/tdykstra\)](https://github.com/tdykstra)
Last updated: 10/22/2015

In this article:

[Prerequisites](#)

[What you'll learn](#)

[Application architecture](#)

[Set up the development environment](#)

[Create an Azure Storage account](#)

[Download the application](#)

[Configure the application to use your storage account](#)

[Run the application locally](#)

[Run the application in the cloud](#)

[Create the application from scratch](#)

[Review the application code](#)

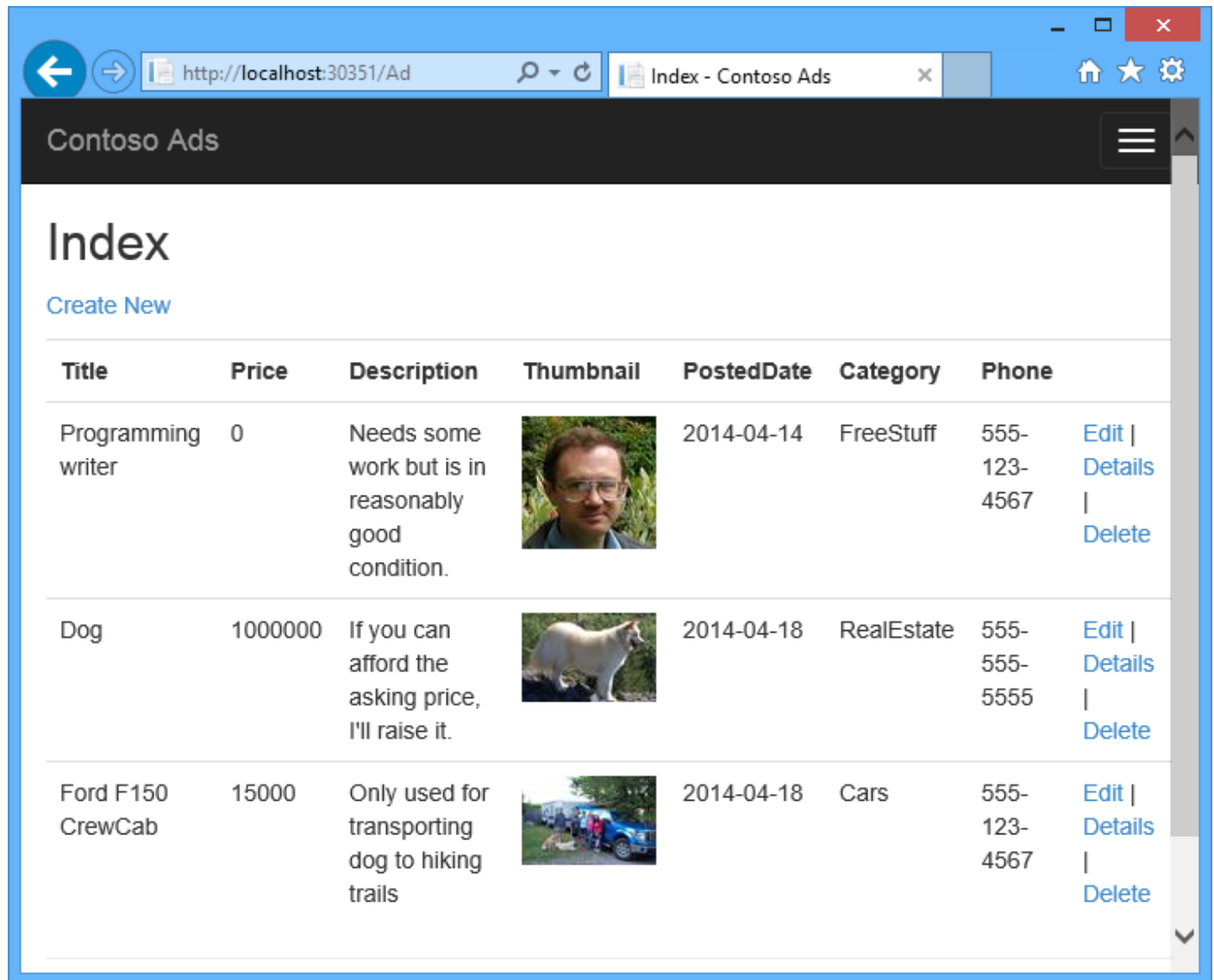
[Next steps](#)

[46 Comments](#)

This tutorial shows how to write code for a simple multi-tier ASP.NET MVC 5 application that uses the [WebJobs SDK \(../websites-dotnet-webjobs-sdk/\)](#) to work with [Azure queues \(http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/queue-centric-work-pattern\)](#) and [Azure blobs \(http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/unstructured-blob-storage\)](#). The tutorial shows how

to deploy the application to [Azure App Service \(http://go.microsoft.com/fwlink/?linkid=529714&clcid=0x409\)](http://go.microsoft.com/fwlink/?linkid=529714&clcid=0x409) and [Azure SQL Database \(http://msdn.microsoft.com/library/azure/ee336279\)](http://msdn.microsoft.com/library/azure/ee336279).

The sample application is an advertising bulletin board. Users can upload images for ads, and a backend process converts the images to thumbnails. The ad list page shows the thumbnails, and the ad details page shows the full size image. Here's a screenshot:



Prerequisites

The tutorial assumes that you know how to work with [ASP.NET MVC 5 \(http://www.asp.net/mvc/tutorials/mvc-5/introduction/getting-started\)](http://www.asp.net/mvc/tutorials/mvc-5/introduction/getting-started) projects in Visual Studio.

The tutorial was written for Visual Studio 2013. If you don't have Visual Studio already, it will be installed for you automatically when you install the Azure SDK for .NET.

The tutorial can be used with Visual Studio 2015, but before you run the application locally you have to change the `Data Source` part of the SQL Server LocalDB connection string in the `Web.config` and `App.config` files from `Data Source=(localdb)\v11.0` to `Data Source=(LocalDb)\MSSQLLocalDB`.

NOTE:

You need an Azure account to complete this tutorial:

- You can [open an Azure account for free \(/pricing/free-trial/?WT.mc_id=A261C142F\)](/pricing/free-trial/?WT.mc_id=A261C142F): You get credits you can use to try out paid Azure services, and even after they're used up you can keep the account and use free Azure services, such as Websites. Your credit card will never be charged, unless you explicitly change your settings and ask to be charged.
- You can [activate MSDN subscriber benefits \(/pricing/member-offers/msdn-benefits-details/?WT.mc_id=A261C142F\)](/pricing/member-offers/msdn-benefits-details/?WT.mc_id=A261C142F): Your MSDN subscription gives you credits every month that you can use for paid Azure services.

If you want to get started with Azure App Service before signing up for an Azure account, go to [Try App Service \(http://go.microsoft.com/fwlink/?linkid=523751&clcid=0x409\)](http://go.microsoft.com/fwlink/?linkid=523751&clcid=0x409), where you can immediately create a short-lived starter web app in App Service. No credit cards required; no commitments.

What you'll learn

The tutorial shows how to do the following tasks:

- Enable your machine for Azure development by installing the Azure SDK.
- Create a Console Application project that automatically deploys as an Azure WebJob when you deploy the associated web project.
- Test a WebJobs SDK backend locally on the development computer.
- Publish an application with a WebJobs backend 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.

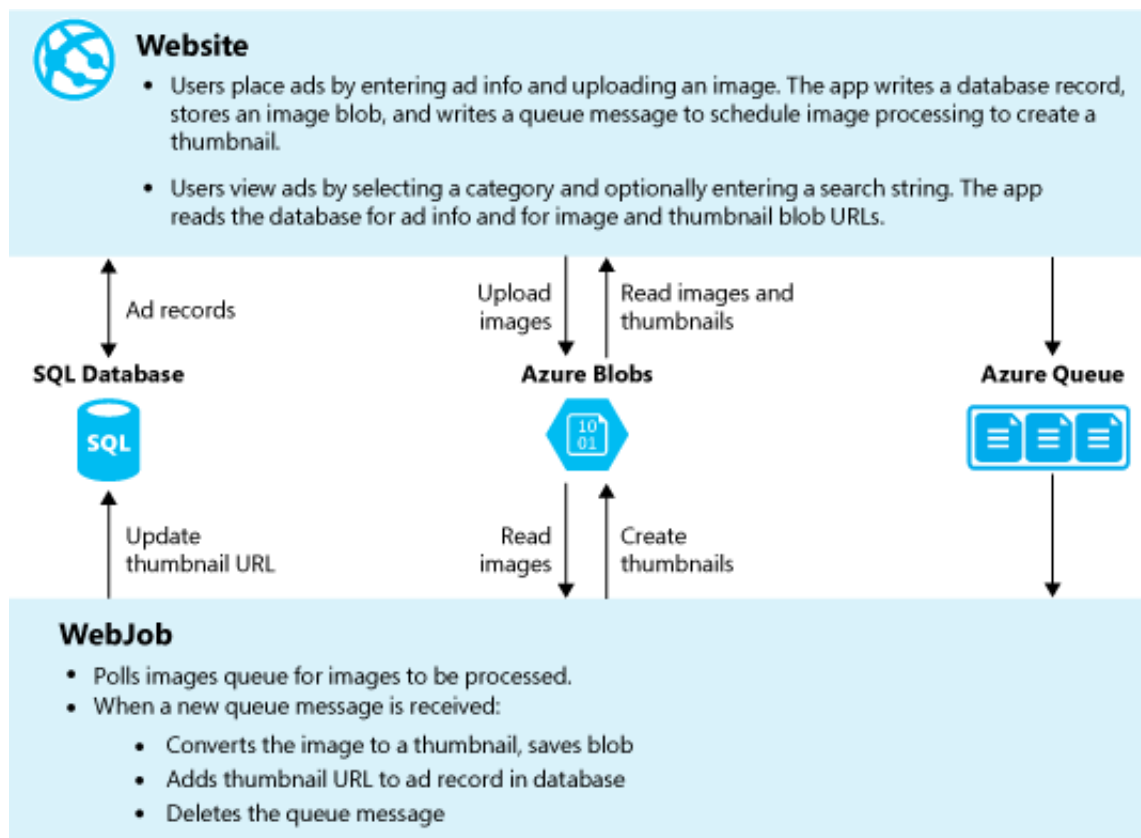
Application architecture

The sample application uses the [queue-centric work pattern \(http://www.asp.net/aspnet/overview/developing-apps-with-windows-azure/building-real-world-cloud-apps-with-windows-azure/queue-centric-work-pattern\)](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 backend process.

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.

dbo.Ads [Design] ✕			
Update		Script File: dbo.Ads.sql	
	Name	Data Type	Allow Nulls
	AddId	int	<input type="checkbox"/>
	Title	nvarchar(100)	<input checked="" type="checkbox"/>
	Price	int	<input type="checkbox"/>
	Description	nvarchar(1000)	<input checked="" type="checkbox"/>
	ImageURL	nvarchar(1000)	<input checked="" type="checkbox"/>
	ThumbnailURL	nvarchar(1000)	<input checked="" type="checkbox"/>
	PostedDate	datetime	<input type="checkbox"/>
	Category	int	<input checked="" type="checkbox"/>
	Phone	nvarchar(12)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

When a user uploads an image, 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) (<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. In a backend process running as an Azure WebJob, the WebJobs SDK polls 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's a diagram that shows how the parts of the application interact:



Set up the development environment

To start, set up your development environment by installing the [Azure SDK for Visual Studio 2015](http://go.microsoft.com/fwlink/?linkid=518003&clcid=0x409) (<http://go.microsoft.com/fwlink/?linkid=518003&clcid=0x409>) or the [Azure SDK for Visual Studio 2013](http://go.microsoft.com/fwlink/?linkid=324322&clcid=0x409) (<http://go.microsoft.com/fwlink/?linkid=324322&clcid=0x409>).

If you don't have Visual Studio installed, use the link for Visual Studio 2015, and Visual Studio will be installed along with the SDK.

NOTE:

Depending on how many of the SDK dependencies you already have on your machine, installing the SDK could take a long time, from several minutes to a half hour or more.

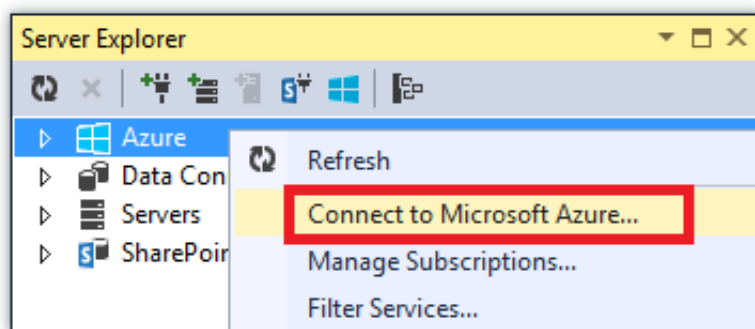
The tutorial instructions apply to Azure SDK for .NET 2.7.1 or later.

Create an Azure Storage account

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

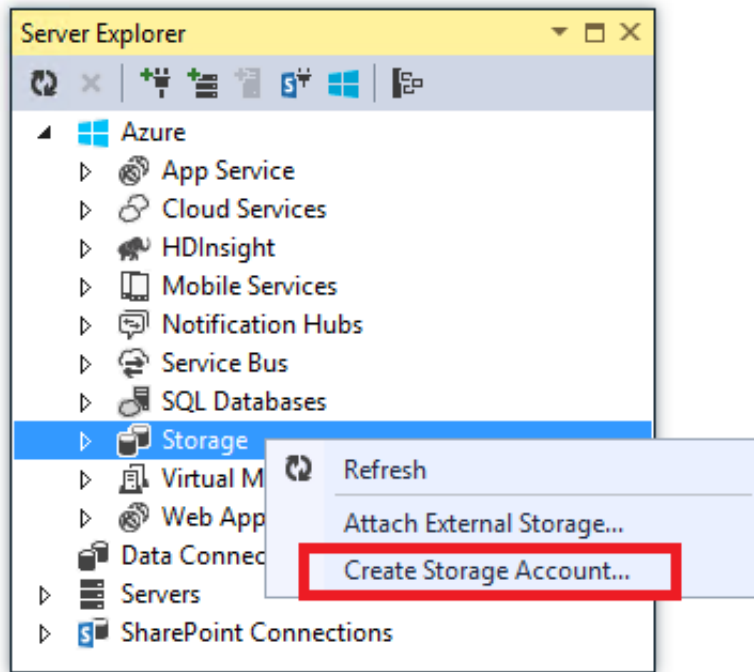
In a real-world application, you typically create separate accounts for application data versus logging data, and separate accounts for test data versus production data. For this tutorial you'll use just one account.

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



3. Sign in using your Azure credentials.

4. Right-click **Storage** under the Azure node, and then click **Create Storage Account**.



5. In the **Create Storage Account** dialog, enter a name for the storage account.

The name must be unique (no other Azure storage account can have the same name). If the name you enter is already in use you'll get a chance to change it.

The URL to access your storage account will be `{name}.core.windows.net`.

6. Set the **Region or Affinity Group** drop-down list to 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.

7. 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 [Create, manage, or delete a storage account \(../storage-create-storage-account/#replication-options\)](#).

8. Click **Create**.

Create Storage Account

Signed in as:

Subscription:
Windows Azure MSDN - Visual Studio Ultimate

Name:
contosoads

Region or Affinity Group:
West US

Replication:
Locally Redundant

[Read more about replication services and pricing details.](#)

Create Close

Download the application

1. Download and unzip the [completed solution \(http://code.msdn.microsoft.com/Simple-Azure-Website-with-b4391eeb\)](http://code.msdn.microsoft.com/Simple-Azure-Website-with-b4391eeb).
2. Start Visual Studio.
3. From the **File** menu choose **Open > Project/Solution**, 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.

5. In **Solution Explorer**, make sure that **ContosoAdsWeb** is selected as the startup project.

Configure the application to use your storage account

1. Open the application *Web.config* file in the ContosoAdsWeb project.

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

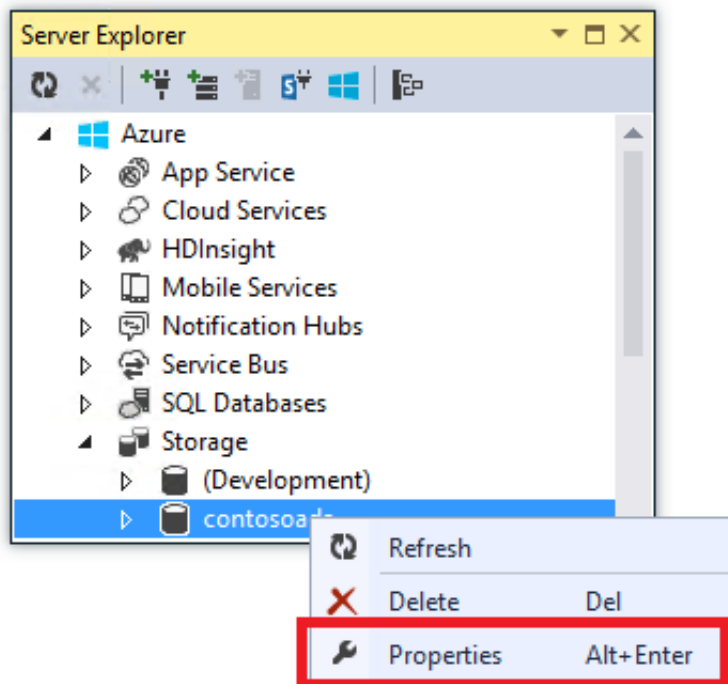
The SQL connection string points to a [SQL Server Express LocalDB \(http://msdn.microsoft.com/library/hh510202.aspx\)](http://msdn.microsoft.com/library/hh510202.aspx) database.

The storage connection string is an example that has placeholders for the storage account name and access key. You'll replace this with a connection string that has the name and key of your storage account.

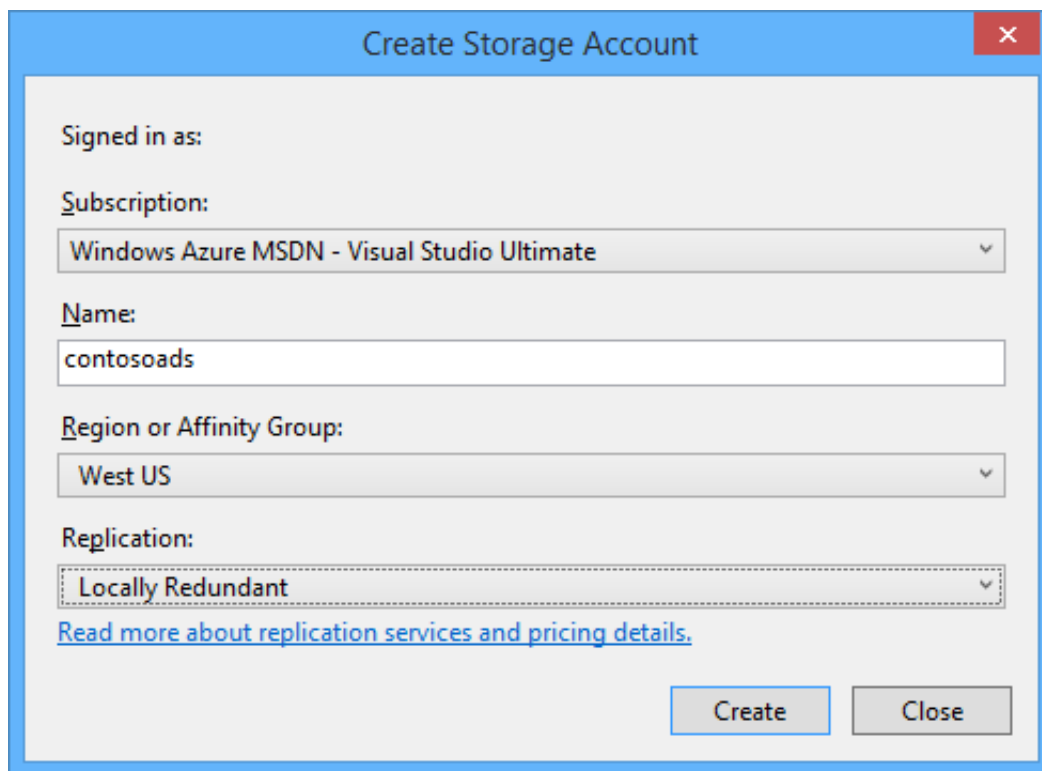
```
<connectionStrings>
  <add name="ContosoAdsContext" connectionString="Data Source=(localdb)\v11.0; Initial
Catalog=ContosoAds; Integrated Security=True; MultipleActiveResultSets=True;"
providerName="System.Data.SqlClient" />
  <add name="AzureWebJobsStorage"
connectionString="DefaultEndpointsProtocol=https;AccountName=[accountname];AccountKey=
[accesskey]"/>
</connectionStrings>
```

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

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

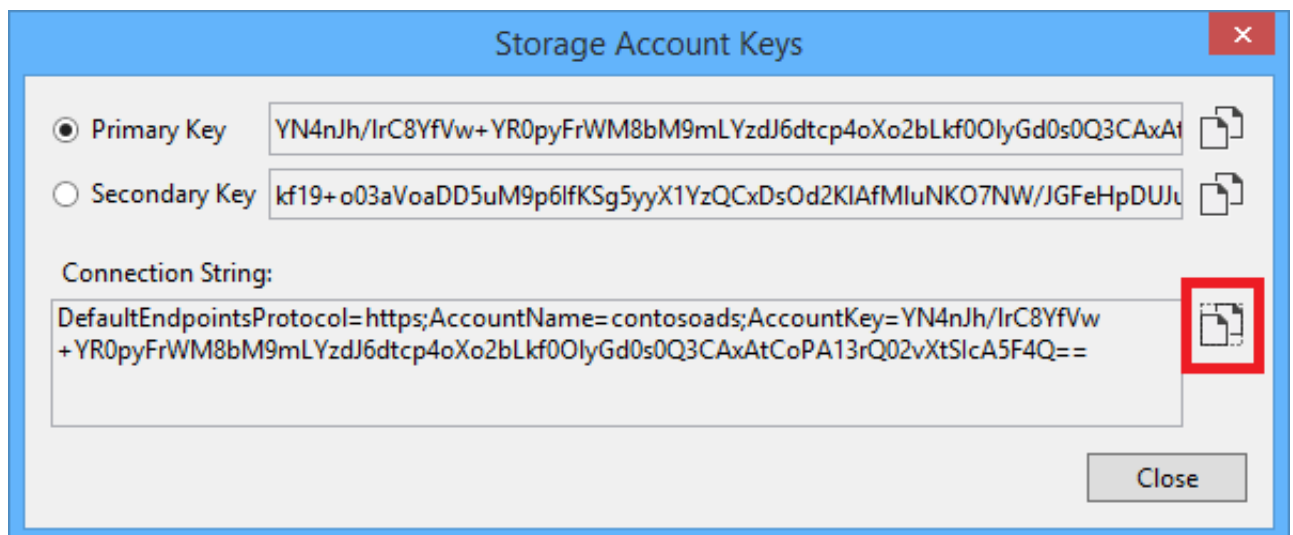


3. In the **Properties** window, click **Storage Account Keys**, and then click the ellipsis.



The 'Create Storage Account' dialog box is shown. It has a blue title bar with a close button. The content area is light gray. It contains the following fields: 'Signed in as:' (empty), 'Subscription:' (dropdown menu showing 'Windows Azure MSDN - Visual Studio Ultimate'), 'Name:' (text box with 'contosoads'), 'Region or Affinity Group:' (dropdown menu showing 'West US'), and 'Replication:' (dropdown menu showing 'Locally Redundant'). Below these is a blue link: 'Read more about replication services and pricing details.' At the bottom right are 'Create' and 'Close' buttons.

4. Copy the **Connection String**.



The 'Storage Account Keys' dialog box is shown. It has a blue title bar with a close button. The content area is light gray. It contains two radio buttons: 'Primary Key' (selected) and 'Secondary Key'. Next to each is a text box containing a long alphanumeric string. Below these is a 'Connection String:' label and a text box containing a long connection string. To the right of the connection string text box is a red square icon with a document symbol. At the bottom right is a 'Close' button.

5. 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 not including the quotation marks before pasting.
6. 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'll 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>
```

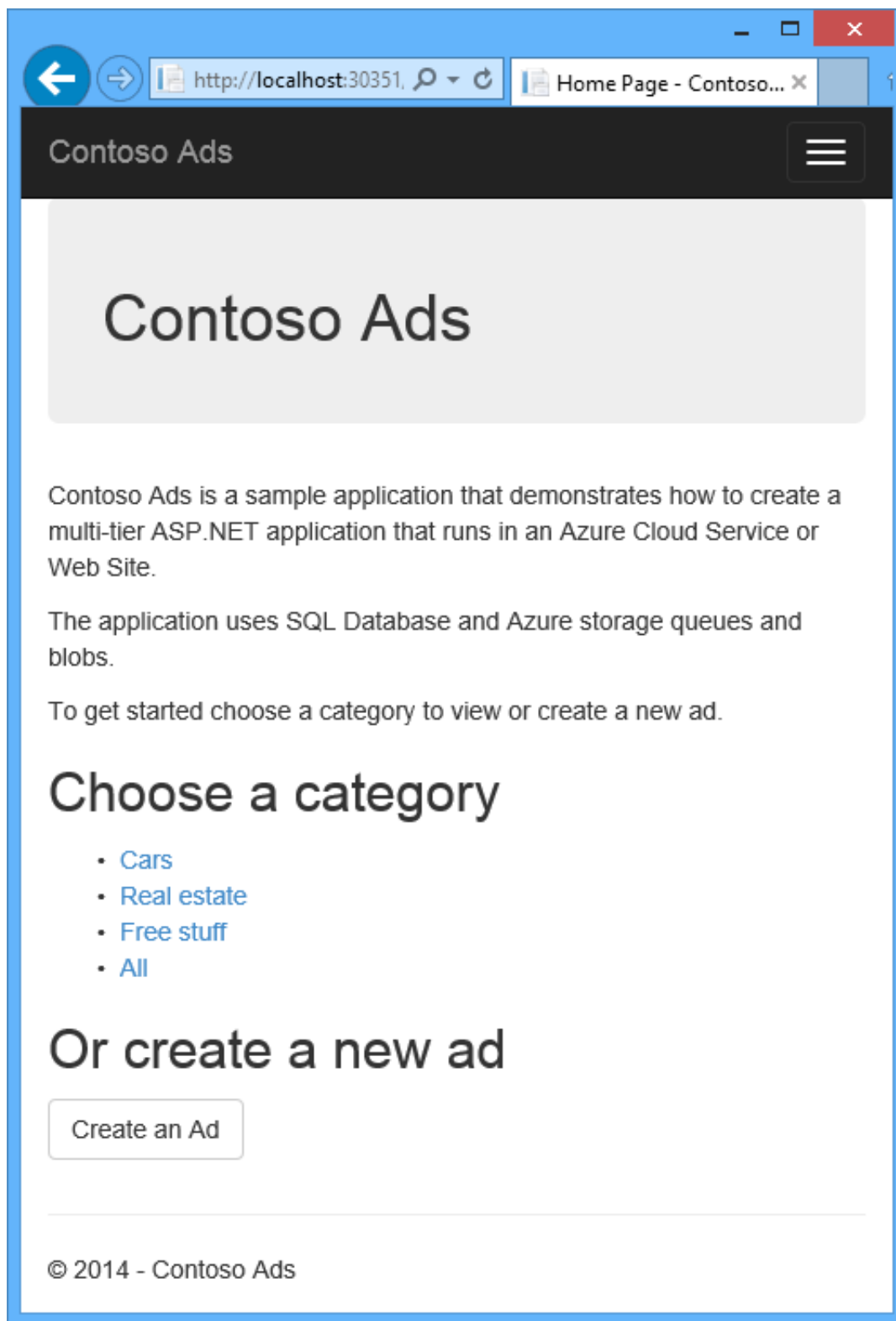
By default, the WebJobs SDK looks for connection strings named AzureWebJobsStorage and AzureWebJobsDashboard. As an alternative, you can [store the connection string however you want and pass it in explicitly to the `JobHost` object](#) ([../websites-dotnet-webjobs-sdk-storage-queues-how-to/#config](#)).

7. Replace both storage connection strings with the connection string you copied earlier.
8. Save your changes.

Run the application locally

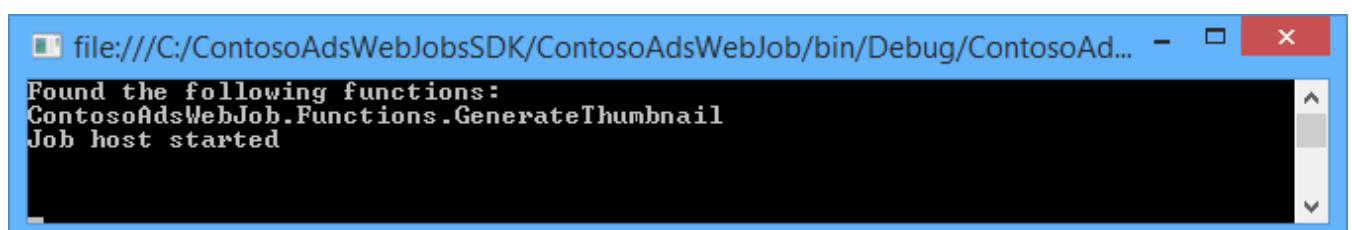
1. To start the web frontend of the application, press CTRL+F5.

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



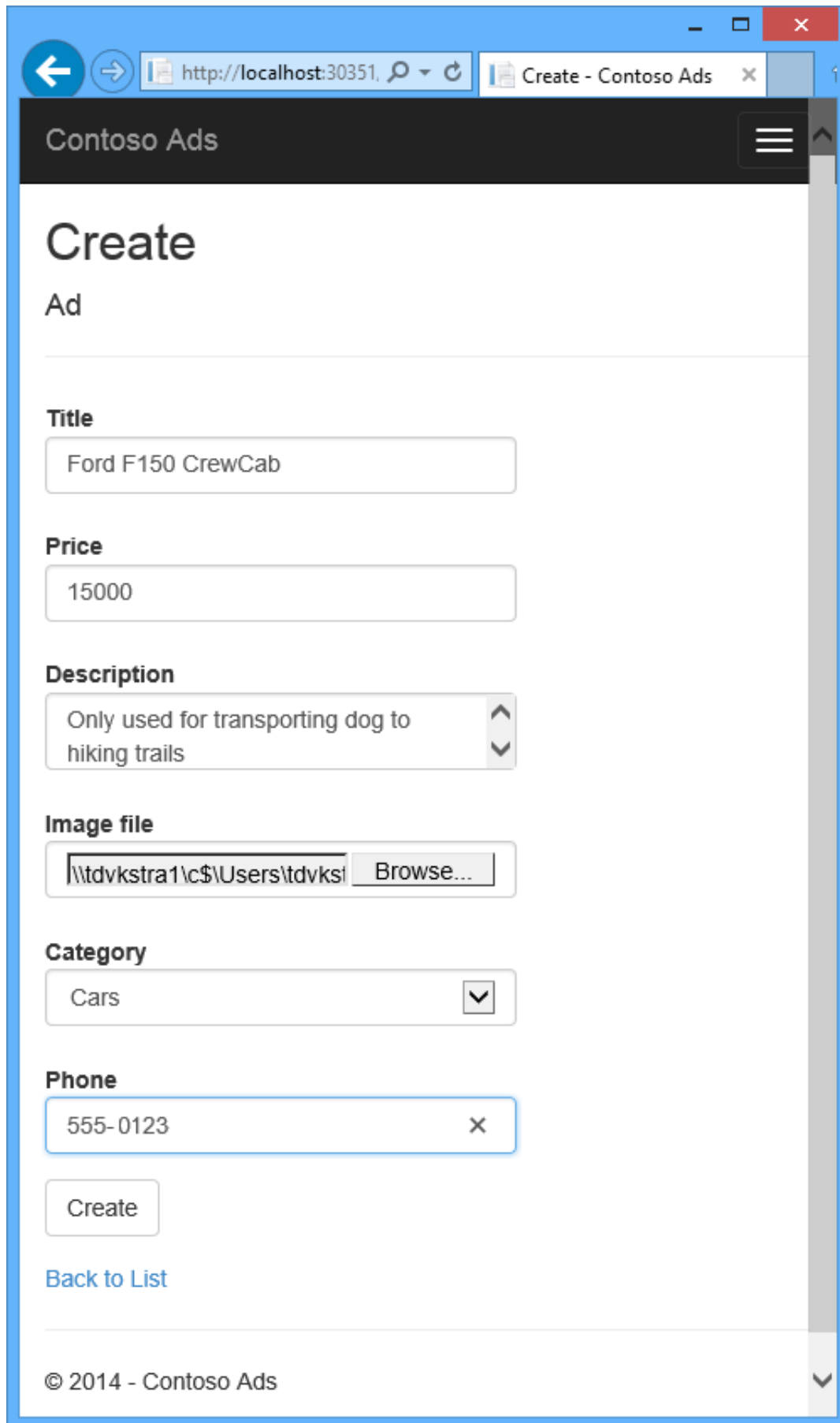
2. To start the WebJob backend of the application, right-click the ContosoAdsWebJob project in **Solution Explorer**, and then click **Debug > Start new instance**.

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



3. In your browser, click **Create an Ad**.

4. Enter some test data and select an image to upload, and then click **Create**.



The screenshot shows a web browser window with the address bar displaying `http://localhost:30351` and the page title "Create - Contoso Ads". The application header is "Contoso Ads" with a menu icon. The main content area is titled "Create Ad".

The form contains the following fields:

- Title:** A text input field containing "Ford F150 CrewCab".
- Price:** A text input field containing "15000".
- Description:** A text area containing "Only used for transporting dog to hiking trails".
- Image file:** A text input field containing a file path and a "Browse..." button.
- Category:** A dropdown menu with "Cars" selected.
- Phone:** A text input field containing "555-0123" and a clear button (X).

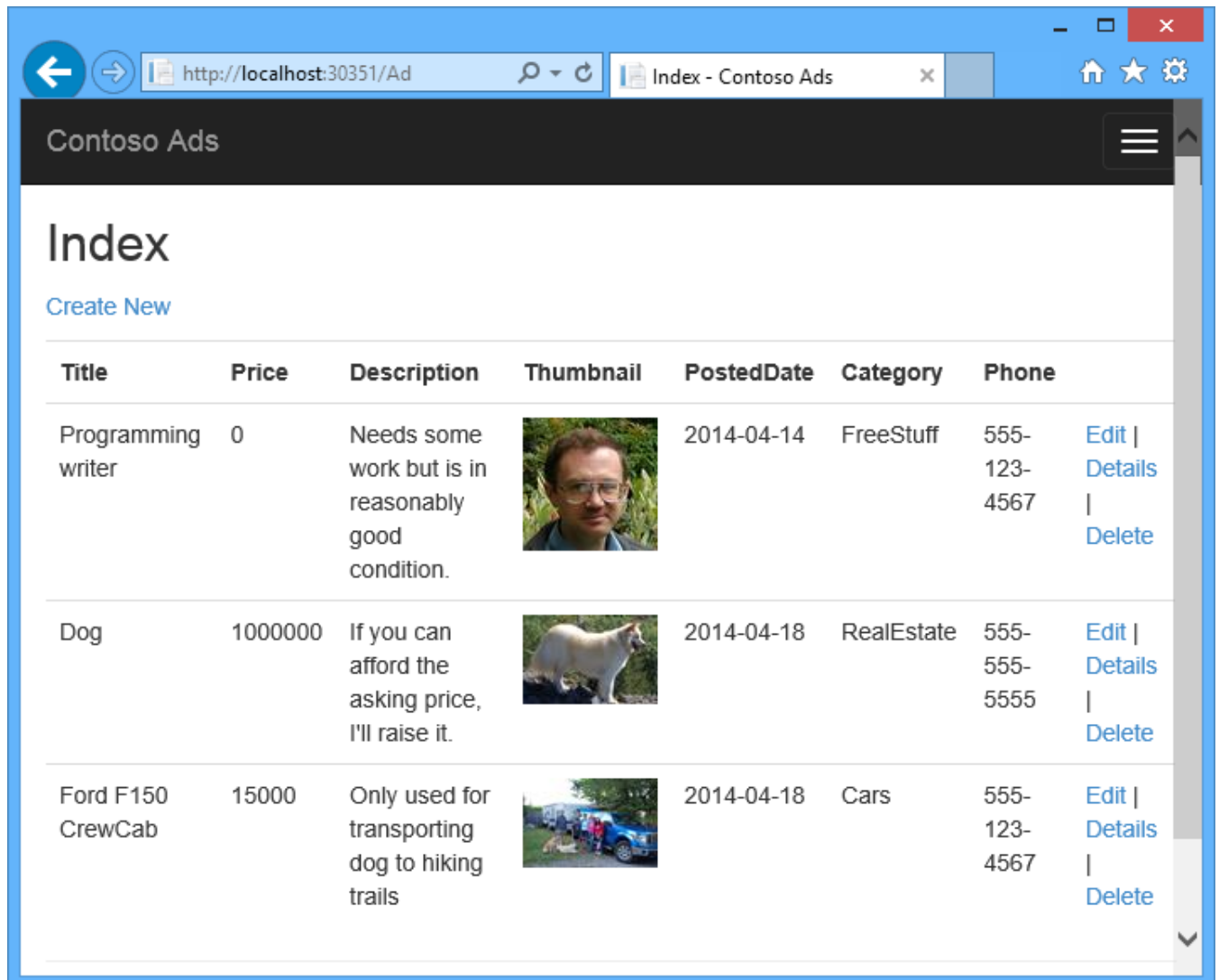
At the bottom of the form is a "Create" button and a link labeled "Back to List". The footer of the page reads "© 2014 - Contoso Ads".

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.

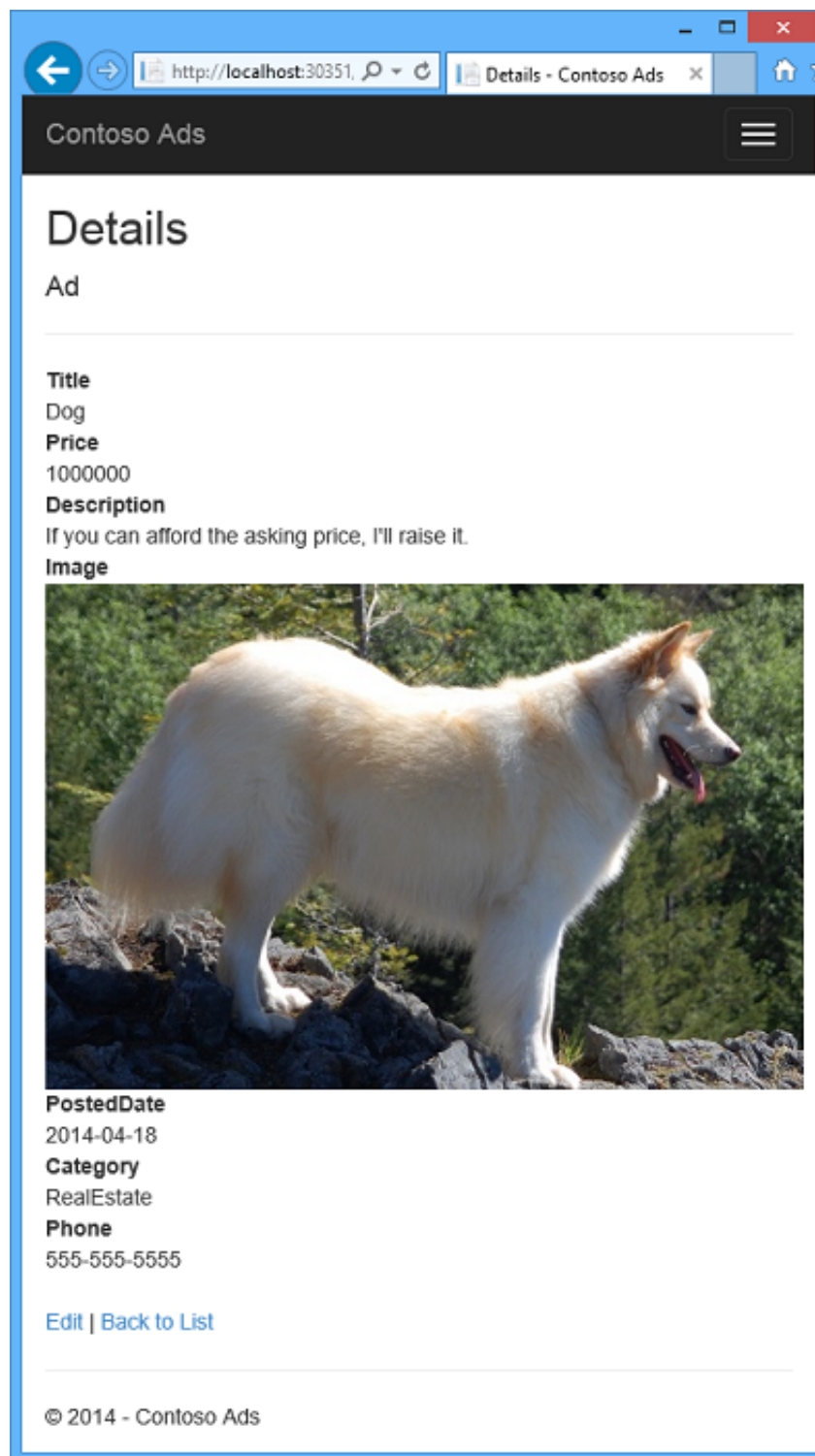
Meanwhile, after a short wait a logging message in the console application window shows that a queue message was received and has been processed.

```
file:///C:/ContosoAdsWebJobsSDK/ContosoAdsWebJob/bin/Debug/ContosoAd...
Found the following functions:
ContosoAdsWebJob.Functions.GenerateThumbnail
Job host started
Executing: 'Functions.GenerateThumbnail' because New queue message detected on '
thumbnailrequest'.
```

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



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



You've been running the application on your local computer, and it's using a SQL Server database located on your computer, but it's 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.

Run the application in the cloud

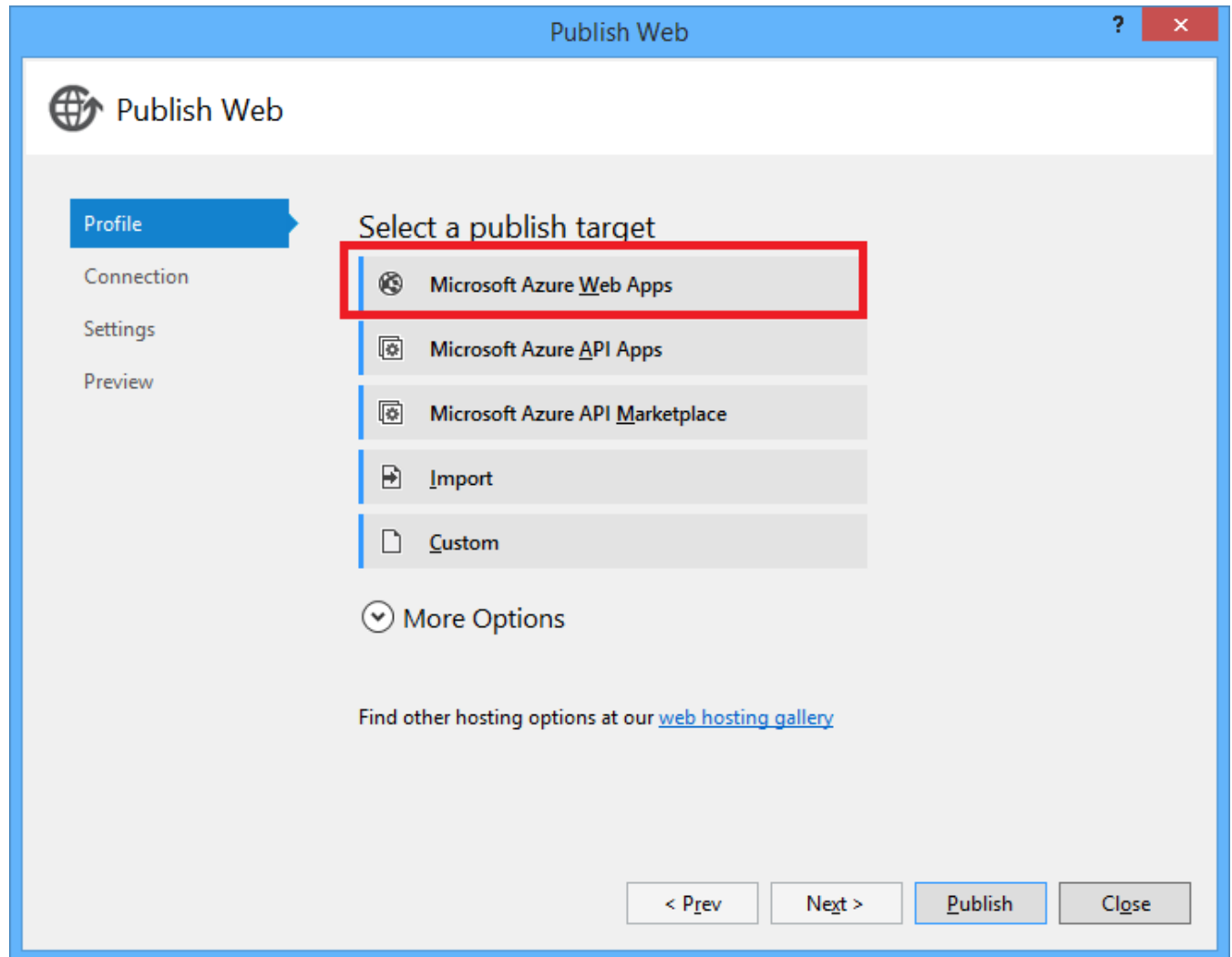
You'll do the following steps to run the application in the cloud:

- Deploy to Web Apps. Visual Studio automatically creates a new web app in App Service and a SQL Database instance.
- Configure the web app to use your Azure SQL 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.

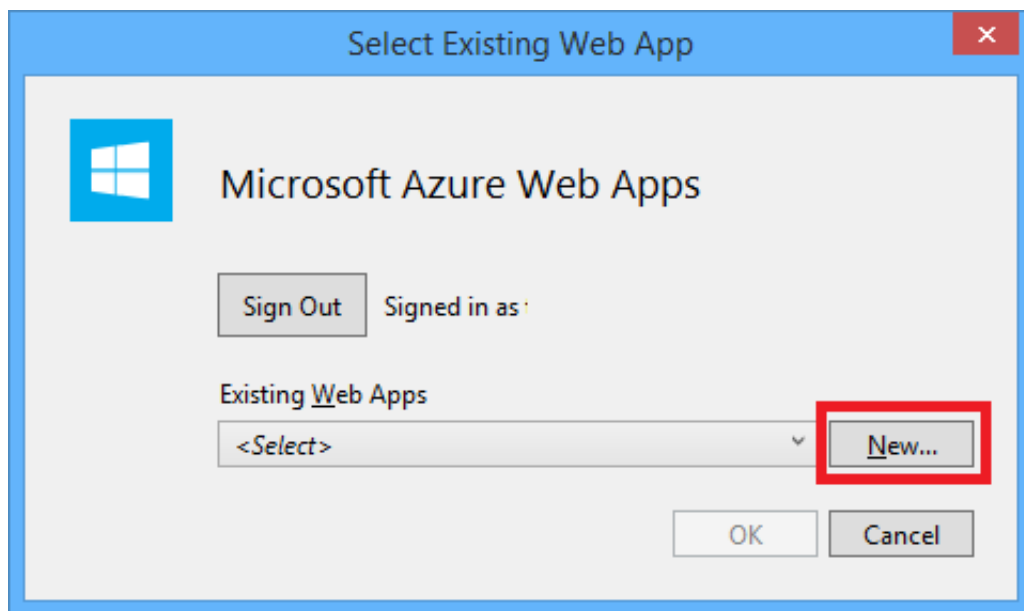
Deploy to Web Apps

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 **Profile** step of the **Publish Web** wizard, click **Microsoft Azure web apps**.



4. Sign in to Azure if you aren't still signed in.
5. Click **New**.

The dialog box may look slightly different depending on which version of the Azure SDK for .NET you have installed.



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

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

7. In the [App Service plan \(../azure-web-sites-web-hosting-plans-in-depth-overview/\)](#) drop-down list choose **Create new App Service plan**. Enter a name for the App Service plan, such as `ContosoAdsPlan`.
8. In the [Resource group \(../resource-group-overview/\)](#) drop-down list choose **Create new resource group**.
9. Enter a name for the resource group, such as `ContosoAdsGroup`.
10. In the **Region** drop-down list, choose the same region you chose for your storage account.

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.

11. In the **Database server** drop-down list choose **Create new server**.
12. Enter a name for the database server, such as `contosoadsserver` + a number or your name to make the server name unique.

The server name must be unique. It can contain lower-case letters, numeric digits, and hyphens. It cannot contain a trailing hyphen.

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

13. Enter an administrator **Database username** and **Database 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.


14. Click **Create**.

The screenshot shows the 'Create Web App on Microsoft Azure' dialog box. At the top, it says 'Create a Web App on Microsoft Azure' with a 'Learn more' link. Below this, there's a 'Sign Out' button and a status 'Signed in as td15426@hotmail.com'. The main form has several fields: 'Web App name' with the value 'ContosoAds8' and a green checkmark; 'App Service plan' with a dropdown menu showing 'Create new App Service plan' and a sub-field 'ContosoAdsPlan'; 'Resource group' with a dropdown menu showing 'Create new resource group' and a sub-field 'ContosoAdsGroup'; 'Region' with a dropdown menu showing 'West US'; 'Database server' with a dropdown menu showing 'Create new server' and a sub-field 'ContosoAdsServer'; 'Database username' with the value 'dbuser'; 'Database password' and 'Confirm password' fields, both masked with dots. At the bottom right, there are 'Create' and 'Cancel' buttons. A disclaimer at the bottom states: 'If you have removed your spending limit or you are using Pay As You Go, there may be monetary impact if you provision additional resources. [legal terms](#)'.

Visual Studio creates the solution, the web project, the web app in Azure, and the Azure SQL Database instance.

15. In the **Connection** step of the **Publish Web** wizard, click **Next**.

Publish Web

 Publish Web

Profile

Connection

Settings

Preview

ContosoAds2

Publish method: Web Deploy

Server: contosoads2.scm.azurewebsites.net:443

Site name: ContosoAds2

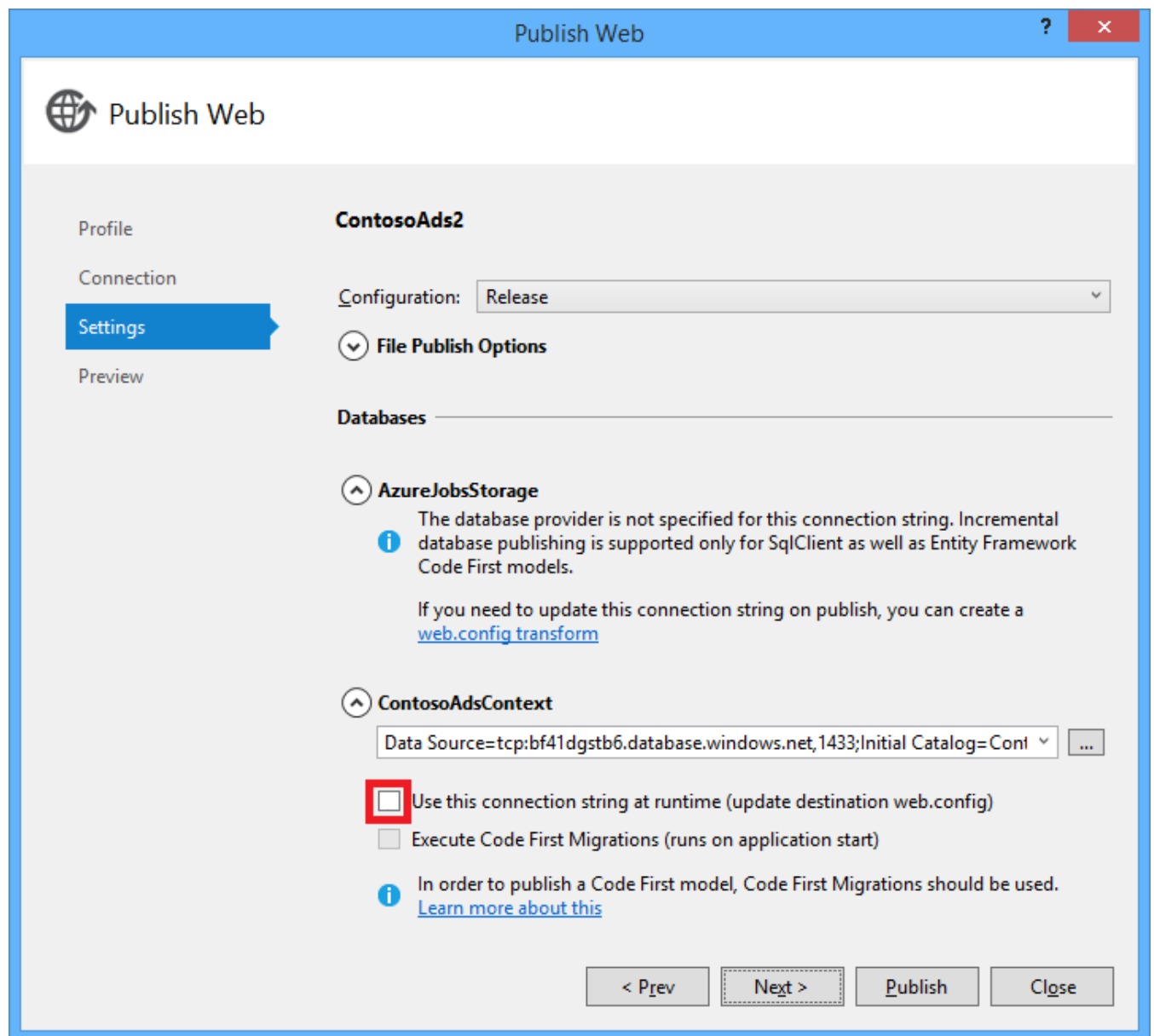
User name: \$ContosoAds2

Password:

☒ Save password

Destination URL: http://contosoads2.azurewebsites.net

16. In the **Settings** step, clear the **Use this connection string at runtime** check box, and then click **Next**.



You don't need to use the publish dialog to set the SQL connection string because you'll set that value in the Azure environment later.

You can ignore the warnings on this page.

- 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 did want to use a different storage account in the cloud, you wouldn't need to transform the connection string because the app uses 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. Code First automatically creates a new database the first time the app tries to access SQL data.

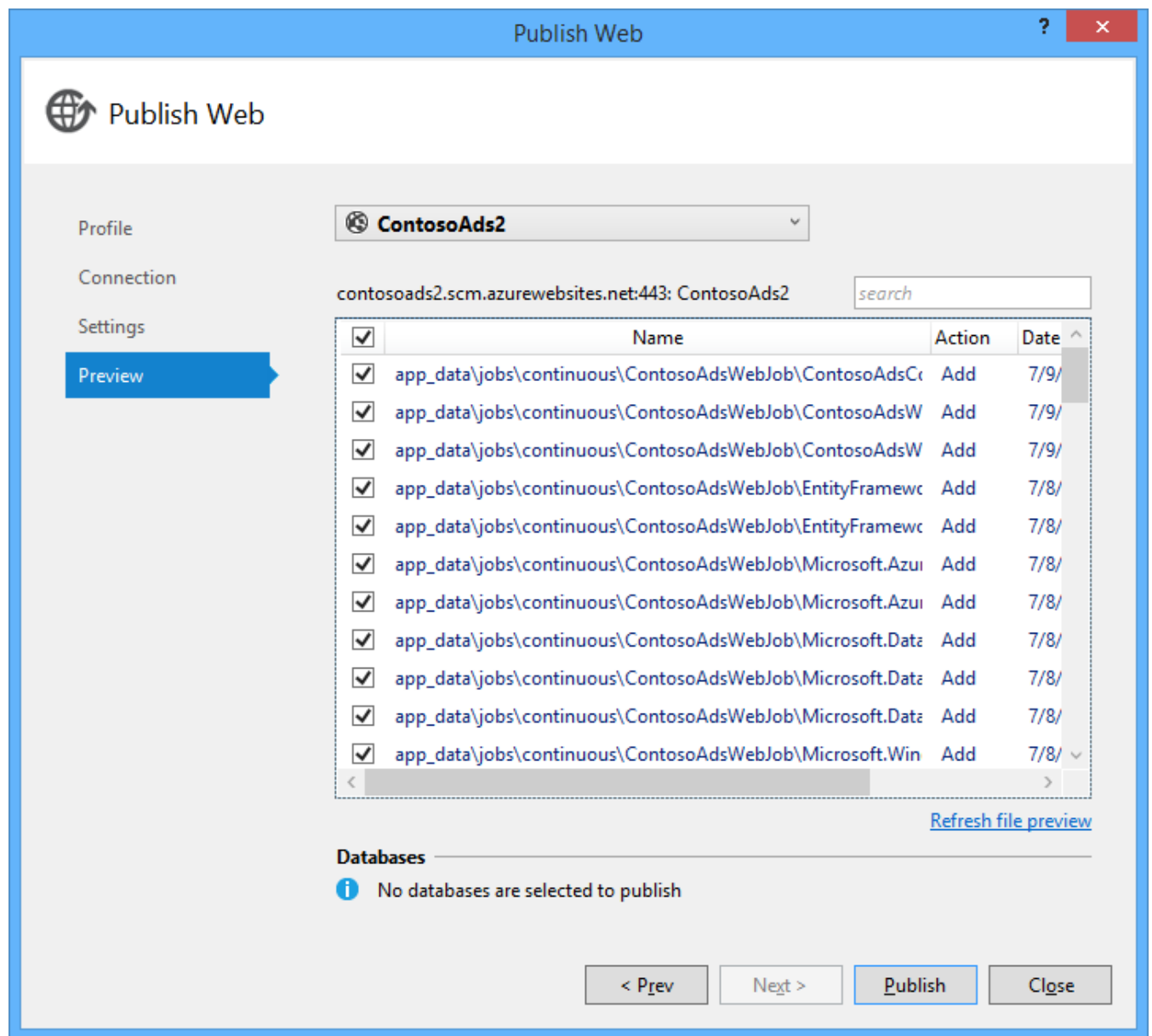
For this tutorial, the default values of the options under **File Publish Options** are fine.

17. In the **Preview** step, click **Start Preview**.



You can ignore the warning about no databases being published. Entity Framework Code First creates 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.



18. 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 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) (<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 \(/blog/2013/07/17/windows-azure-web-sites-how-application-strings-and-connection-strings-work/\)](/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**, right-click your web app under **Azure > {your resource group}**, and then click **View Settings**.

The **Azure Web App** window opens on the **Configuration** tab.

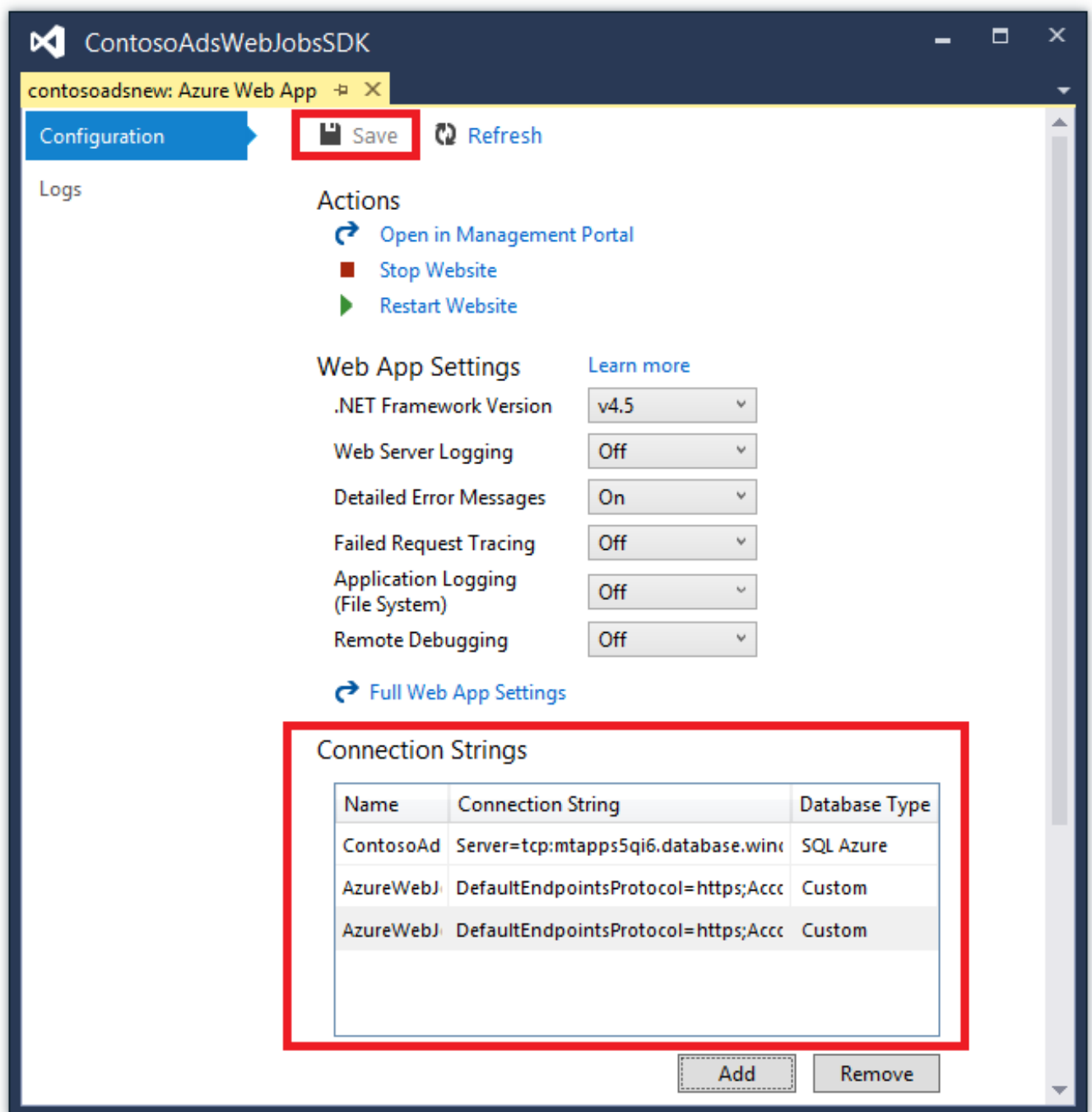
2. Change the name of the DefaultConnection connection string to ContosoAdsContext.

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.

3. Add two new connection strings, named AzureWebJobsStorage and AzureWebJobsDashboard. Set 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.

4. Click **Save**.

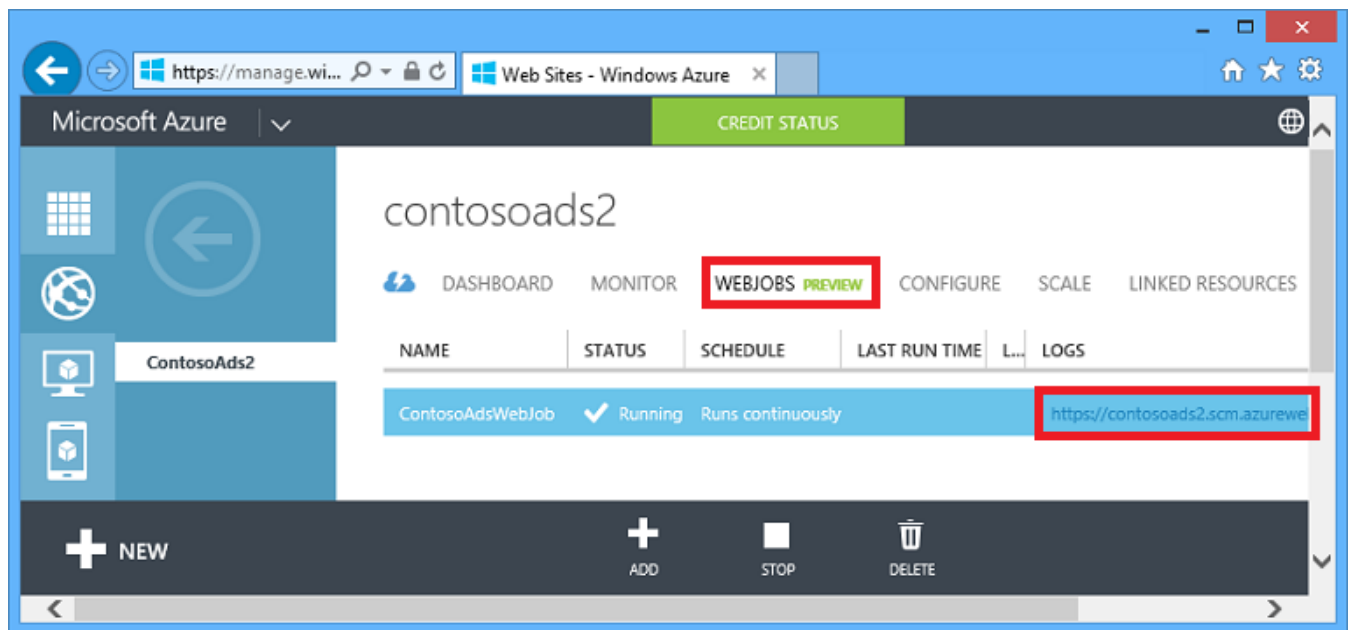


5. In **Server Explorer**, right-click the web app, and then click **Stop**.
6. After the web app stops, right-click the web app again, and then click **Start**.
The WebJob automatically starts when you publish, but it stops when you make a configuration change. To restart it you can either restart the web app or restart the WebJob in the [Azure Portal \(http://go.microsoft.com/fwlink/?linkid=529715&clcid=0x409\)](http://go.microsoft.com/fwlink/?linkid=529715&clcid=0x409). It's generally recommended to restart the web app after a configuration change.
7. Refresh the browser window that has the web app URL in its address bar.
The home page appears.
8. Create an ad, as you did when you ran the application locally.
The Index page shows without a thumbnail at first.
9. Refresh the page after a few seconds, and the thumbnail appears.

If the thumbnail doesn't appear, you may have to wait a minute or so for the WebJob to restart. If after a while you still don't see the thumbnail when you refresh the page, the WebJob may not have started automatically. In that case, go to the WebJobs tab in the [Azure Portal \(https://manage.windowsazure.com\)](https://manage.windowsazure.com) page for your web app, and then click **Start**.

View the WebJobs SDK dashboard

1. In the [Azure Portal \(https://manage.windowsazure.com\)](https://manage.windowsazure.com), select your web app.
2. Click the **WebJobs** tab.
3. Click the URL in the Logs column for your WebJob.



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.

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

←→https://contosoads.scm.azur...Azure WebJobs dashboard

Microsoft Azure WebJobs

WebJobs

Continuous WebJob Details

ContosoAdsWebJob

Running

Run command: ContosoAdsWebJob.exe

Toggle Output

Functions invoked

FUNCTION	STATUS	STATUS DETAIL
ContosoAdsWebJob.Program.GenerateThumbnail ({"BlobUri":"https: ...})	Success	4 minutes ago (4 s running time)

The screenshot shows the Microsoft Azure WebJobs dashboard in a web browser. The browser's address bar displays the URL `https://contosoads.scm.a...` and the page title is 'Azure WebJobs dashboard'. The dashboard header shows 'Microsoft Azure WebJobs' with a menu icon. The breadcrumb trail is 'WebJobs / ContosoAdsWebJob / ContosoAdsWebJob.Program.GenerateThumbnail'. The main section is titled 'Invocation Details' for the function 'ContosoAdsWebJob.Program.GenerateThumbnail', with a JSON snippet `{ "BlobUri": "https: ..." }` below it. A blue 'Replay Function' button is present. A green status bar indicates 'Success 6 minutes ago (4 s running time)'. A lightning bolt icon and text state 'New queue message detected on 'thumbnailrequest''. Below this is a table with three columns: 'PARAMETER', 'VALUE', and 'NOTES'. The table contains three rows: 'blobInfo' with a long JSON string, 'input' with a blob path, and 'outputBlob' with another blob path. The 'NOTES' column for the 'input' row contains performance metrics. At the bottom, there is a 'Toggle Output' button.

Microsoft Azure WebJobs

WebJobs / ContosoAdsWebJob / ContosoAdsWebJob.Program.GenerateThumbnail

Invocation Details

ContosoAdsWebJob.Program.GenerateThumbnail

`{ "BlobUri": "https: ..." }`

[Replay Function](#)

Success 6 minutes ago (4 s running time)

⚡ New queue message detected on 'thumbnailrequest'.

PARAMETER	VALUE	NOTES
blobInfo	<code>{ "BlobUri": "https://contosoads.blob.core.windows.net/images/d3709ece-461a-4eef-9dc3-2f12b7102030.jpg", "BlobName": "d3709ece-461a-4eef-9dc3-2f12b7102030.jpg", "BlobNameWithoutExtension": "d3709ece-461a-4eef-9dc3-2f12b7102030", "AdId": 3 }</code>	
input	images/d3709ece-461a-4eef-9dc3-2f12b7102030.jpg	Read 116,652 bytes (100.15% of total). (about 41 milliseconds spent on I/O)
outputBlob	images/d3709ece-461a-4eef-9dc3-2f12b7102030_thumbnail.jpg	

[Toggle Output](#)

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 Database instance. The web app is free, but the SQL 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 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 Database and Storage account. You can follow a similar procedure to delete the SQL database and storage account when you no longer need them.

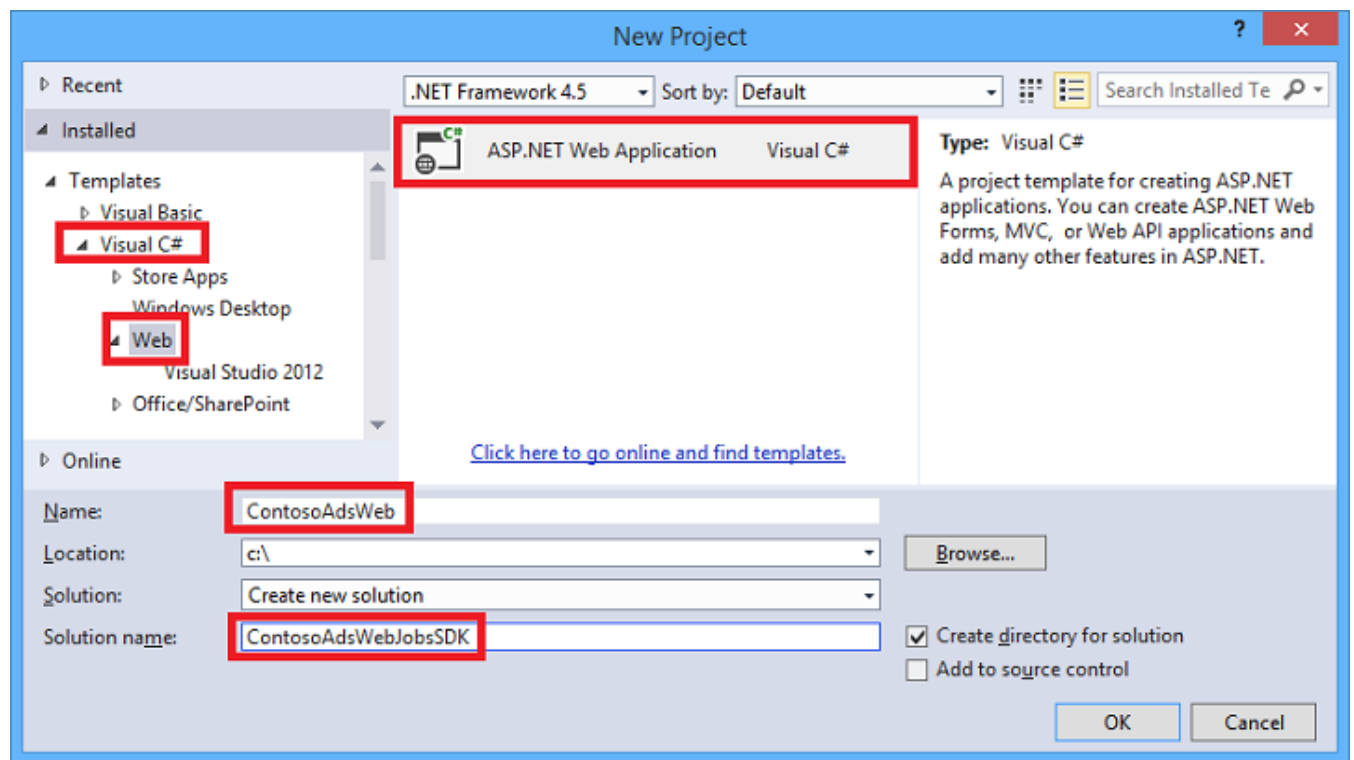
Create the application from scratch

In this section you'll do the following tasks:

- Create a Visual Studio solution with a web project.
- Add a class library project for the data access layer that is shared between front end and backend.
- Add a Console Application project for the backend, 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.

Create a Visual Studio solution with a web project and class library project

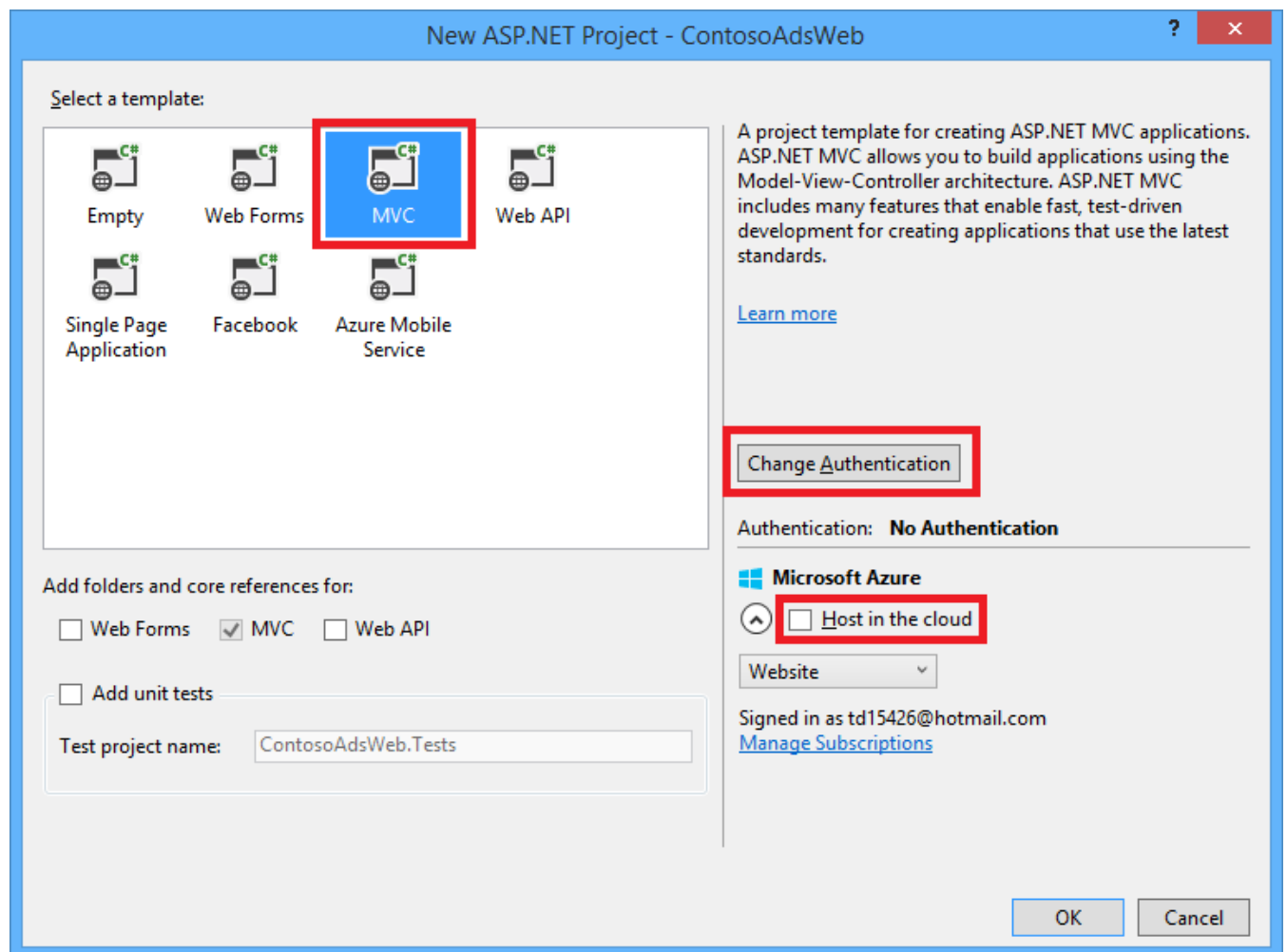
1. In Visual Studio, choose **New > Project** from the **File** menu.
2. In the **New Project** dialog, choose **Visual C# > Web > ASP.NET Web Application**.
3. Name the project ContosoAdsWeb, name the solution ContosoAdsWebJobsSDK (change the solution name if you're putting it in the same folder as the downloaded solution), and then click **OK**.



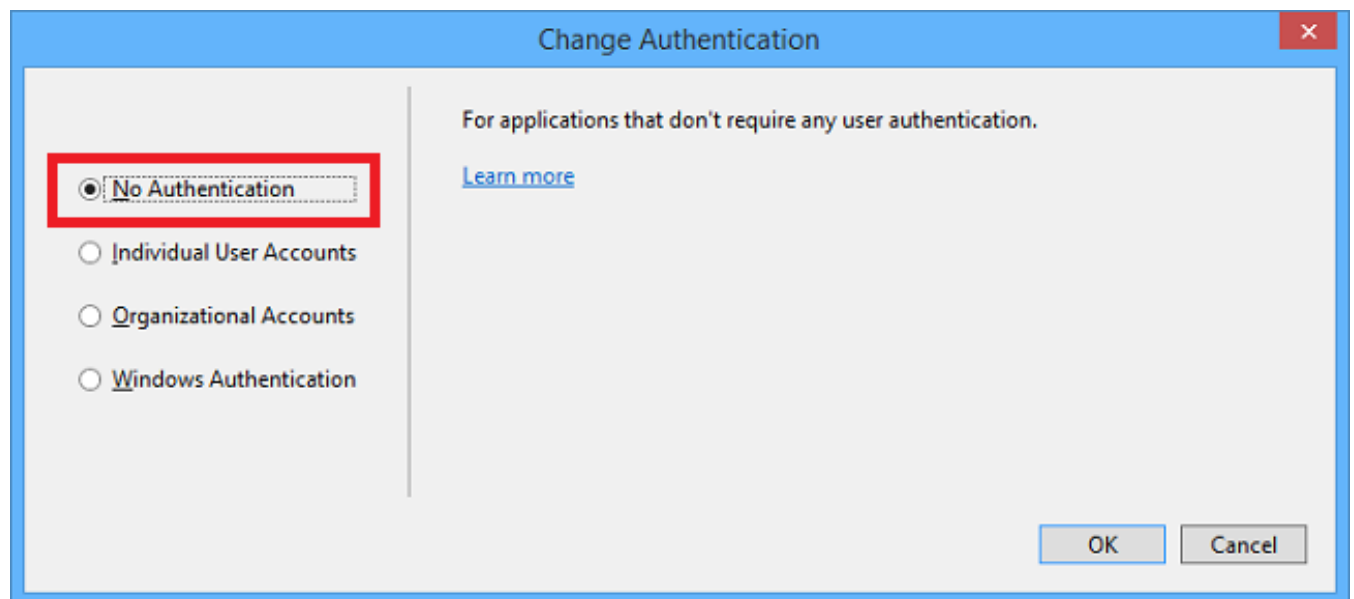
4. In the **New ASP.NET Project** dialog, choose the MVC template, and clear the **Host in the cloud** check box under **Microsoft Azure**.

Selecting **Host in the cloud** enables Visual Studio to automatically create a new Azure web app and SQL 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 database the same way you did earlier when you deployed the application.

5. Click **Change Authentication**.



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



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

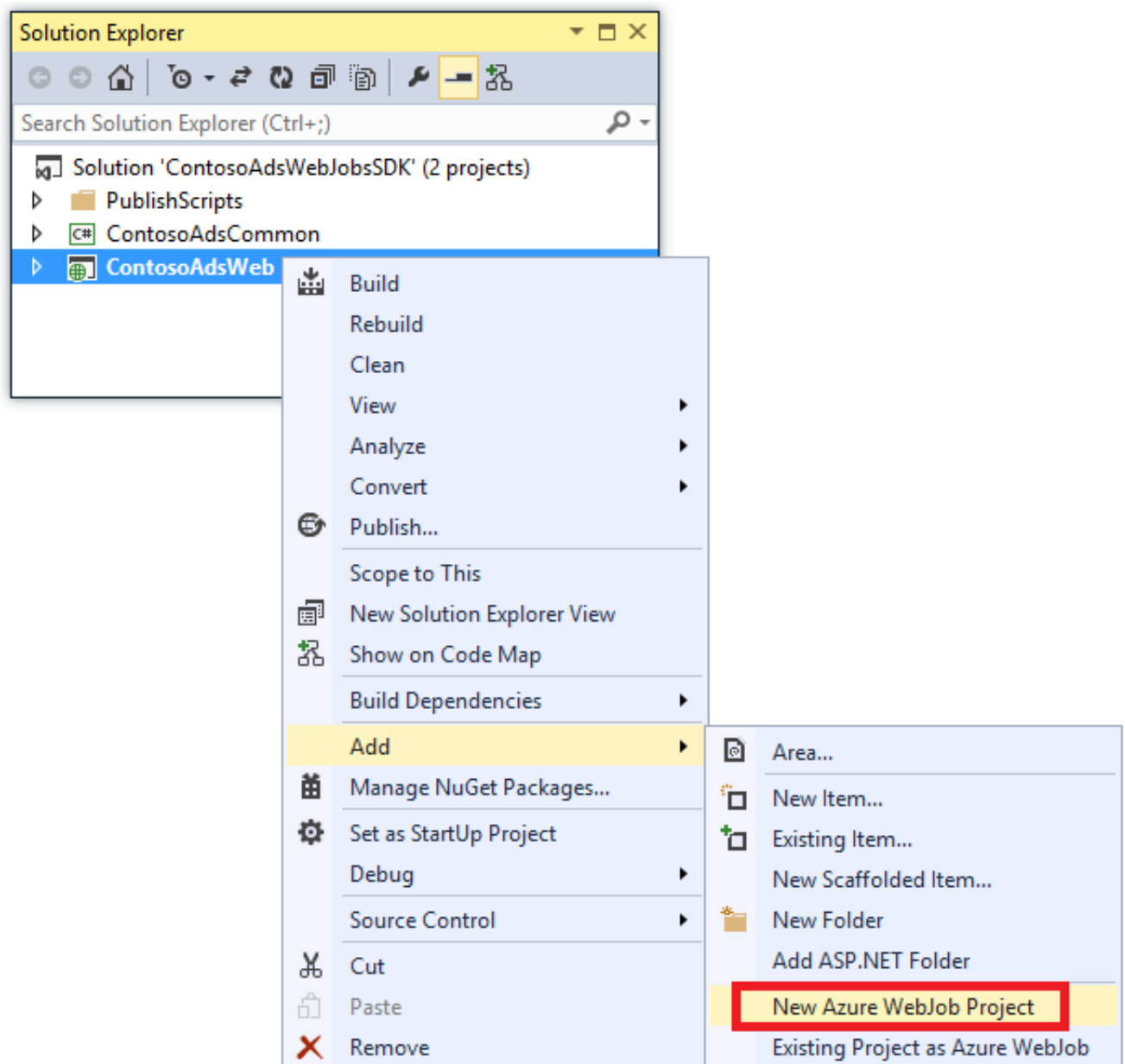
Visual Studio creates the solution and the web project.

8. In **Solution Explorer**, right-click the solution (not the project), and choose **Add > New Project**.
9. In the **Add New Project** dialog, choose **Visual C# > Windows Desktop > Class Library** template.
10. Name the project *ContosoAdsCommon*, and then click **OK**.

This project will contain the Entity Framework context and the data model which both the front end and back end 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.

Add a Console Application project that has WebJobs deployment enabled

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



2. In the **Add Azure WebJob** dialog, enter ContosoAdsWebJob as both the **Project name** and the **WebJob name**. Leave **WebJob run mode** set to **Run Continuously**.
3. 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 \(../websites-dotnet-deploy-webjobs/\)](#).

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 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.

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

Set project references

Both web and WebJob projects work with the SQL 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 click **Add > Reference**. In the **Reference Manager** dialog box, select **Solution > Projects > ContosoAdsCommon**, and then click **OK**.)
2. In the ContosoAdsWebJob project, set a reference to the ContosoAdsCommon project.

The WebJob project needs references for working with images and for accessing connection strings.

3. In the ContosoAdsWebJob project, set a reference to `System.Drawing` and `System.Configuration`.

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) (<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) (<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) (<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 show and explain key parts of the 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 following files from the downloaded project.
 - *Ad.cs*
 - *ContosoAdscontext.cs*
 - *BlobInformation.cs*
2. 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
3. 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.

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, go to the [Program.cs and Functions.cs](#) sections.

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; }
}
```

Copy

ContosoAdsCommon - ContosoAdsContext.cs

The `ContosoAdsContext` class specifies that the `Ad` class is used in a `DbSet` collection, which Entity Framework stores 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; }
}
```

Copy

The class has two constructors. The first 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.

```
public class BlobInformation
{
    public Uri BlobUri { get; set; }

    public string BlobName
    {
        get
        {
            return BlobUri.Segments[BlobUri.Segments.Length - 1];
        }
    }
    public string BlobNameWithoutExtension
    {
        get
        {
            return Path.GetFileNameWithoutExtension(BlobName);
        }
    }
    public int AdId { get; set; }
}
```

Copy

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 are 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.

```
var storageAccount = CloudStorageAccount.Parse
    (ConfigurationManager.ConnectionStrings["AzureWebJobsStorage"].ToString());
```

Copy

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.

```
var blobClient = storageAccount.CreateCloudBlobClient();
var imagesBlobContainer = blobClient.GetContainerReference("images");
if (imagesBlobContainer.CreateIfNotExists())
{
    imagesBlobContainer.SetPermissions(
        new BlobContainerPermissions
        {
            PublicAccess = BlobContainerPublicAccessType.Blob
        });
}
```

Copy

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

```
CloudQueueClient queueClient = storageAccount.CreateCloudQueueClient();
var imagesQueue = queueClient.GetQueueReference("blobnamerequest");
imagesQueue.CreateIfNotExists();
```

Copy

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>
```

Copy

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) (<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.DefaultRequestOptions.RetryPolicy = new LinearRetry(TimeSpan.FromSeconds(3), 3);
imagesBlobContainer = blobClient.GetContainerReference("images");
```

Copy

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");
```

Copy

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) (<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)
```

Copy

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();
}
```

Copy

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;
}
```

Copy

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);
```

Copy

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();
}
```

Copy

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

```
private async Task DeleteAdBlobsAsync(Ad ad)
{
    if (!string.IsNullOrEmpty(ad.ImageURL))
    {
        Uri blobUri = new Uri(ad.ImageURL);
        await DeleteAdBlobAsync(blobUri);
    }
    if (!string.IsNullOrEmpty(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();
}
```

Copy

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

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

```

```

Copy

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

```

```

Copy

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" }))
```

Copy

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

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

Copy

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();
}
```

Copy

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();
    }
}

```

Copy

- 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,
```

Copy

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 5 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)
```

Copy

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.

For more information about how to write functions that use WebJobs SDK attributes, see the following resources:

- [How to use Azure queue storage with the WebJobs SDK \(../websites-dotnet-webjobs-sdk-storage-queues-how-to/\)](#)
- [How to use Azure blob storage with the WebJobs SDK \(../websites-dotnet-webjobs-sdk-storage-blobs-how-to/\)](#)
- [How to use Azure table storage with the WebJobs SDK \(../websites-dotnet-webjobs-sdk-storage-tables-how-to/\)](#)
- [How to use Azure Service Bus with the WebJobs SDK \(../websites-dotnet-webjobs-sdk-service-bus/\)](#)

NOTE:

- If your web app runs on multiple VMs, multiple WebJobs will be running simultaneously, and in some scenarios this can result in the same data getting processed multiple times. This is not a problem if you use the built-in queue, blob, and Service Bus triggers. The SDK ensures that your functions will be processed only once for each message or blob.
- For information about how to implement graceful shutdown, see [Graceful Shutdown \(../websites-dotnet-webjobs-sdk-storage-queues-how-to/#graceful\)](#).
- The code in the `ConvertImageToThumbnailJPG` method (not shown) 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. For more information about image processing options, see [Dynamic Image Generation \(http://www.hanselman.com/blog/BackToBasicsDynamicImageGenerationASPNETControll\)](#) and [Deep Inside Image Resizing \(http://www.hanselminutes.com/313/deep-inside-image-resizing-and-scaling-with-aspnet-and-iis-with-imageresizingnet-author-na\)](#).

Next steps

In this tutorial you've seen a simple multi-tier application that uses the WebJobs SDK for backend processing. This section offers some suggestions for learning more about ASP.NET multi-tier applications and WebJobs.

Missing features

The application has been kept simple for a getting-started tutorial. In a real-world application you would implement [dependency injection \(http://www.asp.net/mvc/tutorials/hands-on-labs/aspnet-mvc-4-dependency-injection\)](#) and 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), 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>), 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, and use 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.

Scaling WebJobs

WebJobs run in the context of a web app and are not scalable separately. For example, if you have one Standard web app instance, you have only one instance of your background process running, and it is using some of the server resources (CPU, memory, etc.) that otherwise would be available to serve web content.

If traffic varies by time of day or day of week, and if the backend processing you need to do can wait, you could schedule your WebJobs to run at low-traffic times. If the load is still too high for that solution, you can run the backend as a WebJob in a separate web app dedicated for that purpose. You can then scale your backend web app independently from your frontend web app.

For more information, see [Scaling WebJobs \(../websites-webjobs-resources/#scale\)](#).

Avoiding web app timeout shut-downs

To make sure your WebJobs are always running, and running on all instances of your web app, you have to enable 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.

Using the WebJobs SDK outside of WebJobs

A program that uses the WebJobs SDK doesn't have to run in Azure in a WebJob. It can run locally, and it can also run in other environments such as a Cloud Service worker role or a Windows service. However, you can only access the WebJobs SDK dashboard through an Azure web app. To use the dashboard you have to connect the web app to the storage account you're using by setting the AzureWebJobsDashboard connection string on the **Configure** tab of the Azure portal. Then you can get to the Dashboard by using the following URL:

`https://{webappname}.scm.azurewebsites.net/azurejobs/#/functions`

For more information, see [Getting a dashboard for local development with the WebJobs SDK](http://blogs.msdn.com/b/jmstall/archive/2014/01/27/getting-a-dashboard-for-local-development-with-the-webjobs-sdk.aspx) (<http://blogs.msdn.com/b/jmstall/archive/2014/01/27/getting-a-dashboard-for-local-development-with-the-webjobs-sdk.aspx>), but note that it shows an old connection string name.

More WebJobs documentation

For more information, see [Azure WebJobs documentation resources](http://go.microsoft.com/fwlink/?linkid=390226&clcid=0x409) (<http://go.microsoft.com/fwlink/?linkid=390226&clcid=0x409>).

Need help?

Go to an MSDN forum (<https://social.msdn.microsoft.com/forums/azure/en-US/home?forum=windowsazureplatform>)

StackOverflow discussion (<http://stackoverflow.com/questions/tagged/azure>)