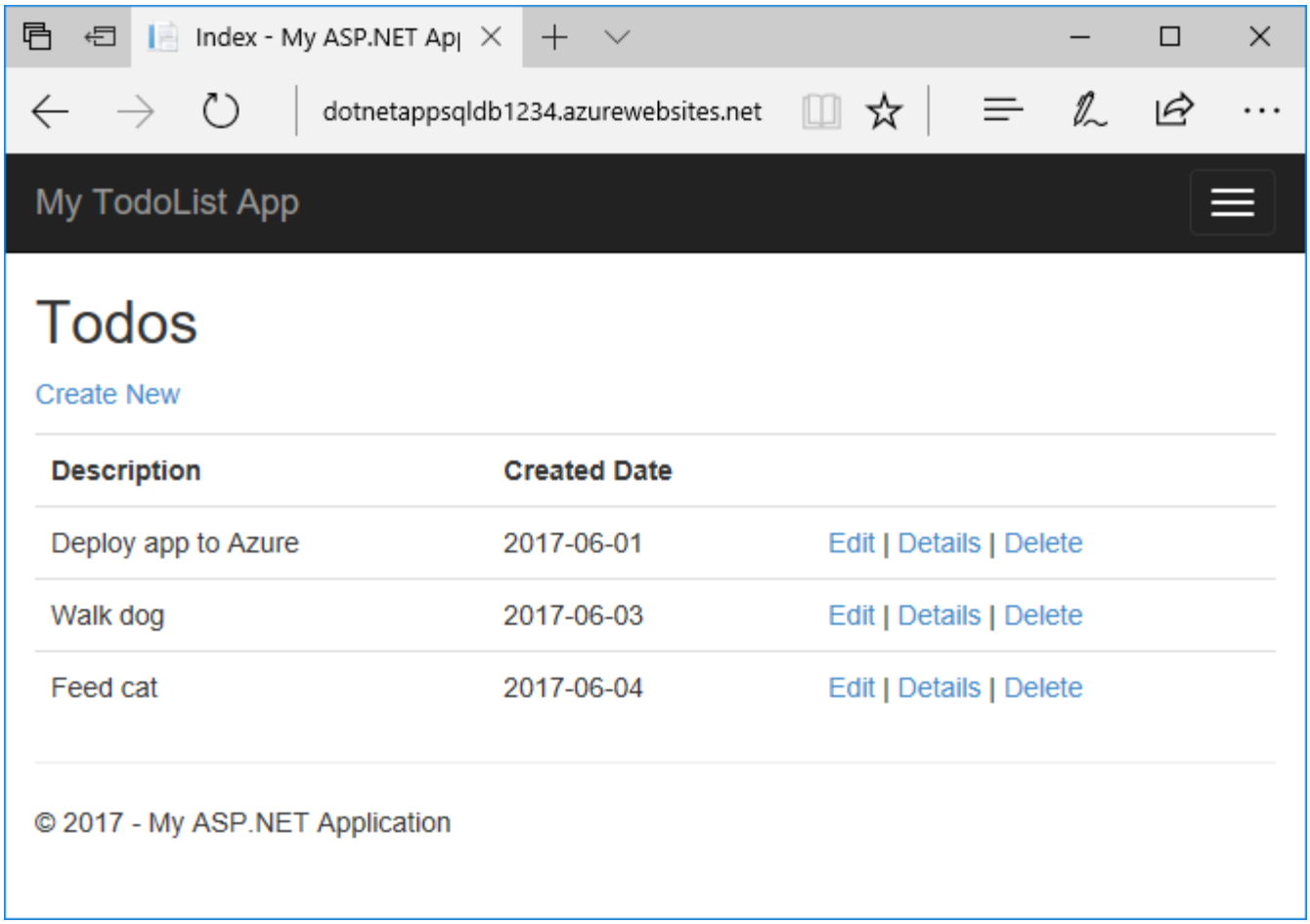
2017 - This tutorial shows you how to deploy a data-driven ASP.NET web app in Azure and connect it to Azure SQL Database. When you're finished, you have a ASP.NET app running in Azure and connected to SQL Database: https://docs.microsoft.com/en-us/azure/app-service/app-service-web-tutorial-dotnet-sqldatabase



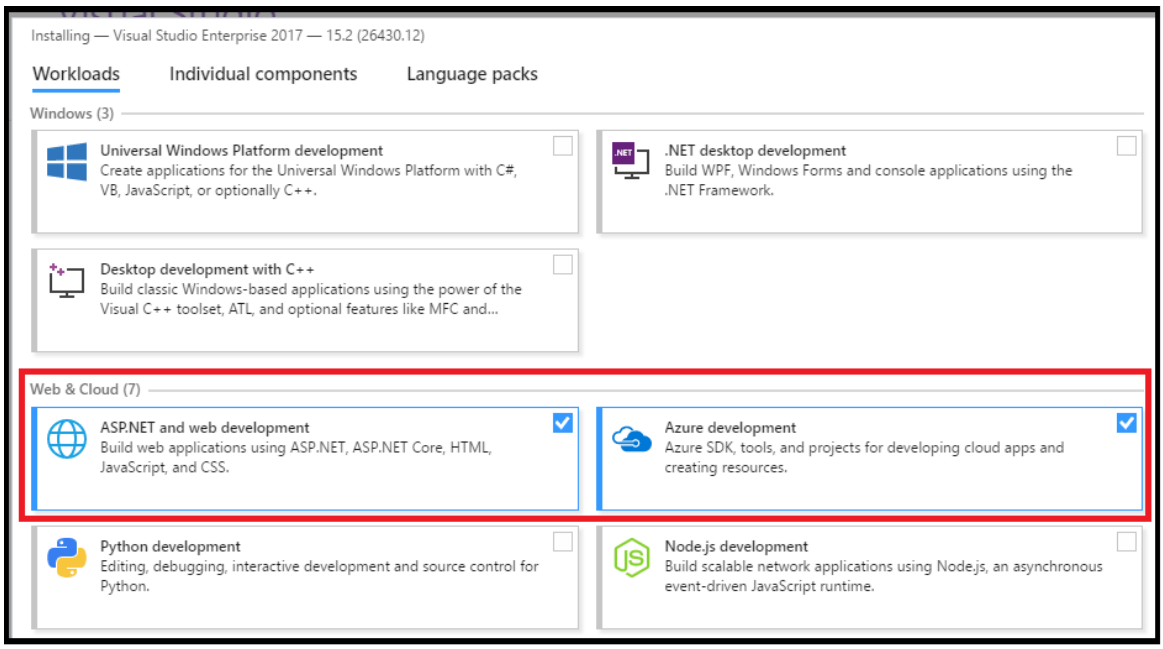
In this tutorial, you learn how to:

- Create a SQL Database in Azure
- Connect an ASP.NET app to SQL Database
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream logs from Azure to your terminal
- Manage the app in the Azure portal

Prerequisites

To complete this tutorial:

- Install Visual Studio 2017 with the following workloads:
  - **ASP.NET and web development**
  - **Azure development**



If you don't have an Azure subscription, create a free account before you begin.

Download the sample

Download the sample project.

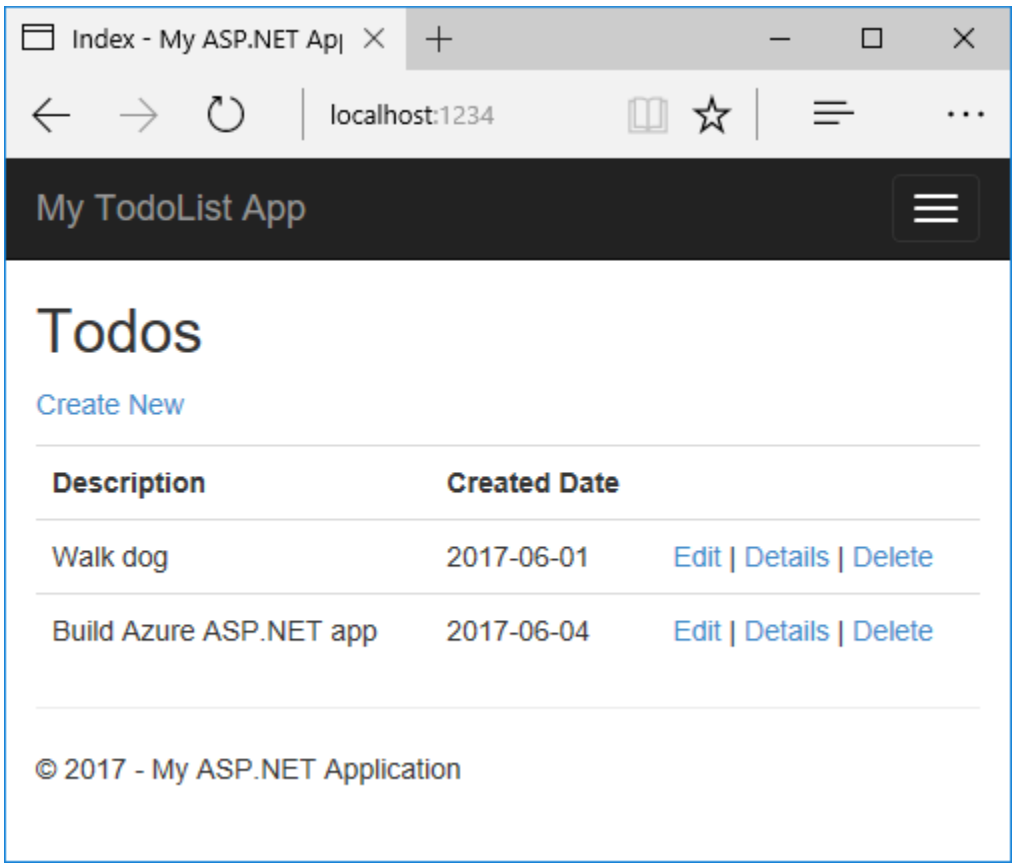
Extract (unzip) the *dotnet-sqldb-tutorial-master.zip* file.

The sample project contains a basic ASP.NET MVC CRUD (create-read-update-delete) app using Entity Framework Code First.

Run the app

Open the *dotnet-sqldb-tutorial-master/DotNetAppSqlDb.sln* file in Visual Studio.

Type `Ctrl+F5` to run the app without debugging. The app is displayed in your default browser. Select the **Create New** link and create a couple *to-do* items.

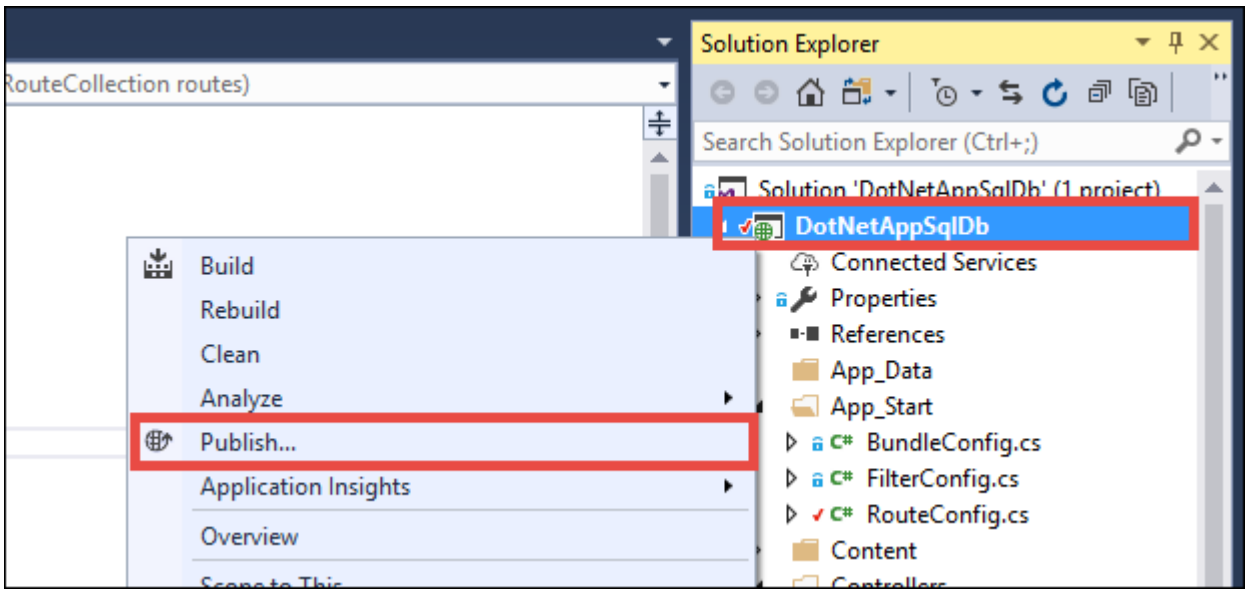


Test the **Edit**, **Details**, and **Delete** links.

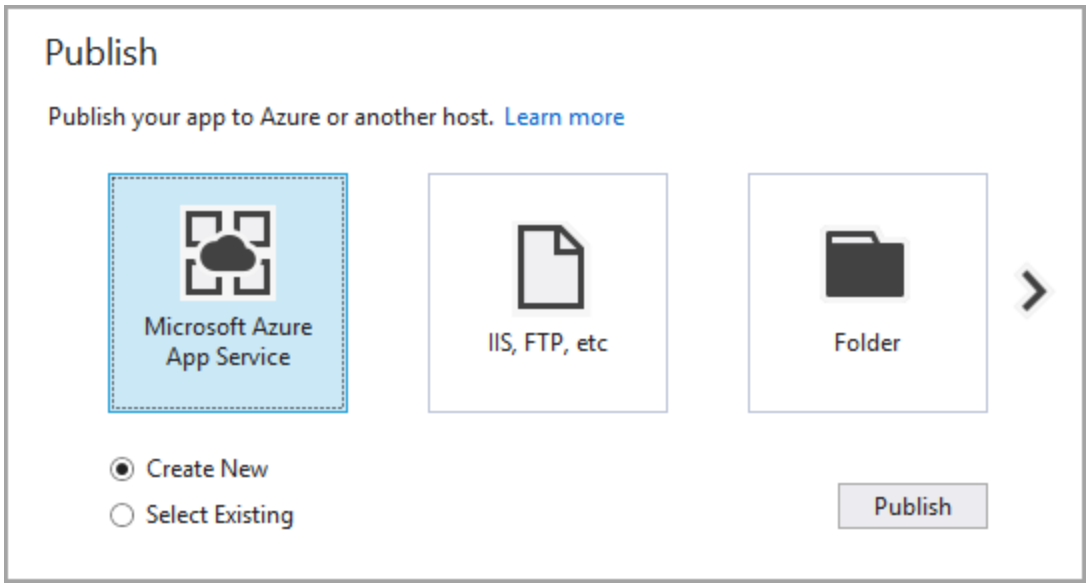
The app uses a database context to connect with the database. In this sample, the database context uses a connection string named `MyDbConnection`. The connection string is set in the *Web.config* file and referenced in the *Models/MyDatabaseContext.cs* file. The connection string name is used later in the tutorial to connect the Azure web app to an Azure SQL Database.

Publish to Azure with SQL Database

In the **Solution Explorer**, right-click your **DotNetAppSqlDb** project and select **Publish**.



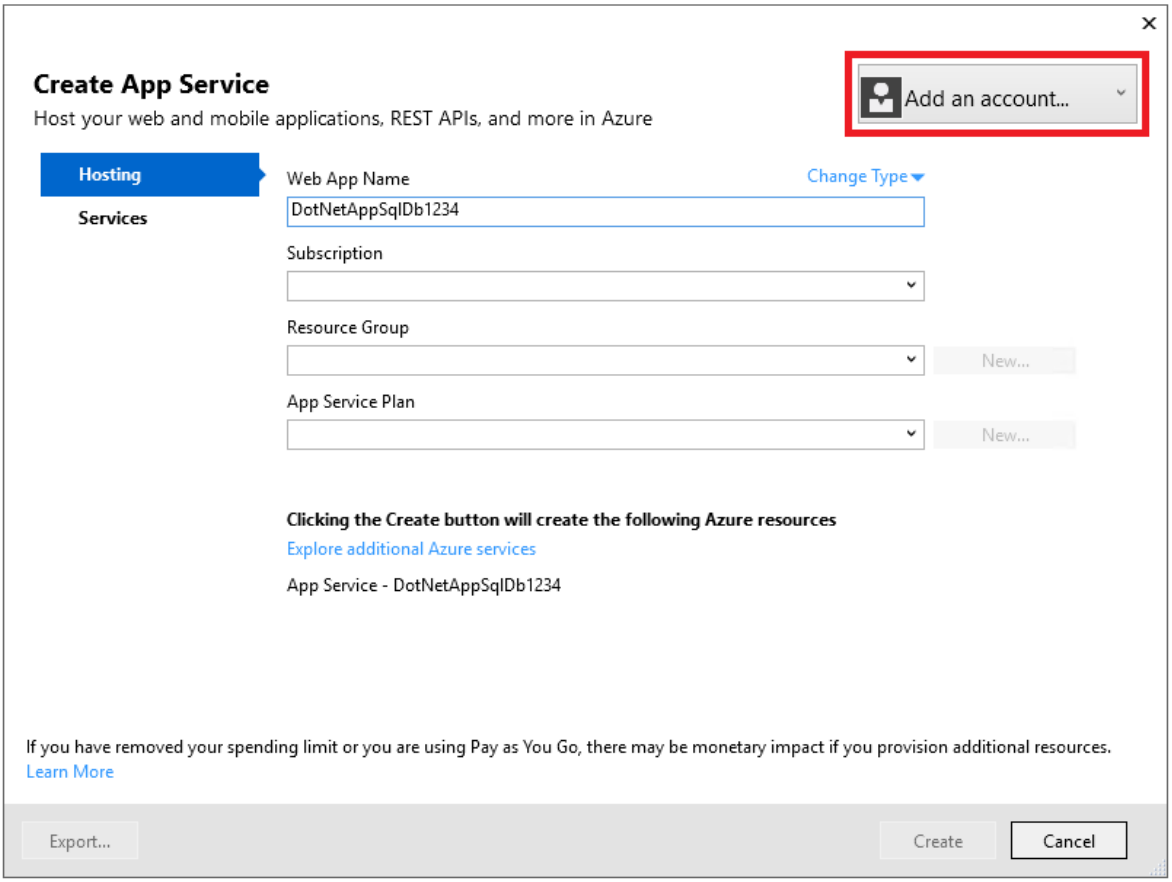
Make sure that **Microsoft Azure App Service** is selected and click **Publish**.



Publishing opens the **Create App Service** dialog, which helps you create all the Azure resources you need to run your ASP.NET web app in Azure.

Sign in to Azure

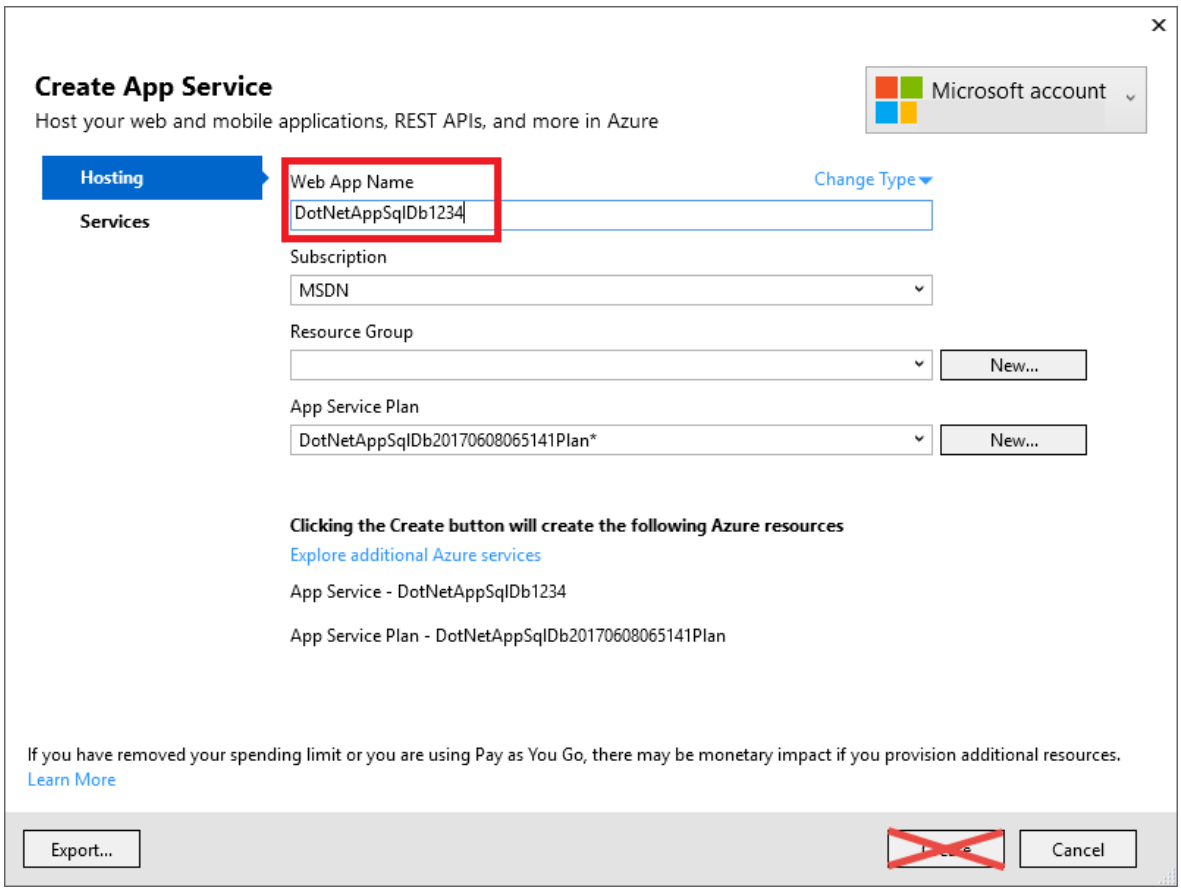
In the **Create App Service** dialog, click **Add an account**, and then sign in to your Azure subscription. If you're already signed into a Microsoft account, make sure that account holds your Azure subscription. If the signed-in Microsoft account doesn't have your Azure subscription, click it to add the correct account.



Once signed in, you're ready to create all the resources you need for your Azure web app in this dialog.

Configure the web app name

You can keep the generated web app name, or change it to another unique name (valid characters are `a-z`, `0-9`, and `-`). The web app name is used as part of the default URL for your app (`<app_name>.azurewebsites.net`, where `<app_name>` is your web app name). The web app name needs to be unique across all apps in Azure.



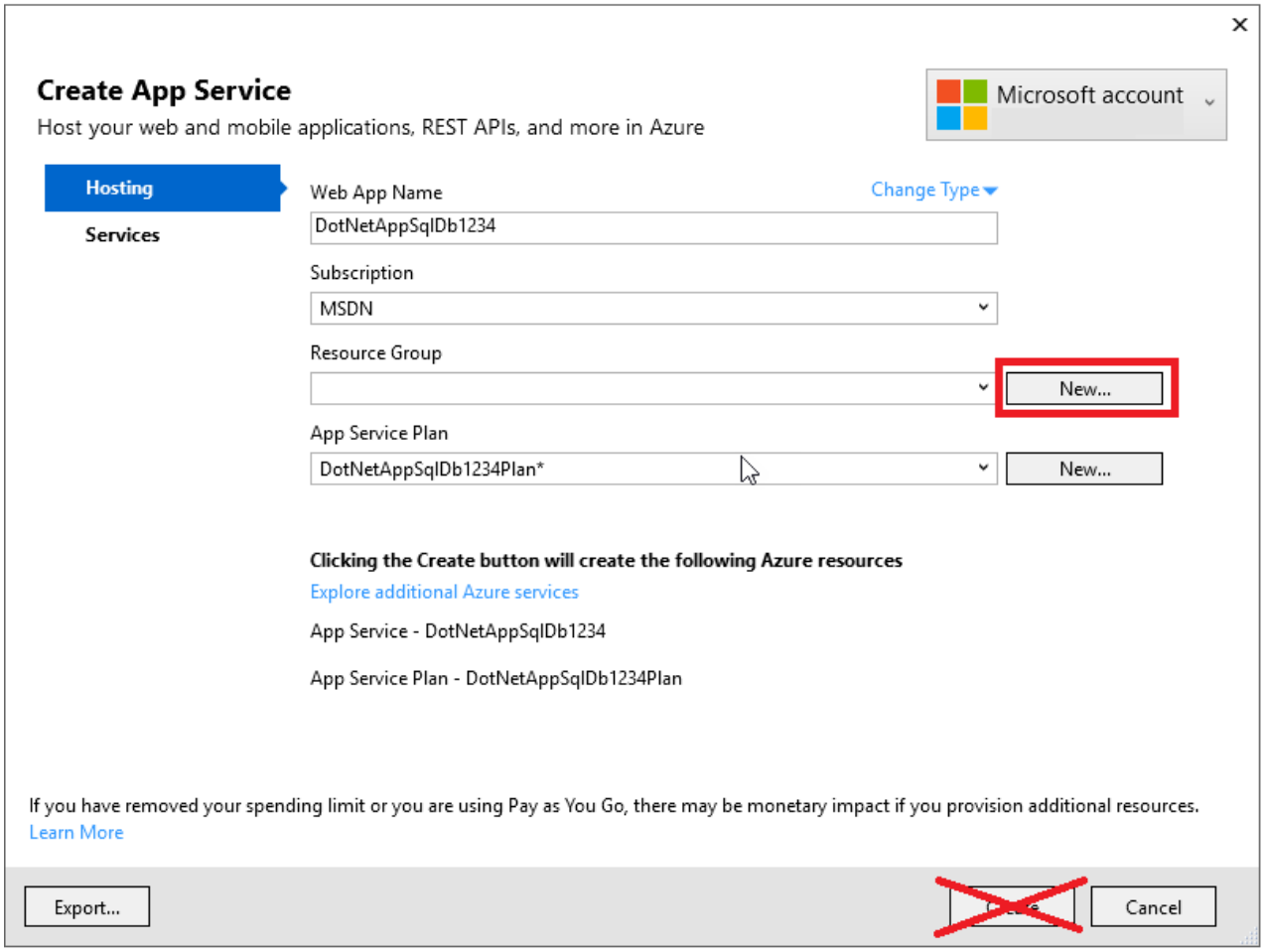
Note

Do not click **Create**. You first need to set up a SQL Database in a later step.

Create a resource group

A resource group is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed.

Next to **Resource Group**, click **New**.



Name the resource group **myResourceGroup**.

Create an App Service plan

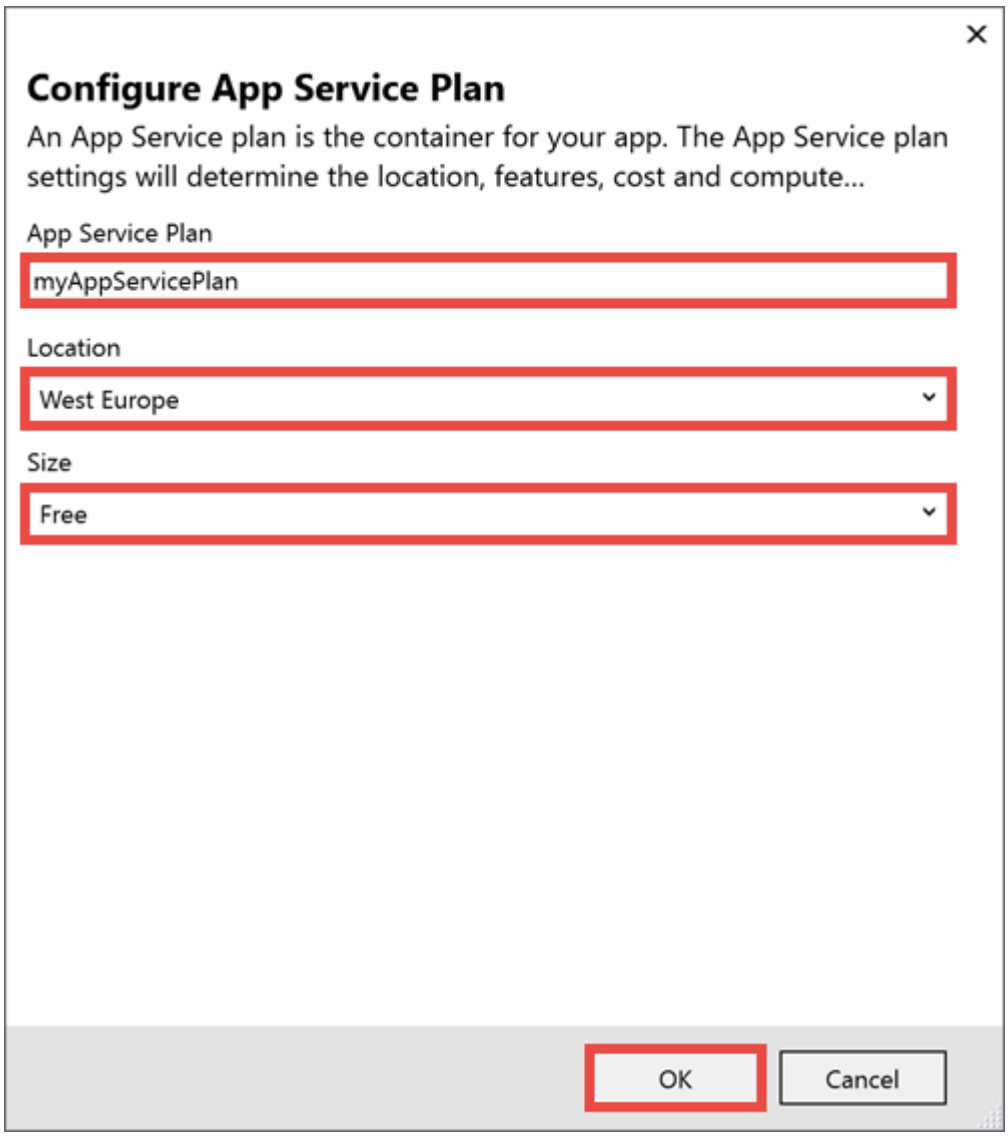
An App Service plan specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan.

App Service plans define:

- Region (for example: North Europe, East US, or Southeast Asia)
- Instance size (small, medium, or large)
- Scale count (1 to 20 instances)
- SKU (Free, Shared, Basic, Standard, or Premium)

Next to **App Service Plan**, click **New**.

In the **Configure App Service Plan** dialog, configure the new App Service plan with the following settings:

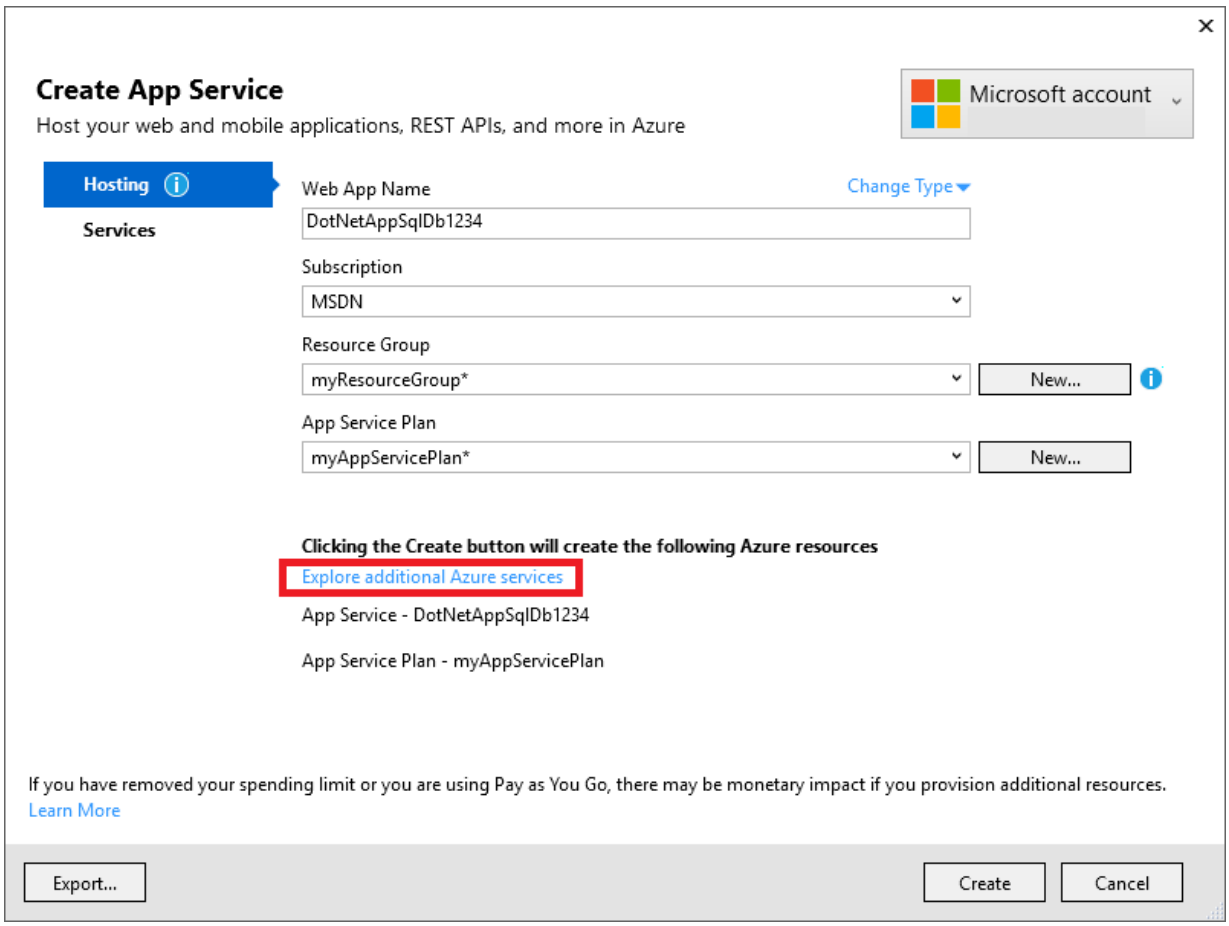


Setting	Suggested value	For more information
App Service Plan	myAppServicePlan	App Service plans
Location	West Europe	Azure regions
Size	Free	Pricing tiers

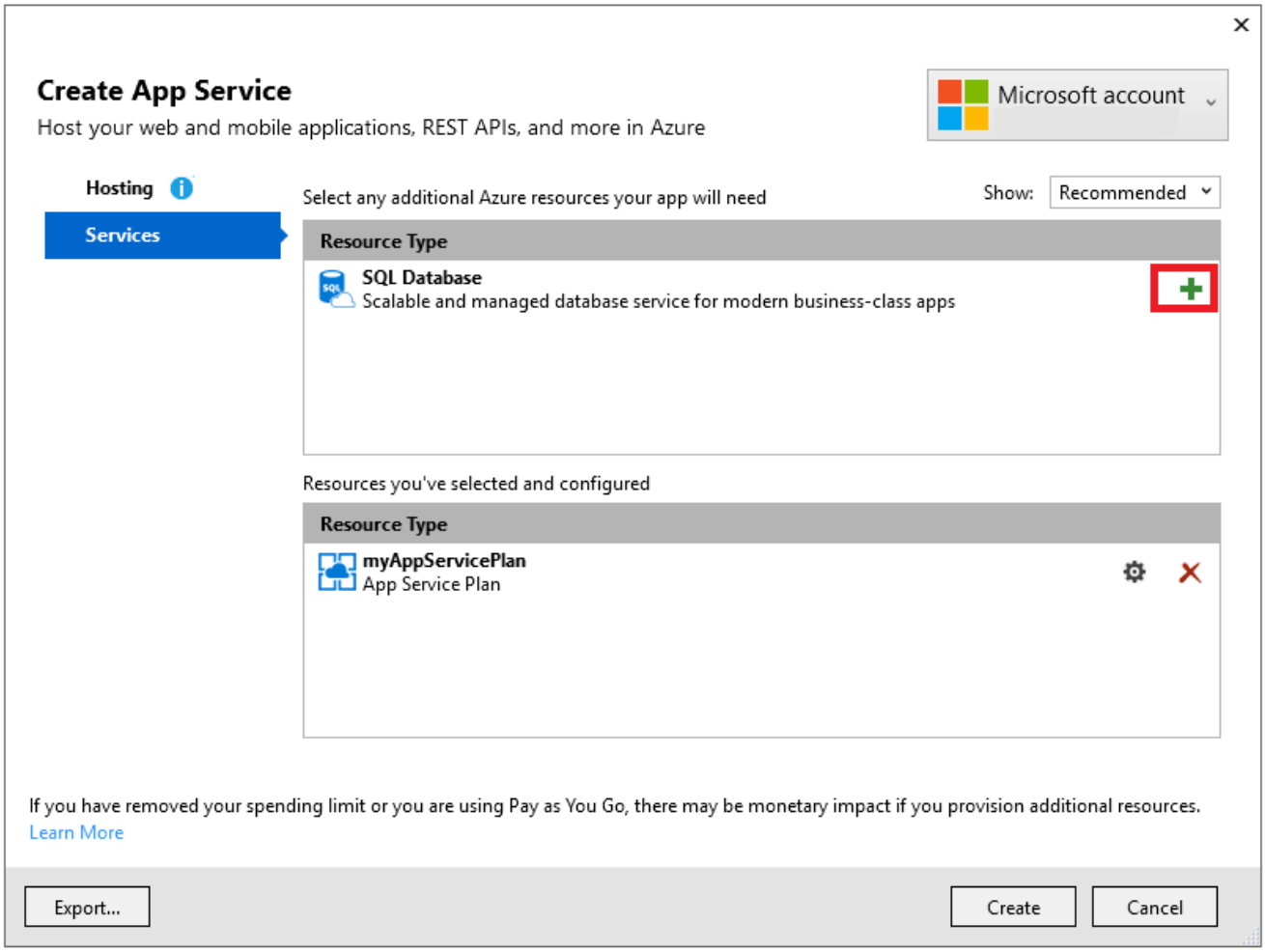
Create a SQL Server instance

Before creating a database, you need an Azure SQL Database logical server. A logical server contains a group of databases managed as a group.

Select **Explore additional Azure services**.



In the **Services** tab, click the **+** icon next to **SQL Database**.

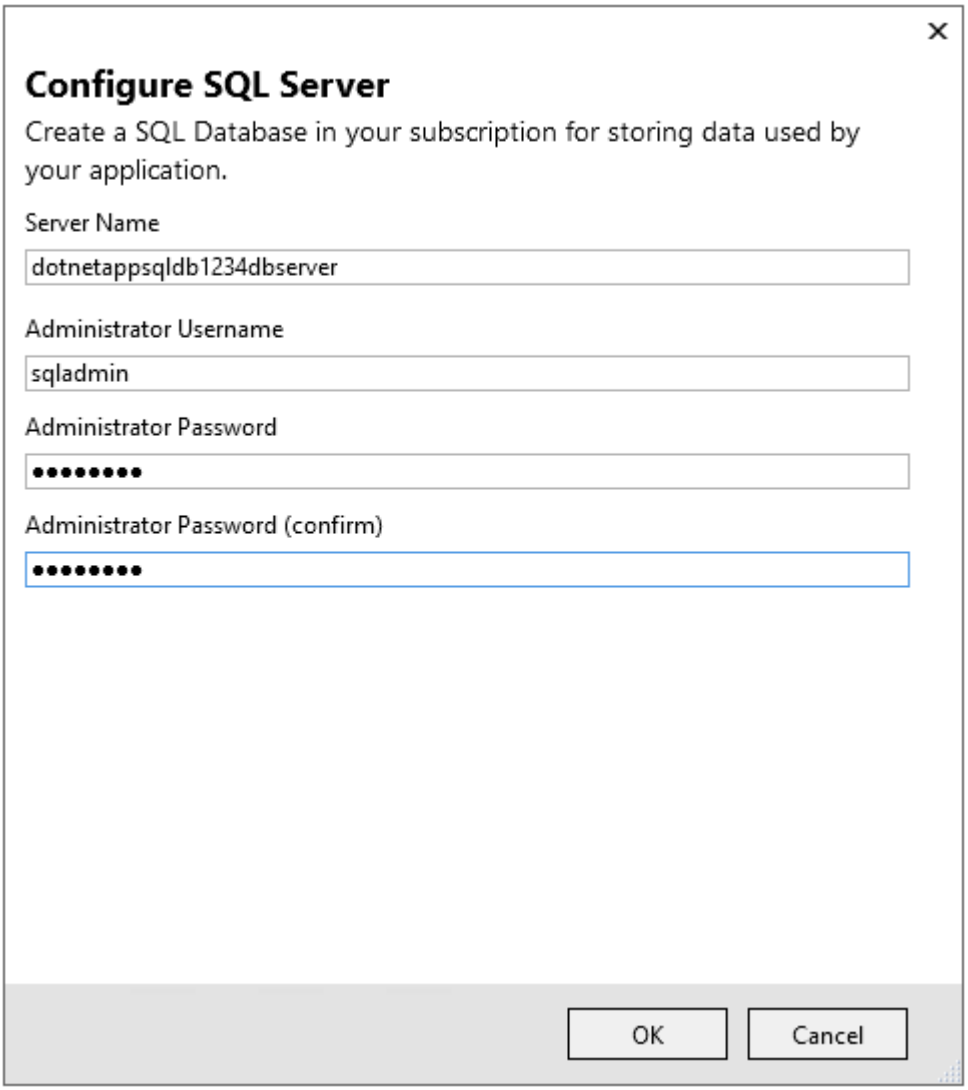


In the **Configure SQL Database** dialog, click **New** next to **SQL Server**.

A unique server name is generated. This name is used as part of the default URL for your logical server, `<server_name>.database.windows.net`. It must be unique across all logical server instances in Azure. You can change the server name, but for this tutorial, keep the generated value.

Add an administrator username and password. For password complexity requirements, see Password Policy.

Remember this username and password. You need them to manage the logical server instance later.

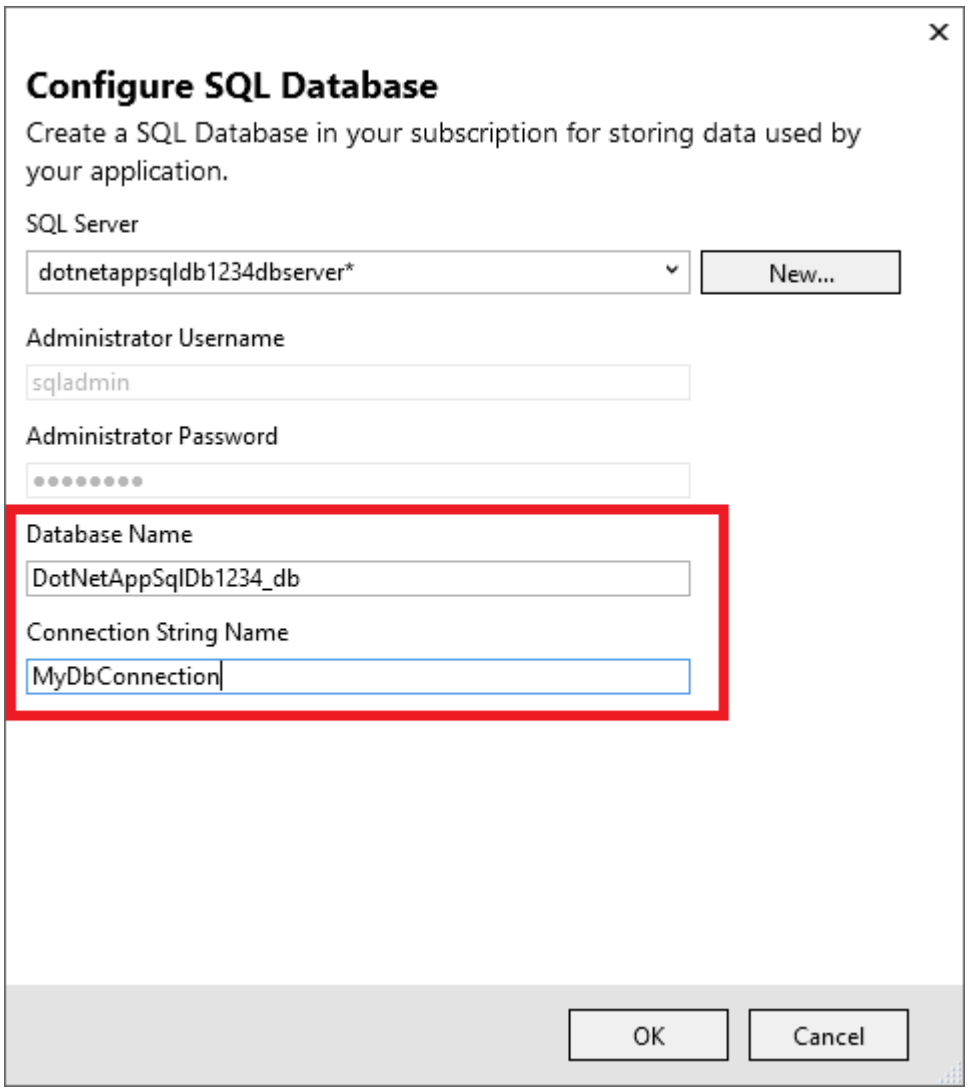


Click **OK**. Don't close the **Configure SQL Database** dialog yet.

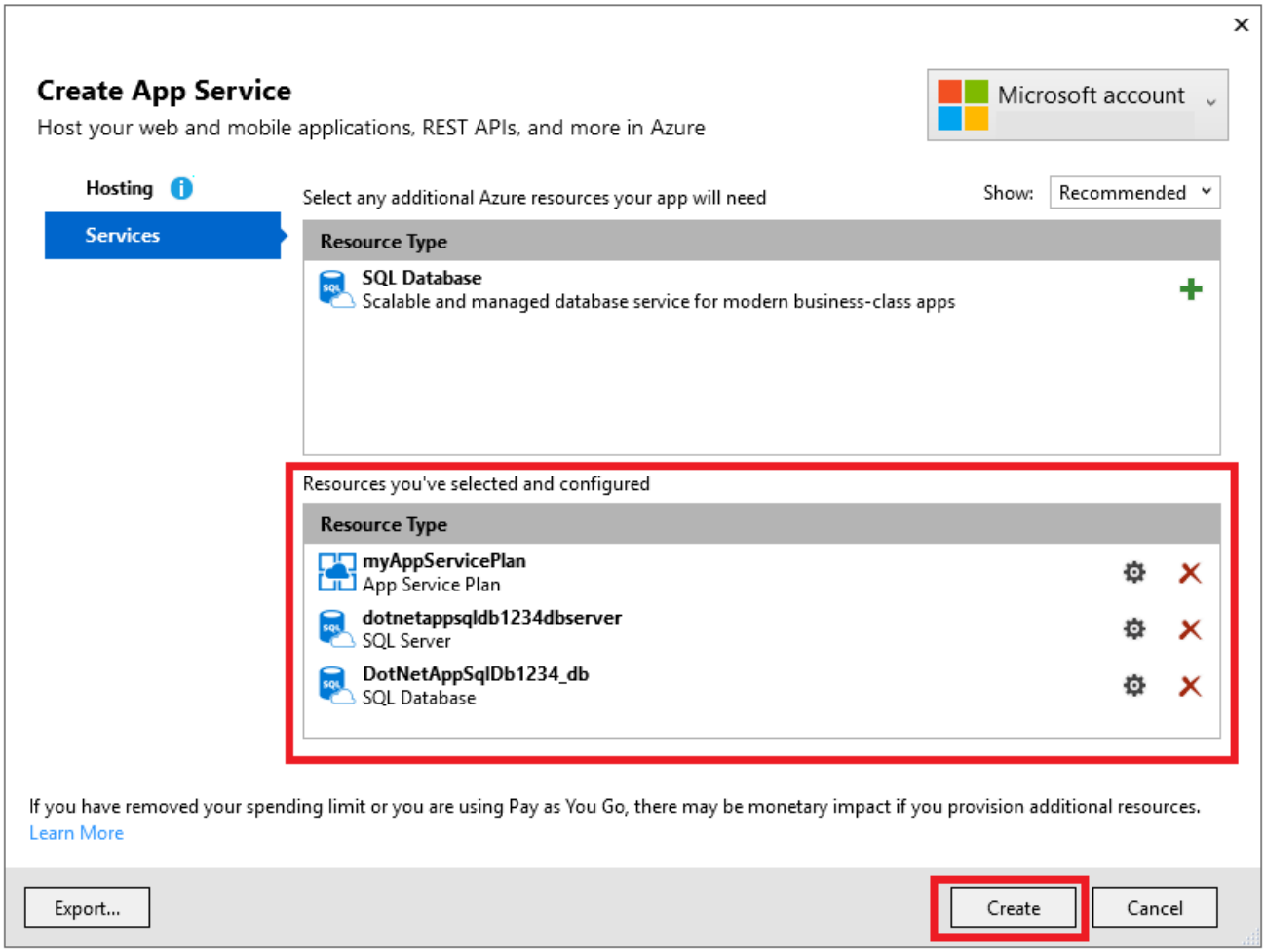
Create a SQL Database

In the **Configure SQL Database** dialog:

- Keep the default generated **Database Name**.
- In **Connection String Name**, type *MyDbConnection*. This name must match the connection string that is referenced in *Models/MyDatabaseContext.cs*.
- Select **OK**.

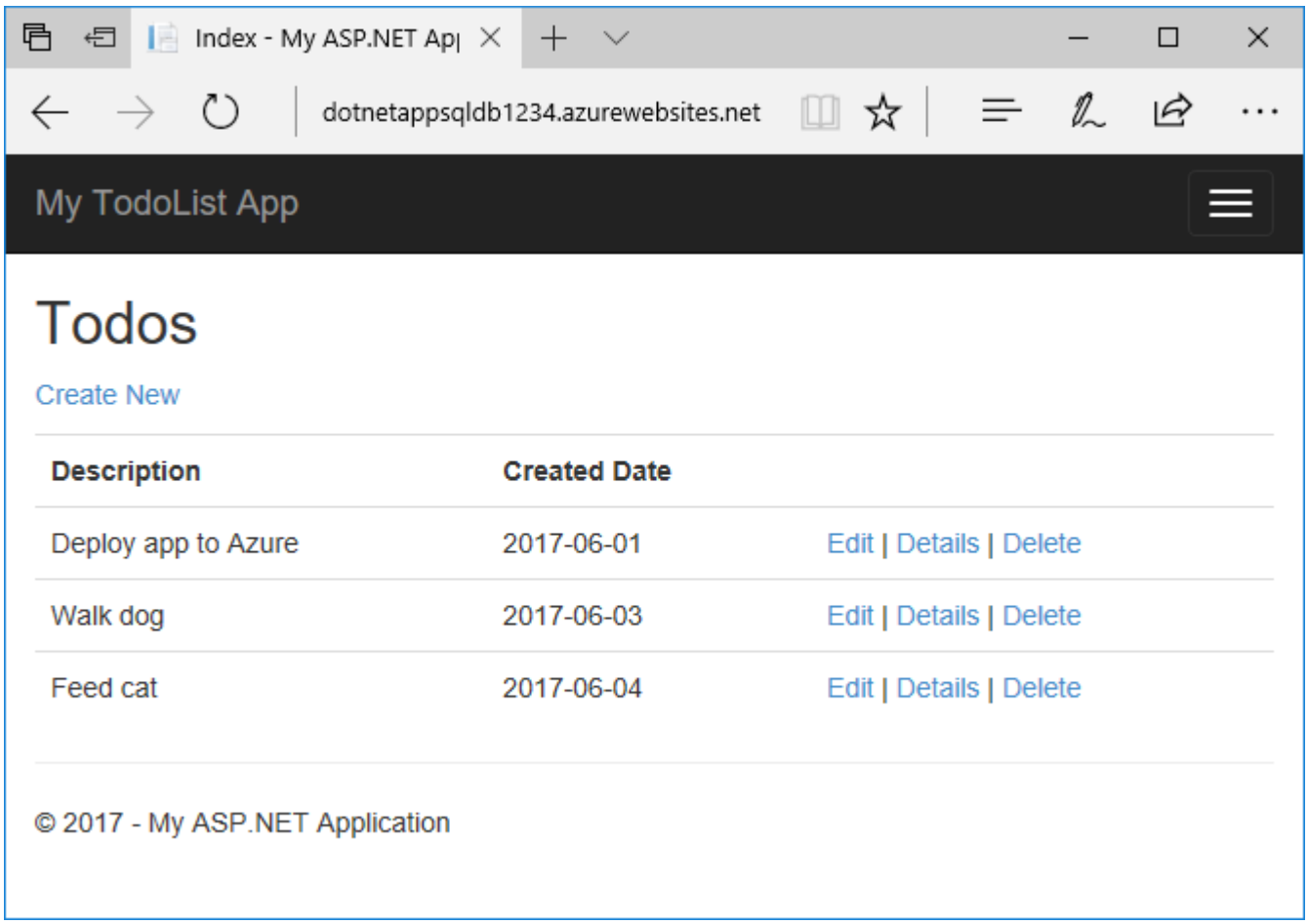


The **Create App Service** dialog shows the resources you've created. Click **Create**.



Once the wizard finishes creating the Azure resources, it publishes your ASP.NET app to Azure. Your default browser is launched with the URL to the deployed app.

Add a few to-do items.



Congratulations! Your data-driven ASP.NET application is running live in Azure App Service.

Access the SQL Database locally

Visual Studio lets you explore and manage your new SQL Database easily in the **SQL Server Object Explorer**.

Create a database connection

From the **View** menu, select **SQL Server Object Explorer**.

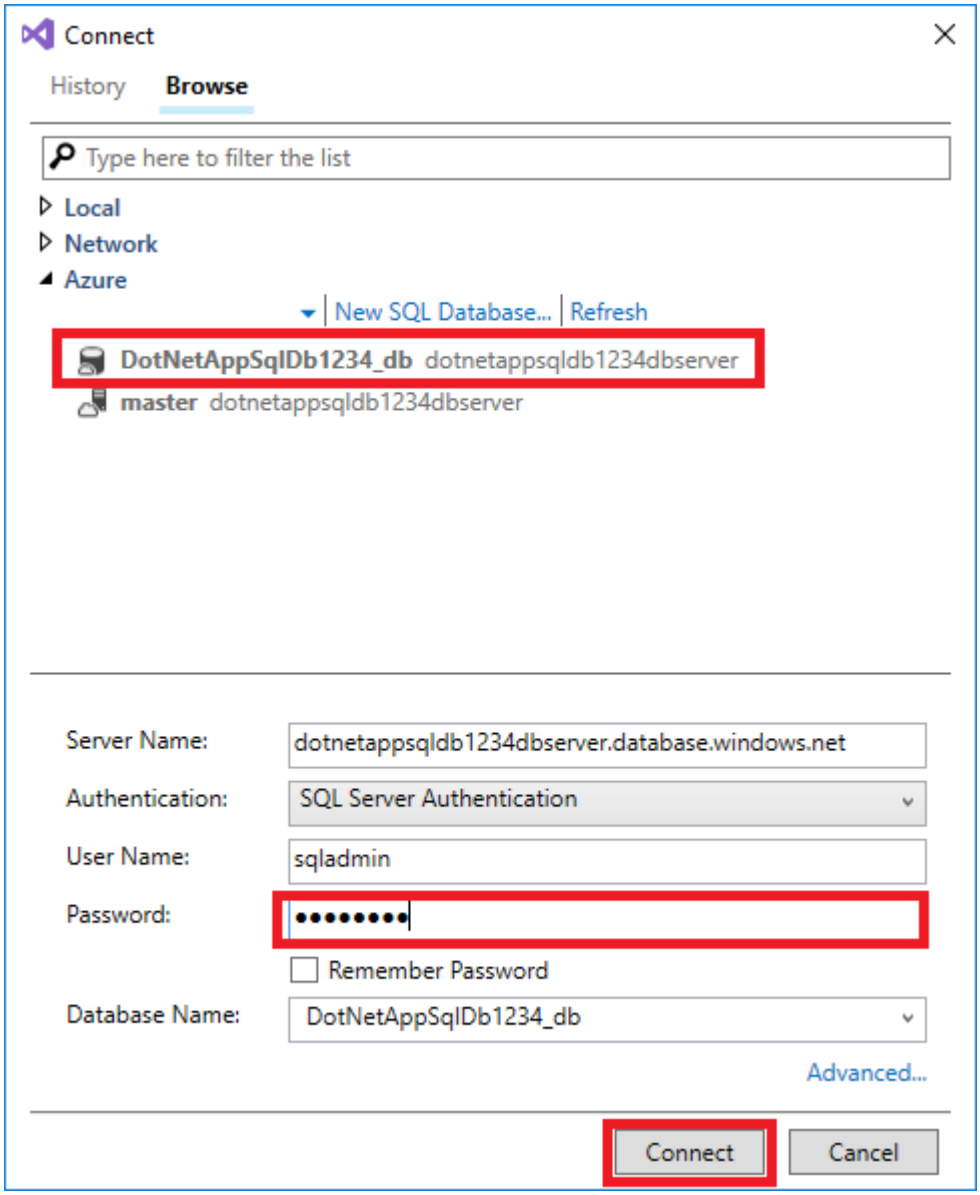
At the top of **SQL Server Object Explorer**, click the **Add SQL Server** button.

Configure the database connection

In the **Connect** dialog, expand the **Azure** node. All your SQL Database instances in Azure are listed here.

Select the SQL Database that you created earlier. The connection you created earlier is automatically filled at the bottom.

Type the database administrator password you created earlier and click **Connect**.

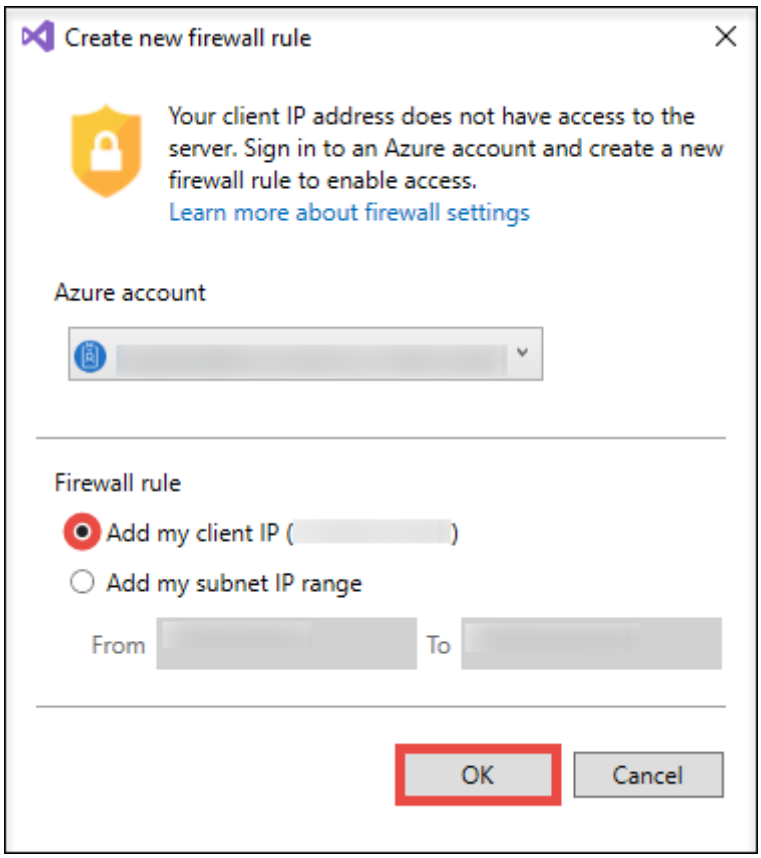


Allow client connection from your computer

The **Create a new firewall rule** dialog is opened. By default, your SQL Database instance only allows connections from Azure services, such as your Azure web app. To connect to your database, create a firewall rule in the SQL Database instance. The firewall rule allows the public IP address of your local computer.

The dialog is already filled with your computer's public IP address.

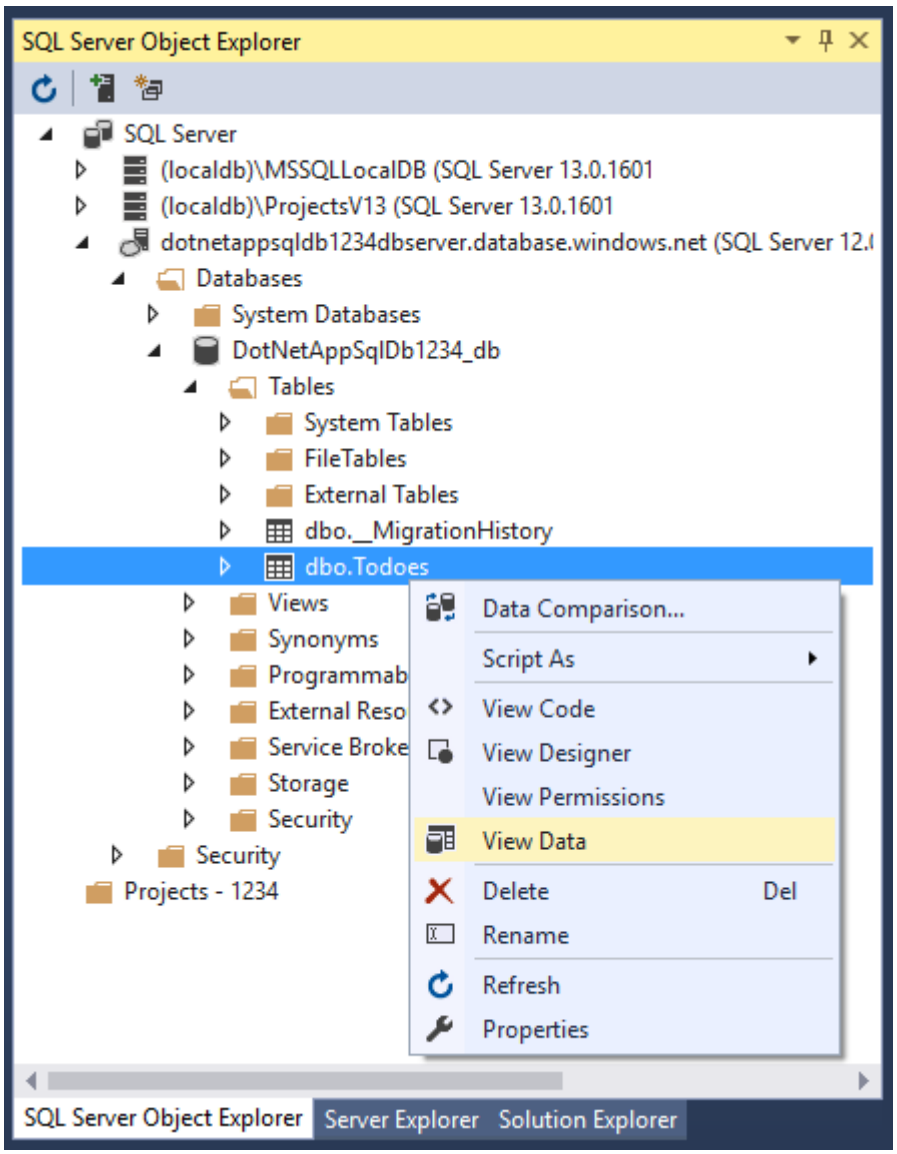
Make sure that **Add my client IP** is selected and click **OK**.



Once Visual Studio finishes creating the firewall setting for your SQL Database instance, your connection shows up in **SQL Server Object Explorer**.

Here, you can perform the most common database operations, such as run queries, create views and stored procedures, and more.

Expand your connection > **Databases** > **<your database>** > **Tables**. Right-click on the `Todoes` table and select **View Data**.



Update app with Code First Migrations

You can use the familiar tools in Visual Studio to update your database and web app in Azure. In this step, you use Code First Migrations in Entity Framework to make a change to your database schema and publish it to Azure.

For more information about using Entity Framework Code First Migrations, see [Getting Started with Entity Framework 6 Code First using MVC 5](#).

Update your data model

Open *Models\Todo.cs* in the code editor. Add the following property to the `ToDo` class:

C#

```
public bool Done { get; set; }
```

Run Code First Migrations locally

Run a few commands to make updates to your local database.

From the **Tools** menu, click **NuGet Package Manager** > **Package Manager Console**.

In the Package Manager Console window, enable Code First Migrations:

PowerShell

```
Enable-Migrations
```

Add a migration:

PowerShell

```
Add-Migration AddProperty
```

Update the local database:

PowerShell

```
Update-Database
```

Type `Ctrl+F5` to run the app. Test the edit, details, and create links.

If the application loads without errors, then Code First Migrations has succeeded. However, your page still looks the same because your application logic is not using this new property yet.

Use the new property

Make some changes in your code to use the `Done` property. For simplicity in this tutorial, you're only going to change the `Index` and `Create` views to see the property in action.

Open *Controllers\TodosController.cs*.

Find the `Create()` method on line 52 and add `Done` to the list of properties in the `Bind` attribute. When you're done, your `Create()` method signature looks like the following code:

C#

```
public ActionResult Create([Bind(Include = "Description,CreatedDate,Done")] Todo todo)
```

Open *Views\Todos\Create.cshtml*.

In the Razor code, you should see a `<div class="form-group">` element that uses `model.Description`, and then another `<div class="form-group">` element that uses `model.CreatedDate`. Immediately following these two elements, add another `<div class="form-group">` element that uses `model.Done`:

C#

```
<div class="form-group">
    @Html.LabelFor(model => model.Done, htmlAttributes: new { @class = "control-label
col-md-2" })
    <div class="col-md-10">
        <div class="checkbox">
            @Html.EditorFor(model => model.Done)
            @Html.ValidationMessageFor(model => model.Done, "", new { @class = "text-
danger" })
        </div>
    </div>
</div>
```

Open *Views\Todos\Index.cshtml*.

Search for the empty `<th></th>` element. Just above this element, add the following Razor code:

C#

```
<th>
    @Html.DisplayNameFor(model => model.Done)
</th>
```

Find the `<td>` element that contains the `Html.ActionLink()` helper methods. *Above* this `<td>`, add another `<td>` element with the following Razor code:

C#

```
<td>
    @Html.DisplayFor(modelItem => item.Done)
</td>
```

That's all you need to see the changes in the `Index` and `Create` views.

Type `Ctrl+F5` to run the app.

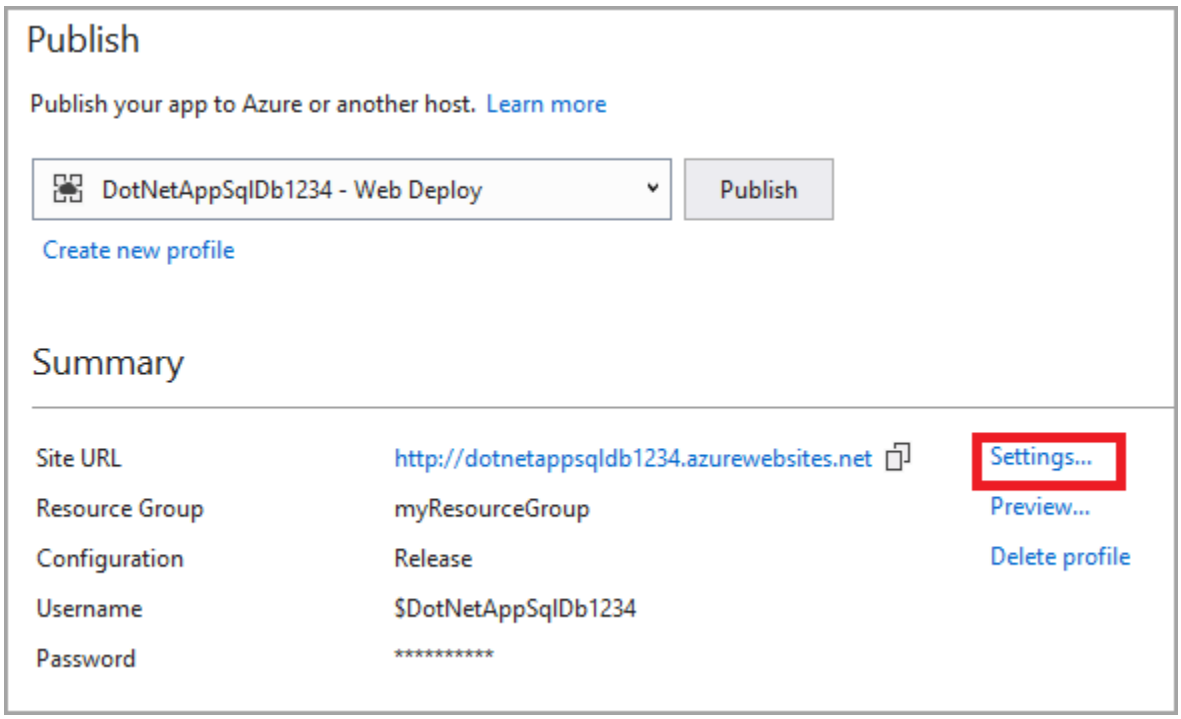
You can now add a to-do item and check **Done**. Then it should show up in your homepage as a completed item. Remember that the `Edit` view doesn't show the `Done` field, because you didn't change the `Edit` view.

Enable Code First Migrations in Azure

Now that your code change works, including database migration, you publish it to your Azure web app and update your SQL Database with Code First Migrations too.

Just like before, right-click your project and select **Publish**.

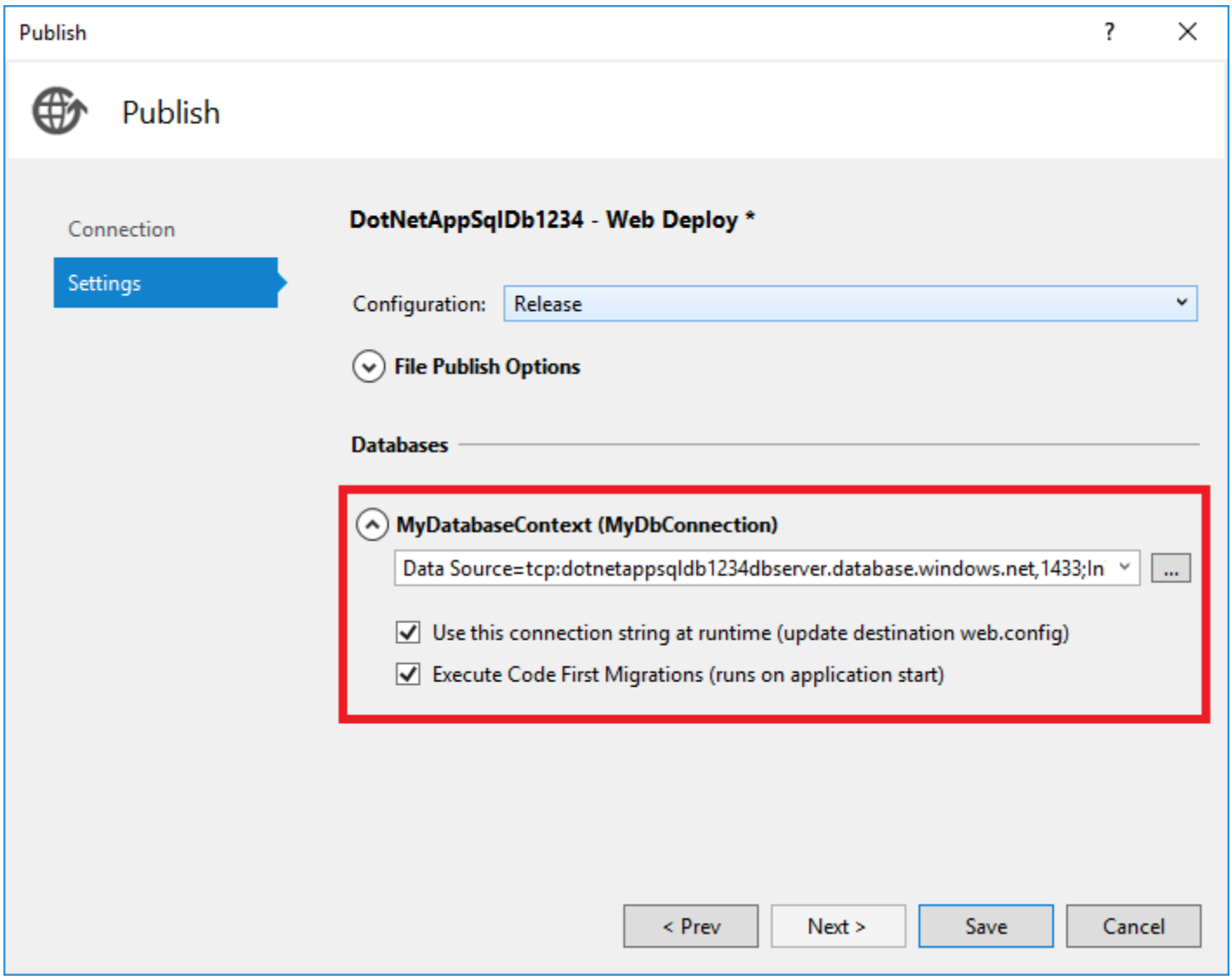
Click **Settings** to open the publish wizard.



In the wizard, click **Next**.

Make sure that the connection string for your SQL Database is populated in **MyDatabaseContext (MyDbConnection)**. You may need to select the **myToDoAppDb** database from the dropdown.

Select **Execute Code First Migrations (runs on application start)**, then click **Save**.

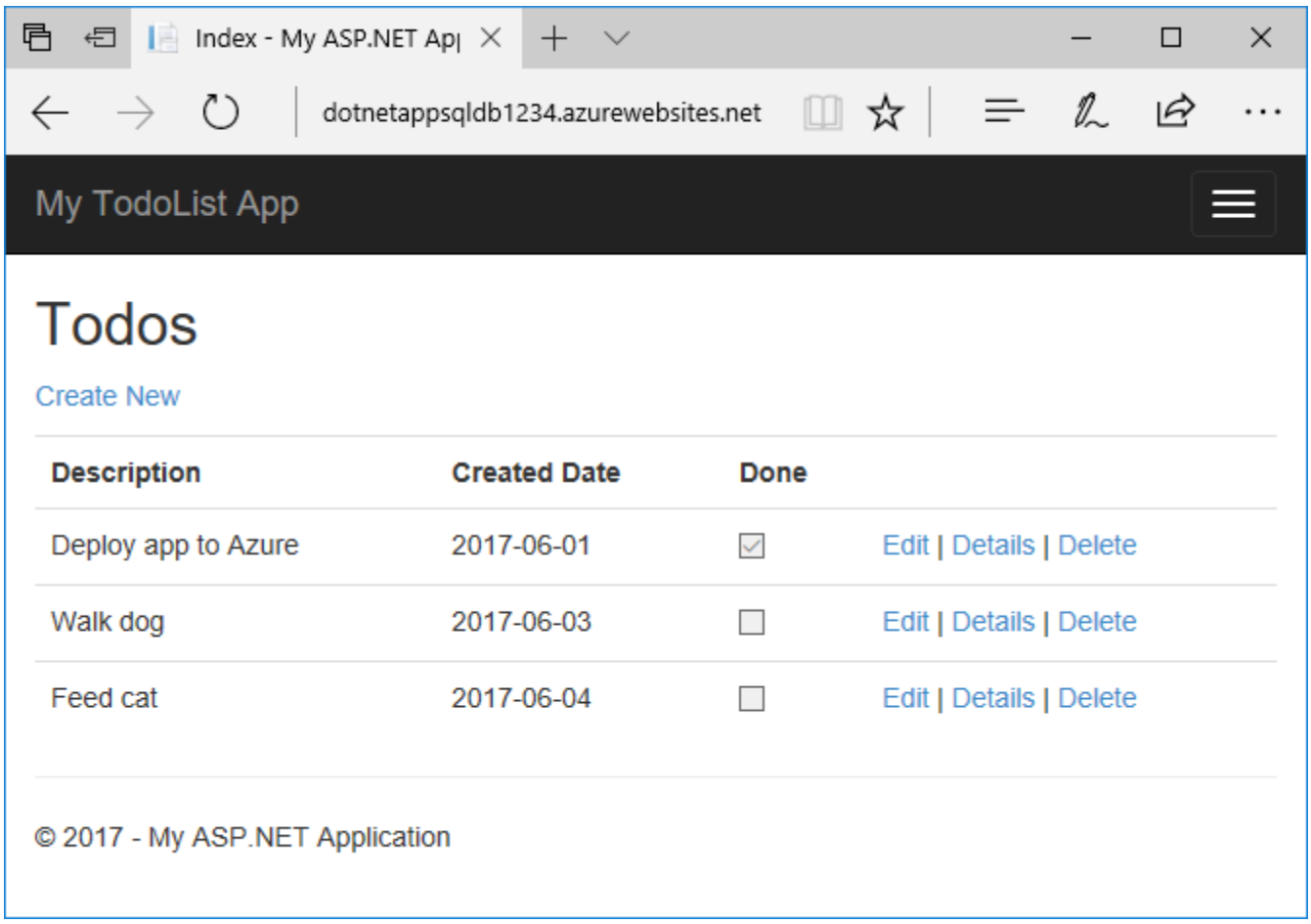


Publish your changes

Now that you enabled Code First Migrations in your Azure web app, publish your code changes.

In the publish page, click **Publish**.

Try adding to-do items again and select **Done**, and they should show up in your homepage as a completed item.



All your existing to-do items are still displayed. When you republish your ASP.NET application, existing data in your SQL Database is not lost. Also, Code First Migrations only changes the data schema and leaves your existing data intact.

Stream application logs

You can stream tracing messages directly from your Azure web app to Visual Studio.

Open *Controllers\TodosController.cs*.

Each action starts with a `Trace.WriteLine()` method. This code is added to show you how to add trace messages to your Azure web app.

Open Server Explorer

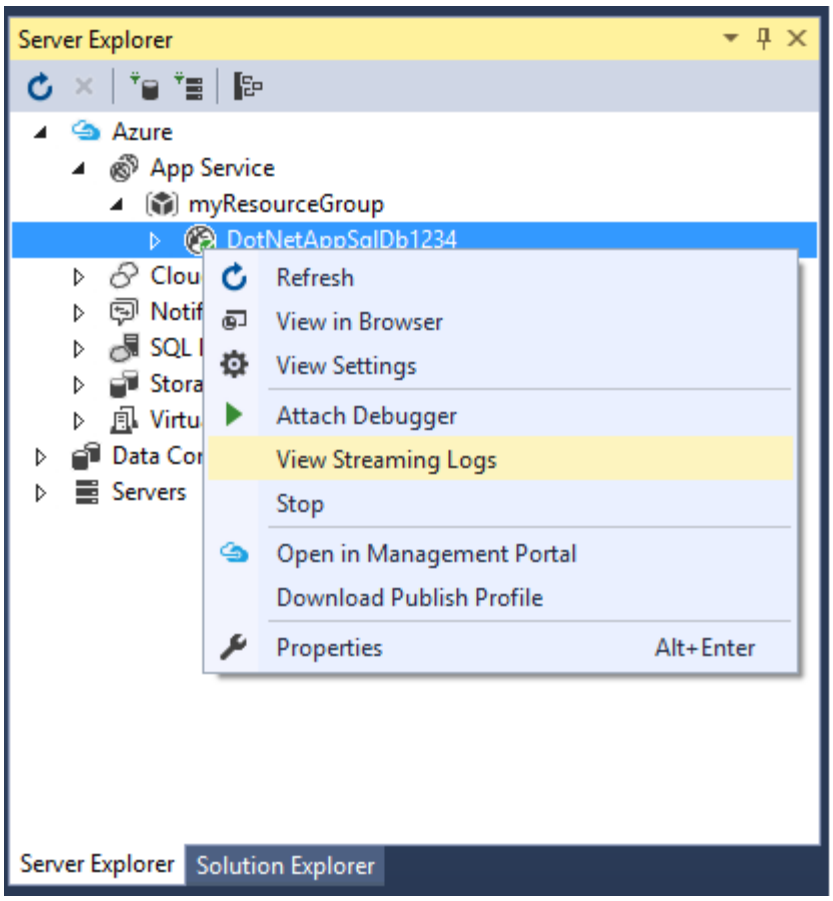
From the **View** menu, select **Server Explorer**. You can configure logging for your Azure web app in **Server Explorer**.

Enable log streaming

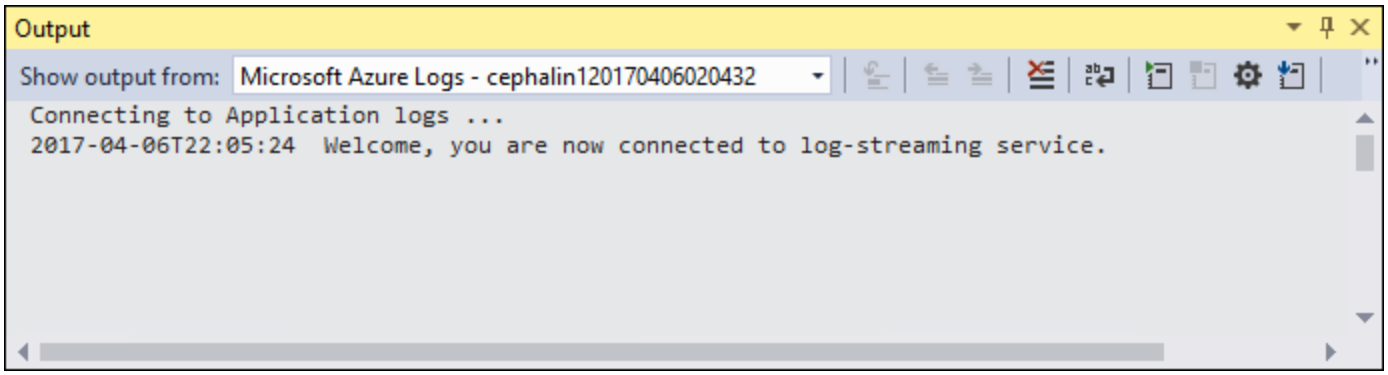
In **Server Explorer**, expand **Azure** > **App Service**.

Expand the **myResourceGroup** resource group, you created when you first created the Azure web app.

Right-click your Azure web app and select **View Streaming Logs**.



The logs are now streamed into the **Output** window.



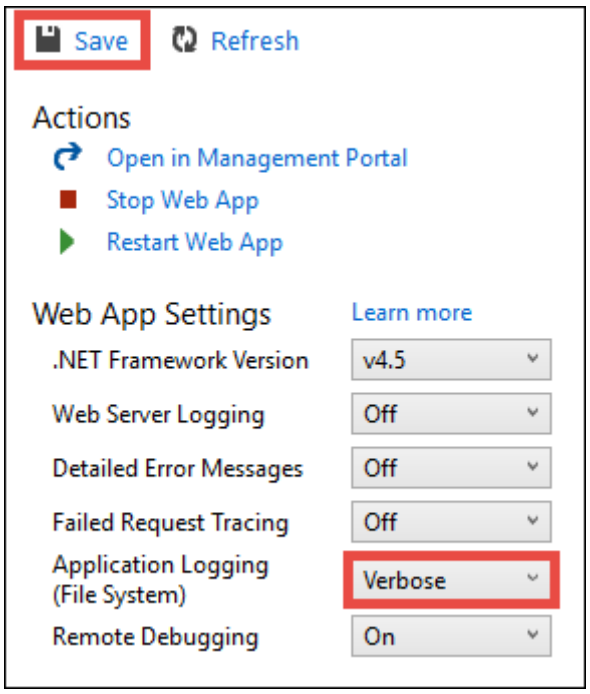
However, you don't see any of the trace messages yet. That's because when you first select **View Streaming Logs**, your Azure web app sets the trace level to `Error`, which only logs error events (with the `Trace.TraceError()` method).

Change trace levels

To change the trace levels to output other trace messages, go back to **Server Explorer**.

Right-click your Azure web app again and select **View Settings**.

In the **Application Logging (File System)** dropdown, select **Verbose**. Click **Save**.



Tip

You can experiment with different trace levels to see what types of messages are displayed for each level. For example, the **Information** level includes all logs created by `Trace.TraceInformation()`, `Trace.TraceWarning()`, and `Trace.TraceError()`, but not logs created by `Trace.WriteLine()`.

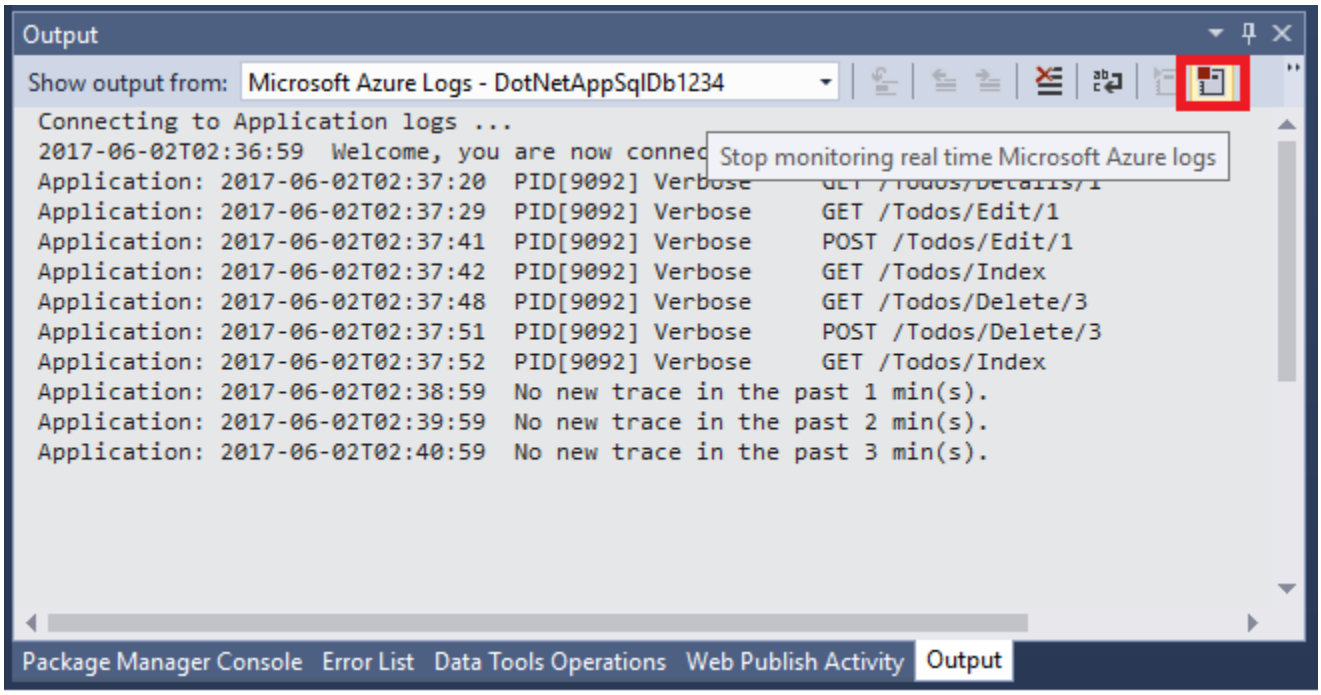
In your browser navigate to your web app again at *http://<your app name>.azurewebsites.net*, then try clicking around the to-do list application in Azure. The trace messages are now streamed to the **Output** window in Visual Studio.

console

```
Application: 2017-04-06T23:30:41  PID[8132] Verbose     GET /Todos/Index
Application: 2017-04-06T23:30:43  PID[8132] Verbose     GET /Todos/Create
Application: 2017-04-06T23:30:53  PID[8132] Verbose     POST /Todos/Create
Application: 2017-04-06T23:30:54  PID[8132] Verbose     GET /Todos/Index
```

Stop log streaming

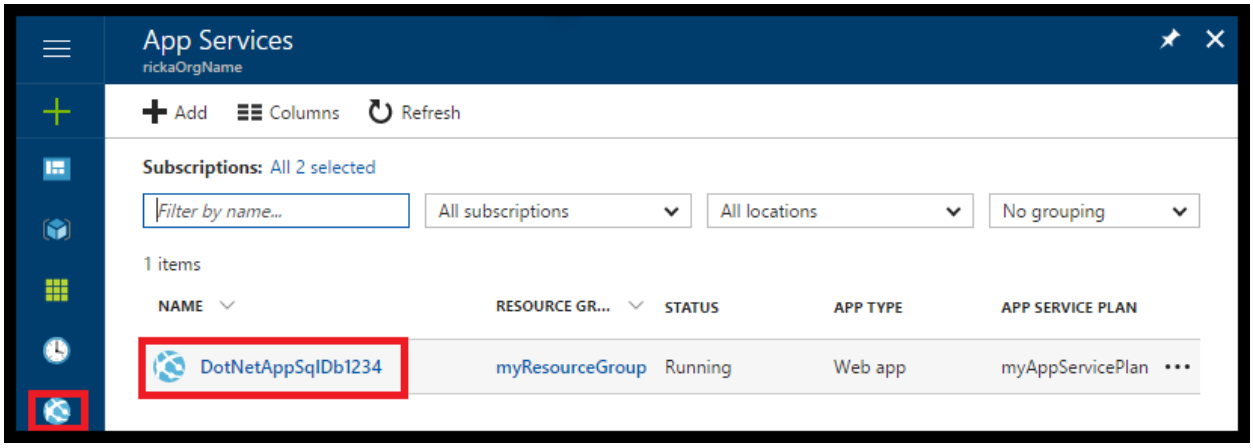
To stop the log-streaming service, click the **Stop monitoring** button in the **Output** window.



Manage your Azure web app

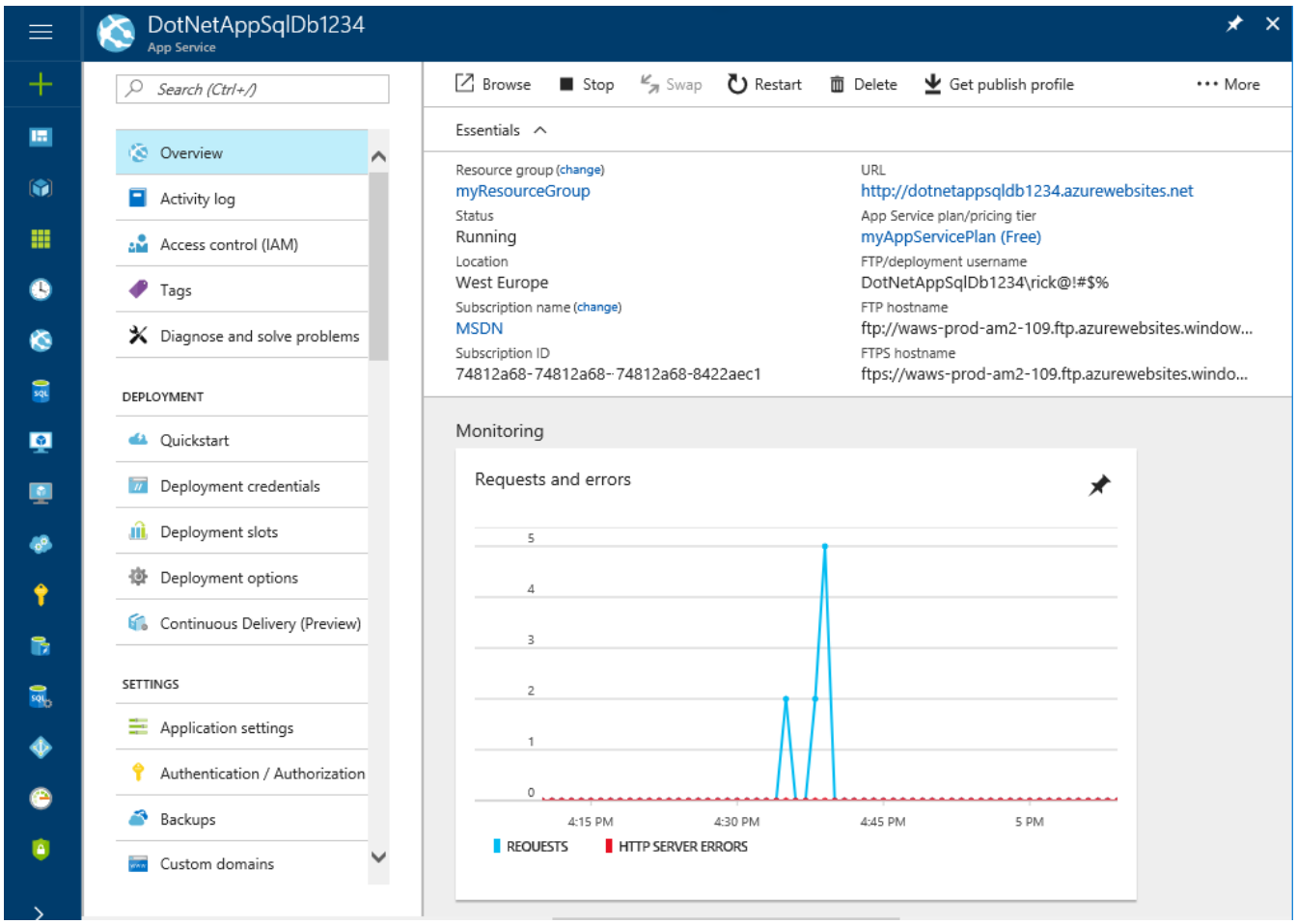
Go to the Azure portal to see the web app you created.

From the left menu, click **App Service**, then click the name of your Azure web app.



You have landed in your web app's page.

By default, the portal shows the **Overview** page. This page gives you a view of how your app is doing. Here, you can also perform basic management tasks like browse, stop, start, restart, and delete. The tabs on the left side of the page show the different configuration pages you can open.



Clean up resources

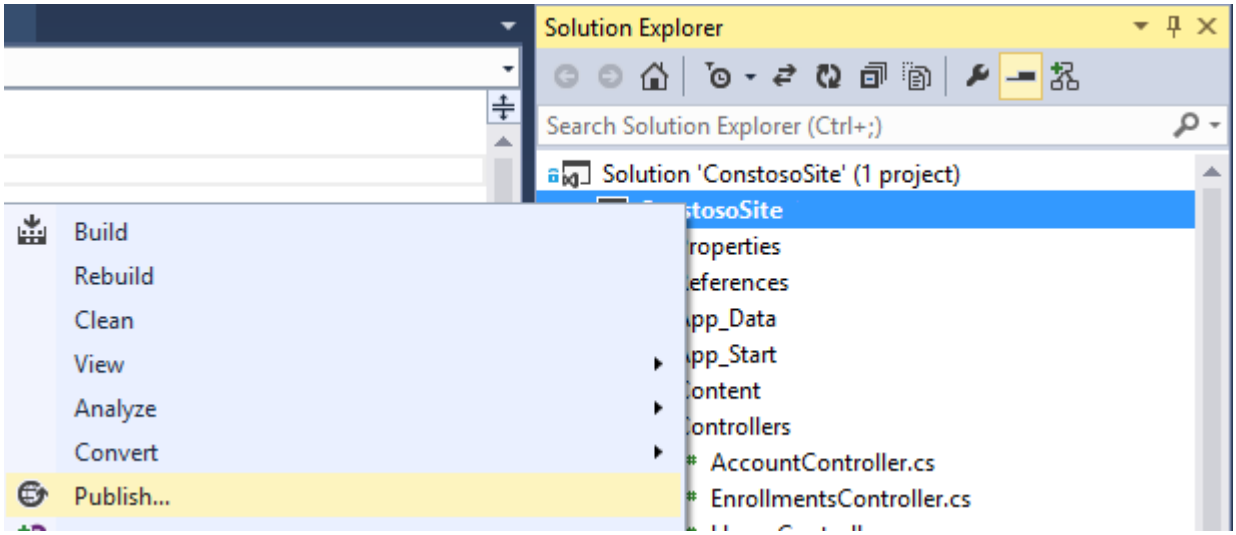
In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, you can delete them by deleting the resource group.

1. From your web app's **Overview** page in the Azure portal, select the **myResourceGroup** link under **Resource group**.
2. On the resource group page, make sure that the listed resources are the ones you want to delete.
3. Select **Delete**, type **myResourceGroup** in the text box, and then select **Delete**.

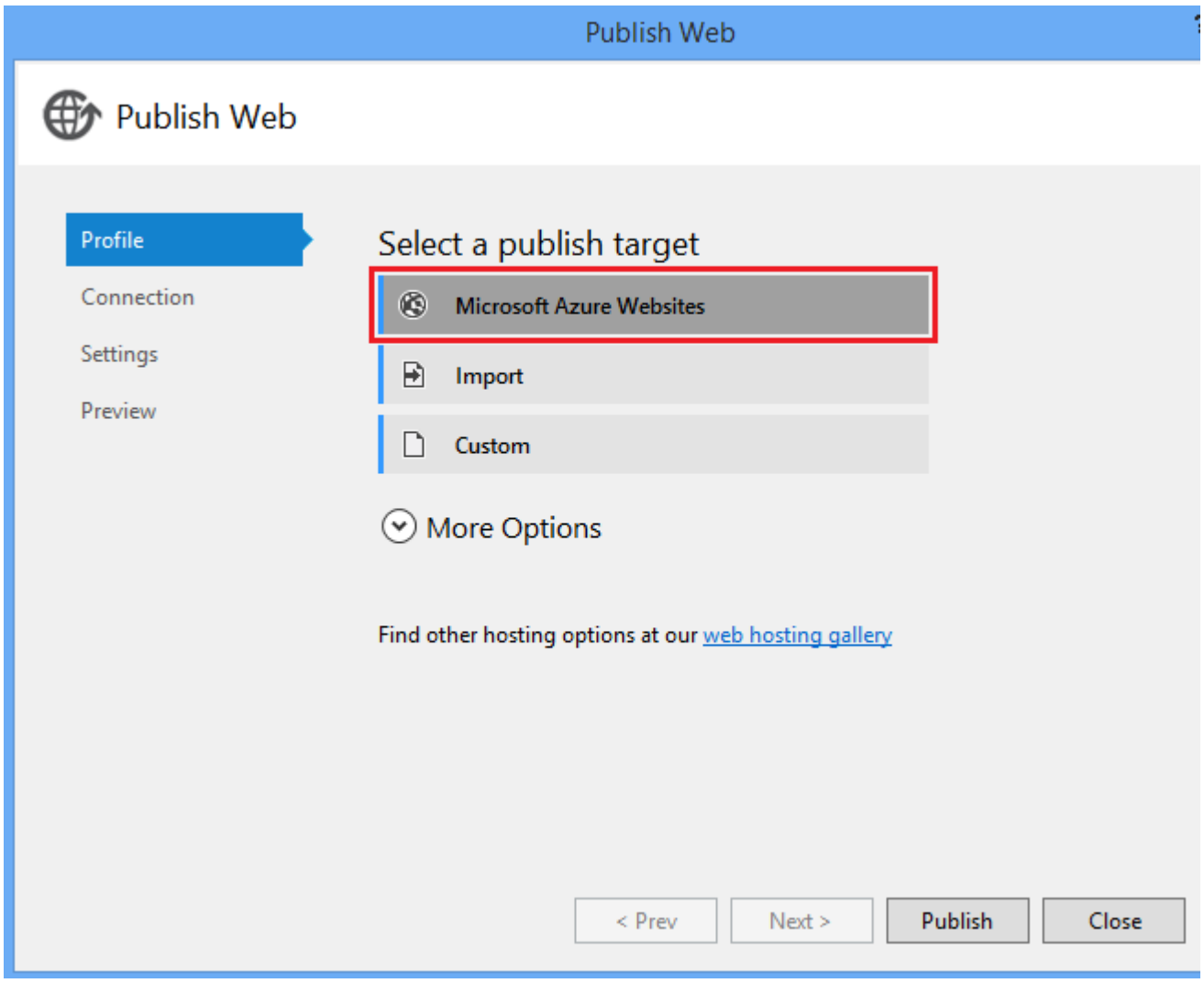
End 2017 Lab

2014 - Deploy a Database With a Web Application Project

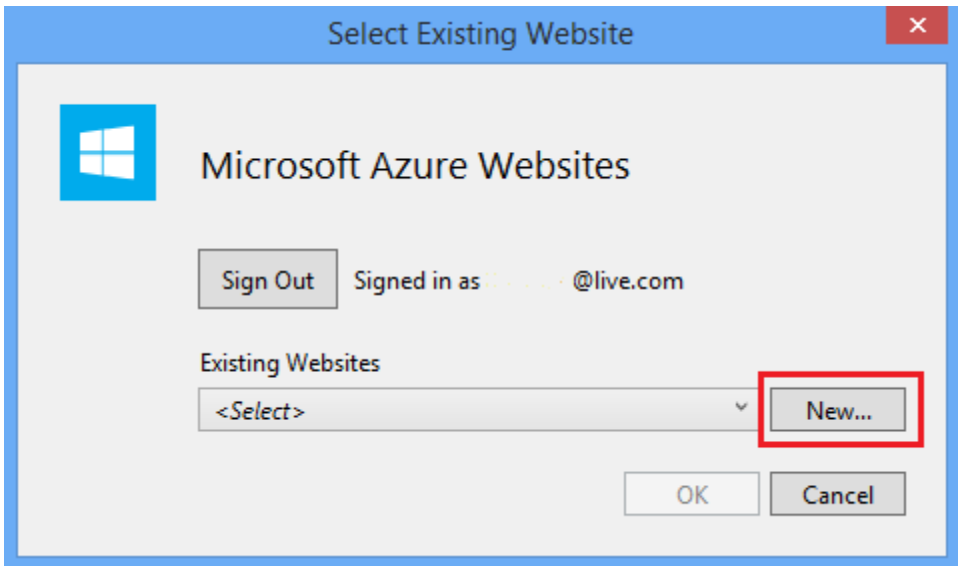
To publish your web app, right-click the project and select **Publish**.



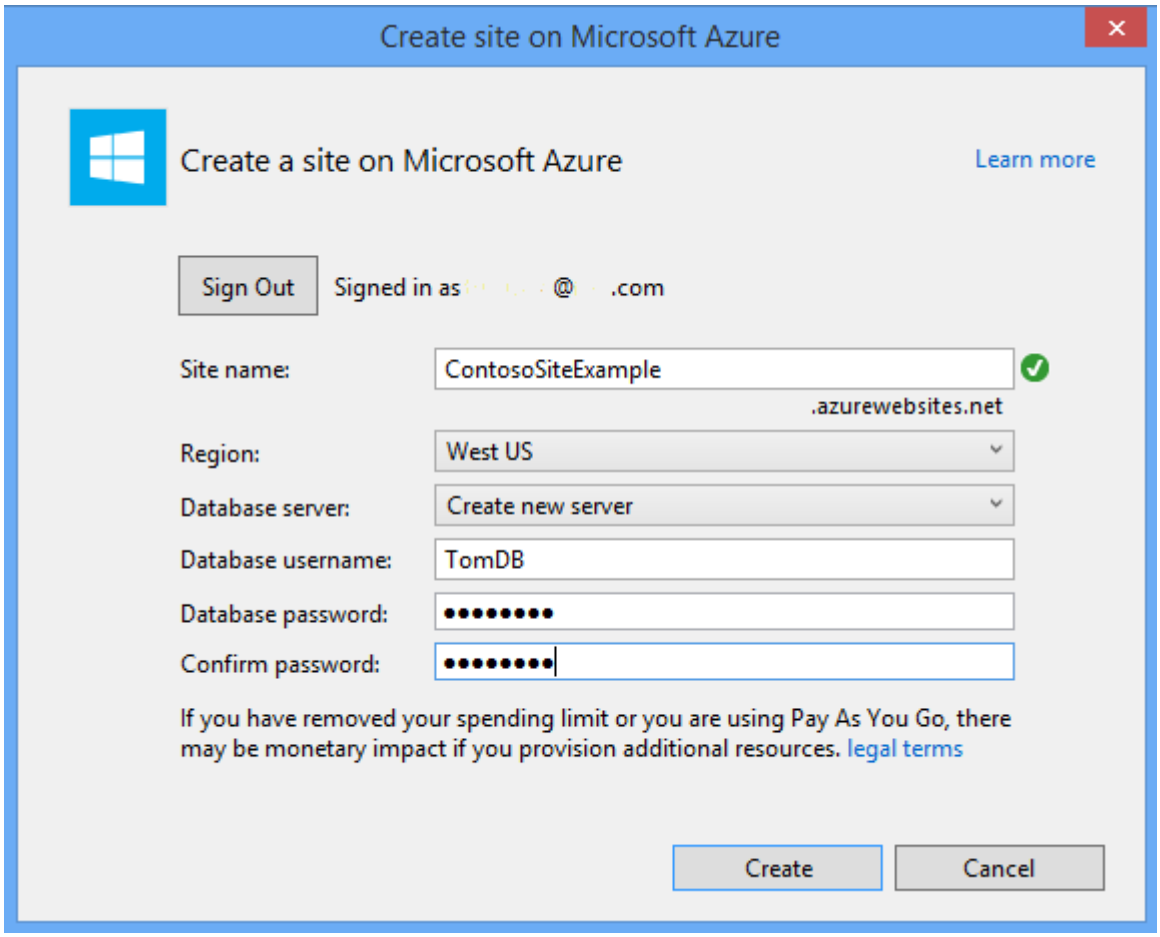
Select Microsoft Azure Websites.



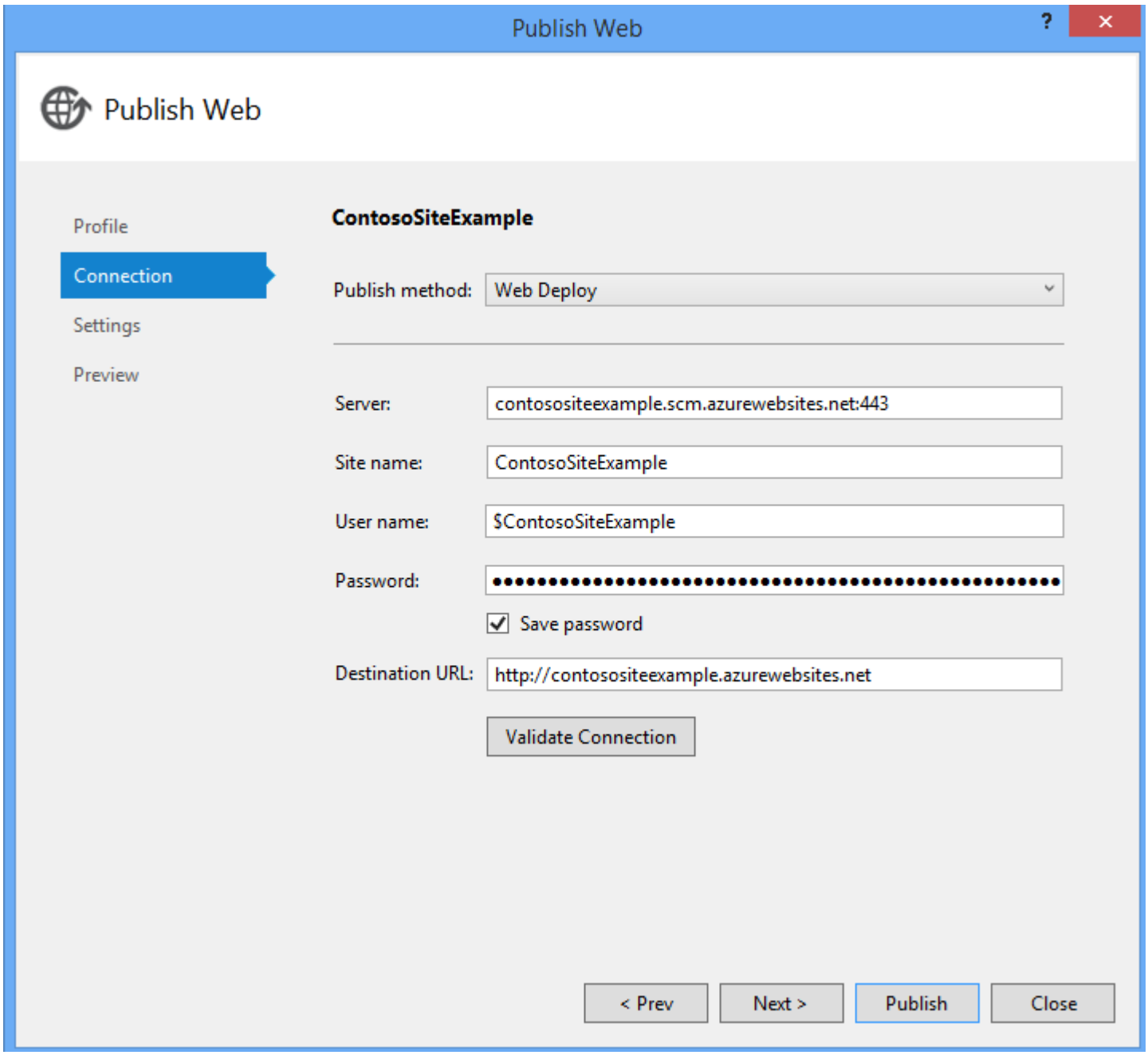
If you are not signed in to Azure, provide your Azure account credentials. Then, select New to create a new web app.



Create a unique name for your web app. You will know the name is unique if you see a green check mark to the right of the name field. Select a region for your web app. Select **Create new server** for the database, and provide a user name and password for this new database server. When finished, click **Create**.

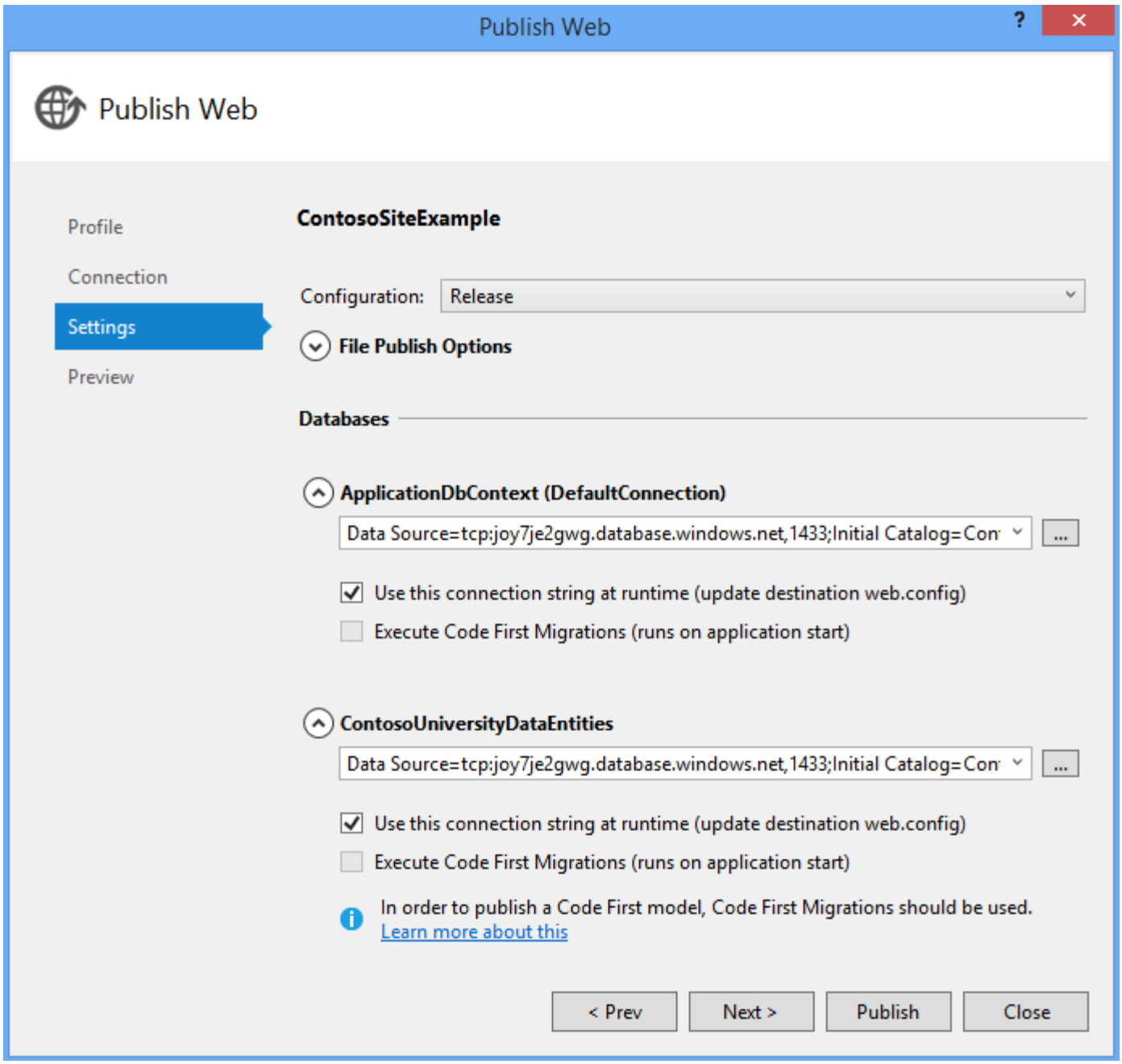


Your connection values are now all set. You can leave these values unchanged.

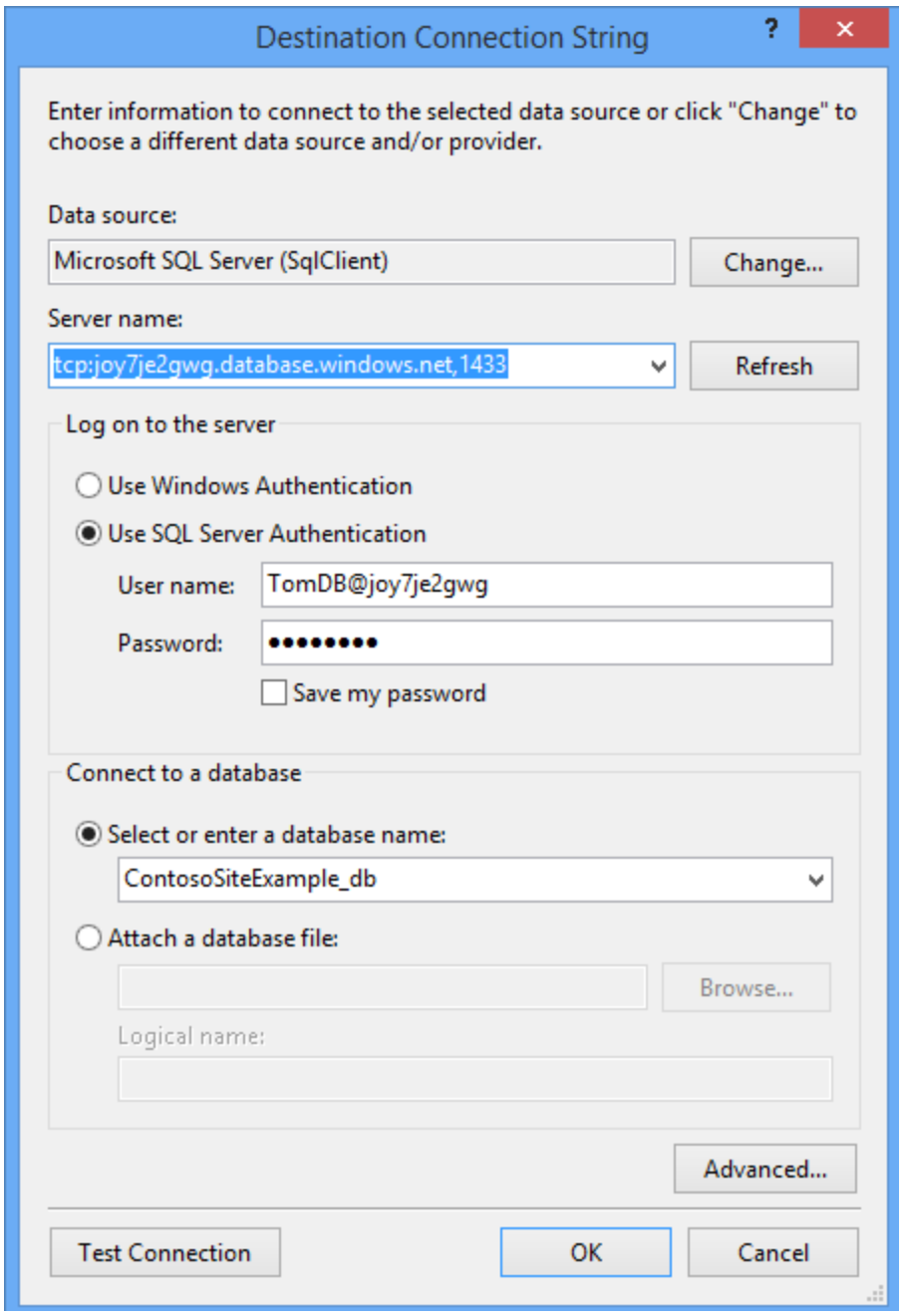


Click **Next**.

For Settings, notice that two database connections are specified - ApplicationDBContext and ContosoUniversityDataEntities. ApplicationDBContext is the connection for user account tables. These values only show the connection strings for the databases. It does not mean that these databases will be published when you publish your site. You will publish your database project after you have finished publishing the web app.



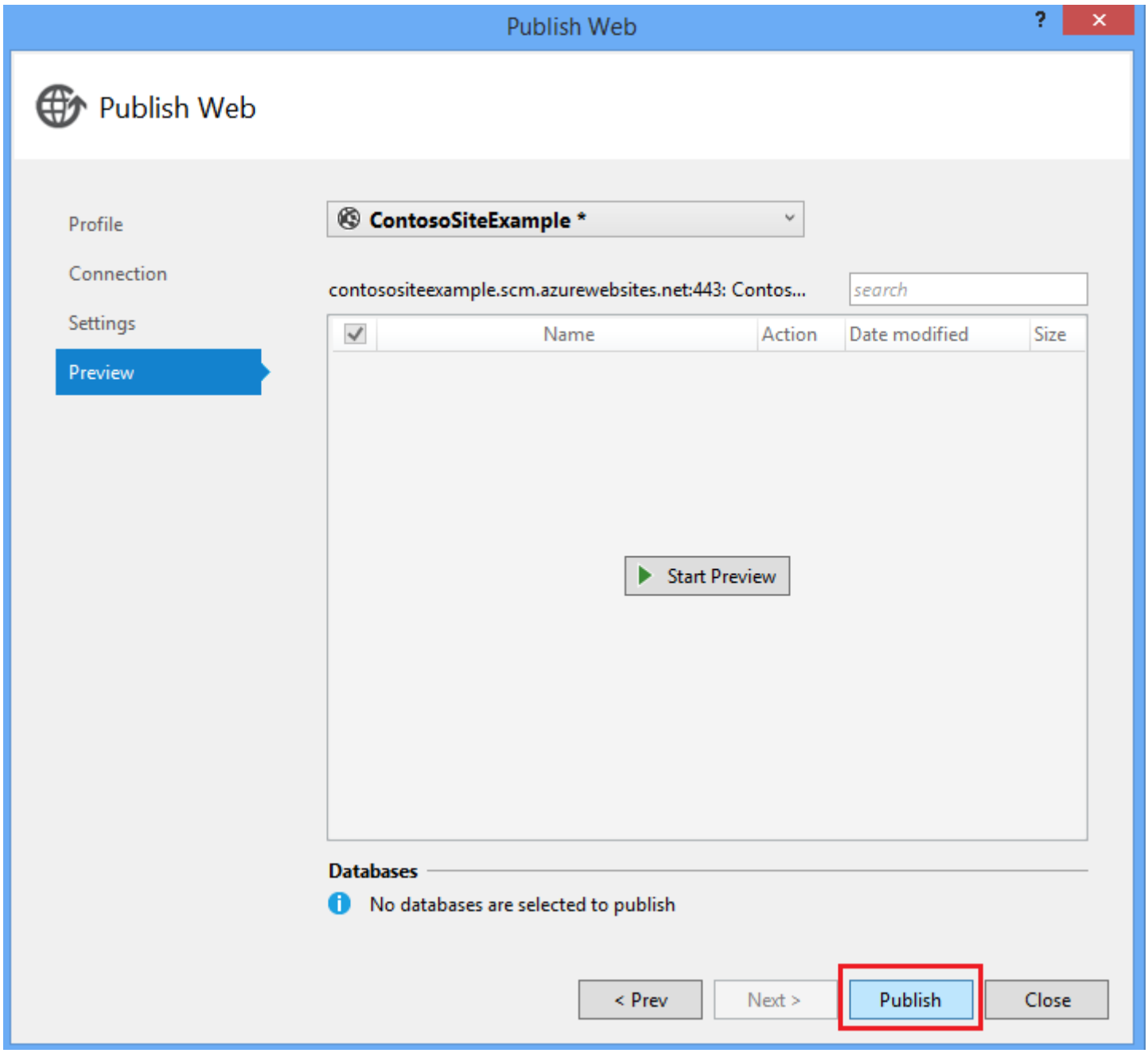
The ellipsis (...) next to the database connection shows you the details of the connection string. Click the ellipsis next to ContosoUniversityDataEntities.



Note the name of the database server and the database. The server name is randomly generated. The database name is simple the name of your site with **_db** appended. You will need both names later when you publish your database.

Click **OK** to close the database connection string window.

In the Publish Web window, click **Next** to see the preview.



You can click Start Preview to see a list of the files to publish. Since this is the first time you have published this site, the list is every relevant file in the project.

Click **Publish**.

The Output pane will display the result of your publication.

```
Output
Show output from: Build
2>Adding file (ContosoSiteExample\Views\Students\Create.cshtml).
2>Adding file (ContosoSiteExample\Views\Students\Delete.cshtml).
2>Adding file (ContosoSiteExample\Views\Students\Details.cshtml).
2>Adding file (ContosoSiteExample\Views\Students\Edit.cshtml).
2>Adding file (ContosoSiteExample\Views\Students\Index.cshtml).
2>Adding file (ContosoSiteExample\Views\Web.config).
2>Adding file (ContosoSiteExample\Views\_ViewStart.cshtml).
2>Adding file (ContosoSiteExample\Web.config).
2>Adding ACL's for path (ContosoSiteExample)
2>Adding ACL's for path (ContosoSiteExample)
2>Publish Succeeded.
2>Site was published successfully http://contosositeexample.azurewebsites.net/
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
Web Publish Activity  Data Tools Operations  Package Manager Console  Error List  Output  Find Results 1
```

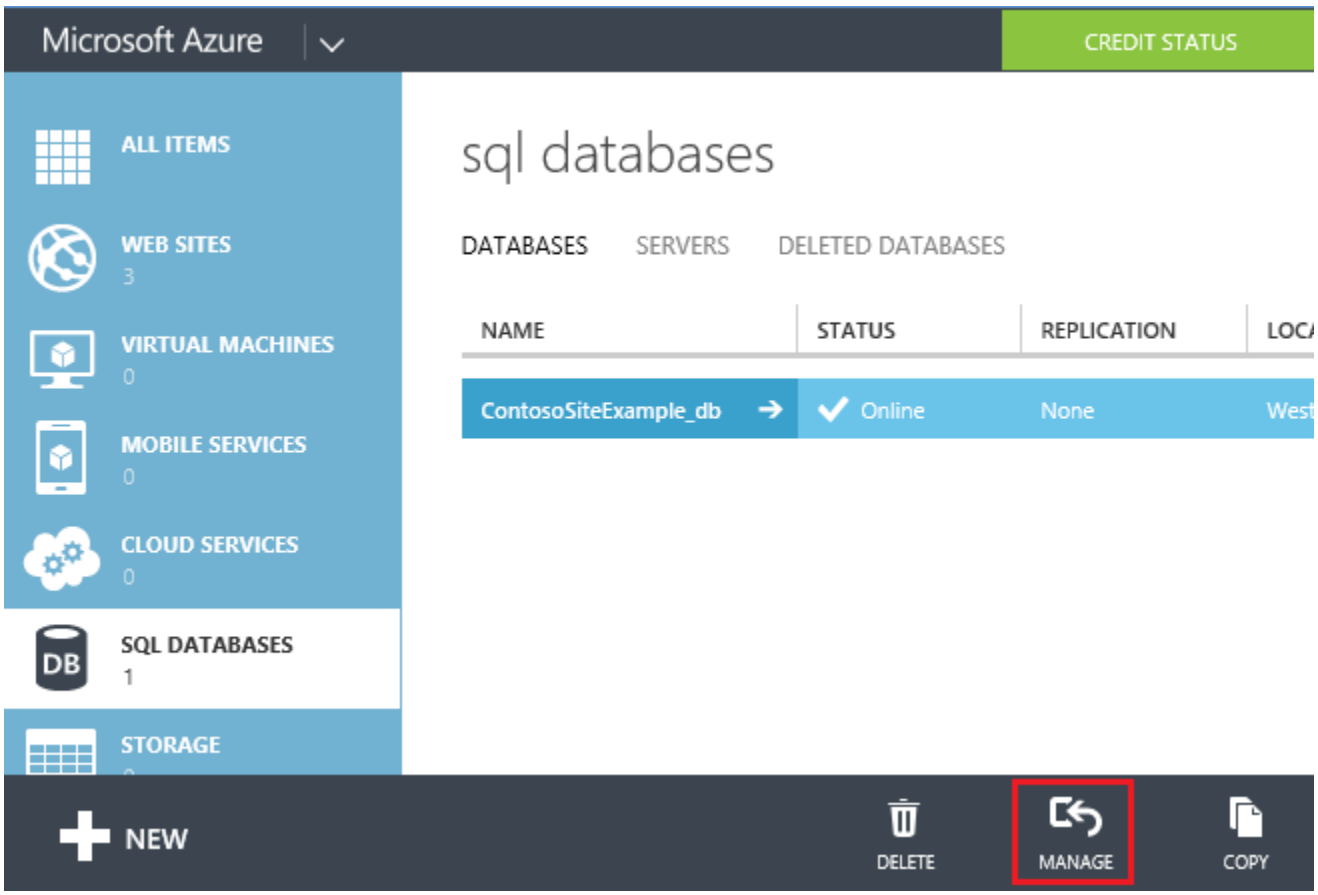
After publication, the site is immedialely launched in a web browser. Your site has been deployed and you can register a new user to the site; however, your tables in the ContosoUniversityData project have not yet been published. If you click on the List of students link you will receive an error.

Publish database to SQL Azure

Before publishing your database, you must make sure your local computer can connect to the database server. The firewall for your database server restricts which machines can connect to the database. You need to add the IP address of your computer to the allowed IP addresses for the firewall.

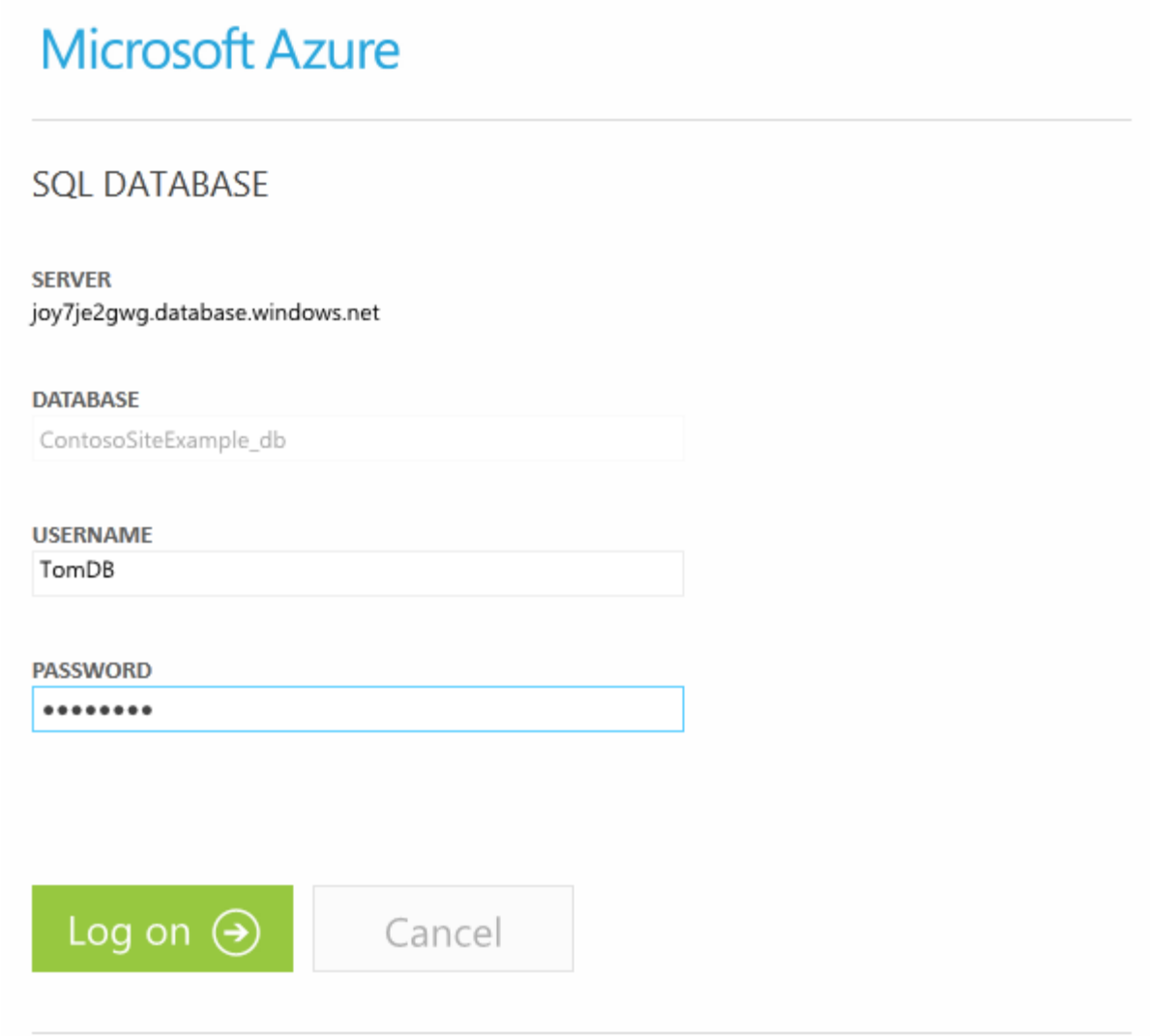
Login to your Azure account through the Azure portal.

Select your new database and select **Manage**.

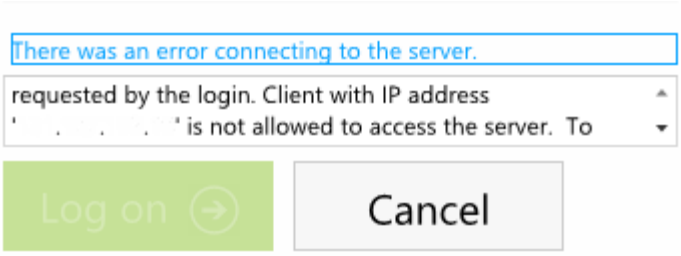


You must configure your database server to allow connections from your computer. When you select Manage, you are asked to add the current IP address as permitted to the database server. Select Yes.

There is a chance that the IP address you added in the previous step is not the only IP address you need to configure for connections. You can attempt to login to the database to see if the connections have been properly set up. Provide the user and password you created earlier.

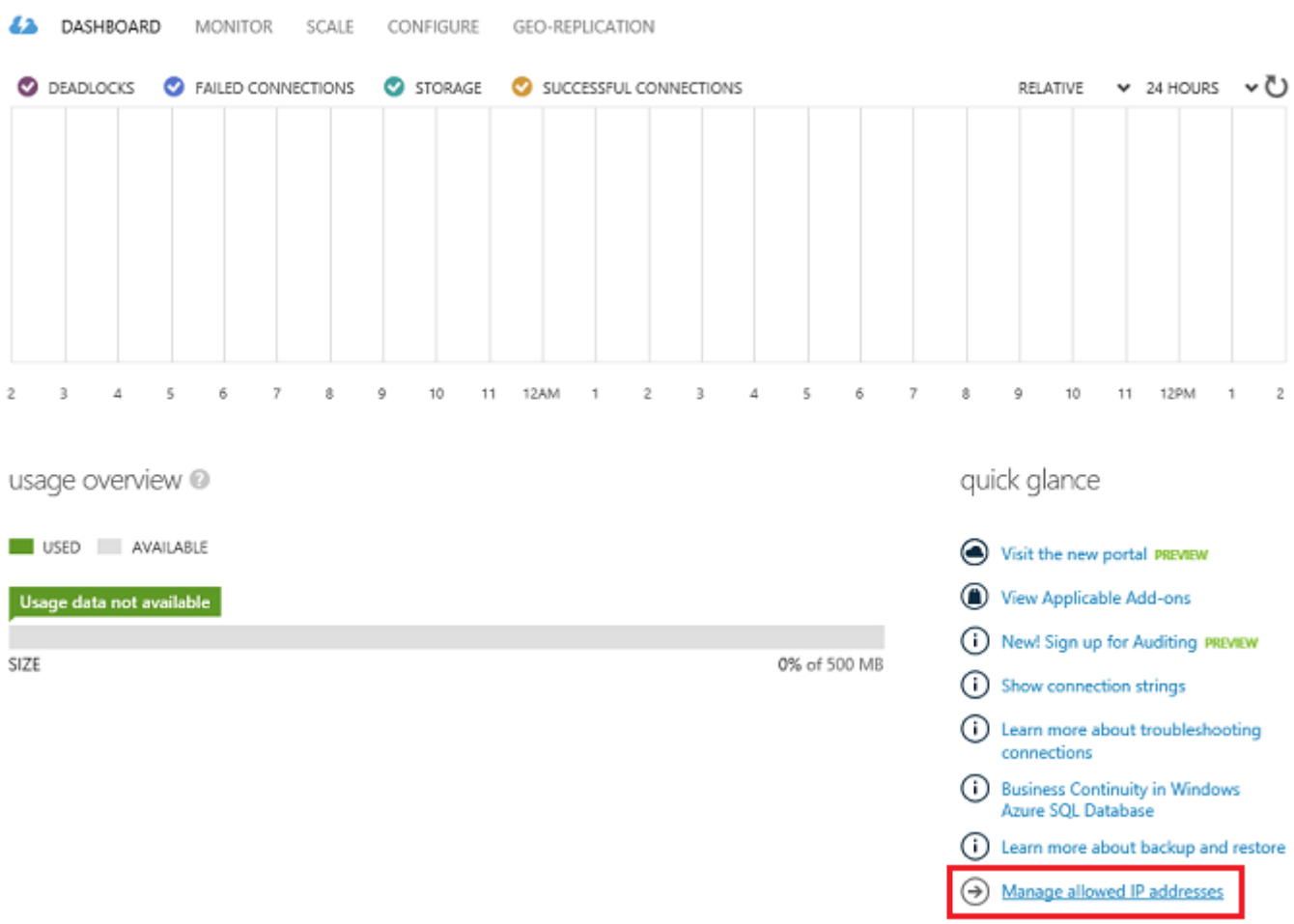


If you receive an error message, you need to add another IP address. Click the error message to see more details about error. In the details you will see the IP address that you need to add. Note this IP address.

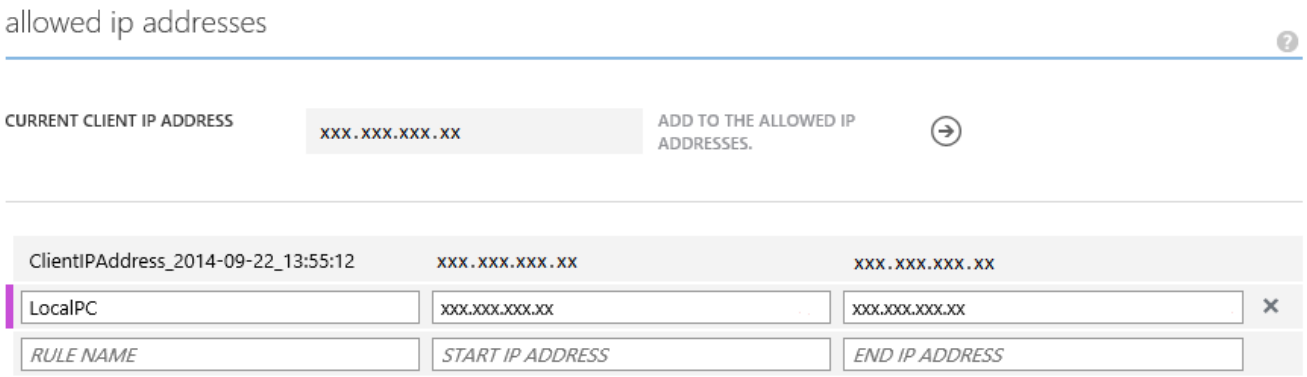


Close this login window, and return to the Azure portal.

Navigate to the Dashboard for your database. Click **Manage allowed IP addresses**.



You must now add the IP address from the error message. Either change the range of allowed IP addresses to include the one from the error message, or add that IP address as a separate entry.



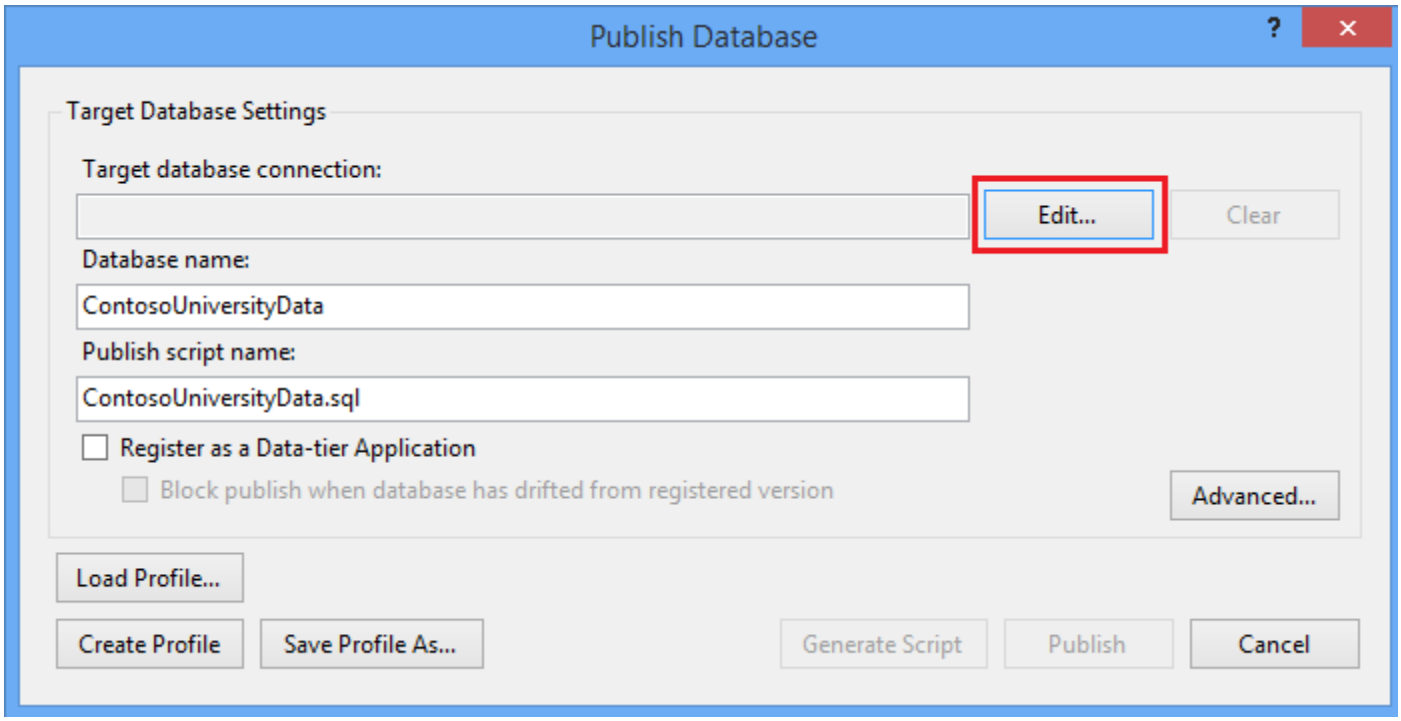
Save the change to allowed IP addresses.

Click Manage, and try logging in again to the database. You may need to wait a few minutes before the allowed IP addresses are correctly configured for the firewall. When you can successfully log in the database, you have finished setting up your connection to the database.

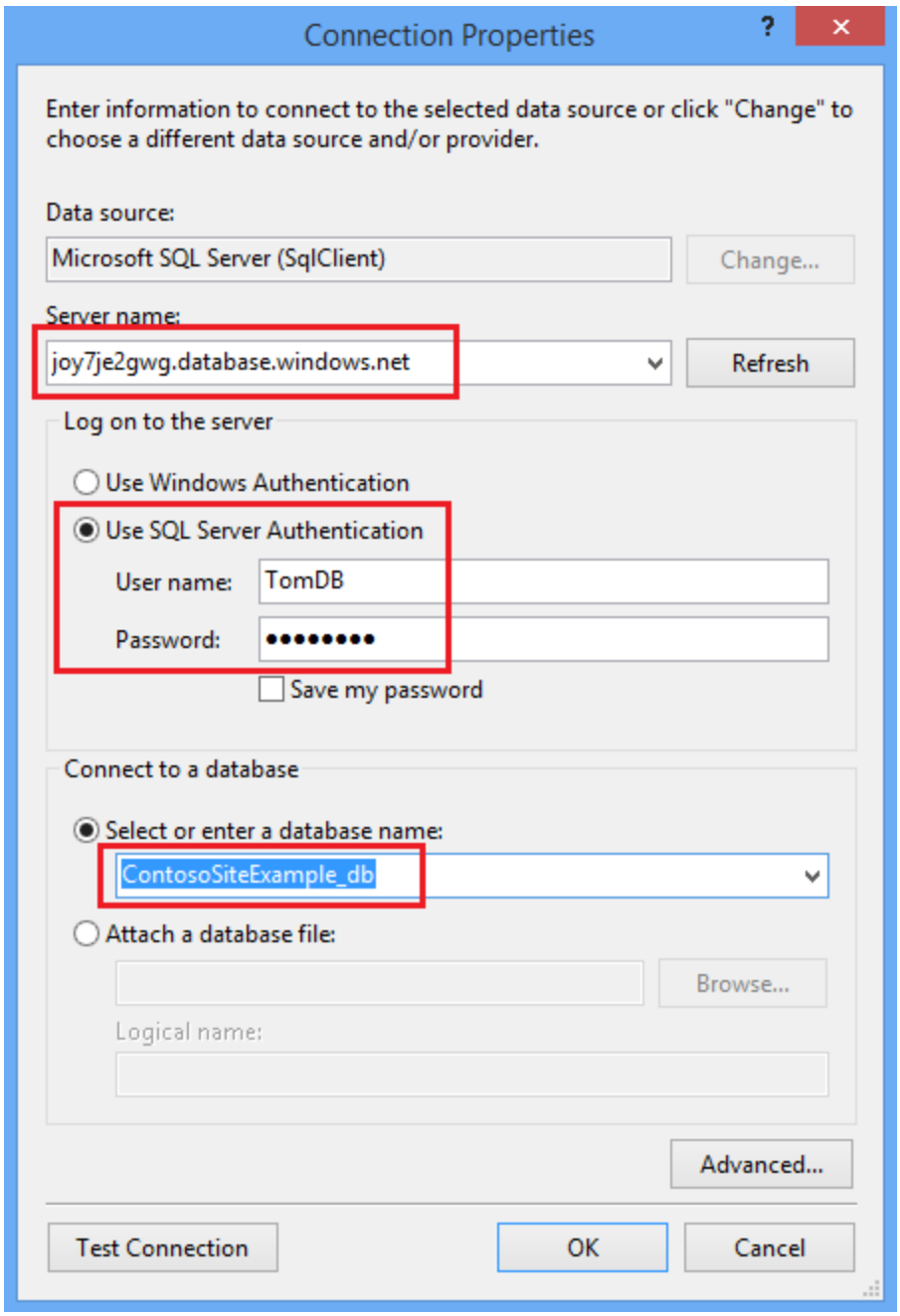
You can leave this management window open because you will check the result of your database deployment shortly.

Return to your database project. Right-click the project and select **Publish**.

In the Publish Database window, select **Edit**.

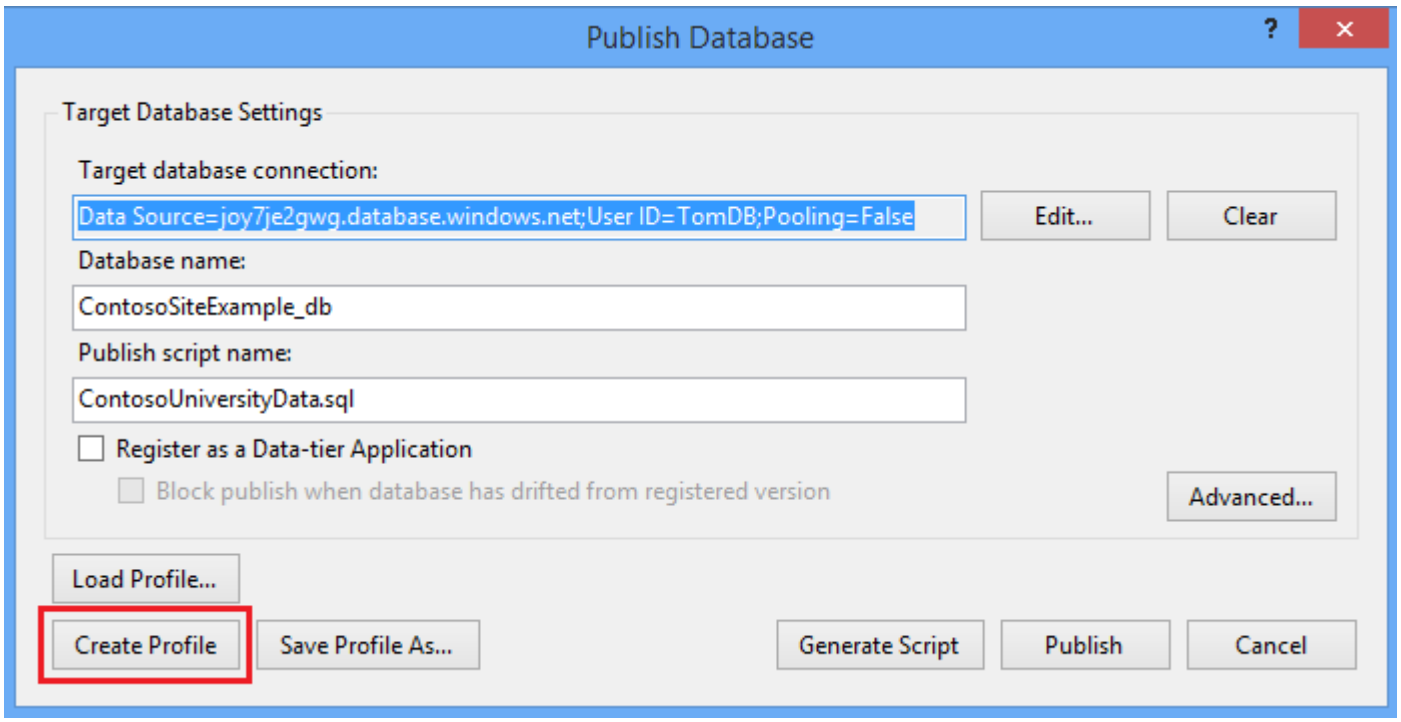


Provide the name of the database server and your authentication credentials for the server. After providing the credentials, select the database you created from the list of available databases. By default, Visual Studio sets the name of the database field to the name of your project which might not be the same as the database you created.



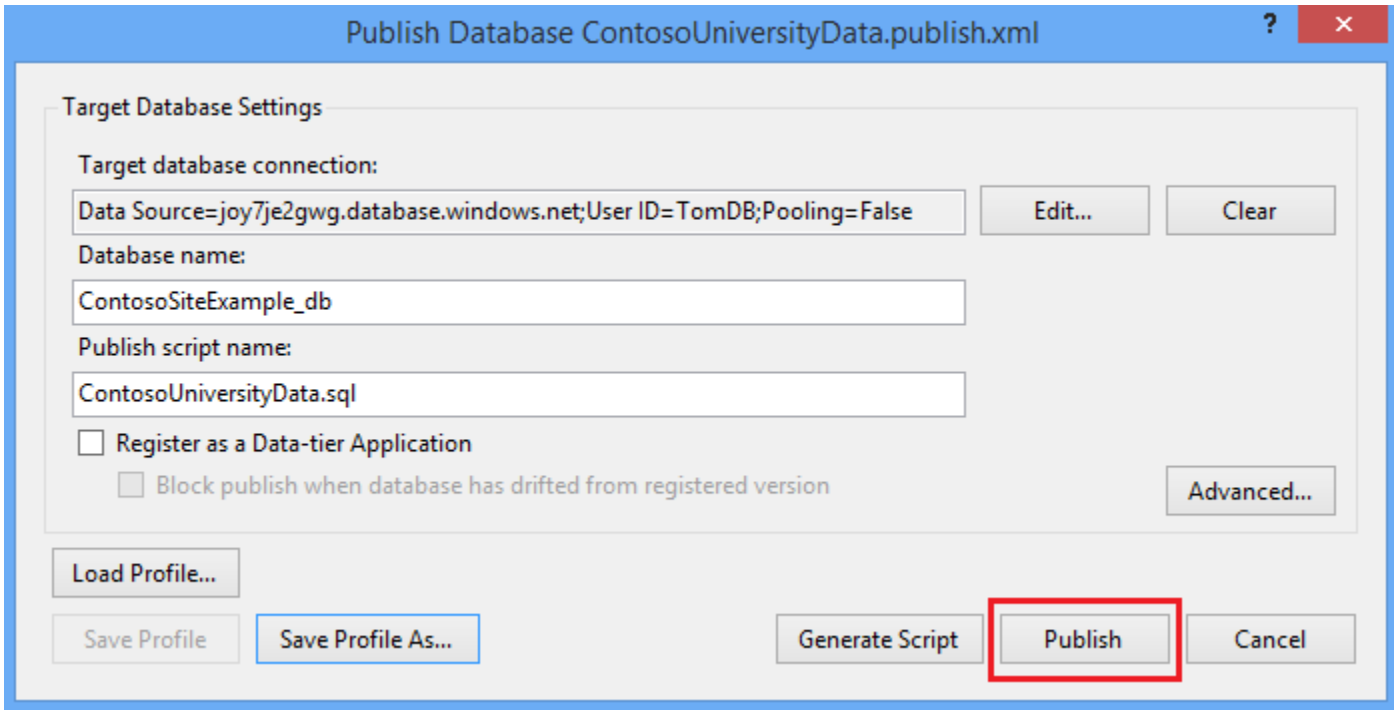
Click OK.

You will probably want to save this profile so you can publish updates in the future without re-entering all of the connection information. Select **Create Profile**.

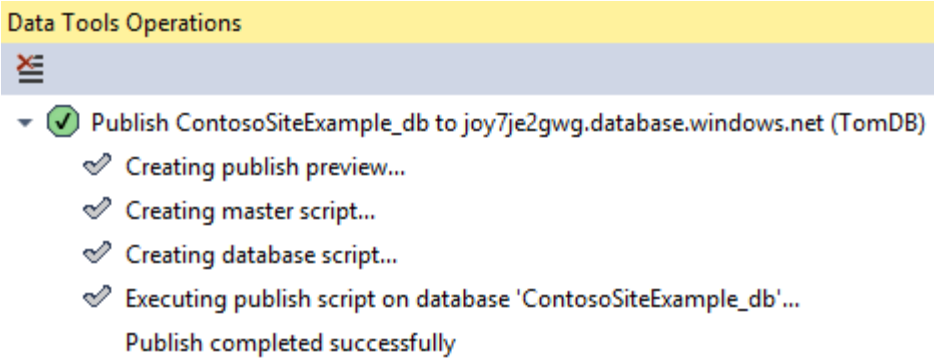


It will create a file in your project named **ContosoUniversityData.publish.xml**. The next time you want to publish the database to Azure, simply load that profile.

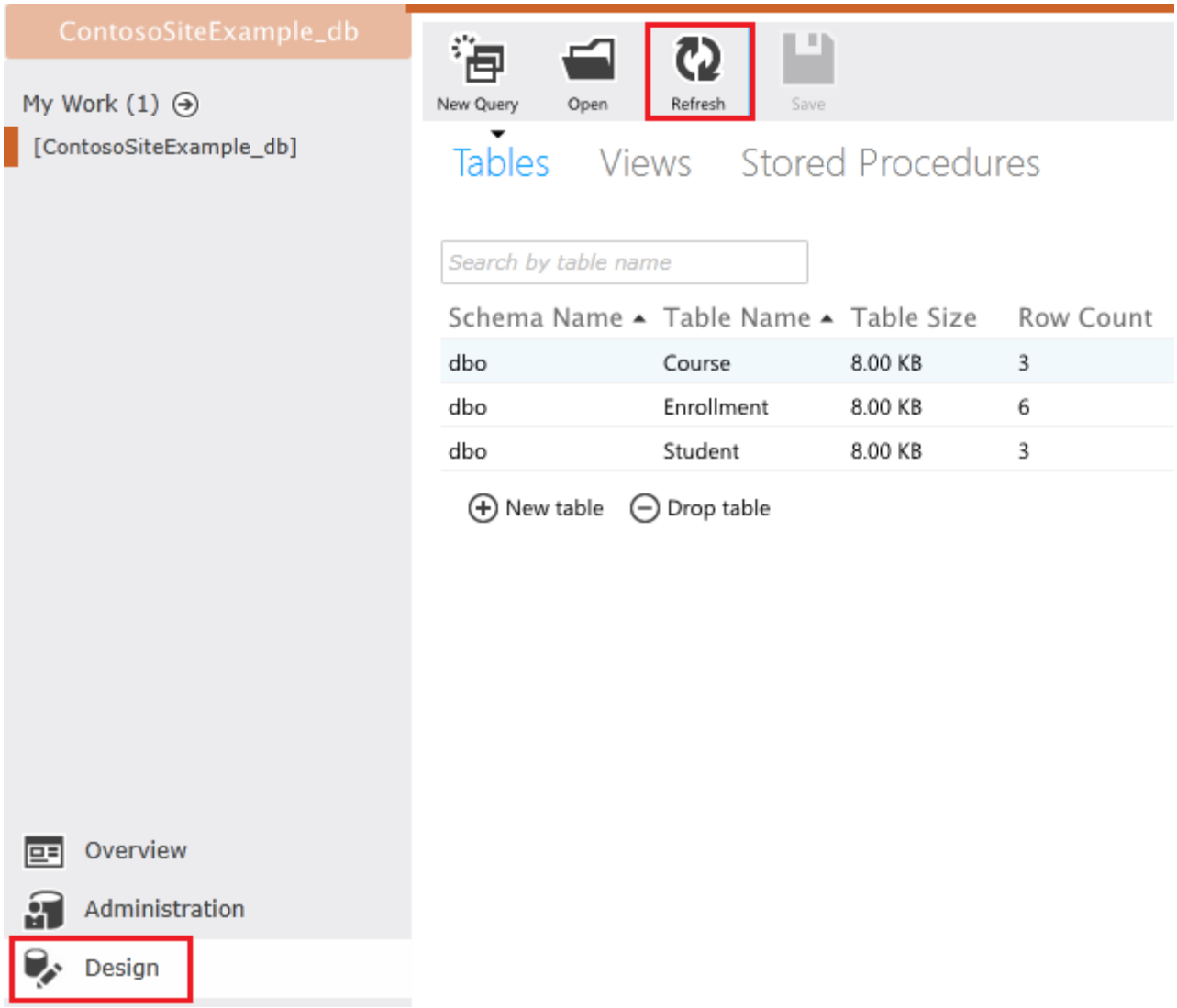
Now, click **Publish** to create the database on Azure.



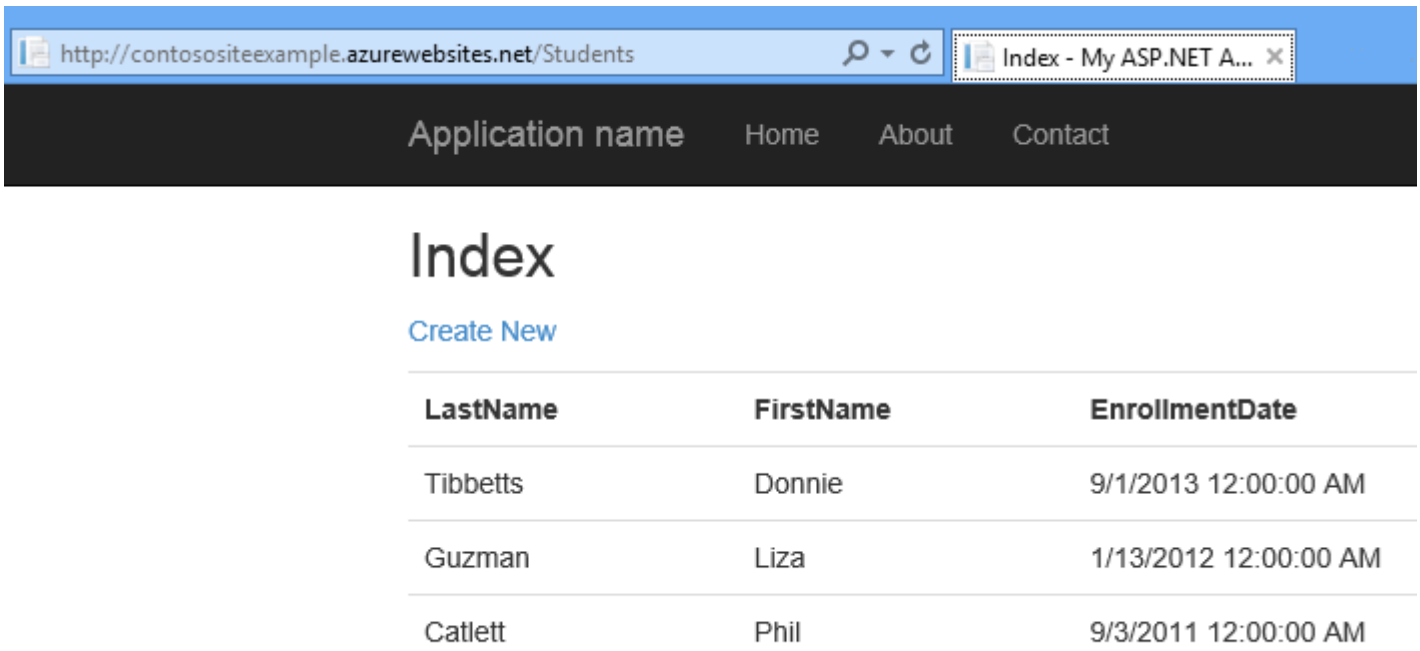
After running for a while, the publishing results are displayed.



Now, you can go back to the management portal for your database. Refresh the design view, and notice the 3 tables with pre-filled data have been deployed.



Now you are ready to test the web app that is deployed to Azure. Navigate to the web app on Azure (such as http://contosositeexample.azurewebsites.net/). Click the link for List of students and you should see the index view for students.



Occasionally, the database and connection need a little time to be properly configured. If you receive an error, wait a few minutes and then try again.

End 2014 lab

Resources

1. What is the Azure SQL Database service?: https://docs.microsoft.com/en-us/azure/sql-database/sql-database-technical-overview
2. Build an ASP.NET app in Azure with SQL Database: 2017 - https://docs.microsoft.com/en-us/azure/app-service/app-service-web-tutorial-dotnet-sqldatabase
3. You can't use localdb with Azure Websites. That said, you can use localdb for development, but change the connection string on deployment to use some other database, such as SQL Azure: http://azure.microsoft.com/blog/2013/07/17/windows-azure-web-sites-how-application-strings-and-connection-strings-work/