Create & deploy a simple Azure Application using WebJobs

Contents

[Overview 2](#_Toc442184288)

[Contoso Ads application 2](#_Toc442184289)

[Prerequisites 3](#_Toc442184290)

[What you'll learn 3](#_Toc442184291)

[Application architecture 4](#_Toc442184292)

[Set up the development environment 5](#_Toc442184293)

[Create an Azure Storage account 5](#_Toc442184294)

[Download the application 7](#_Toc442184295)

[Configure the application to use your storage account 8](#_Toc442184296)

[Run the application locally 10](#_Toc442184297)

[Run the application in the cloud 15](#_Toc442184298)

[**Deploy to Web Apps** 16](#_Toc442184299)

[**Configure the web app to use your Azure SQL database and storage account.** 22](#_Toc442184300)

[**View the WebJobs SDK dashboard** 25](#_Toc442184301)

[Next steps 28](#_Toc442184302)

[**Missing features** 28](#_Toc442184303)

[**Scaling WebJobs** 28](#_Toc442184304)

[**Avoiding web app timeout shut-downs** 28](#_Toc442184305)

[**Using the WebJobs SDK outside of WebJobs** 29](#_Toc442184306)

[**More WebJobs documentation** 29](#_Toc442184307)

## Overview

This tutorial shows how to write code for a simple multi-tier ASP.NET MVC 5 application that uses the [WebJobs SDK](https://azure.microsoft.com/en-us/documentation/articles/websites-dotnet-webjobs-sdk/).

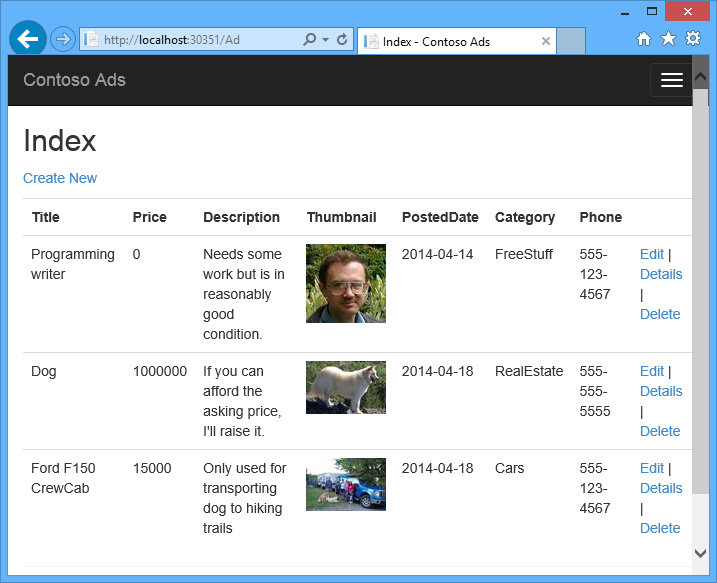
The purpose of the [WebJobs SDK](https://azure.microsoft.com/en-us/documentation/articles/websites-webjobs-resources/) is to simplify the code you write for common tasks that a WebJob can perform, such as image processing, queue processing, RSS aggregation, file maintenance, and sending emails. The WebJobs SDK has built-in features for working with Azure Storage and Service Bus, for scheduling tasks and handling errors, and for many other common scenarios. In addition, it's designed to be extensible, and there's an [open source repository for extensions](https://github.com/Azure/azure-webjobs-sdk-extensions/wiki/Binding-Extensions-Overview).

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.

## 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.

This sample application works 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) and [Azure SQL Database](http://msdn.microsoft.com/library/azure/ee336279).



## 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) 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.

## What you'll learn

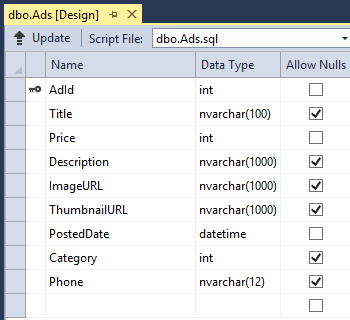
The tutorial shows how to do the following tasks:

* 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.

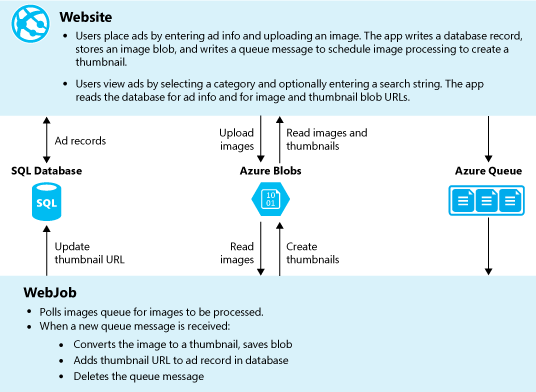
## Application architecture

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) 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.



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), 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) or the [Azure SDK for Visual Studio 2013](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.

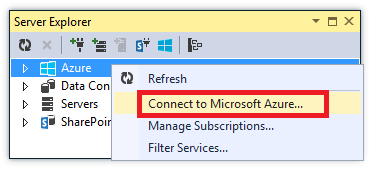
The tutorial instructions apply to Azure SDK for .NET 2.7.1 or later.

## Create an Azure Storage account

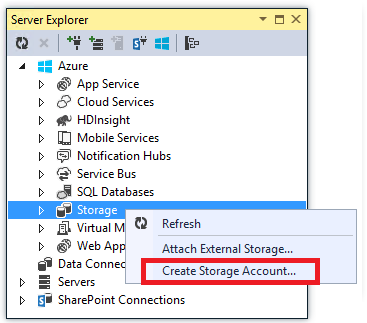
An Azure storage account provides resources for storing queue and blob data in the cloud. It's also used by the WebJobs SDK to store logging data for the dashboard.

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**.



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



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

The name must be 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.

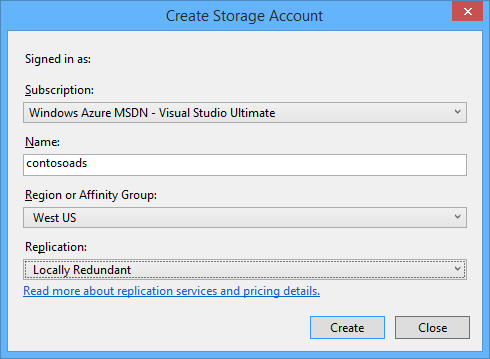
1. Set the **Region or Affinity Group** drop-down list to the region closest to you.

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.

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[Create, manage, or delete a storage account](https://azure.microsoft.com/en-us/documentation/articles/storage-create-storage-account/#replication-options).

1. Click **Create**.



## Download the application

1. Download and unzip the [completed solution](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.

1. 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) 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>

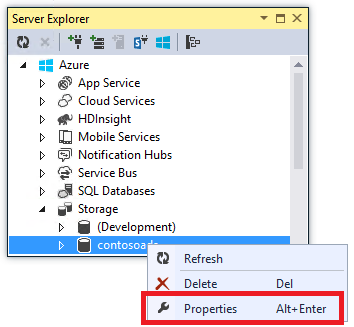
<add name="ContosoAdsContext" connectionString="Data Source=(localdb)\v11.0; Initial Catalog=ContosoAds; Integrated Security=True; MultipleActiveResultSets=True;" providerName="System.Data.SqlClient" />

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

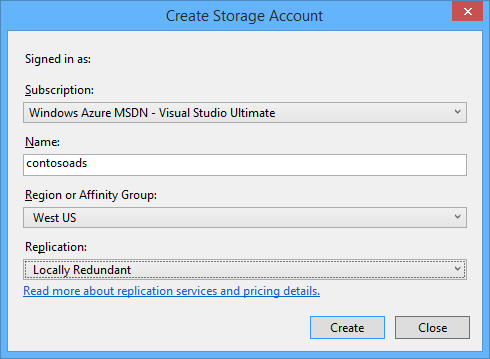
</connectionStrings>

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.

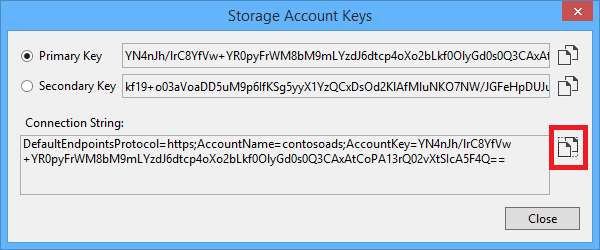
1. In **Server Explorer**, right-click your storage account under the **Storage** node, and then click **Properties**.



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



1. Copy the **Connection String**.



1. Replace the storage connection string in the Web.config file with the connection string you just copied. Make sure you select everything inside the quotation marks but not including the quotation marks before pasting.
2. Open the App.config file in the ContosoAdsWebJob project.

This file has two storage connection strings, one for application data and one for logging. You can use separate storage accounts for application data and logging, and you can use[multiple storage accounts for data](https://github.com/Azure/azure-webjobs-sdk/blob/master/test/Microsoft.Azure.WebJobs.Host.EndToEndTests/MultipleStorageAccountsEndToEndTests.cs). For this tutorial you'll use a single storage account. 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>

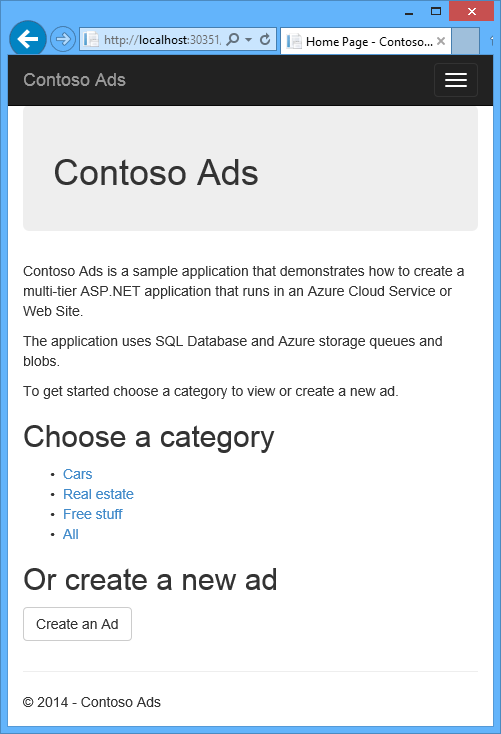
By default, the WebJobs SDK looks for connection strings named AzureWebJobsStorage and AzureWebJobsDashboard. As an alternative, you can [store the connection string however you want and pass it in explicitly to the JobHost object](https://azure.microsoft.com/en-us/documentation/articles/websites-dotnet-webjobs-sdk-storage-queues-how-to/#config).

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

## Run the application locally

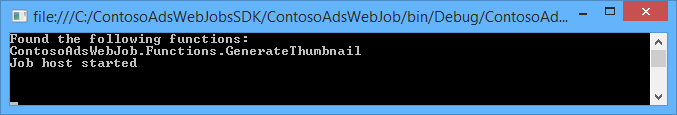
1. To start the web frontend of the application, press CTRL+F5.

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

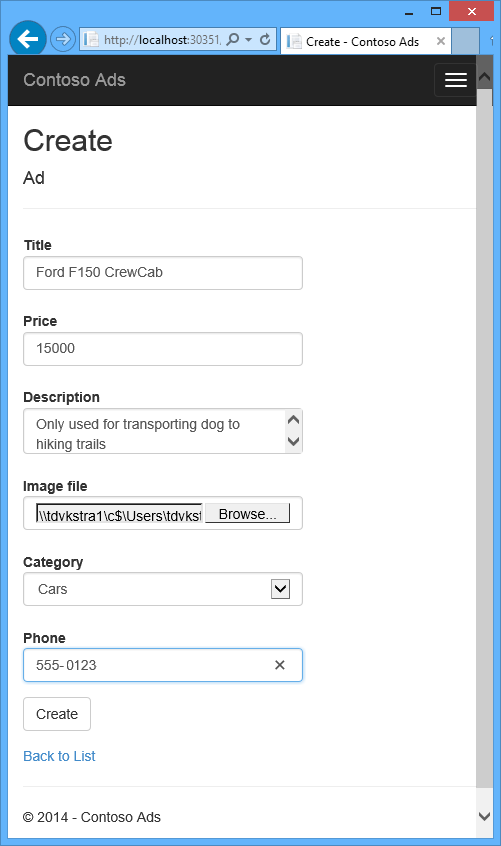


1. To start the WebJob backend of the application, right-click the ContosoAdsWebJob project in **Solution Explorer**, and then click **Debug** > **Start new instance**.

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

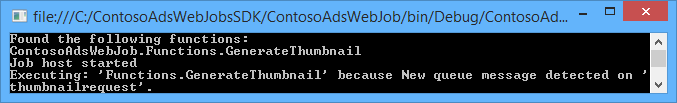


1. In your browser, click **Create an Ad**.
2. Enter some test data and select an image to upload, and then click **Create**.

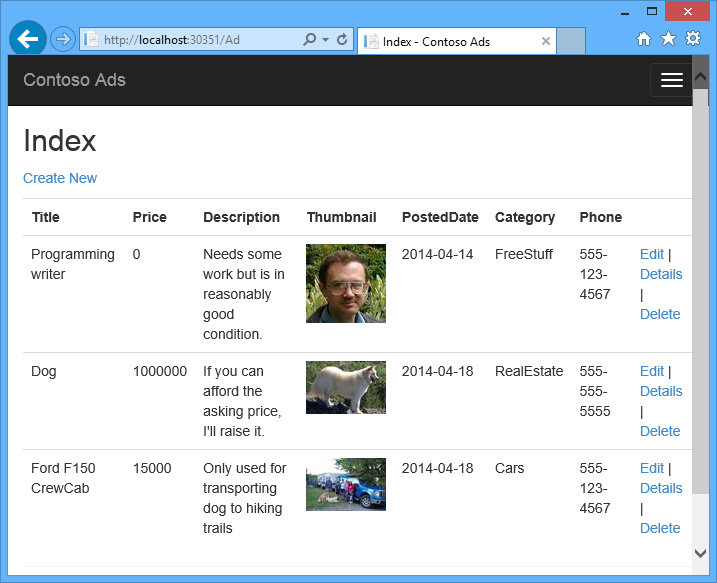


The app goes to the Index page, but it doesn't show a thumbnail for the new ad because that processing hasn't happened yet.

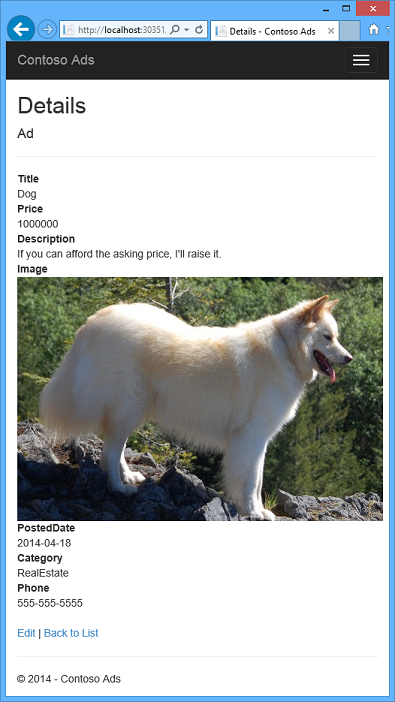
Meanwhile, after a short wait a logging message in the console application window shows that a queue message was received and has been processed.



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



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



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

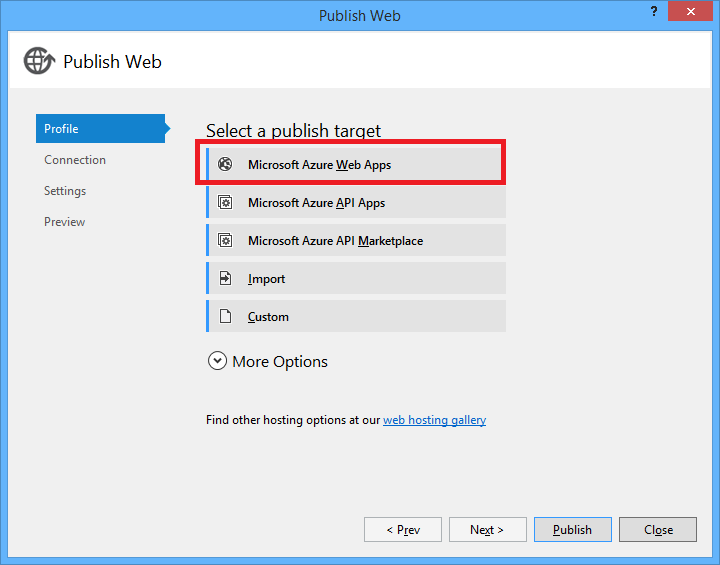
You'll do the following steps to run the application in the cloud:

* 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.

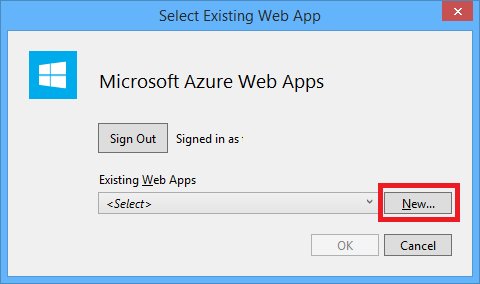
### **Deploy to Web Apps**

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**.



1. Sign in to Azure if you aren't still signed in.
2. Click **New**.

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



1. 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.

1. In the [App Service plan](https://azure.microsoft.com/en-us/documentation/articles/azure-web-sites-web-hosting-plans-in-depth-overview/) drop-down list choose **Create new App Service plan**. Enter a name for the App Service plan, such as ContosoAdsPlan.
2. In the [Resource group](https://azure.microsoft.com/en-us/documentation/articles/resource-group-overview/) drop-down list choose **Create new resource group**.
3. Enter a name for the resource group, such as ContosoAdsGroup.
4. 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.

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

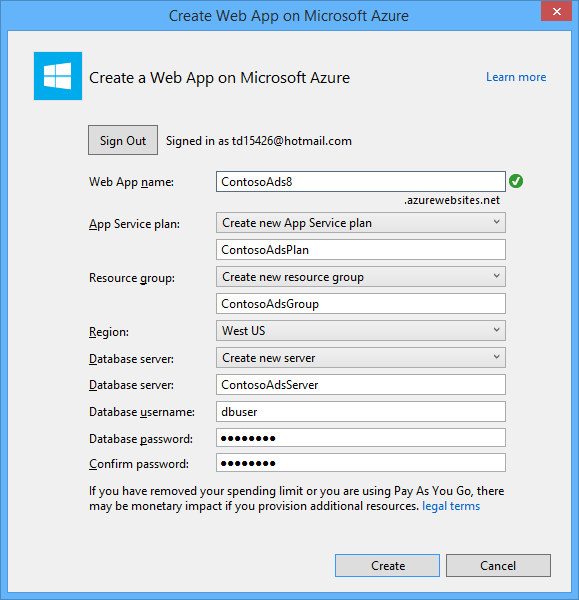
The server name must be unique. It can contain lower-case letters, numeric digits, and hyphens. It cannot contain a trailing hyphen.

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

1. Enter an administrator **Database username** and **Database password**.

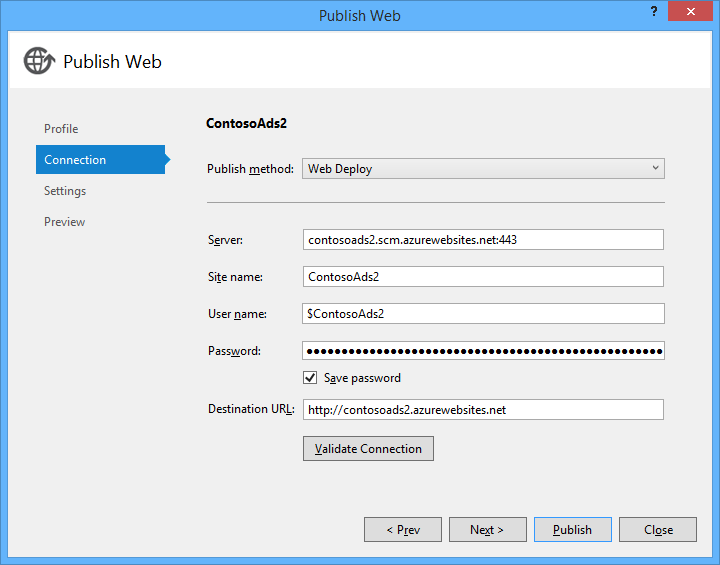
If you selected **New SQL Database server** you aren't entering an existing name and password here, you're entering a new name and password that you're defining now to use later when you access the database. If you selected a server that you created previously, you'll be prompted for the password to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

1. Click **Create**.

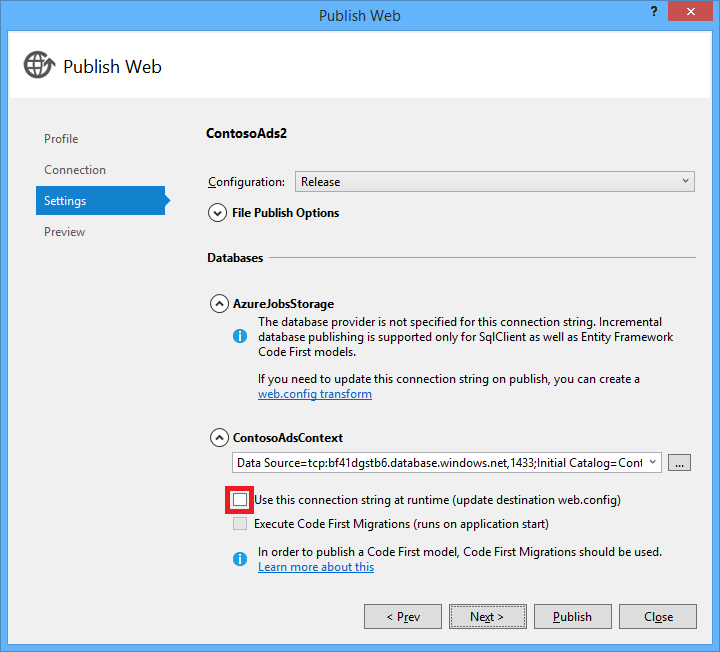


Visual Studio creates the solution, the web project, the web app in Azure, and the Azure SQL Database instance.

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



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



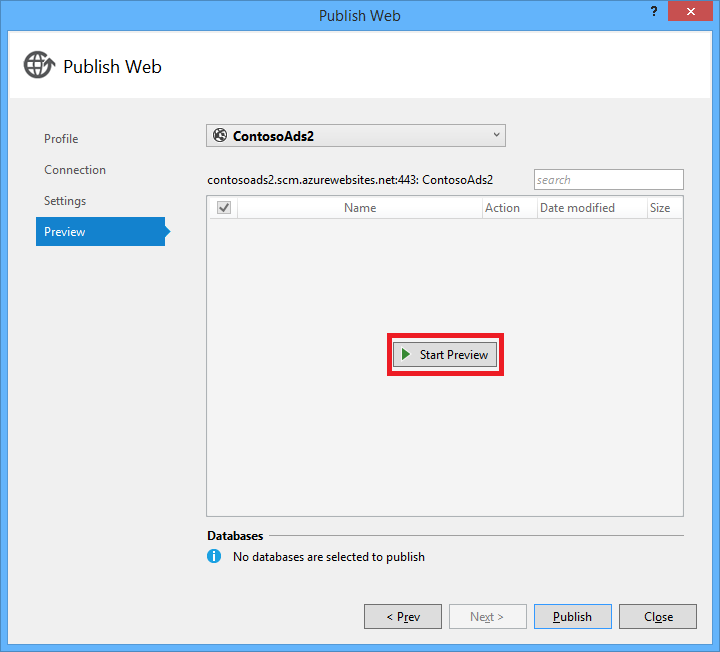
You don't need to use the publish dialog to set the SQL connection string because you'll set that value in the Azure environment later.

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.

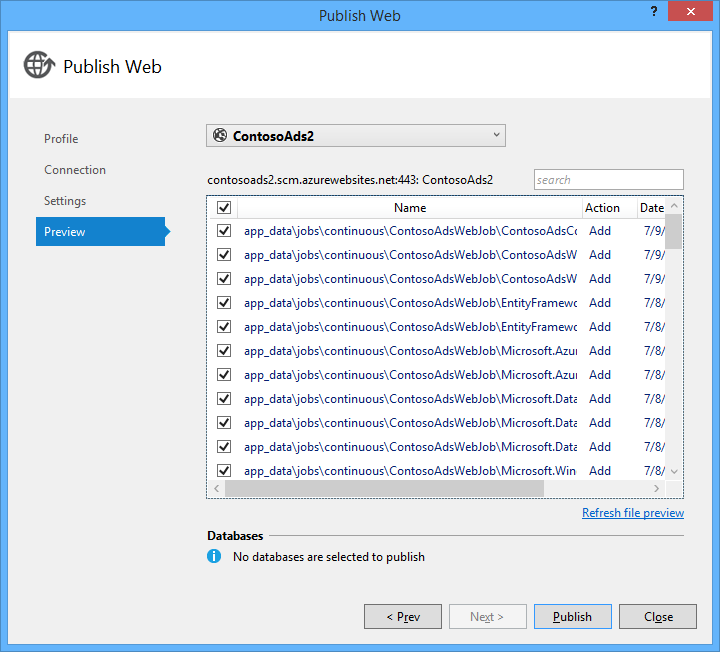
For this tutorial, the default values of the options under **File Publish Options** are fine.

1. 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.



1. Click **Publish**.

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

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

### **Configure the web app to use your Azure SQL database and storage account.**

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

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

1. In **Server Explorer**, 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.

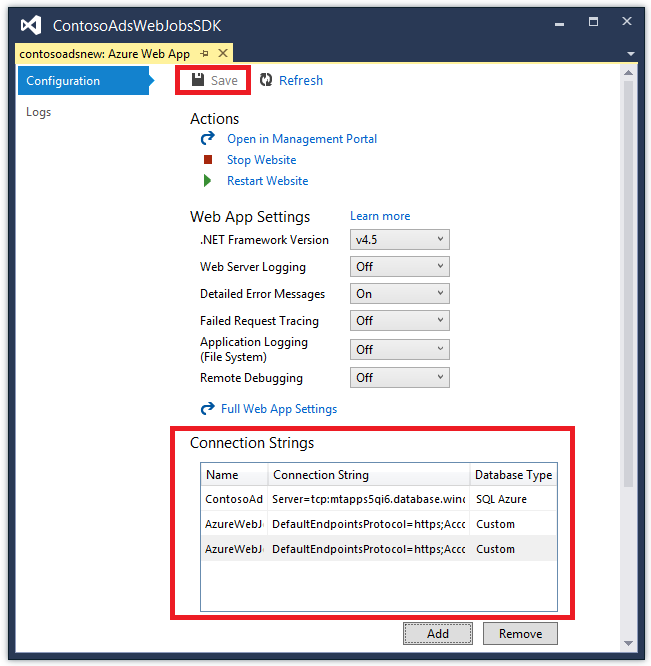
1. Change the name of the DefaultConnection connection string to ContosoAdsContext.

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

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

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

1. Click **Save**.



1. In **Server Explorer**, right-click the web app, and then click **Stop**.
2. 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). It's generally recommended to restart the web app after a configuration change.

1. Refresh the browser window that has the web app URL in its address bar.

The home page appears.

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

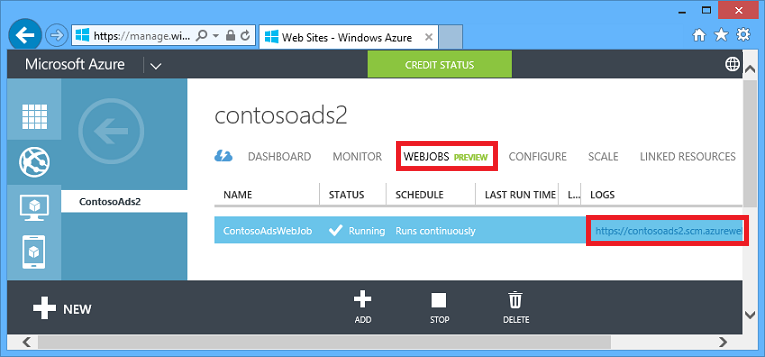
The Index page shows without a thumbnail at first.

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

If the thumbnail doesn't appear, 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 [classic portal](https://manage.windowsazure.com/) page for your web app, and then click **Start**.

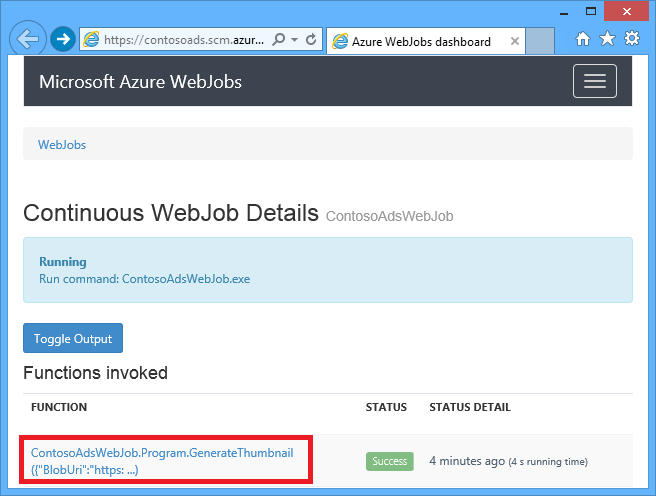
### **View the WebJobs SDK dashboard**

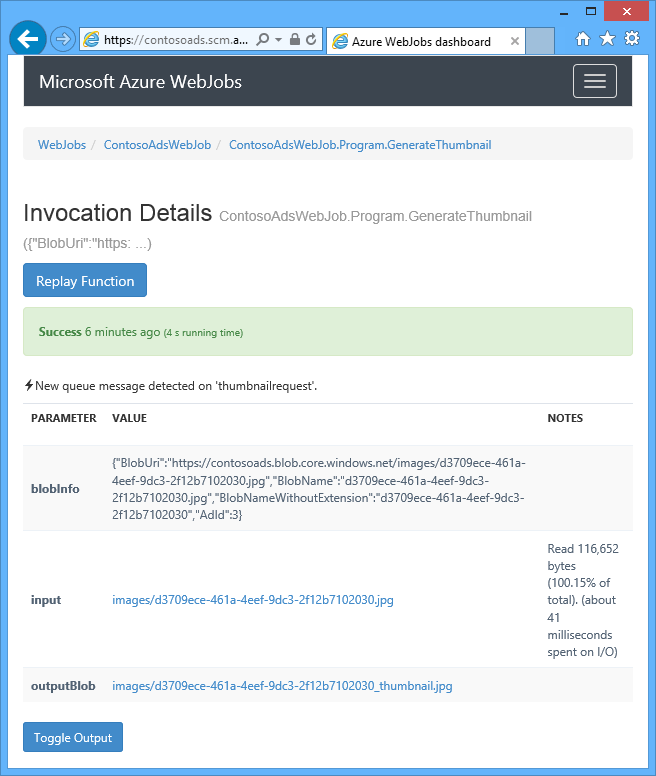
1. In the [classic portal](https://manage.windowsazure.com/), select your web app.
2. Click the **WebJobs** tab.
3. Click the URL in the Logs column for your WebJob.



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.

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





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

##### **NOTE:**

When you're finished testing, delete the web app and the SQL 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 classic 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.

## 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](https://azure.microsoft.com/en-us/documentation/articles/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 classic 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), 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).

End of the document.