**The lecture 10**

**Build an ASP.NET app in Azure with SQL Database**

[Azure App Service](https://docs.microsoft.com/en-gb/azure/app-service/overview) provides a highly scalable, self-patching web hosting service. This tutorial shows you how to deploy a data-driven ASP.NET app in App Service and connect it to [Azure SQL Database](https://docs.microsoft.com/en-gb/azure/sql-database/sql-database-technical-overview). When you're finished, you have an ASP.NET app running in Azure and connected to SQL Database.

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

If you don't have an [Azure subscription](https://docs.microsoft.com/en-gb/azure/guides/developer/azure-developer-guide#understanding-accounts-subscriptions-and-billing), create a [free account](https://azure.microsoft.com/free/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio) before you begin.

## Prerequisites

To complete this tutorial:

Install [Visual Studio 2019](https://www.visualstudio.com/downloads/) with the **ASP.NET and web development** workload.

If you've installed Visual Studio already, add the workloads in Visual Studio by clicking **Tools** > **Get Tools and Features**.

## Download the sample

* [Download the sample project](https://github.com/Azure-Samples/dotnet-sqldb-tutorial/archive/master.zip).
* Extract (unzip) the dotnet-sqldb-tutorial-master.zip file.

The sample project contains a basic [ASP.NET MVC](https://www.asp.net/mvc) create-read-update-delete (CRUD) app using [Entity Framework Code First](https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application).

### 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 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 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.

Note

If you're already signed in, don't select **Create** yet.



### 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.

### Create a resource group

A [resource group](https://docs.microsoft.com/en-gb/azure/azure-resource-manager/management/overview#terminology) is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

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



Name the resource group **myResourceGroup**.

### Create an App Service plan

An [App Service plan](https://docs.microsoft.com/en-gb/azure/app-service/overview-hosting-plans) 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](https://docs.microsoft.com/en-gb/azure/app-service/overview-hosting-plans) |
| **Location** | West Europe | [Azure regions](https://azure.microsoft.com/regions/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio) |
| **Size** | Free | [Pricing tiers](https://azure.microsoft.com/pricing/details/app-service/?ref=microsoft.com&utm_source=microsoft.com&utm_medium=docs&utm_campaign=visualstudio) |

### Create a SQL Server instance

Before creating a database, you need an [Azure SQL Database logical server](https://docs.microsoft.com/en-gb/azure/sql-database/sql-database-features). A logical server contains a group of databases managed as a group.

Click **Create a 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](https://docs.microsoft.com/en-us/sql/relational-databases/security/password-policy).

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

Important

Even though your password in the connection strings is masked (in Visual Studio and also in App Service), the fact that it's maintained somewhere adds to the attack surface of your app. App Service can use [managed service identities](https://docs.microsoft.com/en-gb/azure/app-service/overview-managed-identity) to eliminate this risk by removing the need to maintain secrets in your code or app configuration at all. For more information, see [Next steps](https://docs.microsoft.com/en-gb/azure/app-service/app-service-web-tutorial-dotnet-sqldatabase#next-steps).



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 configured. 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 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 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](https://docs.microsoft.com/aspnet/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application).

### 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 app and update your SQL Database with Code First Migrations too.

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

Click **Configure** to open the publish settings.



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 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 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 app.

### Open Server Explorer

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

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

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