ASP.NET Web Deployment using Visual Studio

Tom Dykstra

Step-by-Step

Microsoft

ASP.NET Web Deployment using Visual Studio

Tom Dykstra

Summary: This tutorial series shows you how to deploy (publish) an ASP.NET web application to a Windows Azure Web Site or a third-party hosting provider, by using Visual Studio 2012 or Visual Studio 2010. You develop a web application in order to make it available to people over the Internet. But web programming tutorials typically stop right after they've shown you how to get something working on your development computer. This tutorial begins where the others leave off: you've built a web site, tested it, and it's ready to go. What's next? This tutorial shows you how to deploy first to IIS on your local development computer for testing, and then to Windows Azure or a third-party hosting provider for staging and production. The sample application that you'll deploy is a web application project that uses the Entity Framework, SQL Server, and the ASP.NET membership system. The sample application uses ASP.NET Web Forms, but the procedures shown apply also to ASP.NET MVC and Web API.

Category: Step-by-Step
Applies to: ASP.NET, Visual Studio 2010, Visual Studio 2012, Windows Azure
Source: ASP.NET Site (http://asp.net/web-forms/tutorials/deployment/visual-studio-web-deployment/introduction)
E-book publication date: April, 2013

Microsoft

ASP.NET Web Deployment using Visual Studio

Contents

Introduction

This tutorial series shows you how to deploy (publish) an ASP.NET web application to a Windows Azure Web Site or a third-party hosting provider, by using Visual Studio 2012 or Visual Studio 2010.

You develop a web application in order to make it available to people over the Internet. But web programming tutorials typically stop right after they've shown you how to get something working on your development computer. This series of tutorials begins where the others leave off: you've built a web site, tested it, and it's ready to go. What's next? These tutorials show you how to deploy first to IIS on your local development computer for testing, and then to Windows Azure or a third-party hosting provider for staging and production. The sample application that you'll deploy is a web application project that uses the Entity Framework, SQL Server, and the ASP.NET membership system. The sample application uses ASP.NET Web Forms, but the procedures shown apply also to ASP.NET MVC and Web API.

These tutorials assume you know how to work with ASP.NET in Visual Studio. If you don't, a good place to start is a basic ASP.NET Web Forms Tutorial or a basic ASP.NET MVC Tutorial.

Visual Studio 2012 is recommended, but you can complete most of the tutorial steps by using Visual Studio 2010. You'll need to install the latest updates and the Windows Azure SDK, as explained in the Prerequisites section.

If you have questions that are not directly related to the tutorial, you can post them to the ASP.NET Deployment forum or StackOverflow.

Overview

These tutorials guide you through deploying an ASP.NET web application that includes SQL Server databases. You'll deploy first to IIS on your local development computer for testing, and then to a Windows Azure Web Site and Windows Azure SQL Database for staging and production. You'll see how to deploy by using Visual Studio one-click publish, and you'll see how to deploy using the command line.

The number of tutorials might make the deployment process seem daunting. In fact, the basic procedures are simple. However, in real-world situations, you often need information about some small but important extra deployment task — for example, setting folder permissions on the target server. We've illustrated many of these additional tasks, in the hope that the tutorials don't leave out information that might prevent you from successfully deploying a real application.

The tutorials are designed to run in sequence, and each part builds on the previous part. However, you can skip parts that aren't relevant to your situation. (Skipping parts might require you to adjust the procedures in later tutorials.)

Intended audience

The tutorials are aimed at ASP.NET developers who work in small organizations or other environments where:

- A continuous integration process (automated builds and deployment) is not used.
- The production environment is Windows Azure Web Sites or a third-party hosting provider.
- One person typically fills multiple roles (the same person develops, tests, and deploys).

In enterprise environments, it's more typical to implement continuous integration processes, and the production environment is usually hosted by the company's own servers. Different people also typically perform different roles. For information about enterprise deployment, see Deploying Web Applications in Enterprise Scenarios.

Using a third-party hosting provider

The tutorials take you through the process of setting up a Windows Azure account and deploying the application to a Windows Azure Web Site for staging and production. However, you can use the same basic procedures for deploying to a third-party hosting provider of your choice. Where the tutorials go over processes unique to Windows Azure, they explain that and advise what differences you can expect at a third-party hosting provider.

Deploying web site projects

The sample application that you download and deploy for these tutorials is a Visual Studio web application project. However, if you install the latest Web Publish Update for Visual Studio, you can use the same deployment methods and tools for web site projects.

Deploying ASP.NET MVC projects

The sample application is an ASP.NET Web Forms project, but everything you learn how to do is applicable to ASP.NET MVC as well. A Visual Studio MVC project is just another form of web application project. The only difference is that if you're deploying to a hosting provider that does not support ASP.NET MVC or your target version of it, you must make sure that you have installed the appropriate (MVC 3 or MVC 4) NuGet package in your project.

Programming language

The sample application uses C# but the tutorials do not require knowledge of C#, and the deployment techniques shown by the tutorials are not language-specific.

Database deployment methods

There are three ways that you can deploy a SQL Server database along with web deployment in Visual Studio:

- Entity Framework Code First Migrations
- The dbDacFx Web Deploy provider
- The dbFullSql Web Deploy provider

In this tutorial you will use the first two of these methods. The dbFullSql Web Deploy provider is a legacy method that is no longer recommended except for some specific scenarios such as migrating from SQL Server Compact to SQL Server.

The methods shown in this tutorial are for SQL Server databases, not SQL Server Compact. For information about how to deploy a SQL Server Compact database, see Visual Studio Web Deployment With SQL Server Compact.

The methods shown in this tutorial require that you use the Web Deploy publish method. If you prefer a different publish method, such as FTP, File System, or FPSE, see Deploying a database separately from web application deployment in the Web Deployment Content Map for Visual Studio and ASP.NET.

Entity Framework Code First Migrations

In the Entity Framework version 4.3, Microsoft introduced Code First Migrations. Code First Migrations automates the process of making incremental changes to a data model and propagating those changes to the database. In earlier versions of Code First, you typically let the Entity Framework drop and re-create the database each time you change the data model. This is not a problem in development because test data is easily re-created, but in production you usually want to update the database schema without dropping the database. The Migrations feature enables Code First to update the database without dropping and re-creating it. You can let Code First automatically decide how to make the required schema changes, or you can write code that customizes the changes. For an introduction to Code First Migrations, see Code First Migrations.

When you are deploying a web project, Visual Studio can automate the process of deploying a database that is managed by Code First Migrations. When you create the publish profile, you select a check box that is labeled Execute Code First Migrations (runs on application start). This setting causes the deployment process to automatically configure the application Web.config file on the destination server so that Code First uses the `MigrateDatabaseToLatestVersion` initializer class.

Visual Studio does not do anything with the database during the deployment process. When the deployed application accesses the database for the first time after deployment, Code First automatically creates the database or updates the database schema to the latest version. If the application implements a Migrations Seed method, the method runs after the database is created or the schema is updated.

In this tutorial, you'll use Code First Migrations to deploy the application database.

The dbDacFx Web Deploy provider

For a SQL Server database that isn't managed by Entity Framework Code First, you can select a check box that is labeled Update database when you configure the publish profile. During the initial deployment, the dbDacFx provider creates tables and other database objects in the destination database to match the source database. On subsequent deployments, the provider determines what is different between the source and destination databases, and it updates the schema of the destination database to match the source database. By default, the provider won't make any changes that cause data loss, such as when a table or column is dropped.

This method does not automate the deployment of data in database tables, but you can create scripts to do that and configure Visual Studio to run them during deployment. Another reason to run scripts during deployment is to make schema changes that can't be done automatically because they would cause data loss.

In this tutorial, you'll use the dbDacFx provider to deploy the ASP.NET membership database.

Troubleshooting during this tutorial

When an error happens during deployment, or if the deployed site does not run correctly, the error messages don't always provide an obvious solution. To help you with some common problem scenarios, a troubleshooting reference page is available. If you get an error message or something doesn't work as you go through the tutorials, be sure to check the troubleshooting page.

Comments welcome

Comments on the tutorials are welcome, and when the tutorial is updated every effort will be made to take into account corrections or suggestions for improvements that are provided in tutorial comments.

Prerequisites

Before you start, make sure that you have the following products installed on your computer:

- Windows 8 or Windows 7.
- One of the following:
  - Visual Studio 2012 with Visual Studio 2012 Update 1
  - Visual Studio Express 2012 with Visual Studio 2012 Update 1
  - Visual Studio 2010 SP1
- One of the following:
  - Windows Azure SDK for Visual Studio 2010
  - Windows Azure SDK for Visual Studio 2012 (If you don't already have Visual Studio, install this version of the SDK, and that will automatically install Visual Studio Express 2012 for Web.)
- If you are using Visual Studio 2010, you also need the following software which is installed by default with Visual Studio 2012:

- SQL Server Express LocalDB
- SQL Server Data Tools

Note: Depending on how many of the SDK dependencies you already have on your machine, installing the Windows Azure SDK could take a long time, from several minutes to a half hour or more. You need the Windows Azure SDK even if you plan to publish to a third-party hosting provider instead of to Windows Azure, because the SDK includes the latest updates to Visual Studio web publish features.

The instructions and screen shots are based on Windows 8, but the tutorials explain differences for Windows 7.

The instructions and screen shots are based on Visual Studio 2012. The tutorials explain known differences for Visual Studio 2010 but have not been tested with Visual Studio 2010, so you might have to adjust for minor differences that aren't noted.

Some other software is required in order to complete the tutorial, but you don't have to have that loaded yet. The tutorial will walk you through the steps for installing it when you need it.

Download the sample application

The application that you'll deploy is named Contoso University and has already been created for you. It's a simplified version of a university web site, based loosely on the Contoso University application described in the Entity Framework tutorials on the ASP.NET site.

When you have the prerequisites installed, download the Contoso University web application. The *.zip* file contains multiple versions of the project and a PDF file that contains all of the tutorials. To work through the steps of the tutorial, start with ContosoUniversity-Begin. To see what the project looks like at the end of the tutorials, open ContosoUniversity-End.

To prepare the project for working through the tutorial steps, perform the following steps:

1. Save ContosoUniversity-Begin to whatever folder you use for working with Visual Studio projects.

   By default this is the following folder for Visual Studio 2012:

   ```
   C:\Users\<username>\Documents\Visual Studio 2012\Projects
   ```

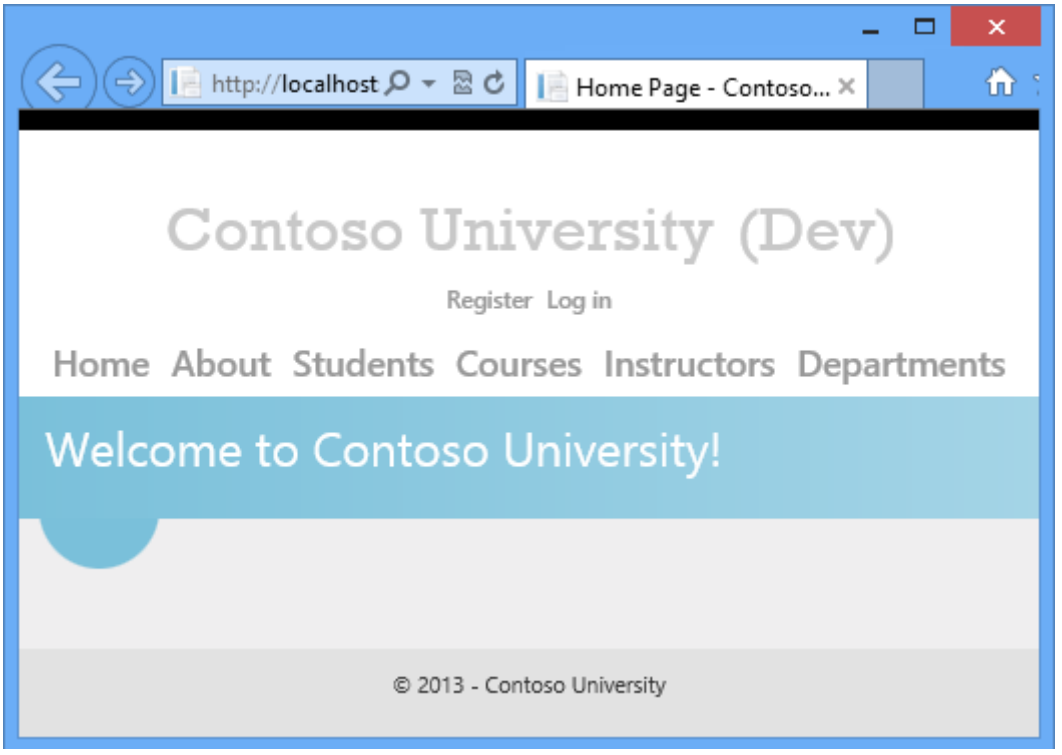
   (For the screen shots in this tutorial, the project folder is located in the root directory on the `C:` drive.)

2. Start Visual Studio and open the project.
3. In **Solution Explorer**, right-click the solution and click **EnableNuGet Package Restore**.
4. Build the solution.
5. If you get compile errors, manually restore the NuGet packages:

1. In **Solution Explorer**, right-click the solution, and then click **Manage NuGet Packages for Solution**.
2. At the top of the **Manage NuGet Packages** dialog box you'll see **Some NuGet packages are missing from this solution. Click to restore.** Click the **Restore** button.
3. Rebuild the solution.

6. Press CTRL-F5 to run the application.

   The application opens to the Contoso University home page.

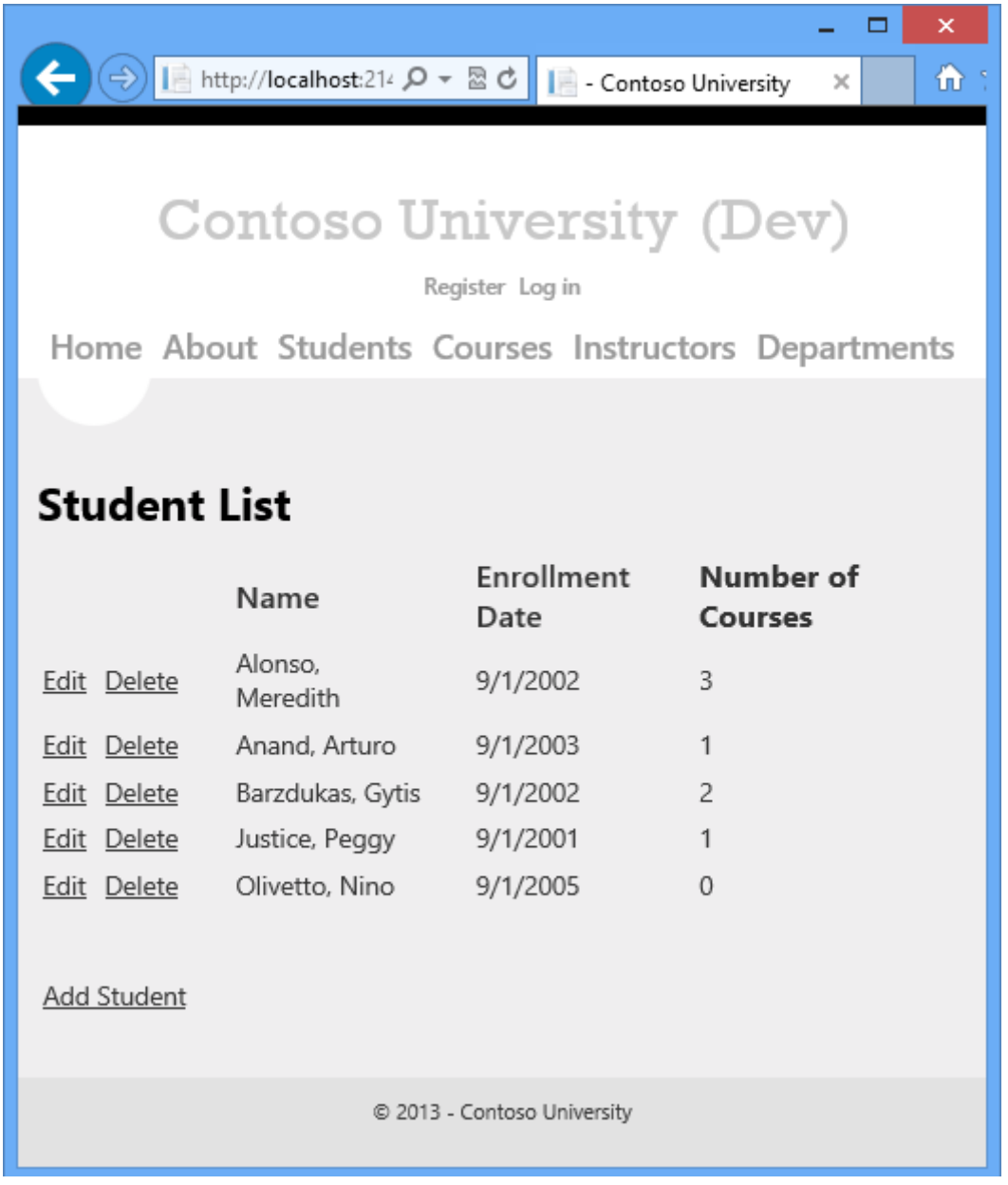


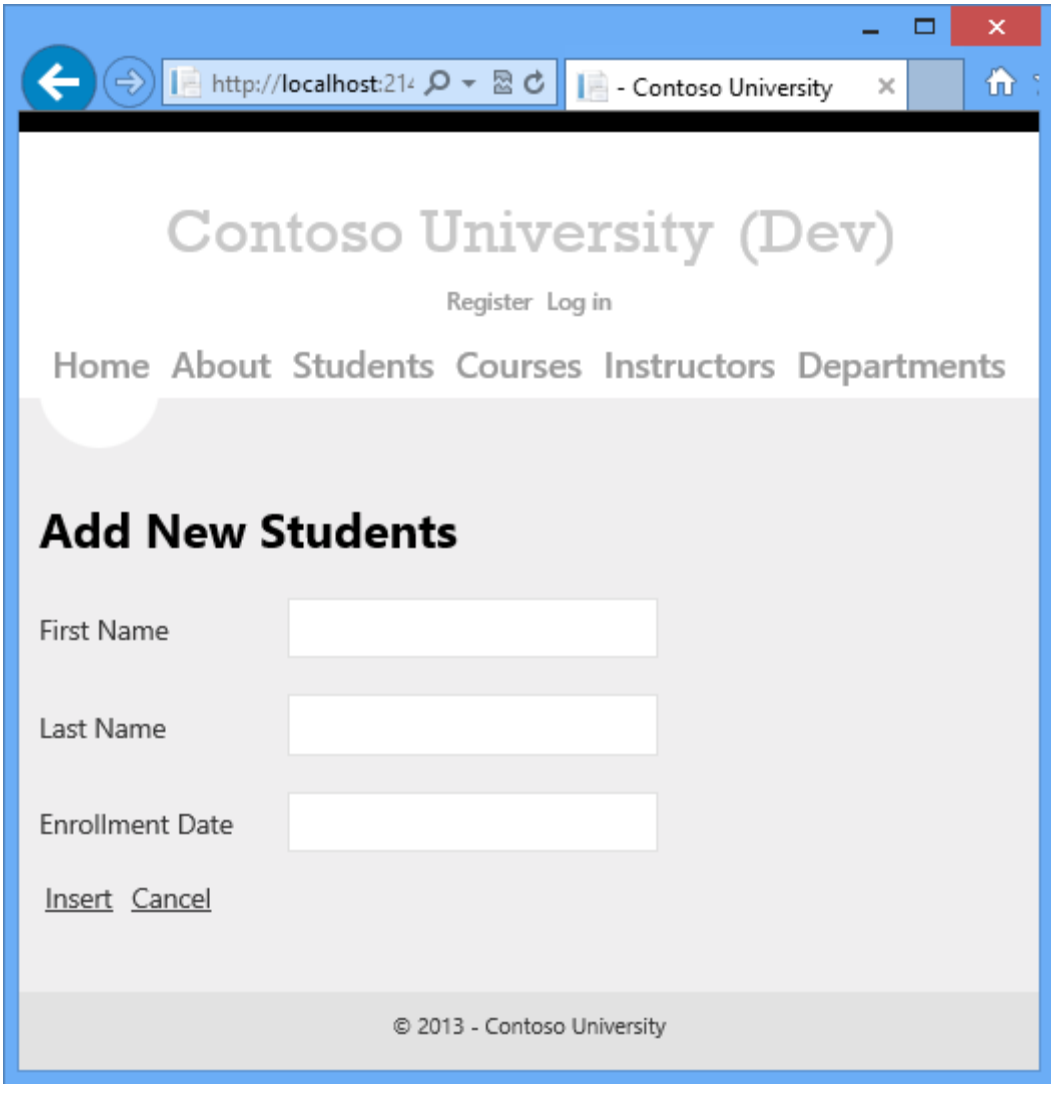
   (There might be a wait time while Visual Studio starts up the SQL Server Express LocalDB instance, and you might get a timeout error if that process takes too long. In that case just start the project again.)

The website pages are accessible from the menu bar and let you perform the following functions:

- Display student statistics (the About page).
- Display, edit, delete, and add students.
- Display and edit courses.
- Display and edit instructors.
- Display and edit departments.

Following are screen shots of a few representative pages.


http://localhost:214
- Contoso University
Contoso University (Dev)
Register Log in
Home About Students Courses Instructors Departments
Student List
Name
Enrollment Date
Number of Courses
Edit Delete
Alonso, Meredith
9/1/2002
3
Edit Delete
Anand, Arturo
9/1/2003
1
Edit Delete
Barzdukas, Gytis
9/1/2002
2
Edit Delete
Justice, Peggy
9/1/2001
1
Edit Delete
Olivetto, Nino
9/1/2005
0
Add Student
© 2013 - Contoso University



Review application features that affect deployment

The following features of the application affect how you deploy it or what you have to do to deploy it. Each of these is explained in more detail in the following tutorials in the series.

- Contoso University uses a SQL Server database to store application data such as student and instructor names. The database contains a mix of test data and production data, and when you deploy to production you need to exclude the test data.
- The application uses the ASP.NET membership system, which stores user account information in a SQL Server database. The application defines an administrator user who has access to some restricted information. You need to deploy the membership database without test accounts but with an administrator account.
- The application uses a third-party error logging and reporting utility. This utility is provided in an assembly which must be deployed with the application.
- The error logging utility writes error information in XML files to a file folder. You have to make sure that the account that ASP.NET runs under in the deployed site has write

permission to this folder, and you have to exclude this folder from deployment. (Otherwise, error log data from the test environment might be deployed to production and/or production error log files might be deleted.)

- The application includes some settings that must be changed in in the deployed *Web.config* file depending on the destination environment (test, staging, or production), and other settings that must be changed depending on the build configuration (Debug or Release).
- The Visual Studio solution includes a class library project. Only the assembly that this project generates should be deployed, not the project itself.

Summary

In this first tutorial in the series, you have downloaded the sample Visual Studio project and reviewed site features that affect how you deploy the application. In the following tutorials, you prepare for deployment by setting up some of these things to be handled automatically. Others you take care of manually.

Preparing for Database Deployment

Overview

This tutorial shows how to get the project ready for database deployment. The database structure and some (not all) of the data in the application's two databases must be deployed to test, staging, and production environments.

Typically, as you develop an application, you enter test data into a database that you don't want to deploy to a live site. However, you might also have some production data that you do want to deploy. In this tutorial you'll configure the Contoso University project and prepare SQL scripts so that the correct data is included when you deploy.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

SQL Server Express LocalDB

The sample application uses SQL Server Express LocalDB. SQL Server Express is the free edition of SQL Server. It is commonly used during development because it is based on the same database engine as full versions of SQL Server. You can test with SQL Server Express and be assured that the application will behave the same in production, with a few exceptions for features that vary between SQL Server Editions.

LocalDB is a special execution mode of SQL Server Express that enables you to work with databases as *.mdf* files. Typically, LocalDB database files are kept in the *App_Data* folder of a web project. The user instance feature in SQL Server Express also enables you to work with *.mdf* files, but the user instance feature is deprecated; therefore, LocalDB is recommended for working with *.mdf* files.

Typically SQL Server Express is not used for production web applications. LocalDB in particular is not recommended for production use with a web application because it is not designed to work with IIS.

In Visual Studio 2012, LocalDB is installed by default with Visual Studio. In Visual Studio 2010 and earlier versions, SQL Server Express (without LocalDB) is installed by default with Visual Studio; that is why you installed it as one of the prerequisites in the first tutorial in this series.

For more information about SQL Server editions, including LocalDB, see the following resources Working with SQL Server Databases.

Entity Framework and Universal Providers

For database access, the Contoso University application requires the following software that must be deployed with the application because it is not included in the .NET Framework:

- ASP.NET Universal Providers (enables the ASP.NET membership system to use Windows Azure SQL Database)
- Entity Framework 5.0

Because this software is included in NuGet packages, the project is already set up so that the required assemblies are deployed with the project. (The links point to the current versions of these packages, which might be newer than what is installed in the starter project that you downloaded for this tutorial.)

If you are deploying to a third-party hosting provider instead of to Windows Azure, make sure that you use Entity Framework 5.0 or later. Earlier versions of Code First Migrations require Full Trust, and most hosting providers will run your application in Medium Trust. For more information about Medium Trust, see the Deploy to IIS as a Test Environment tutorial.

Configure Code First Migrations for application database deployment

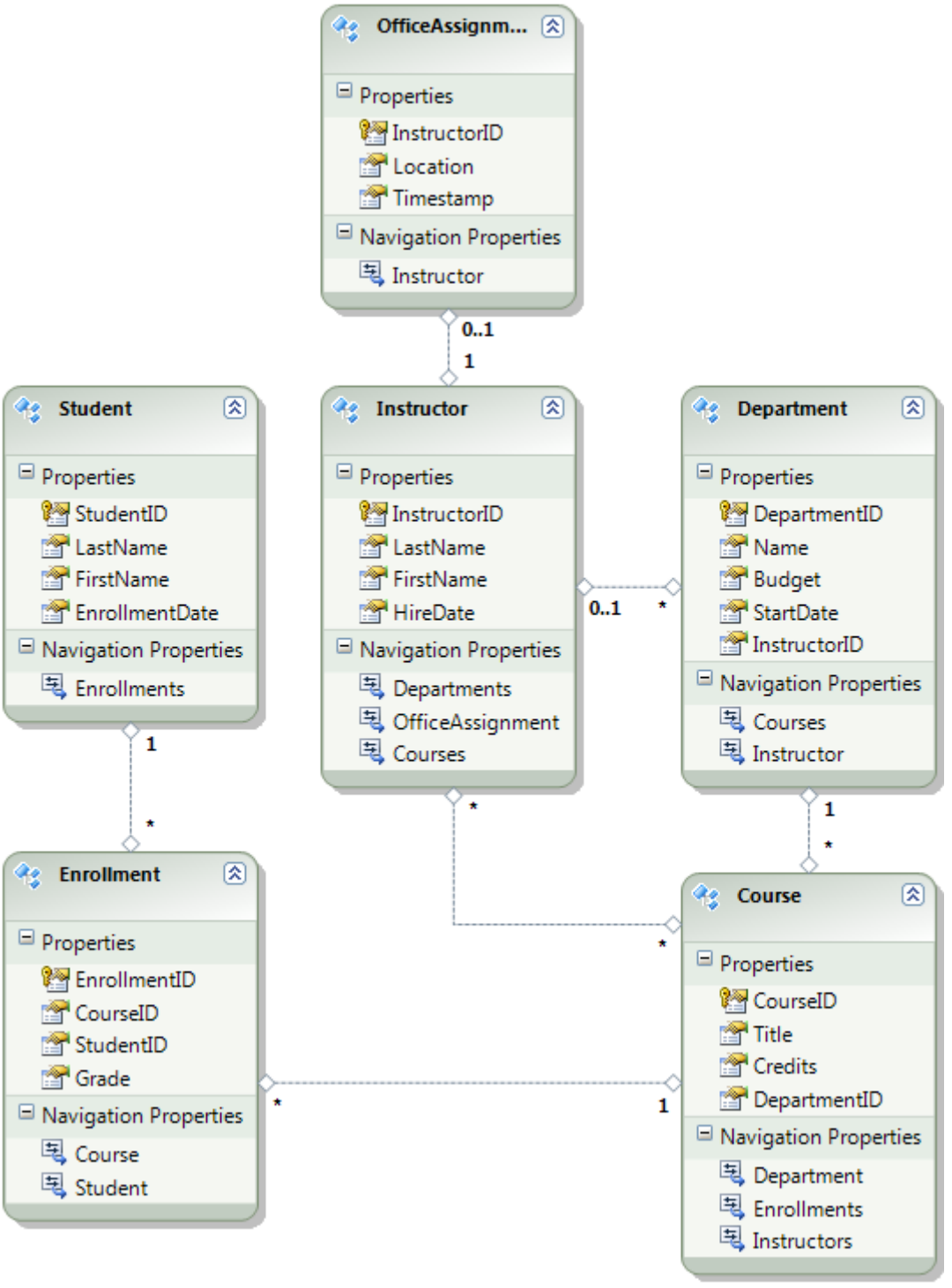
The Contoso University application database is managed by Code First, and you'll deploy it by using Code First Migrations. For an overview of database deployment by using Code First Migrations, see the first tutorial in this series.

When you deploy an application database, typically you don't simply deploy your development database with all of the data in it to production, because much of the data in it is probably there only for testing purposes. For example, the student names in a test database are fictional. On the other hand, you often can't deploy just the database structure with no data in it at all. Some of the data in your test database might be real data and must be there when users begin to use the application. For example, your database might have a table that contains valid grade values or real department names.

To simulate this common scenario, you'll configure a Code First Migrations `Seed` method that inserts into the database only the data that you want to be there in production. This `Seed` method shouldn't insert test data because it will run in production after Code First creates the database in production.

In earlier versions of Code First before Migrations was released, it was common for `Seed` methods to insert test data also, because with every model change during development the database had to be completely deleted and re-created from scratch. With Code First Migrations, test data is retained after database changes, so including test data in the `Seed` method is not necessary. The project that you downloaded uses the method of including all data in the `Seed` method of an initializer class. In this tutorial you'll disable that initializer class and `enable Migrations. Then you'll update the `Seed` method in the Migrations configuration class so that it inserts only data that you want to be inserted in production.

The following diagram illustrates the schema of the application database:



For these tutorials, you'll assume that the `Student` and `Enrollment` tables should be empty when the site is first deployed. The other tables contain data that has to be preloaded when the application goes live.

Disable the initializer

Since you will be using Code First Migrations, you do not have to use the `DropCreateDatabaseIfModelChanges` Code First initializer. The code for this initializer is in the *SchoolInitializer.cs* file in the ContosoUniversity.DAL project. A setting in the `appSettings` element of the *Web.config* file causes this initializer to run whenever the application tries to access the database for the first time:

```
 <appSettings>
  <add key="Environment" value="Dev" />
  <add key="DatabaseInitializerForType ContosoUniversity.DAL.SchoolContext,
ContosoUniversity.DAL" value="ContosoUniversity.DAL.SchoolInitializer,
ContosoUniversity.DAL" />
</appSettings>
```

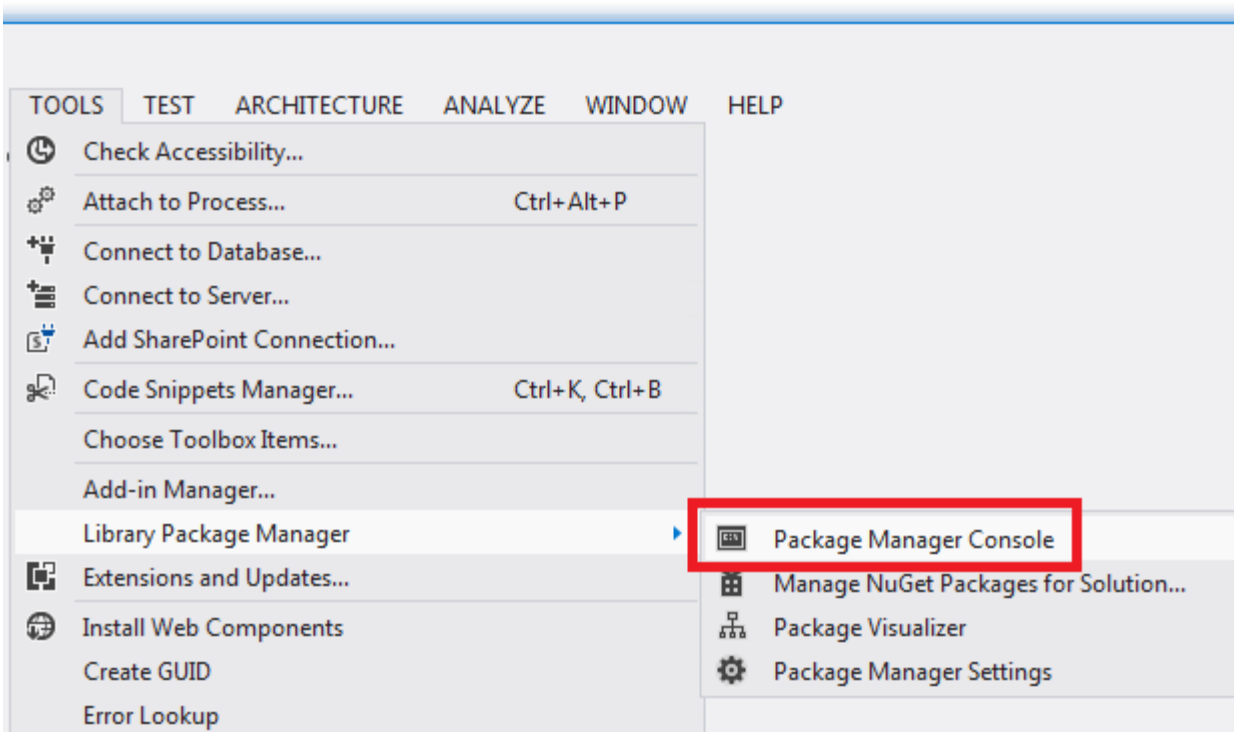
Open the application *Web.config* file and remove or comment out the `add` element that specifies the Code First initializer class. The `appSettings` element now looks like this:

```
 <appSettings>
  <add key="Environment" value="Dev" />
</appSettings>
```

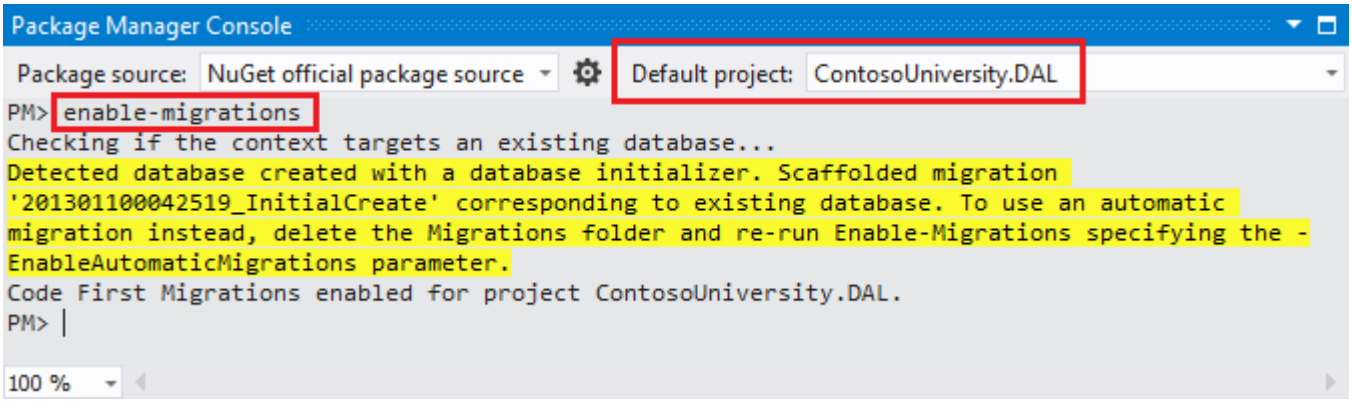
Note: Another way to specify an initializer class is do it by calling `Database.SetInitializer` in the `Application_Start` method in the *Global.asax* file. If you are enabling Migrations in a project that uses that method to specify the initializer, remove that line of code.

Enable Code First Migrations

1. Make sure that the ContosoUniversity project (not ContosoUniversity.DAL) is set as the startup project. In **Solution Explorer**, right-click the ContosoUniversity project and select **Set as Startup Project**. Code First Migrations will look in the startup project to find the database connection string.
2. From the **Tools** menu, click **Library Package Manager** and then **Package Manager Console**.

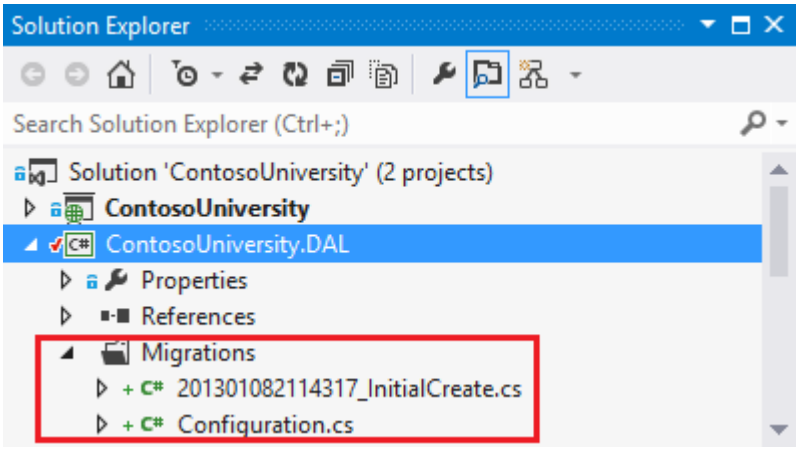


3. At the top of the **Package Manager Console** window select ContosoUniversity.DAL as the default project and then at the `PM>` prompt enter "enable-migrations".



(If you get an error saying the *enable-migrations* command is not recognized, enter the command *update-package EntityFramework -Reinstall* and try again.)

This command creates a *Migrations* folder in the ContosoUniversity.DAL project, and it puts in that folder two files: a *Configuration.cs* file that you can use to configure Migrations, and an *InitialCreate.cs* file for the first migration that creates the database.



You selected the DAL project in the **Default project** drop-down list of the **Package Manager Console** because the `enable-migrations` command must be executed in the project that contains the Code First context class. When that class is in a class library project, Code First Migrations looks for the database connection string in the startup project for the solution. In the ContosoUniversity solution, the web project has been set as the startup project. If you don't want to designate the project that has the connection string as the startup project in Visual Studio, you can specify the startup project in the PowerShell command. To see the command syntax, enter the command `get-help enable-migrations`.

The `enable-migrations` command automatically created the first migration because the database already exists. An alternative is to have Migrations create the database. To do that, use **Server Explorer** or **SQL Server Object Explorer** to delete the ContosoUniversity database before you enable Migrations. After you enable migrations, create the first migration manually by entering the command "add-migration InitialCreate". You can then create the database by entering the command "update-database".

Set up the Seed method

For this tutorial you'll add fixed data by adding code to the `Seed` method of the Code First Migrations `Configuration` class. Code First Migrations calls the `Seed` method after every migration.

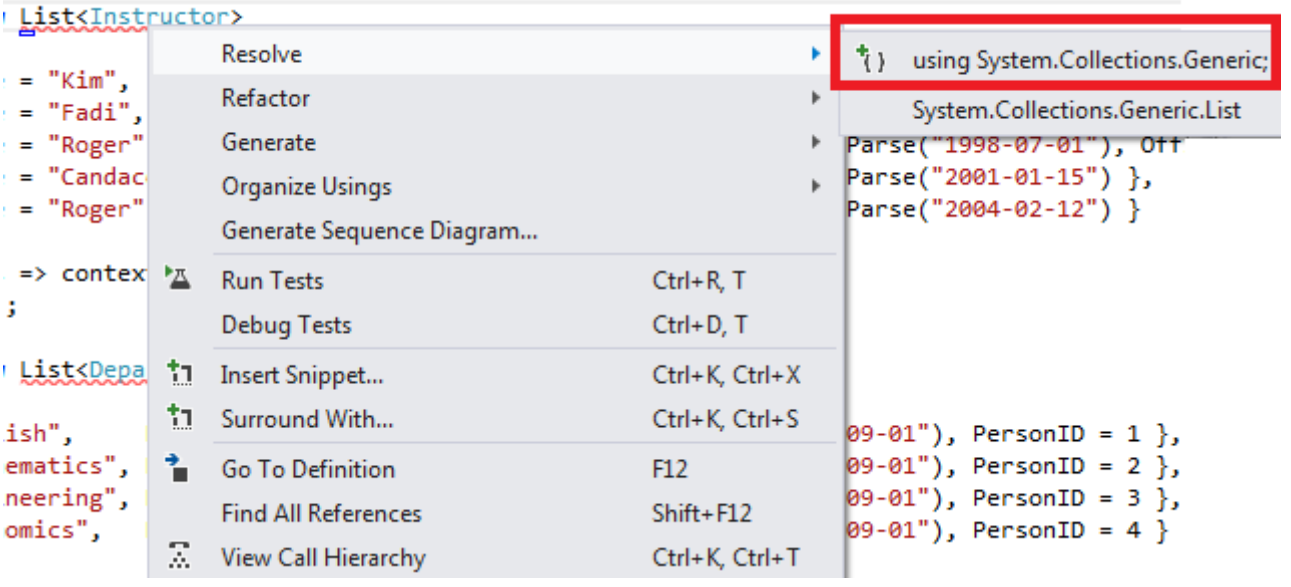
Since the `Seed` method runs after every migration, there is data already in the tables after the first migration. To handle this situation you'll use the `AddOrUpdate` method to update rows that have already been inserted, or insert them if they don't exist yet. The `AddOrUpdate` method might not be the best choice for your scenario. For more information, see Take care with EF 4.3 AddOrUpdate Method on Julie Lerman's blog.

1. Open the *Configuration.cs* file and replace the comments in the `Seed` method with the following code:

```
var instructors = new List<Instructor>
{
    new Instructor { FirstMidName = "Kim",     LastName =
"Abercrombie", HireDate = DateTime.Parse("1995-03-11"),
OfficeAssignment = new OfficeAssignment { Location = "Smith 17" } },
    new Instructor { FirstMidName = "Fadi",    LastName =
"Fakhouri",    HireDate = DateTime.Parse("2002-07-06"),
OfficeAssignment = new OfficeAssignment { Location = "Gowan 27" } },
    new Instructor { FirstMidName = "Roger",   LastName =
"Harui",       HireDate = DateTime.Parse("1998-07-01"),
OfficeAssignment = new OfficeAssignment { Location = "Thompson 304" }
},
    new Instructor { FirstMidName = "Candace", LastName =
"Kapoor",      HireDate = DateTime.Parse("2001-01-15") },
    new Instructor { FirstMidName = "Roger",   LastName =
"Zheng",       HireDate = DateTime.Parse("2004-02-12") }
};
instructors.ForEach(s => context.Instructors.AddOrUpdate(i =>
i.LastName, s));
context.SaveChanges();

var departments = new List<Department>
{
    new Department { Name = "English",     Budget = 350000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 1 },
    new Department { Name = "Mathematics", Budget = 100000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 2 },
    new Department { Name = "Engineering", Budget = 350000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 3 },
    new Department { Name = "Economics",   Budget = 100000, StartDate =
DateTime.Parse("2007-09-01"), PersonID = 4 }
};
departments.ForEach(s => context.Departments.AddOrUpdate(d => d.Name,
s));
context.SaveChanges();

var courses = new List<Course>
{
    new Course { CourseID = 1050, Title = "Chemistry",      Credits =
3, DepartmentID = 3 },
    new Course { CourseID = 4022, Title = "Microeconomics", Credits =
3, DepartmentID = 4 },
    new Course { CourseID = 4041, Title = "Macroeconomics", Credits =
3, DepartmentID = 4 },
    new Course { CourseID = 1045, Title = "Calculus",       Credits =
4, DepartmentID = 2 },
    new Course { CourseID = 3141, Title = "Trigonometry",   Credits =
4, DepartmentID = 2 },
    new Course { CourseID = 2021, Title = "Composition",    Credits =
3, DepartmentID = 1 },
    new Course { CourseID = 2042, Title = "Literature",     Credits =
4, DepartmentID = 1 }
};
courses.ForEach(s => context.Courses.AddOrUpdate(s));
context.SaveChanges();

courses[0].Instructors.Add(instructors[0]);
```

```
courses[0].Instructors.Add(instructors[1]);
courses[1].Instructors.Add(instructors[2]);
courses[2].Instructors.Add(instructors[2]);
courses[3].Instructors.Add(instructors[3]);
courses[4].Instructors.Add(instructors[3]);
courses[5].Instructors.Add(instructors[3]);
courses[6].Instructors.Add(instructors[3]);
context.SaveChanges();
```

2. The references to `List` have red squiggly lines under them because you don't have a `using` statement for its namespace yet. Right-click one of the instances of `List` and click **Resolve**, and then click **using System.Collections.Generic**.



   This menu selection adds the following code to the `using` statements near the top of the file.

```
using System.Collections.Generic;
```

3. Press CTRL-SHIFT-B to build the project.

The project is now ready to deploy the *ContosoUniversity* database. After you deploy the application, the first time you run it and navigate to a page that accesses the database, Code First will create the database and run this `Seed` method.

Note: Adding code to the `Seed` method is one of many ways that you can insert fixed data into the database. An alternative is to add code to the `Up` and `Down` methods of each migration class. The `Up` and `Down` methods contain code that implements database changes. You'll see examples of them in the Deploying a Database Update tutorial.

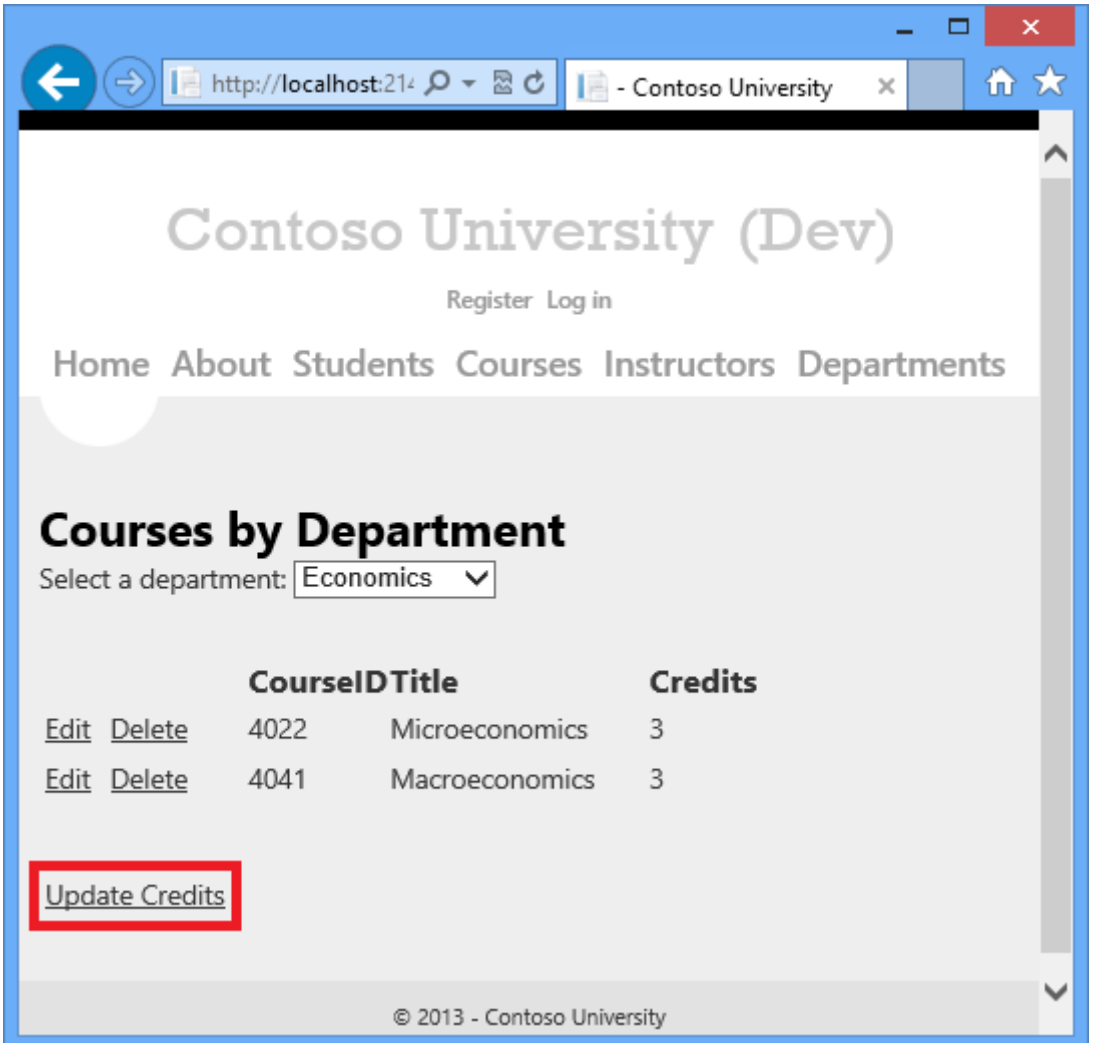
You can also write code that executes SQL statements by using the `Sql` method. For example, if you were adding a Budget column to the Department table and wanted to initialize all department budgets to $1,000.00 as part of a migration, you could add the folllowing line of code to the `Up` method for that migration:

```
Sql("UPDATE Department SET Budget = 1000");
```

Create scripts for membership database deployment

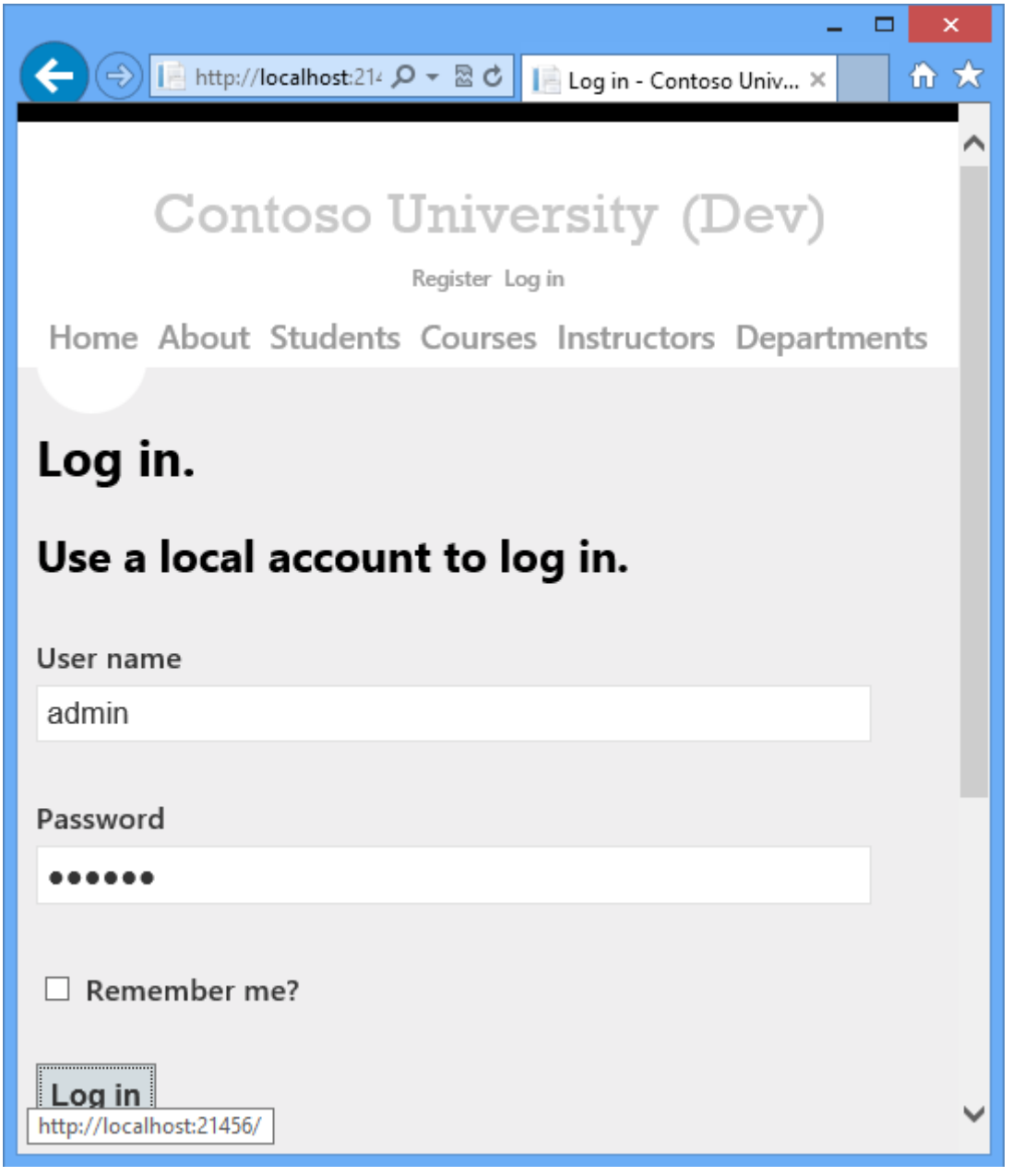
The Contoso University application uses the ASP.NET membership system and forms authentication to authenticate and authorize users. The **Update Credits** page is accessible only to users who are in the Administrator role.

Run the application and click **Courses**, and then click **Update Credits**.

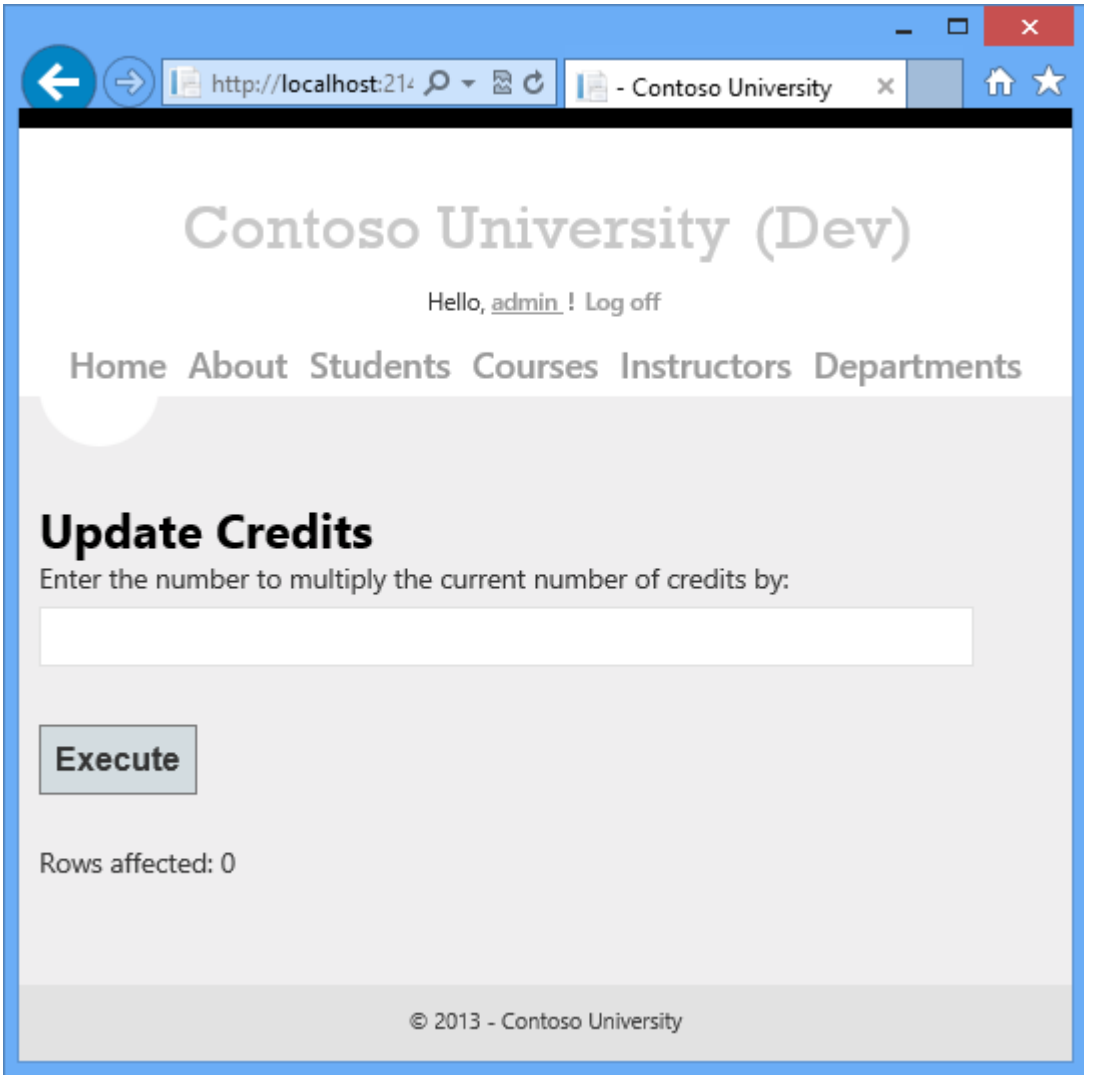


The **Log in** page appears because the **Update Credits** page requires administrative privileges.

Enter *admin* as the user name and *devpwd* as the password and click **Log in**.



The **Update Credits** page appears.



User and role information is in the *aspnet-ContosoUniversity* database that is specified by the **DefaultConnection** connection string in the *Web.config* file.

This database is not managed by Entity Framework Code First, so you can't use Migrations to deploy it. You'll use the dbDacFx provider to deploy the database schema, and you'll configure the publish profile to run a script that will insert initial data into database tables.

Here too, you typically don't want the same data in production that you have in development. When you deploy a site for the first time, it is common to exclude most or all of the user accounts you create for testing. Therefore, the downloaded project has two membership databases: *aspnet-ContosoUniversity.mdf* with development users and *aspnet-ContosoUniversity-Prod.mdf* with production users. For this tutorial the user names are the same in both databases: *admin* and *nonadmin*. Both users have the password *devpwd* in the development database and *prodpwd* in the production database. In this tutorial you'll deploy the development users to the test environment and the production users to staging and production.

Note: The membership database stores a hash of account passwords. In order to deploy accounts from one machine to another, you must make sure that hashing routines don't generate different hashes on the destination server than they do on the source computer. They will generate the same hashes when you use the ASP.NET Universal Providers, as long as you don't change the default algorithm. The default algorithm is HMACSHA256 and is specified in the **validation** attribute of the **machineKey** element in the Web.config file.

You can create data deployment scripts manually, by using SQL Server Management Studio (SSMS), or by using a third-party tool. This remainder of this tutorial will show you how to do it in SSMS, but if you don't want to install and use SSMS you can get the scripts from the completed version of the project and skip to the section where you store them in the solution folder.

To install SSMS, install it from Download Center: Microsoft SQL Server 2012 Express by clicking ENU\x64\SQLManagementStudio_x64_ENU.exe or ENU\x86\SQLManagementStudio_x86_ENU.exe. If you choose the wrong one for your system it will fail to install and you can try the other one.

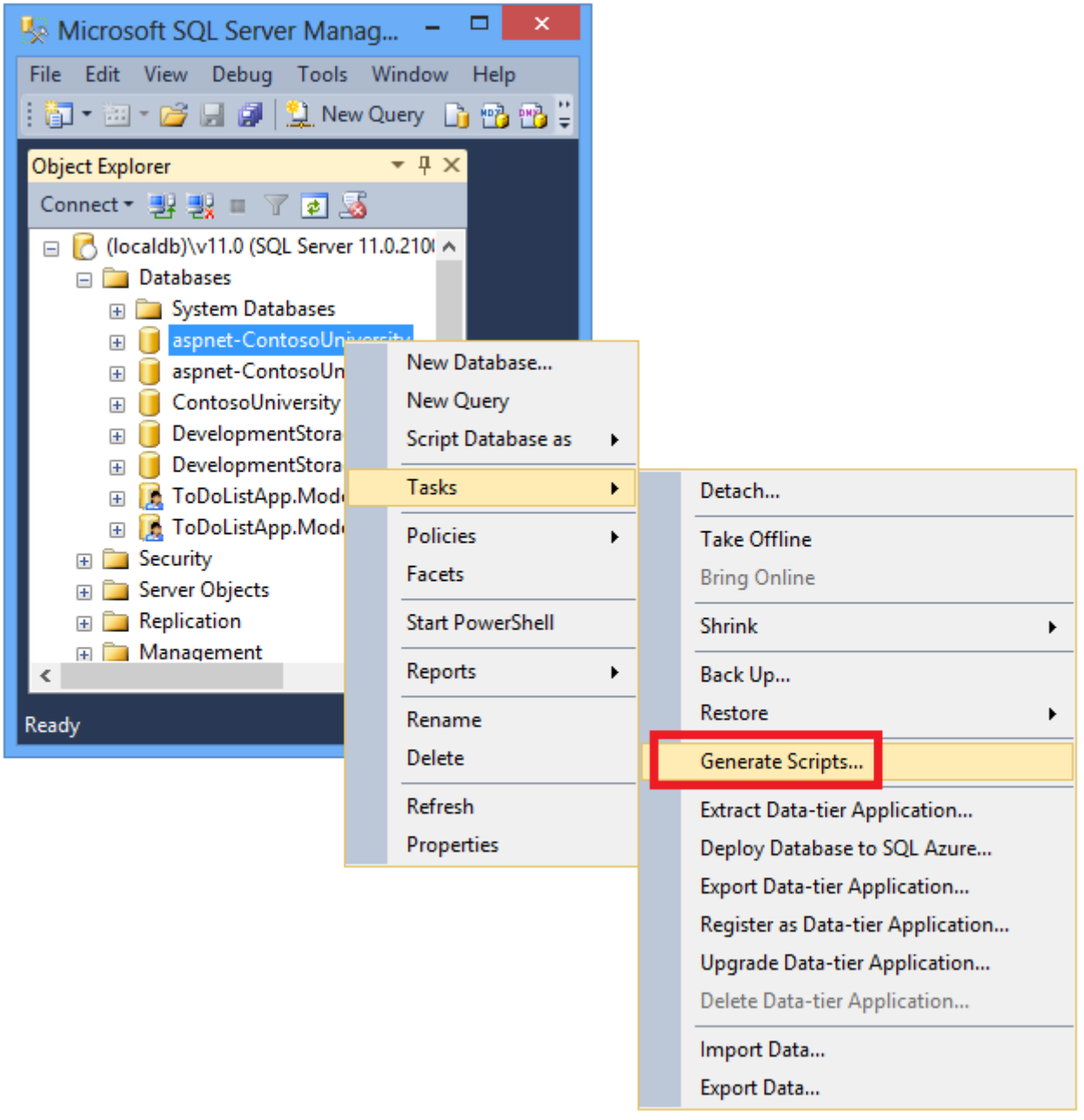
(Note that this is a 600 megabyte download. It may take a long time to install and will require a reboot of your computer.)

On the first page of the SQL Server Installation Center, click **New SQL Server stand-alone installation or add features to an existing installation**, and follow the instructions, accepting the default choices.

Create the development database script

1. Run SSMS.
2. In the **Connect to Server** dialog box, enter *(localdb)\v11.0* as the **Server name**, leave **Authentication** set to **Windows Authentication**, and then click **Connect**.

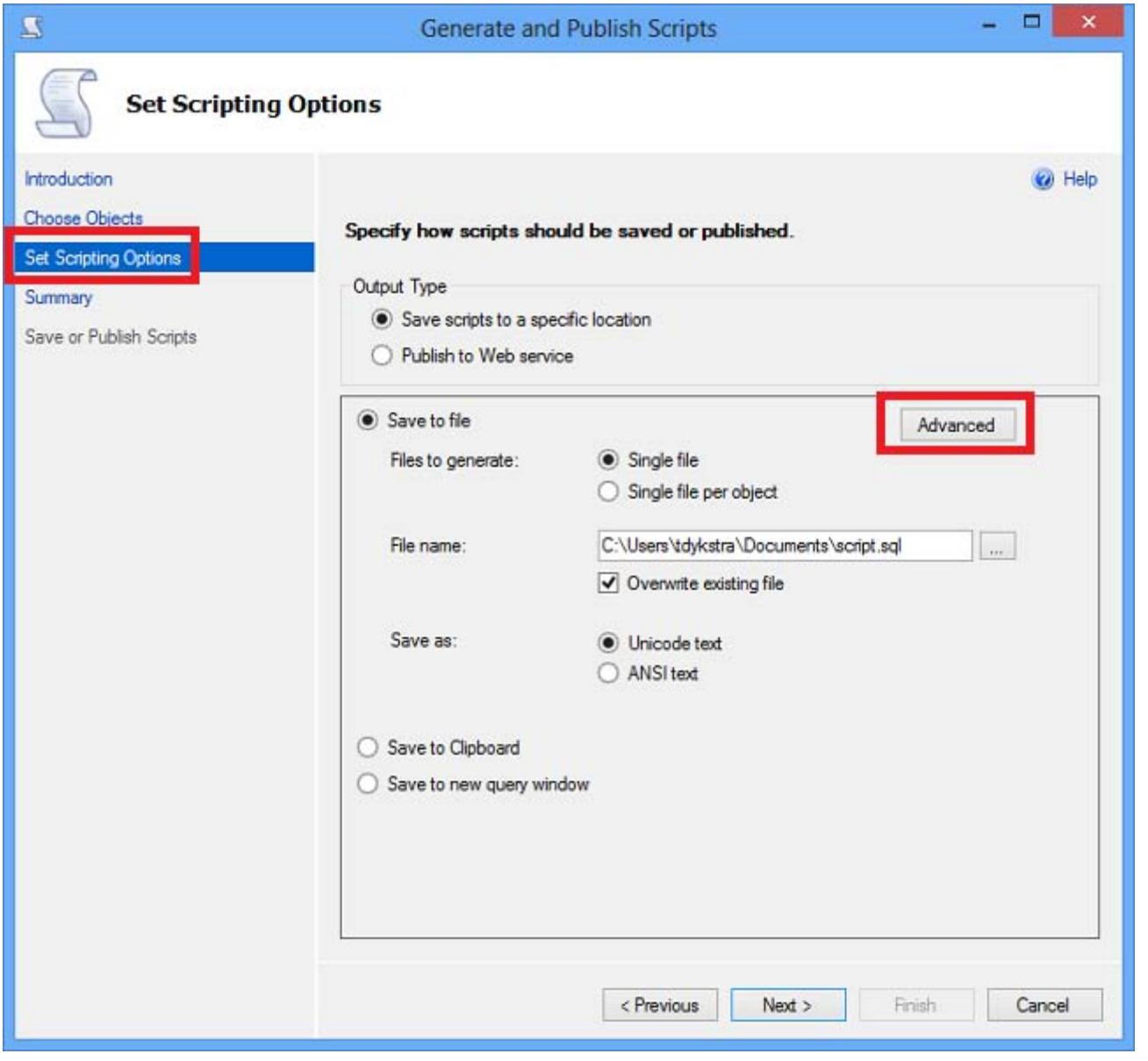
3. In the **Object Explorer** window, expand **Databases**, right-click **aspnet-ContosoUniversity**, click **Tasks**, and then click **Generate Scripts**.



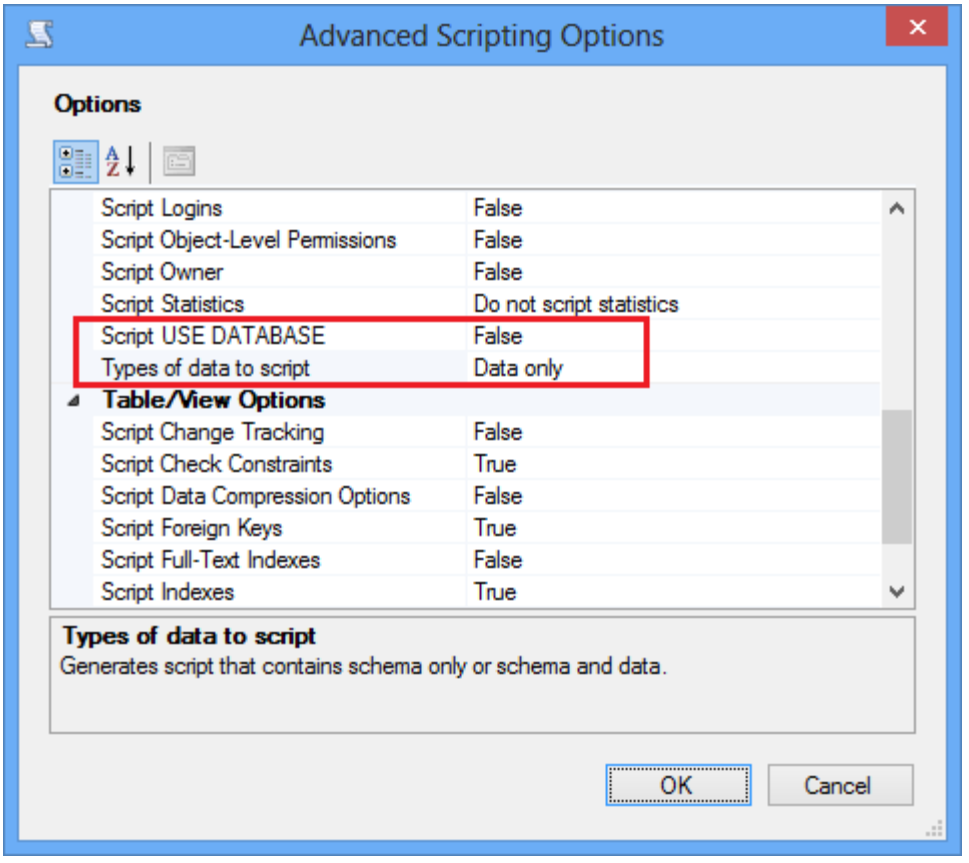
4. In the **Generate and Publish Scripts** dialog box, click **Set Scripting Options**.

   You can skip the **Choose Objects** step because the default is **Script entire database and all database objects** and that is what you want.

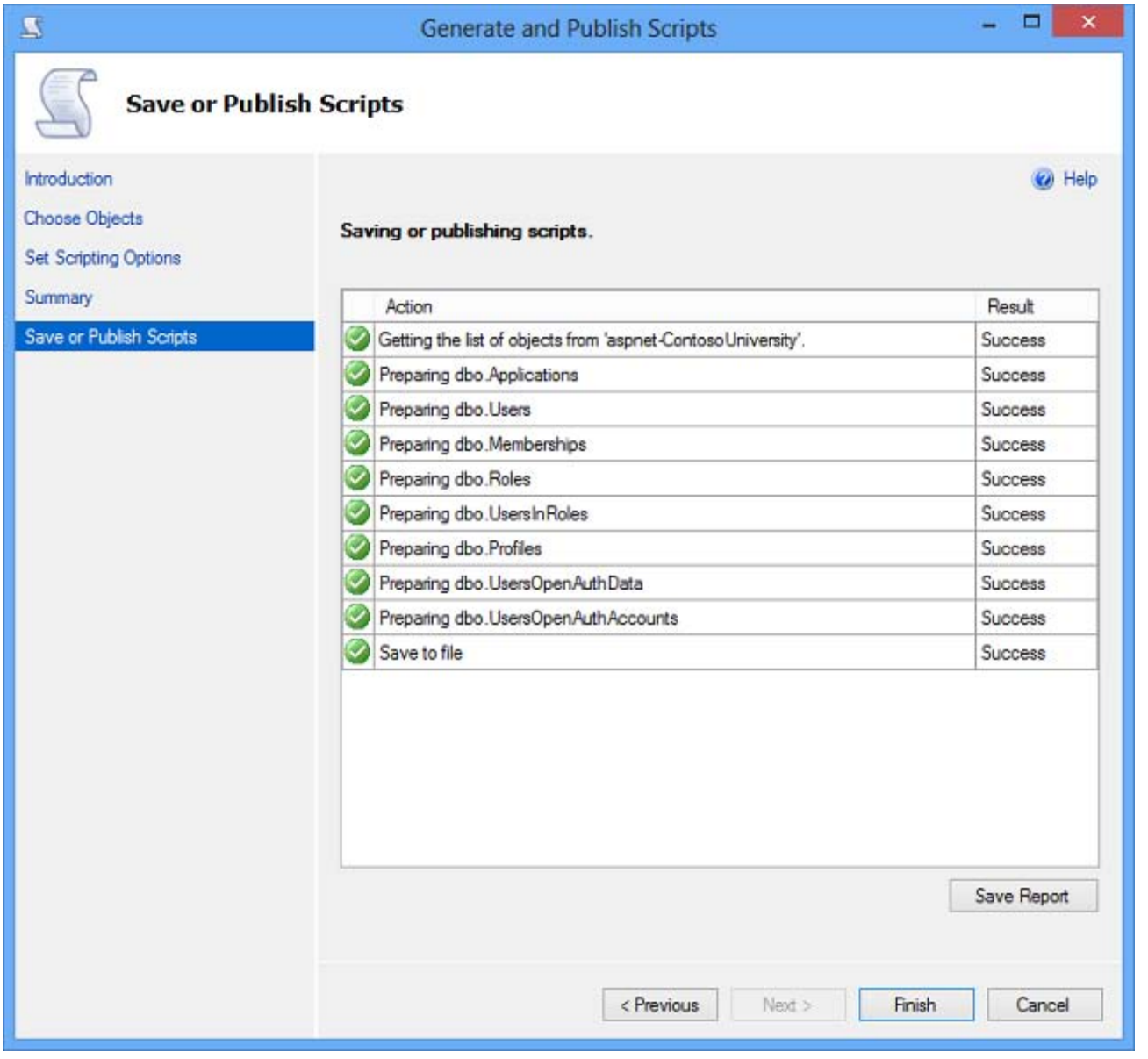
5. Click **Advanced**.



6. In the **Advanced Scripting Options** dialog box, scroll down to **Types of data to script**, and click the **Data only** option in the drop-down list.
7. Change **Script USE DATABASE** to **False**. USE statements aren't valid for Windows Azure SQL Database and aren't needed for deployment to SQL Server Express in the test environment.



8. Click **OK**.
9. In the **Generate and Publish Scripts** dialog box, the **File name** box specifies where the script will be created. Change the path to your solution folder (the folder that has your ContosoUniversity.sln file) and the file name to *aspnet-data-dev.sql*.
10. Click **Next** to go to the **Summary** tab, and then click **Next** again to create the script.

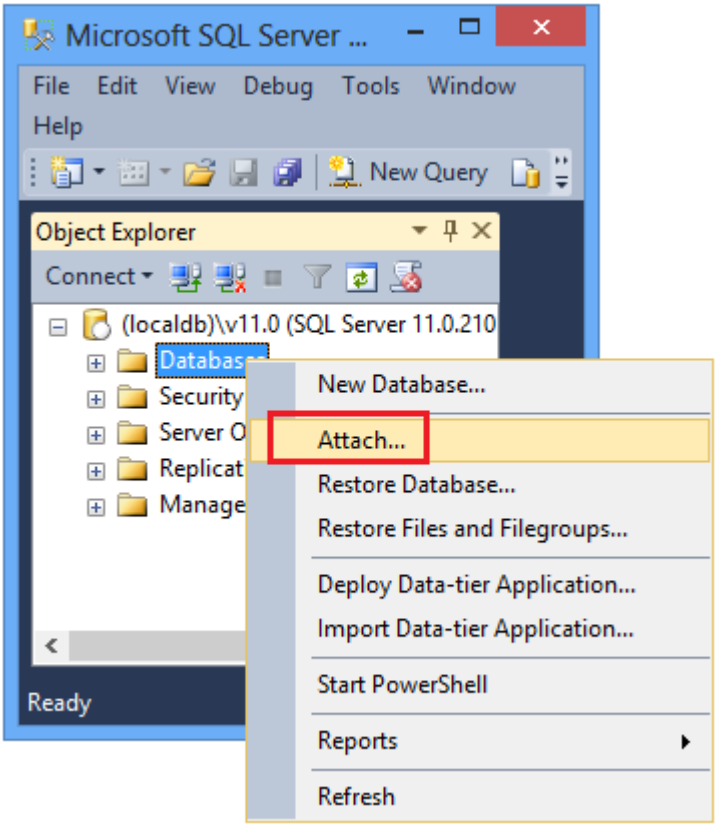


11. Click **Finish**.

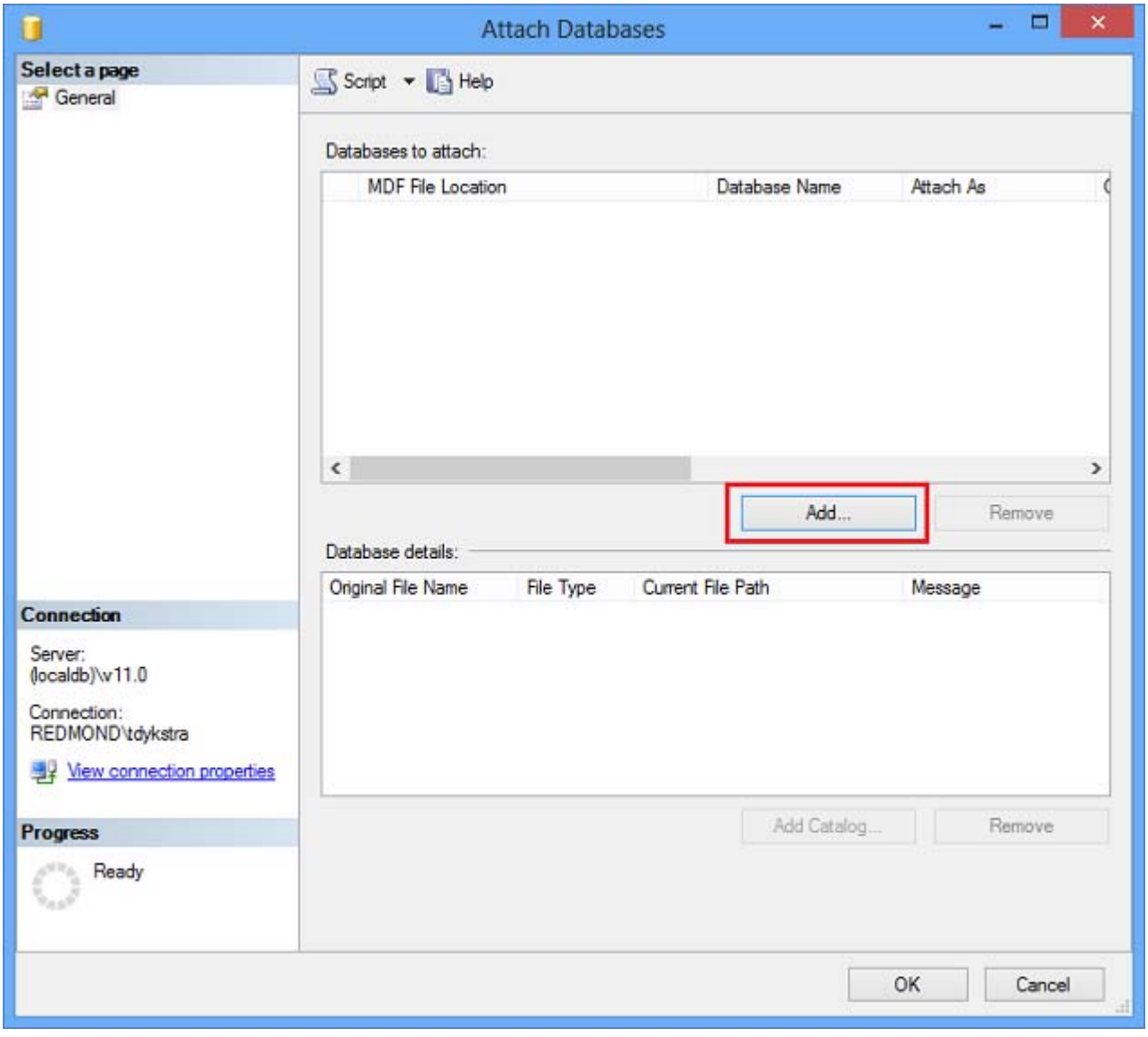
Create the production database script

Since you haven't run the project with the production database, it isn't attached yet to the LocalDB instance. Therefore you need to attach the database first.

1. In the SSMS **Object Explorer**, right-click **Databases** and click **Attach**.



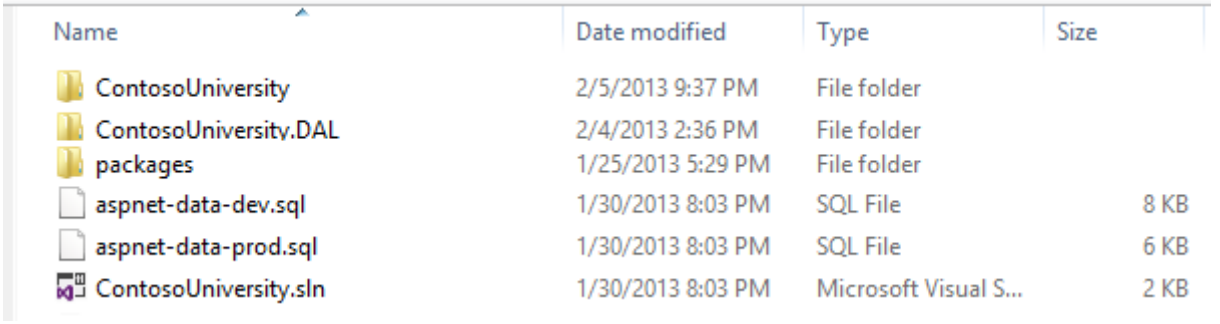
2. In the **Attach Databases** dialog box, click **Add** and then navigate to the *aspnet-ContosoUniversity-Prod.mdf* file in the *App_Data* folder.



3. Click **OK**.
4. Follow the same procedure you used earlier to create a script for the production file. Name the script file *aspnet-data-prod.sql*.

Summary

Both databases are now ready to be deployed and you have two data deployment scripts in your solution folder.



Name	Date modified	Type	Size
ContosoUniversity	2/5/2013 9:37 PM	File folder	
ContosoUniversity.DAL	2/4/2013 2:36 PM	File folder	
packages	1/25/2013 5:29 PM	File folder	
aspnet-data-dev.sql	1/30/2013 8:03 PM	SQL File	8 KB
aspnet-data-prod.sql	1/30/2013 8:03 PM	SQL File	6 KB
ContosoUniversity.sln	1/30/2013 8:03 PM	Microsoft Visual S...	2 KB

In the following tutorial you configure project settings that affect deployment, and you set up automatic *Web.config* file transformations for settings that must be different in the deployed application.

More Information

For more information on NuGet, see Manage Project Libraries with NuGet and NuGet Documentation. If you don't want to use NuGet, you'll need to learn how to analyze a NuGet package to determine what it does when it is installed. (For example, it might configure *Web.config* transformations, configure PowerShell scripts to run at build time, etc.) To learn more about how NuGet works, see Creating and Publishing a Package and Configuration File and Source Code Transformations.

Web.config File Transformations

Overview

This tutorial shows you how to automate the process of changing the *Web.config* file when you deploy it to different destination environments. Most applications have settings in the *Web.config* file that must be different when the application is deployed. Automating the process of making these changes keeps you from having to do them manually every time you deploy, which would be tedious and error prone.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Web.config transformations versus Web Deploy parameters

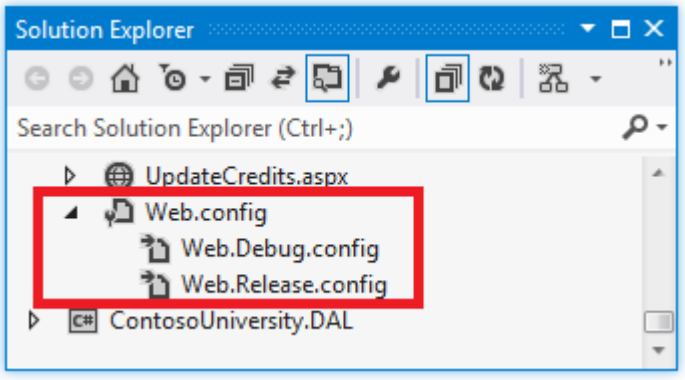
There are two ways to automate the process of changing *Web.config* file settings: Web.config transformations and Web Deploy parameters. A *Web.config* transformation file contains XML markup that specifies how to change the *Web.config* file when it is deployed. You can specify different changes for specific build configurations and for specific publish profiles. The default build configurations are Debug and Release, and you can create custom build configurations. A publish profile typically corresponds to a destination environment. (You'll learn more about publish profiles in the Deploying to IIS as a Test Environment tutorial.)

Web Deploy parameters can be used to specify many different kinds of settings that must be configured during deployment, including settings that are found in *Web.config* files. When used to specify *Web.config* file changes, Web Deploy parameters are more complex to set up, but they are useful when you do not know the value to be set until you deploy. For example, in an enterprise environment, you might create a *deployment package* and give it to a person in the IT department to install in production, and that person has to be able to enter connection strings or passwords that you do not know.

For the scenario that this tutorial series covers, you know in advance everything that has to be done to the *Web.config* file, so you do not need to use Web Deploy parameters. You'll configure some transformations that differ depending on the build configuration used, and some that differ depending on the publish profile used.

Default transformation files

In **Solution Explorer**, expand *Web.config* to see the *Web.Debug.config* and *Web.Release.config* transformation files that are created by default for the two default build configurations.



You can create transformation files for custom build configurations by right-clicking the Web.config file and choosing **Add Config Transforms** from the context menu. For this tutorial you don't need to do that because you aren't creating custom build configurations.

Later you'll create three more transformation files, one each for the test, staging, and production publish profiles. A typical example of a setting that you would handle in a publish profile transformation file because it depends on the destination environment is a WCF endpoint that is different for test versus production. You'll create publish profile transformation files in later tutorials after you create the publish profiles that they go with.

Disable debug mode

An example of a setting that depends on build configuration rather than destination environment is the `debug` attribute. For a Release build, you typically want debugging disabled regardless of which environment you are deploying to. Therefore, by default the Visual Studio project templates create *Web.Release.config* transform files with code that removes the `debug` attribute from the `compilation` element. Here is the default *Web.Release.config*: in addition to some sample transformation code that is commented out, it includes code in the `compilation` element that removes the `debug` attribute:

```
<?xml version="1.0" encoding="utf-8"?>

<!-- For more information on using web.config transformation visit
http://go.microsoft.com/fwlink/?LinkId=125889 -->

<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-
Transform">
  <!--
    In the example below, the "SetAttributes" transform will change the value
of
    "connectionString" to use "ReleaseSQLServer" only when the "Match"
locator
    finds an attribute "name" that has a value of "MyDB".

    <connectionStrings>
      <add name="MyDB"
        connectionString="Data Source=ReleaseSQLServer;Initial
```

```
Catalog=MyReleaseDB;Integrated Security=True"
        xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
    </connectionStrings>
  -->
  <system.web>
    <compilation xdt:Transform="RemoveAttributes(debug)" />
    <!--
      In the example below, the "Replace" transform will replace the entire
      <customErrors> section of your web.config file.
      Note that because there is only one customErrors section under the
      <system.web> node, there is no need to use the "xdt:Locator" attribute.

      <customErrors defaultRedirect="GenericError.htm"
        mode="RemoteOnly" xdt:Transform="Replace">
        <error statusCode="500" redirect="InternalError.htm"/>
      </customErrors>
    -->
  </system.web>
</configuration>
```

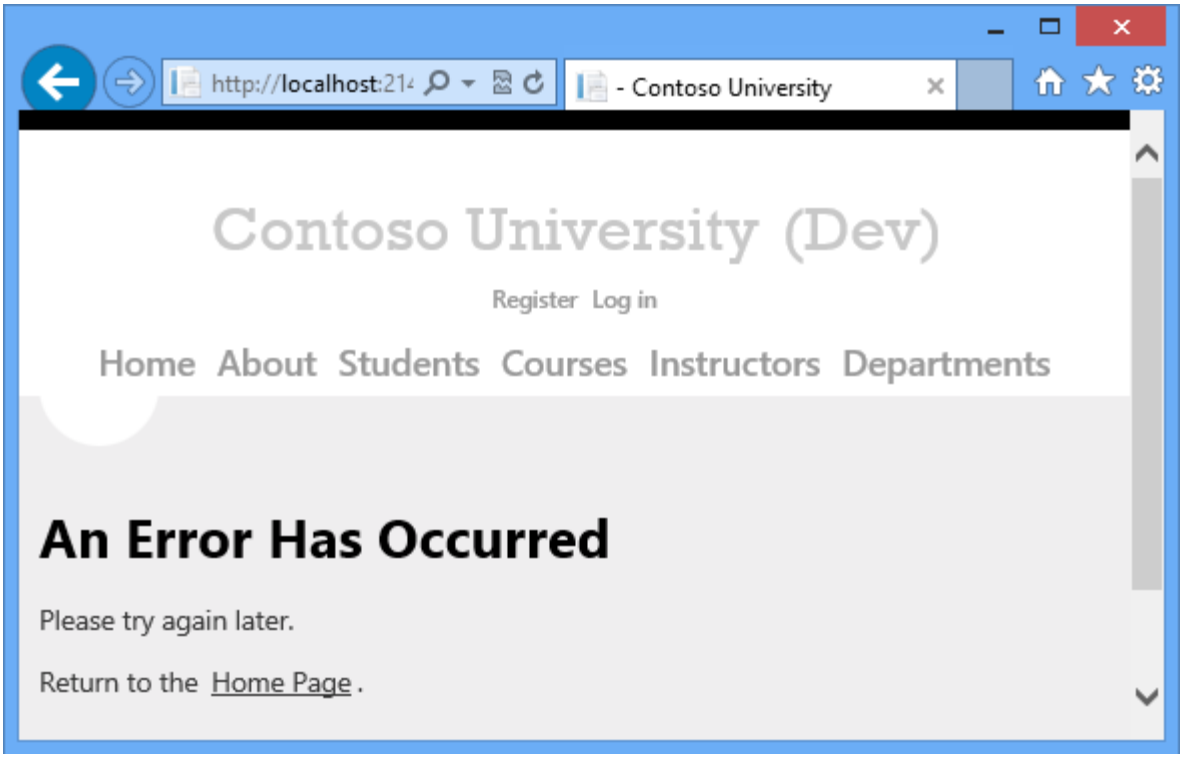
The `xdt:Transform="RemoveAttributes(debug)"` attribute specifies that you want the `debug` attribute to be removed from the `system.web/compilation` element in the deployed *Web.config* file. This will be done every time you deploy a Release build.

Limit error log access to administrators

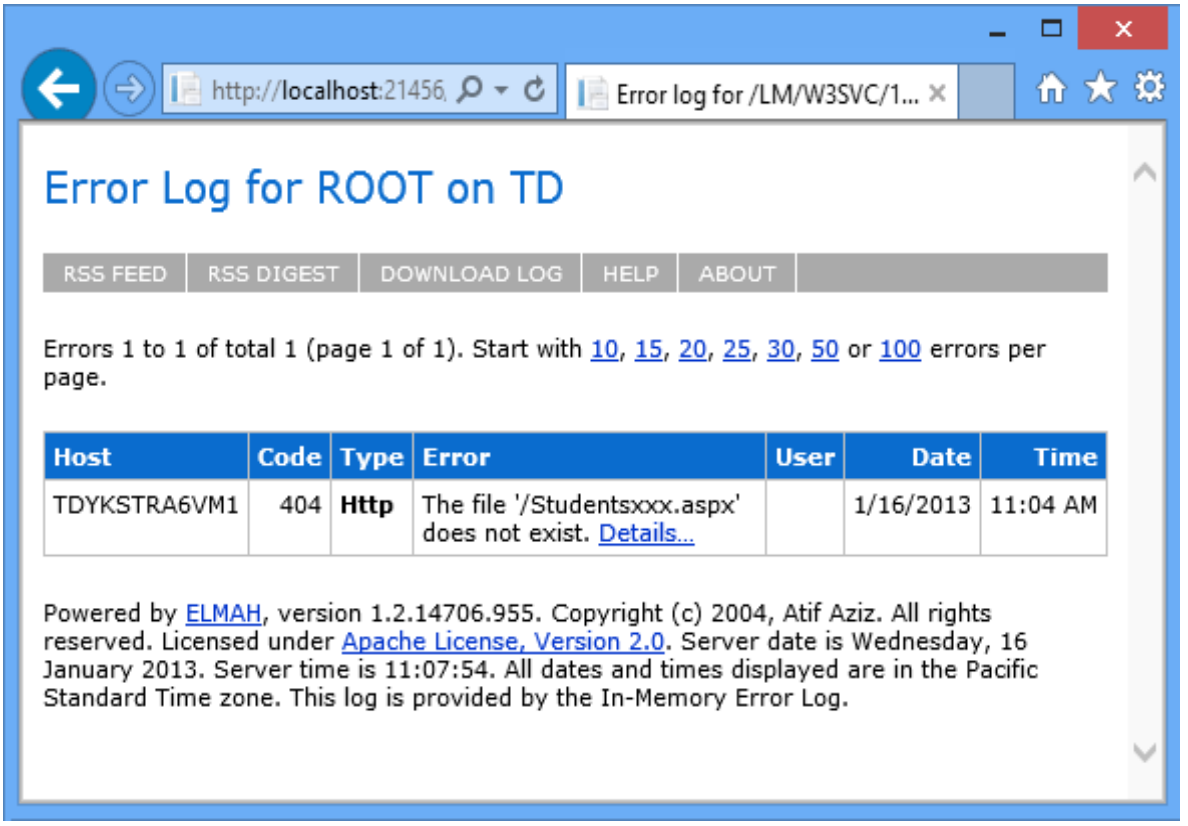
If there's an error while the application runs, the application displays a generic error page in place of the system-generated error page, and it uses the Elmah NuGet package for error logging and reporting. The `customErrors` element in the application *Web.config* file specifies the error page:

```
<customErrors mode="RemoteOnly" defaultRedirect="~/GenericErrorPage.aspx">
  <error statusCode="404" redirect="~/GenericErrorPage.aspx" />
</customErrors>
```

To see the error page, temporarily change the `mode` attribute of the `customErrors` element from "RemoteOnly" to "Off" and run the application from Visual Studio. Cause an error by requesting an invalid URL, such as *Studentsxxx.aspx*. Instead of an IIS-generated "The resource cannot be found" error page, you see the *GenericErrorPage.aspx* page.



To see the error log, replace everything in the URL after the port number with *elmah.axd* (for example, *http://localhost:51130/elmah.axd*) and press Enter:



Don't forget to set the `customErrors` element back to "RemoteOnly" mode when you're done.

On your development computer it's convenient to allow free access to the error log page, but in production that would be a security risk. For the production site, you want to add an authorization rule that restricts error log access to administrators, and to make sure that the restriction works you want it in test and staging also. Therefore this is another change that you want to implement every time you deploy a Release build, and so it belongs in the *Web.Release.config* file.

Open *Web.Release.config* and add a new `location` element immediately before the closing `configuration` tag, as shown here.

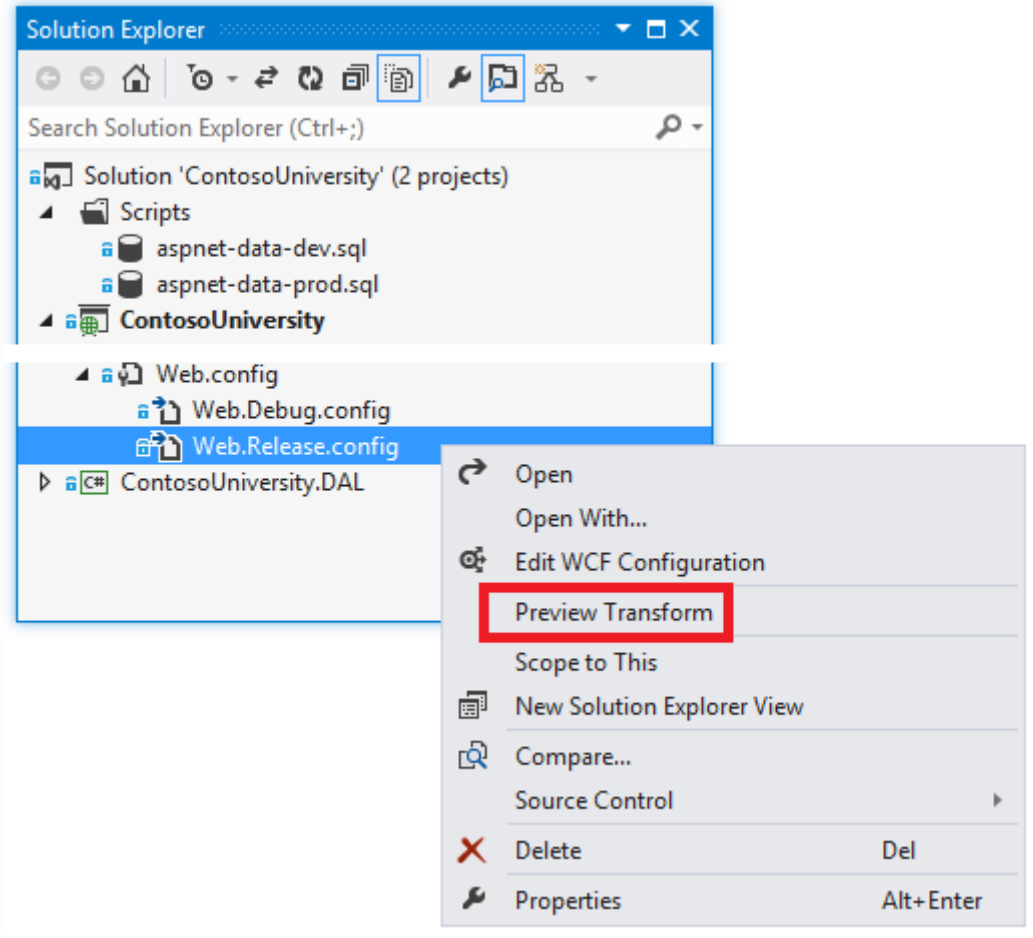
```
<configuration xmlns:xdt="http://schemas.microsoft.com/XML-Document-
Transform">
  <!--
    In the example below, the "SetAttributes" transform will change the value
of
    "connectionString" to use "ReleaseSQLServer" only when the "Match"
locator
    finds an attribute "name" that has a value of "MyDB".

    <connectionStrings>
      <add name="MyDB"
        connectionString="Data Source=ReleaseSQLServer;Initial
Catalog=MyReleaseDB;Integrated Security=True"
```

```
        xdt:Transform="SetAttributes" xdt:Locator="Match(name)"/>
    </connectionStrings>
  -->
  <system.web>
    <compilation xdt:Transform="RemoveAttributes(debug)" />
    <!--
      In the example below, the "Replace" transform will replace the entire
      <customErrors> section of your web.config file.
      Note that because there is only one customErrors section under the
      <system.web> node, there is no need to use the "xdt:Locator" attribute.

      <customErrors defaultRedirect="GenericError.htm"
        mode="RemoteOnly" xdt:Transform="Replace">
        <error statusCode="500" redirect="InternalError.htm"/>
      </customErrors>
    -->
  </system.web>
  <location path="elmah.axd" xdt:Transform="Insert">
    <system.web>
      <authorization>
        <allow roles="Administrator" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>
</configuration>
```

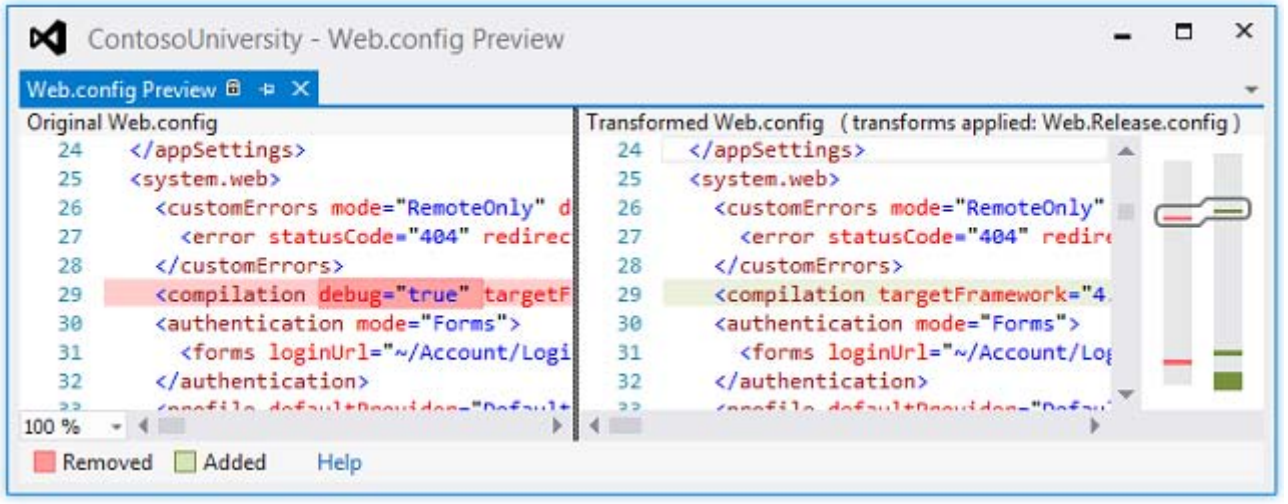
The `Transform` attribute value of "Insert" causes this `location` element to be added as a sibling to any existing `location` elements in the *Web.config* file. (There is already one `location` element that specifies authorization rules for the **Update Credits** page.)

Now you can preview the transform to make sure that you coded it correctly.

In **Solution Explorer**, right-click *Web.Release.config* and click **Preview Transform**.



A page opens that shows you the development *Web.config* file on the left and what the deployed *Web.config* file will look like on the right, with changes highlighted.



(In the preview, you might notice some additional changes that you didn't write transforms for: these typically involve the removal of white space that doesn't affect functionality.)

When you test the site after deployment, you'll also test to verify that the authorization rule is effective.

Security Note: Never display error details to the public in a production application, or store that information in a public location. Attackers can use error information to discover vulnerabilities in a site. If you use ELMAH in your own application, be sure to investigate ways in which ELMAH can be configured to minimize security risks. The ELMAH example in this tutorial should not be considered a recommended configuration. It is an example that was chosen in order to illustrate how to handle a folder that the application must be able to create files in.

A setting that you'll handle in publish profile transformation files

A common scenario is to have *Web.config* file settings that must be different in each environment that you deploy to. For example, an application that calls a WCF service might need a different endpoint in test and production environments. The Contoso University application includes a setting of this kind also. This setting controls a visible indicator on a site's pages that tells you which environment you are in, such as development, test, or production. The setting value determines whether the application will append "(Dev)" or "(Test)" to the main heading in the *Site.Master* master page:

The environment indicator is omitted when the application is running in staging or production.

The Contoso University web pages read a value that is set in `appSettings` in the *Web.config* file in order to determine what environment the application is running in:

```
<appSettings>
    <add key="Environment" value="Dev" />
</appSettings>
```

The value should be "Test" in the test environment, and "Prod" for staging and production.

The following code in a transform file will implement this transformation:

```
<appSettings>
    <add key="Environment" value="Test" xdt:Transform="SetAttributes"
xdt:Locator="Match(key)"/>
</appSettings>
```

The `xdt:Transform` attribute value "SetAttributes" indicates that the purpose of this transform is to change attribute values of an existing element in the *Web.config* file. The `xdt:Locator` attribute value "Match(key)" indicates that the element to be modified is the one whose `key` attribute matches the `key` attribute specified here. The only other attribute of the `add` element is `value`, and that is what will be changed in the deployed *Web.config* file. The code shown here causes the `value` attribute of the `Environment appSettings` element to be set to "Test" in the *Web.config* file that is deployed.

This transform belongs in the publish profile transform files, which you haven't created yet. You'll create and update the transform files that implement this change when you create the publish profiles for the test, staging, and production environments. You'll do that in the deploy to IIS and deploy to production tutorials.

Setting connection strings

Although the default transform file contains an example that shows how to update a connection string, in most cases you do not need to set up connection string transformations, because you can specify connection strings in the publish profile. You'll do that in the deploy to IIS and deploy to production tutorials.

Summary

You have now done as much as you can with *Web.config* transformations before you create the publish profiles, and you've seen a preview of what will be in the deployed Web.config file.

In the following tutorial, you'll take care of deployment set-up tasks that require setting project properties.

More Information

For more information about topics covered by this tutorial, see Using Web.config transformations to change settings in the destination Web.config file or app.config file during deployment in the Web Deployment Content Map for Visual Studio and ASP.NET.

Project Properties

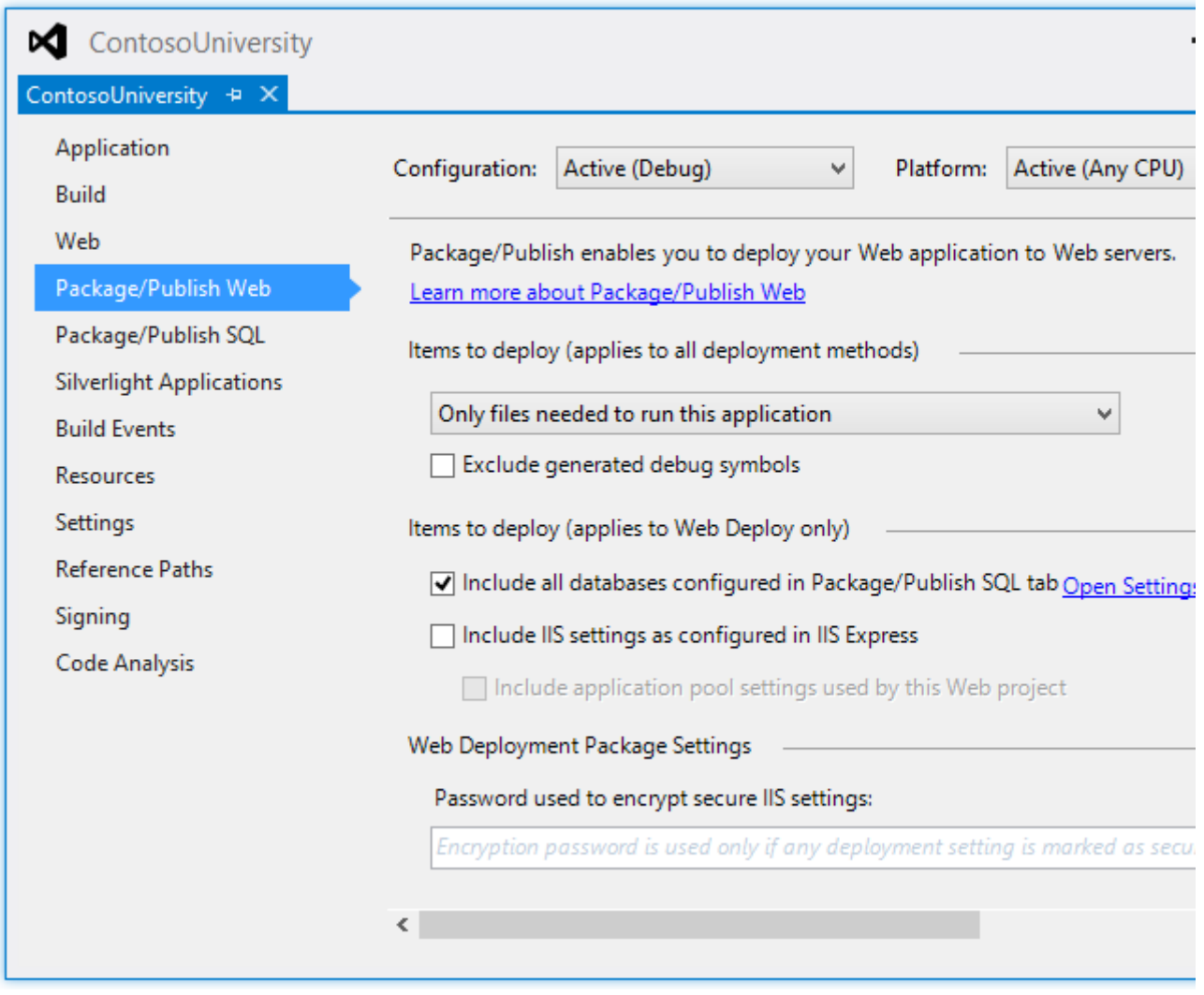
Overview

Some deployment options are configured in project properties that are stored in the project file (the *.csproj* or *.vbproj* file). In most cases, the default values of these settings are what you want, but you can use the **Project Properties** UI built into Visual Studio to work with these settings if you have to change them. In this tutorial you review the deployment settings in **Project Properties**. You also create a placeholder file that causes an empty folder to be deployed.

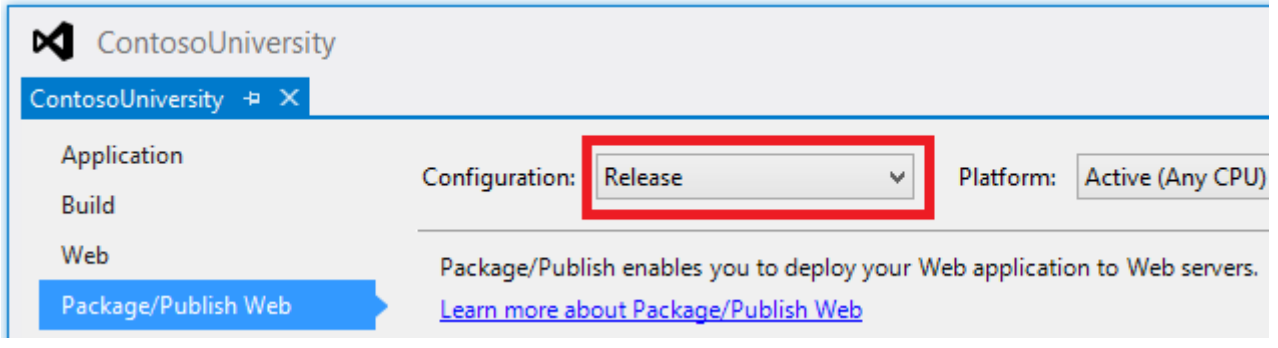
Configure deployment settings in the project properties window

Most settings that affect what happens during deployment are included in the publish profile, as you'll see in the following tutorials. A few settings that you should be aware of are located in the **Package/Publish** tabs of the **Project Properties** window. These settings are specified for each build configuration — that is, you can have different settings for a Release build than you have for a Debug build.

In **Solution Explorer**, right-click the **ContosoUniversity** project, select **Properties**, and then select the **Package/Publish Web** tab.



When the window is displayed, it defaults to showing settings for whichever build configuration is currently active for the solution. If the **Configuration** box does not indicate **Active (Release)**, select **Release** in order to display settings for the Release build configuration. You'll deploy Release builds to both your test and production environments.



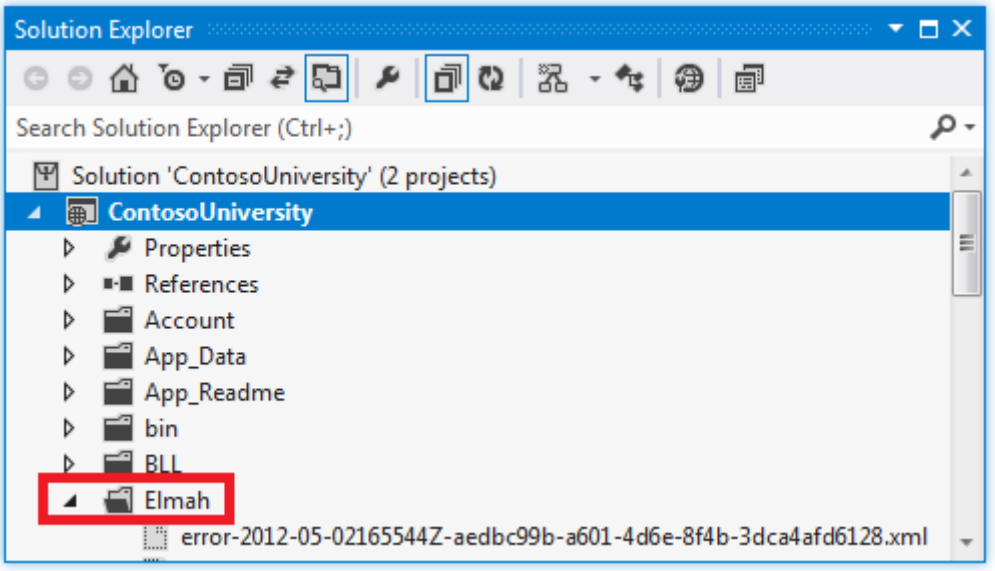
With **Active (Release)** or **Release** selected, you see the values that are effective when you deploy using the Release build configuration:

- In the **Items to deploy** box, **Only files needed to run the application** is selected. Other options are **All files in this project** or **All files in this project folder**. By leaving the default selection unchanged you avoid deploying source code files, for example. This setting is the reason why the folders that contain the SQL Server Compact binary files had to be included in the project. For more information about this setting, see **Why don't all of the files in my project folder get deployed?** in ASP.NET Web Application Project Deployment FAQ.
- **Exclude generated debug symbols** is selected. You won't be debugging when you use this build configuration.
- **Include all databases configured in Package/Publish SQL tab** is selected. Specifies whether Visual Studio will deploy databases as well as files. Although the check box label only mentions the **Package/Publish SQL** tab, clearing this check box would also disable database deployment that is configured in the publish profile. You will be doing that later, so the check box must remain selected. The **Package/Publish SQL** tab is used for a legacy database publishing method that you won't be using in these tutorials.
- The **Web Deployment Package Settings** section does not apply because you're using one-click publish in these tutorials.

Change the **Configuration** drop-down box to Debug to see the default settings for Debug builds. The values are the same, except **Exclude generated debug symbols** is cleared so that you can debug when you deploy a Debug build.

Make sure that the Elmah folder gets deployed

As you saw in the previous tutorial, the Elmah NuGet package provides functionality for error logging and reporting. In the Contoso University application Elmah has been configured to store error details in a folder named *Elmah*:



Excluding specific files or folders from deployment is a common requirement; another example would be a folder that users can upload files to. You don't want log files or uploaded files that were created in your development environment to be deployed to production. And if you are deploying an update to production you don't want the deployment process to delete files that exist in production. (Depending on how you set a deployment option, if a file exists in the destination site but not the source site when you deploy, Web Deploy deletes it from the destination.)

As you saw earlier in this tutorial, the **Items to deploy** option in the **Package/Publish Web** tab is set to **Only Files Needed to run this application**. As a result, log files that are created by Elmah in development will not be deployed, which is what you want to happen. (To be deployed, they would have to be included in the project and their **Build Action** property would have to be set to **Content**. For more information, see **Why don't all of the files in my project folder get deployed?** in ASP.NET Web Application Project Deployment FAQ). However, Web Deploy will not create a folder in the destination site unless there's at least one file to copy to it. Therefore, you'll add a *.txt* file to the folder to act as a placeholder so that the folder will be copied.

In **Solution Explorer**, right-click the *Elmah* folder, select **Add New Item**, and create a text file named *Placeholder.txt*. Put the following text in it: "This is a placeholder file to ensure that the folder gets deployed." and save the file. That's all you have to do in order to make sure that Visual Studio deploys this file and the folder it's in, because the **Build Action** property of *.txt* files is set to **Content** by default.

Summary

You have now completed all of the deployment set-up tasks. In the next tutorial, you'll deploy the Contoso University site to the test environment and test it there.

Deploying to Test

Overview

This tutorial shows how to deploy an ASP.NET web application to IIS on the local computer.

When you develop an application, you generally test by running it in Visual Studio. By default, web application projects in Visual Studio 2012 use IIS Express as the development web server. IIS Express behaves more like full IIS than the Visual Studio Development Server (also known as Cassini), which Visual Studio 2010 uses by default. But neither development web server works exactly like IIS. As a result, it's possible that an application will run correctly when you test it in Visual Studio, but fail when it's deployed to IIS.

You can test your application more reliably in these ways:

1. Deploy the application to IIS on your development computer by using the same process that you'll use later to deploy it to your production environment. You can configure Visual Studio to use IIS when you run a web project, but doing that would not test your deployment process. This method validates your deployment process in addition to validating that your application will run correctly under IIS.
2. Deploy the application to a test environment that is nearly identical to your production environment. Since the production environment for these tutorials is a Windows Azure web site, the ideal test environment is an additional Windows Azure web site. You would use this second web site only for testing, but it would be set up the same way as the production web site.

Option 2 is the most reliable way to test, and if you do that, you don't necessarily have to do option 1. However, if you are deploying to a third-party hosting provider option 2 might not be feasible or might be expensive, so this tutorial series shows both methods. Guidance for option 2 is provided in the Deploying to the Production Environment tutorial.

For more information about using web servers in Visual Studio, see Web Servers in Visual Studio for ASP.NET Web Projects.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Install IIS

To deploy to IIS on your development computer, you must have IIS and Web Deploy installed. Web Deploy is installed by default with Visual Studio, but IIS is not included in the default Windows 8 or Windows 7 configuration. If you have already installed IIS and the default application pool is already set to .NET 4, skip to the next section.

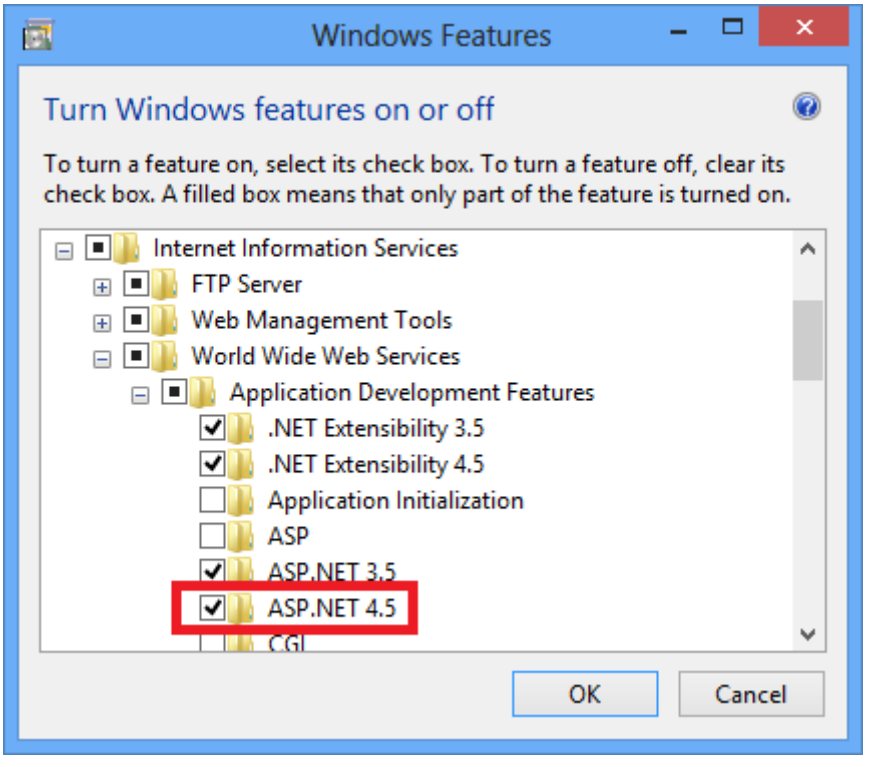
1. Using the Web Platform Installer is the preferred way to install IIS and Web Deploy, because the Web Platform Installer installs a recommended configuration for IIS and it automatically installs the prerequisites for IIS and Web Deploy if necessary.

   To run Web Platform Installer to install IIS and Web Deploy, use the following link. If you already have installed IIS, Web Deploy or any of their required components, the Web Platform Installer installs only what is missing.

   - Install IIS and Web Deploy using WebPI

   You'll see messages indicating that IIS 7 will be installed. The link works for IIS 8 in Windows 8, but for Windows 8 make sure that ASP.NET 4.5 is installed by performing the following steps:

   2. Open **Control Panel**, **Programs and Features**, **Turn Windows features on or off**.
   3. Expand **Internet Information Services**, **World Wide Web Services**, and **Application Development Features**.
   4. Make sure that **ASP.NET 4.5** is selected.

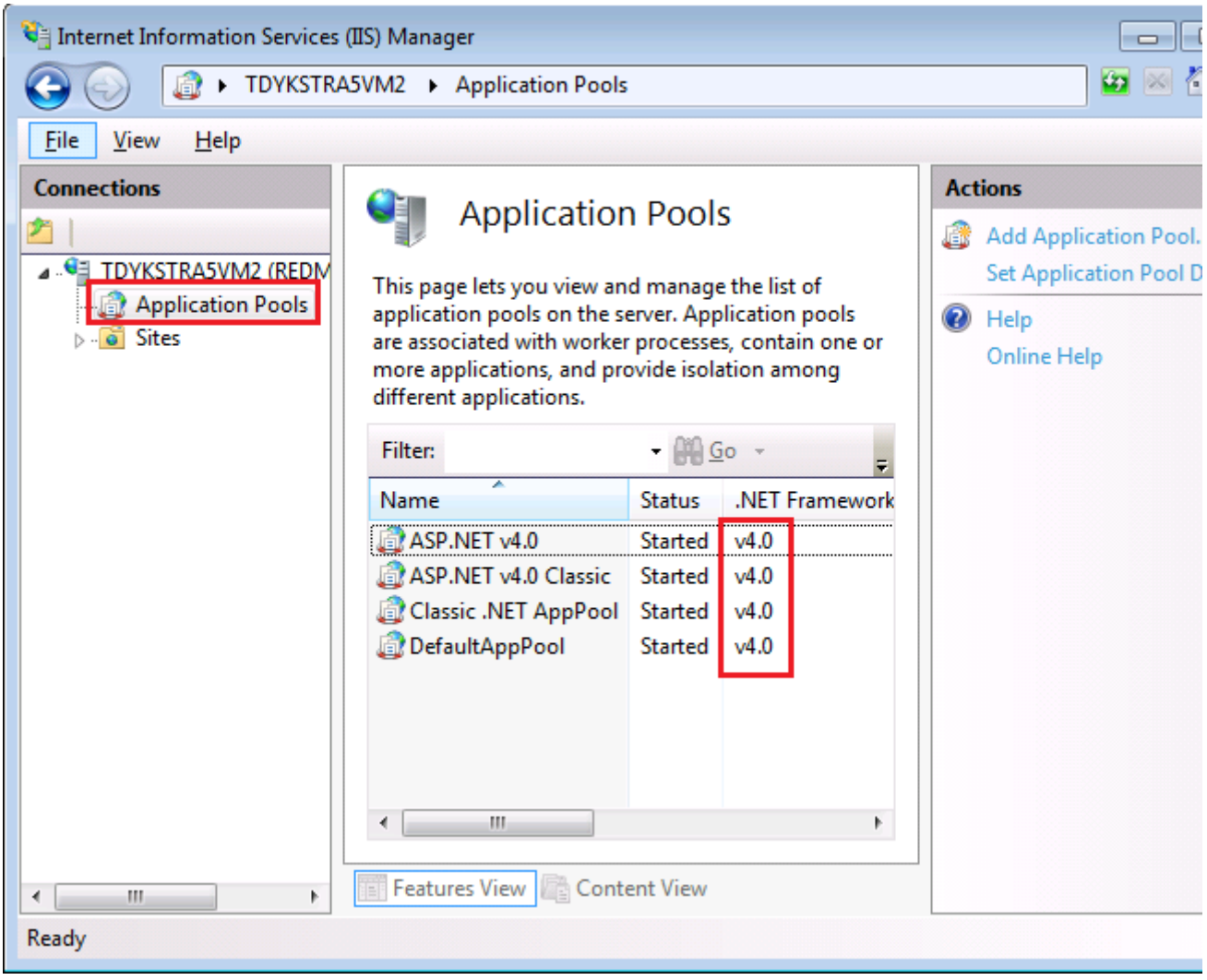


After installing IIS, run **IIS Manager** to make sure that the .NET Framework version 4 is assigned to the default application pool.

1. Press WINDOWS+R to open the **Run** dialog box.

(Or in Windows 8 enter "run" on the **Start** page, or in Windows 7 select **Run** from the **Start** menu. If **Run** isn't in the **Start** menu, right-click the taskbar, click **Properties**, select the **Start Menu** tab, click **Customize**, and select **Run command**.)

2. Enter "inetmgr", and then click **OK**.
3. In the **Connections** pane, expand the server node and select **Application Pools**. In the **Application Pools** pane, if **DefaultAppPool** is assigned to the .NET framework version 4 as in the following illustration, skip to the next section.



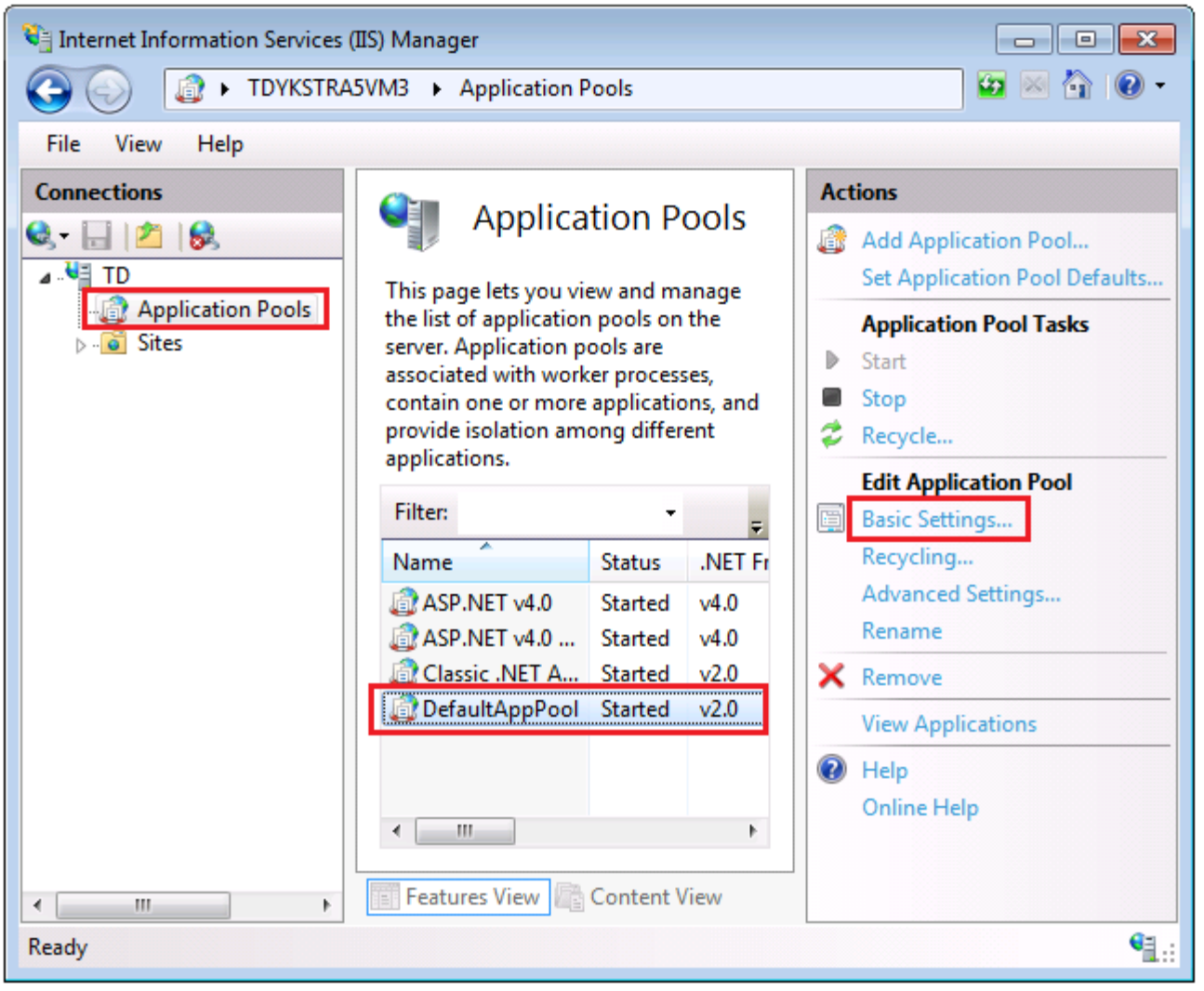
4. If you see only two application pools and both of them are set to the .NET Framework 2.0, you have to install ASP.NET 4 in IIS.

   For Windows 8, see the instructions in the previous section for making sure that ASP.NET 4.5 is installed. For Windows 7, open a command prompt window by right-clicking **Command Prompt** in the Windows **Start** menu and selecting **Run as Administrator**. Then run aspnet_regiis.exe to install ASP.NET 4 in IIS, using the following commands. (In 32-bit systems, replace "Framework64" with "Framework".)

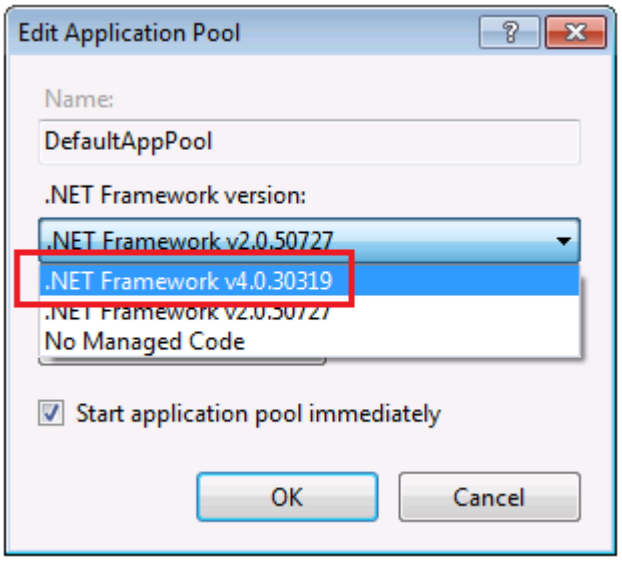
```
cd %windir%\Microsoft.NET\Framework64\v4.0.30319
aspnet_regiis.exe -i
```

This command creates new application pools for the .NET Framework 4, but the default application pool will still be set to 2.0. You'll be deploying an application that targets .NET 4 to that application pool, so you have to change the application pool to .NET 4.

5. If you closed **IIS Manager**, run it again, expand the server node, and click **Application Pools** to display the **Application Pools** pane again.
6. In the **Application Pools** pane, click **DefaultAppPool**, and then in the **Actions** pane click **Basic Settings**.



7. In the **Edit Application Pool** dialog box, change **.NET Framework version** to **.NET Framework v4.0.30319** and click **OK**.



IIS is now ready for you to publish a web application to it, but before you can do that you have to create the databases that you will use in the test environment.

Install SQL Server Express

LocalDB is not designed to work in IIS, so for your test environment you need to have SQL Server Express installed. If you are using Visual Studio 2010 SQL Server Express is already installed by default. If you are using Visual Studio 2012, you have to install it.

To install SQL Server Express, install it from Download Center: Microsoft SQL Server 2012 Express by clicking ENU\x64\SQLEXPR_x64_ENU.exe or ENU\x86\SQLEXPR_x86_ENU.exe. If you choose the wrong one for your system it will fail to install and you can try the other one.

On the first page of the SQL Server Installation Center, click **New SQL Server stand-alone installation or add features to an existing installation**, and follow the instructions, accepting the default choices. In the installation wizard accept the default settings. For more information about installation options, see Install SQL Server 2012 from the Installation Wizard (Setup).

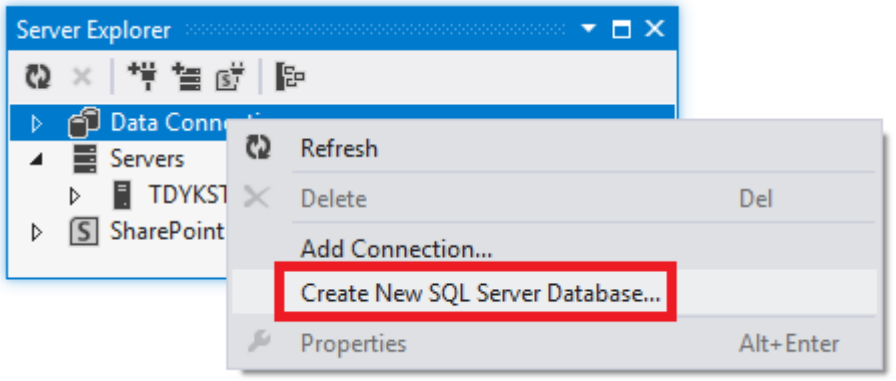
Create SQL Server Express databases for the test environment

The Contoso University application has two databases: the membership database and the application database. You can deploy these databases to two separate databases or to a single database. You might want to combine them in order to facilitate database joins between your application database and your membership database. If you are deploying to a third-party hosting provider, your hosting plan might also provide a reason to combine them. For example, the

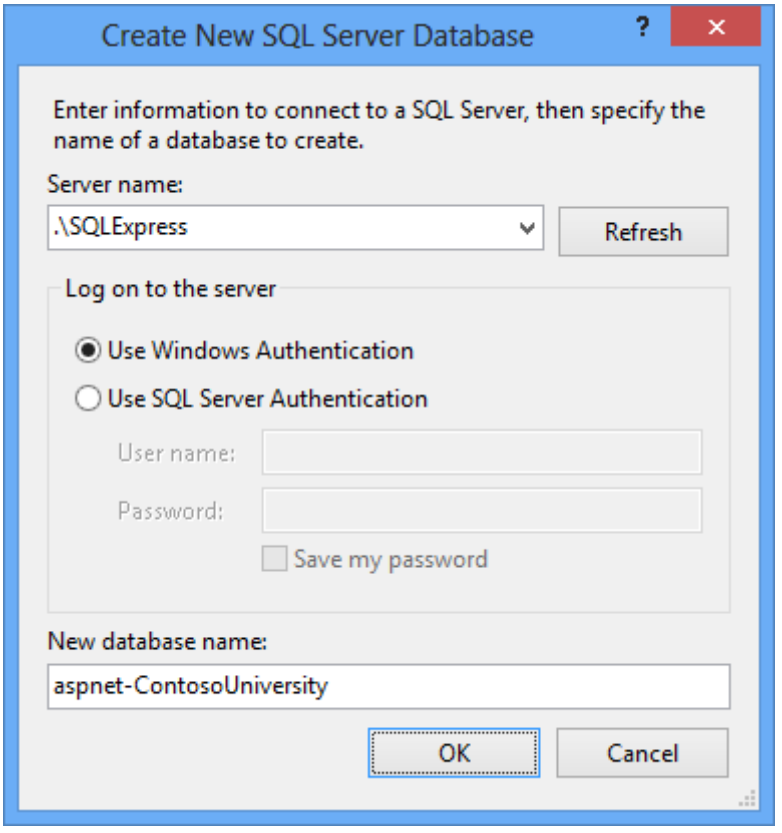
hosting provider might charge more for multiple databases or might not even allow more than one database.

In this tutorial, you'll deploy to two databases in the test environment, and to one database in the staging and production environments.

From the **View** menu select **Server Explorer** (**Database Explorer** in Visual Web Developer), and then right-click **Data Connections** and select **Create New SQL Server Database**.

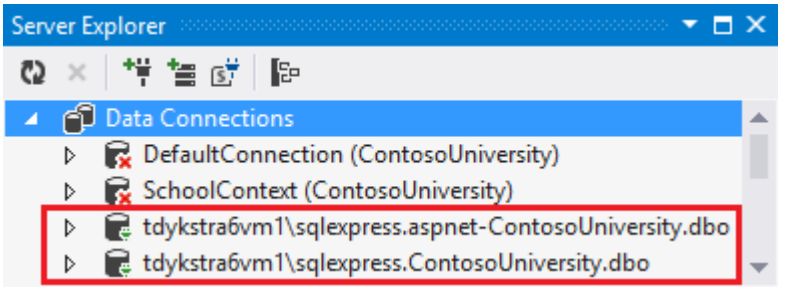


In the **Create New SQL Server Database** dialog box, enter ".\SQLExpress" in the **Server name** box and "aspnet-ContosoUniversity" in the **New database name** box, then click **OK**.



Follow the same procedure to create a new SQL Server Express School database named "ContosoUniversity".

Server Explorer now shows the two new databases.



Create a grant script for the new databases

When the application runs in IIS on your development computer, the application accesses the database by using the default application pool's credentials. However, by default, the application pool identity does not have permission to open the databases. So you have to run a script to grant that permission. In this section you create the script that you'll run later to make sure that the application can open the databases when it runs in IIS.

Right-click the solution (not one of the projects), and click **Add New Item**, and then create a new **SQL File** named *Grant.sql*. Copy the following SQL commands into the file, and then save and close the file:

```
IF NOT EXISTS (SELECT name FROM sys.server_principals WHERE name = 'IIS
APPPOOL\DefaultAppPool')
BEGIN
    CREATE LOGIN [IIS APPPOOL\DefaultAppPool]
      FROM WINDOWS WITH DEFAULT_DATABASE=[master],
      DEFAULT_LANGUAGE=[us_english]
END
GO
CREATE USER [ContosoUniversityUser]
  FOR LOGIN [IIS APPPOOL\DefaultAppPool]
GO
EXEC sp_addrolemember 'db_owner', 'ContosoUniversityUser'
GO
```

Note: This script is designed to work with SQL Server Express 2012 and with the IIS settings in Windows 8 or Windows 7 as they are specified in this tutorial. If you're using a different version of SQL Server or of Windows, or if you set up IIS on your computer differently, changes to this script might be required. For more information about SQL Server scripts, see SQL Server Books Online.

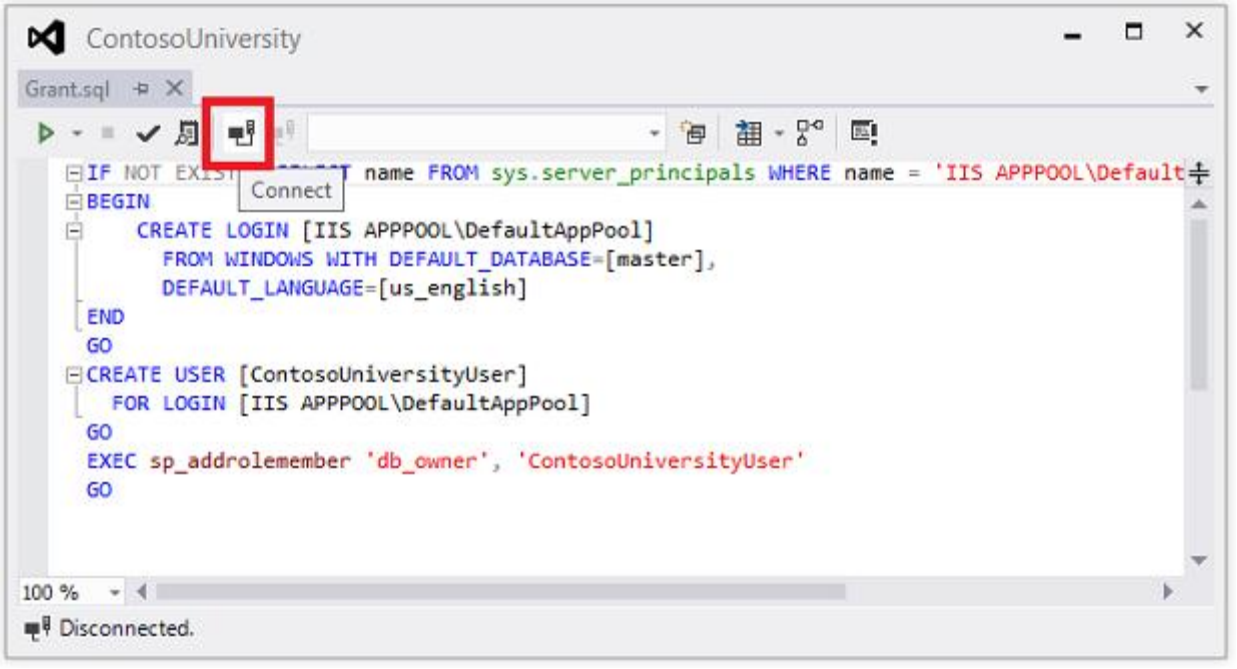
Security Note: This script gives db_owner permissions to the user that accesses the database at run time, which is what you'll have in the production environment. In some scenarios you might

want to specify a user that has full database schema update permissions only for deployment, and specify for run time a different user that has permissions only to read and write data. For more information, see Reviewing the Automatic Web.config Changes for Code First Migrations later in this tutorial.

Run the grant script in the application database

You can configure the publish profile to run the grant script in the membership database during deployment because that database deployment uses the dbDacFx provider. You can't run scripts during Code First Migrations deployment, which is how you're deploying the application database. Therefore, you have to manually run the script before deployment in the application database.

1. In Visual Studio, open the *Grant.sql* file that you created earlier.
2. Click **Connect**.



3. In the **Connect to Server** dialog box, enter .*SQLExpress* as the **Server Name**, and then click **Connect**.
4. In the database drop-down list select **ContosoUniversity**, and then click **Execute**.

The default application pool identity now has sufficient permissions in the application database for Code First Migrations to create the database tables when the application runs.

Publish to IIS

There are several ways you can deploy to IIS using Visual Studio and Web Deploy:

- Use Visual Studio one-click publish.
- Publish from the command line.
- Create a *deployment package* and install it using the IIS Manager UI. The deployment package consists of a *.zip* file that contains all the files and metadata needed to install a site in IIS.
- Create a deployment package and install it using the command line.

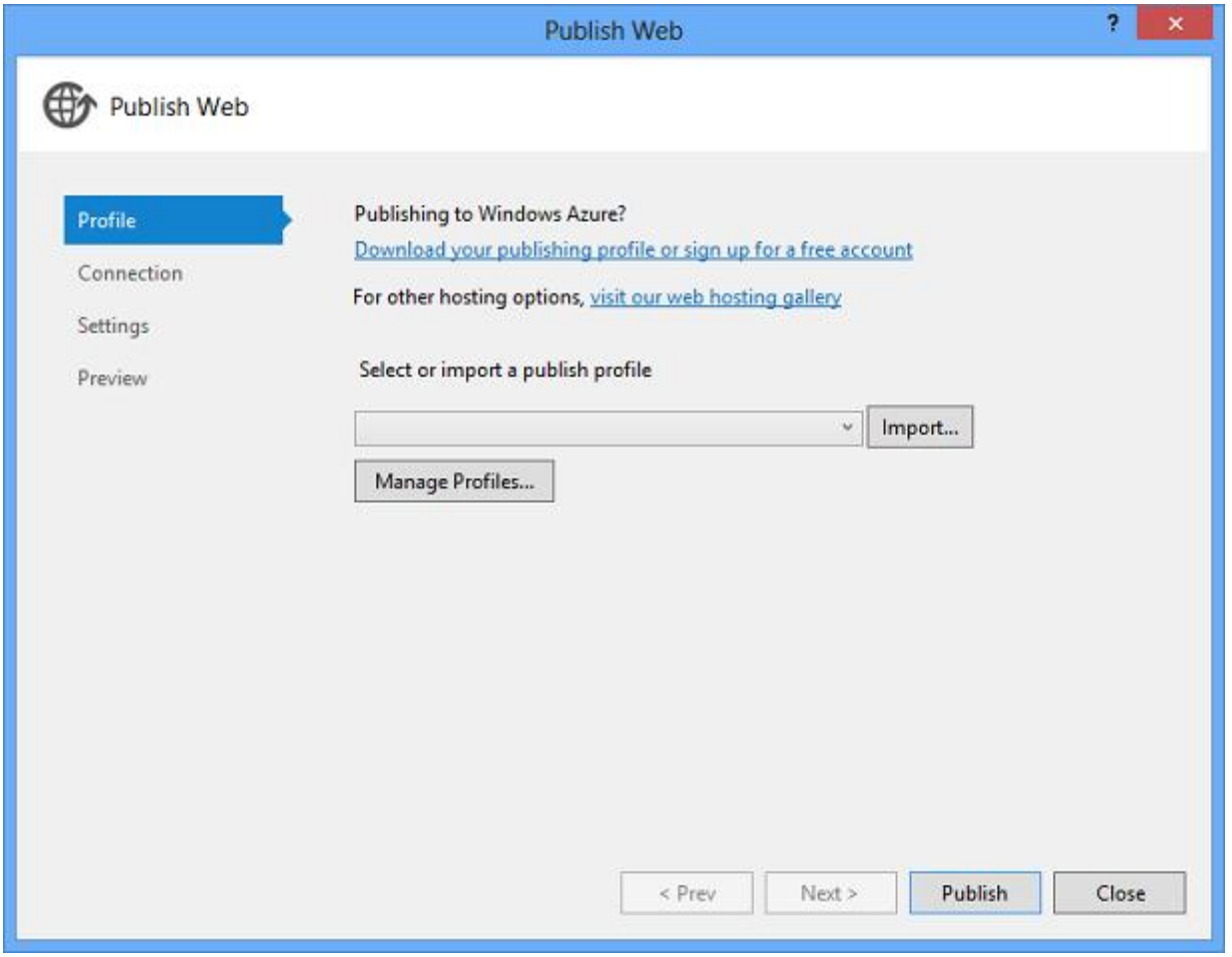
The process you went through in the previous tutorials to set up Visual Studio to automate deployment tasks applies to all of these methods. In these tutorials you'll use the first two of these methods. For information about using deployment packages, see Deploying a web application by creating and installing a web deployment package in the Web Deployment Content Map for Visual Studio and ASP.NET.

Before publishing, make sure that you are running Visual Studio in administrator mode. If you don't see **(Administrator)** in the title bar, close Visual Studio. In the Windows 8 **Start** page or the Windows 7 **Start** menu, right-click the icon for the version of Visual Studio you're using and select **Run as Administrator**. Administrator mode is required for publishing only when you are publishing to IIS on the local computer.

Create the publish profile

1. In **Solution Explorer**, right-click the ContosoUniversity project (not the ContosoUniversity.DAL project) and select **Publish**.

   The **Publish Web** wizard appears.



2. In the drop-down list, select **<New...>**.
3. In the **New Profile** dialog box, enter "Test", and then click **OK**.

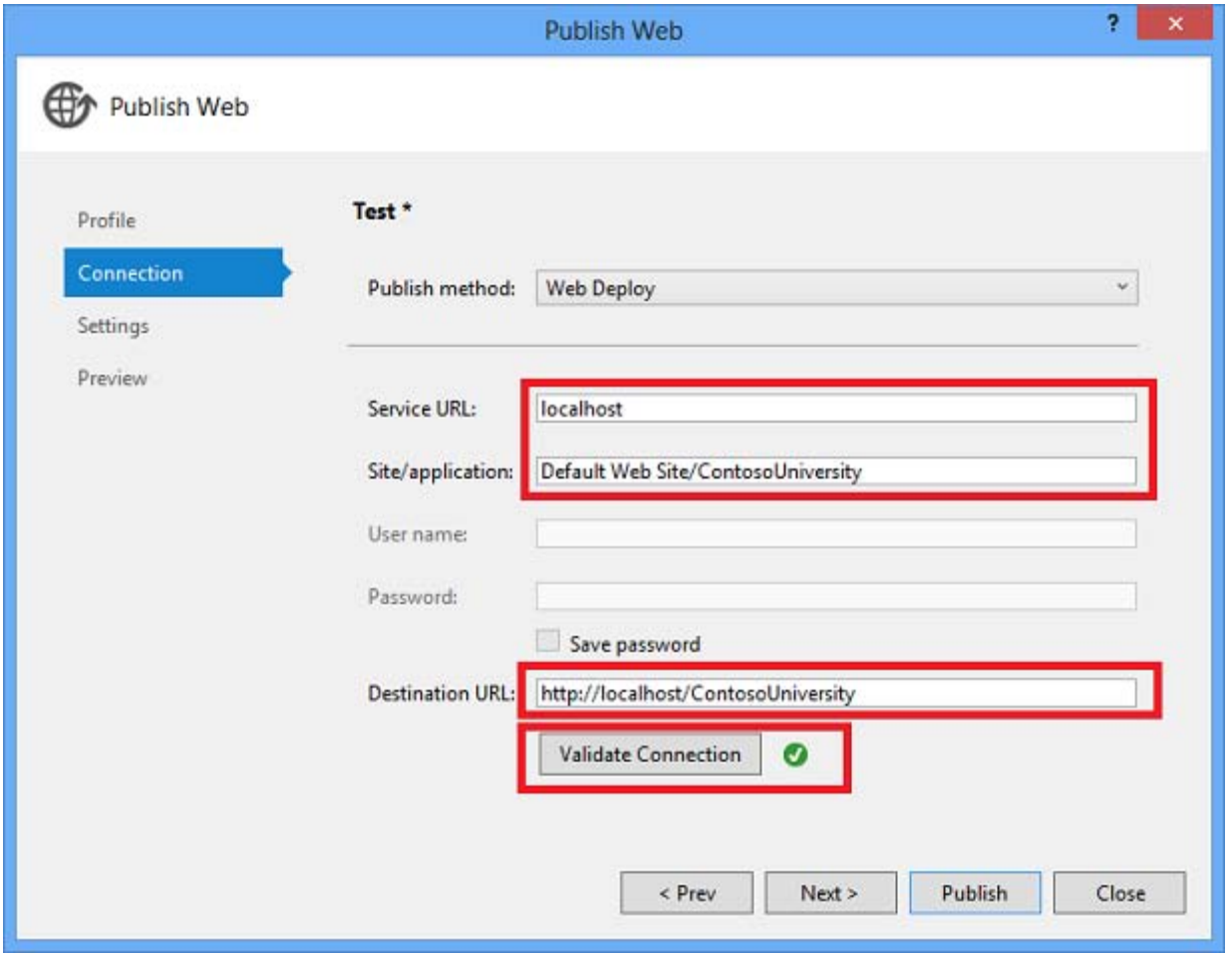
   The wizard automatically advances to the **Connection** tab.

4. In the **Service URL** box, enter *localhost*.
5. In the **Site/application** box, enter *Default Web Site/ContosoUniversity*
6. In the **Destination URL** box, enter *http://localhost/ContosoUniversity*

   The **Destination URL** setting isn't required. When Visual Studio finishes deploying the application, it automatically opens your default browser to this URL. If you don't want the browser to open automatically after deployment, leave this box blank.

7. Click **Validate Connection** to verify that the settings are correct and you can connect to IIS on the local computer.

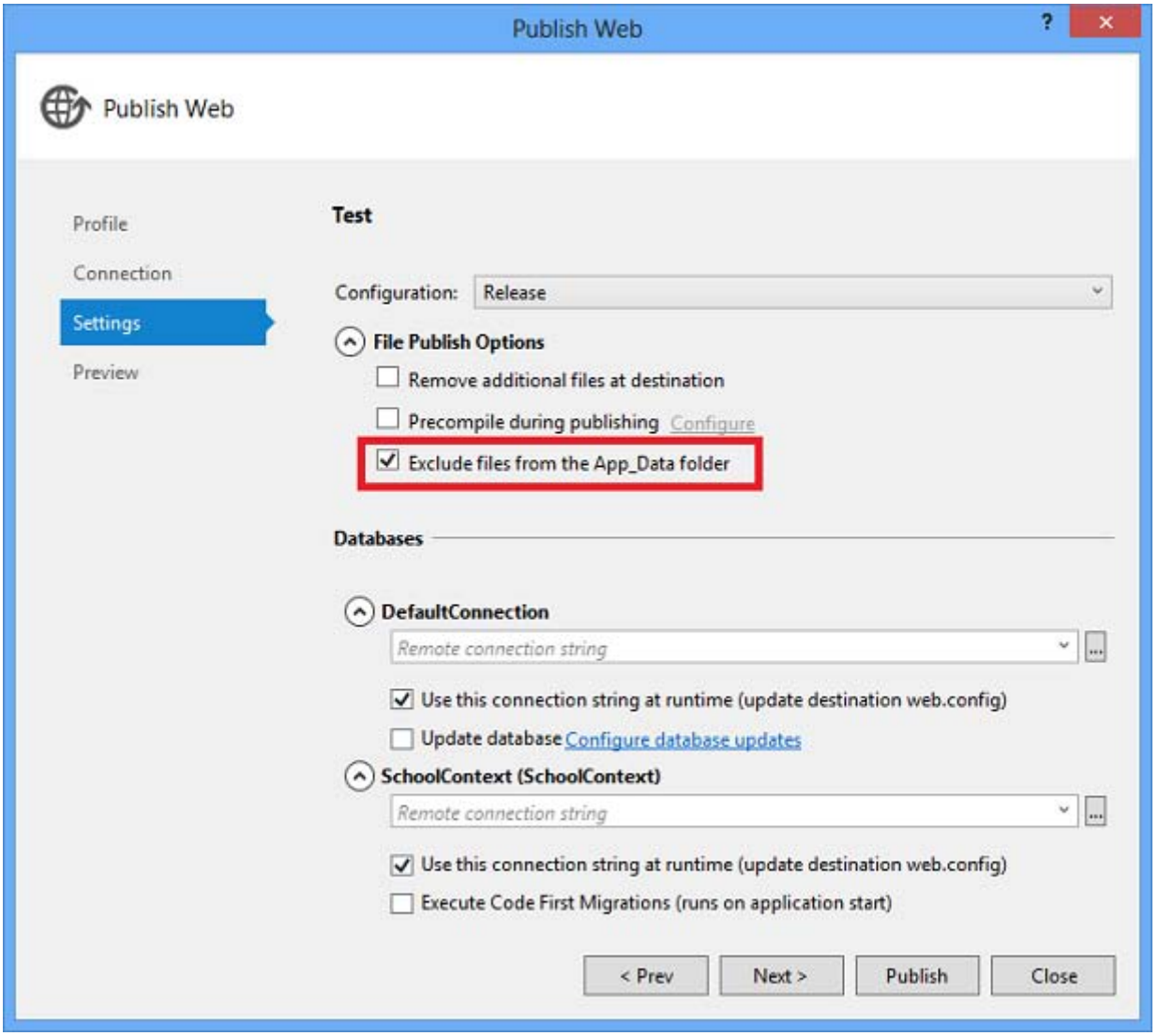
   A green check mark verifies that the connection is successful.



8. Click **Next** to advance to the **Settings** tab.
9. The **Configuration** drop-down box specifies the build configuration to deploy. Leave it set to the default value of Release. You won't be deploying Debug builds in this tutorial.
10. Expand **File Publish Options**, and then select **Exclude files from the App_Data folder**.

    In the test environment the application will access the databases that you created in the local SQL Server Express instance, not the .mdf files in the *App_Data* folder.

11. Leave the **Precompile during publishing** and **Remove additional files at destination** check boxes cleared.



Precompiling is an option that is useful mainly for very large sites; it can reduce page startup time for the first time a page is requested after the site is published.

You don't need to remove additional files since this is your first deployment and there won't be any files in the destination folder yet.

Caution: If you select **Remove additional files** for a subsequent deployment to the same site, make sure that you use the preview feature so that you see in advance which files will be deleted before you deploy. The expected behavior is that Web Deploy will delete files on the destination server that you have deleted in your project. However, the entire folder structure under the source and destination folders is compared, and in some scenarios Web Deploy might delete files you don't want to delete.

For example, if you have a web application in a subfolder on the server when you deploy a project to the root folder, the subfolder will be deleted. You might have one project for the main site at contoso.com and another project for a blog at contoso.com/blog. The blog

application is in a subfolder. If you select Remove additional files at destination when you deploy the main site, the blog application will be deleted.

For another example, your App_Data folder might get deleted unexpectedly. Certain databases such as SQL Server Compact store database files in the App_Data folder. After the initial deployment you don't want to keep copying the database files in subsequent deployments so you select Exclude App_Data on the Package/Publish Web tab. After you do that, if you have Remove additional files at destination selected, your database files and the App_Data folder itself will be deleted the next time you publish.

Configure deployment for the membership database

The following steps apply to the **DefaultConnection** database in the **Databases** section of the dialog box.

1. In the **Remote connection string** box, enter the following connection string that points to the new SQL Server Express membership database.

   ```
   Data Source=.\SQLExpress;Initial Catalog=aspnet-
   ContosoUniversity;Integrated Security=True
   ```

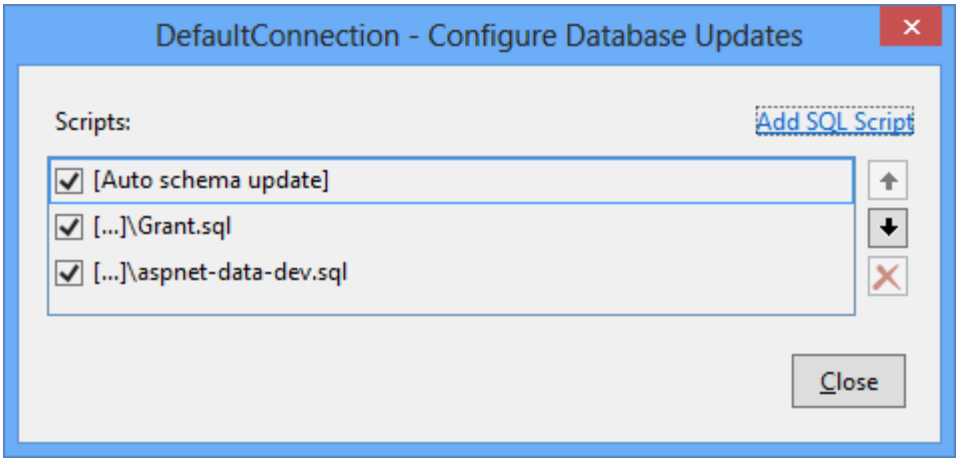
   The deployment process will put this connection string in the deployed Web.config file because **Use this connection string at runtime** is selected.

   You can also get the connection string from **Server Explorer**. In **Server Explorer**, expand **Data Connections** and select the **<machinename>\sqlexpress.aspnet-ContosoUniversity** database, then from the **Properties** window copy the **Connection String** value. That connection string will have one additional setting that you can delete: `Pooling=False`.

2. Select **Update database**.

   This will cause the database schema to be created in the destination database during deployment. In the following steps you specify the additional scripts that you need to run: one to grant database access to the default application pool and one to deploy data.

3. Click **Configure database updates**.
4. In the **Configure Database Updates** dialog box, click **Add SQL Script** and then navigate to the *Grant.sql* script that you saved earlier in the solution folder.
5. Repeat the process to add the *aspnet-data-dev.sql* script.



6. Click **Close**.

Configure deployment for the application database

When Visual Studio detects an Entity Framework `DbContext` class, it creates an entry in the **Databases** section that has an **Execute Code First Migrations** check box instead of an **Update Database** check box. For this tutorial you'll use that check box to specify Code First Migrations deployment.

In some scenarios, you might be using a `DbContext` database but you want to use the dbDacFx provider instead of Migrations to deploy the database. In that case, see How do I deploy a Code First database without Migrations? in the ASP.NET Web Deployment FAQ on MSDN.

The following steps apply to the **SchoolContext** database in the **Databases** section of the dialog box.

1. In the **Remote connection string** box, enter the following connection string that points to the new SQL Server Express application database.

   ```
   Data Source=.\SQLExpress;Initial Catalog=ContosoUniversity;Integrated Security=True
   ```

   The deployment process will put this connection string in the deployed Web.config file because **Use this connection string at runtime** is selected.

   You can also get the application database connection string from **Server Explorer** the same way you got the membership database connection string.

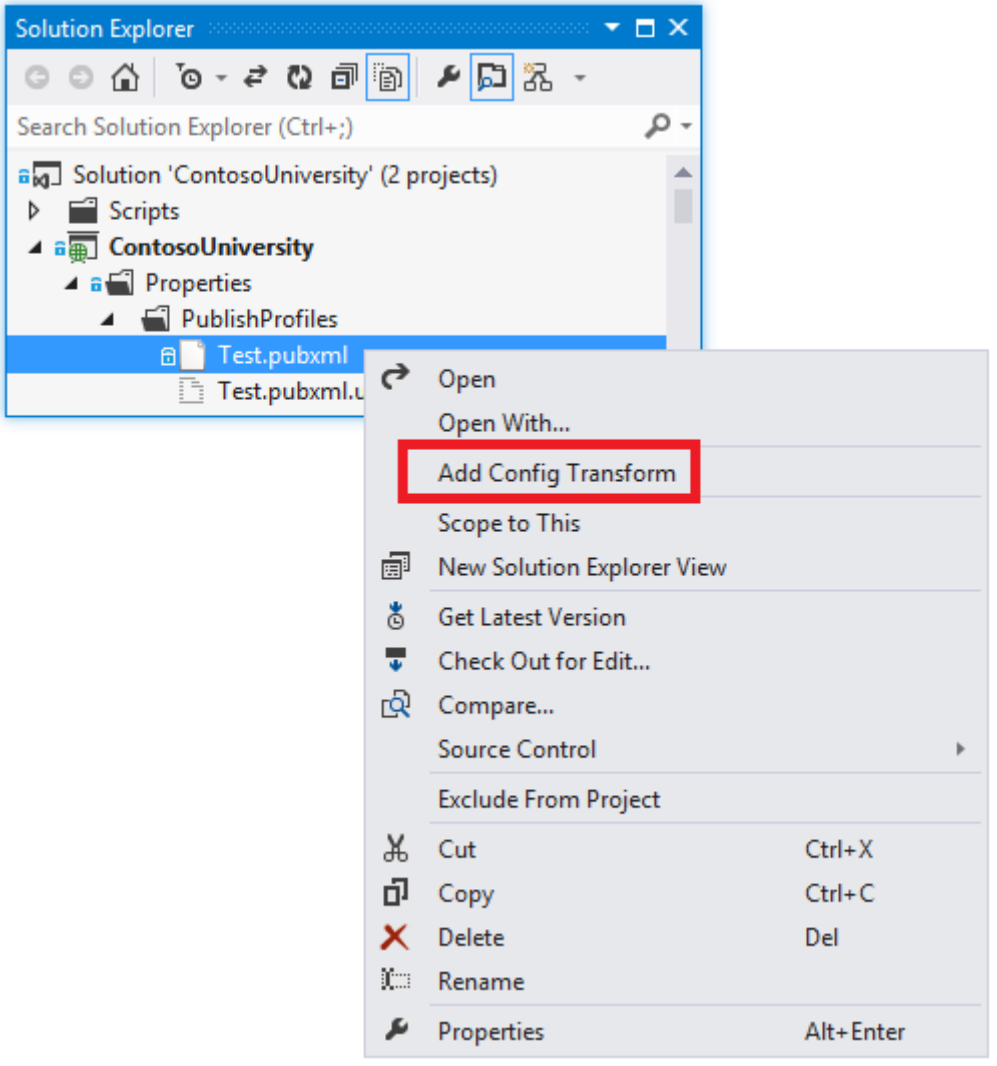
2. Select **Execute Code First Migrations (runs on application start)**.

   This option causes the deployment process to configure the deployed Web.config file to specify the `MigrateDatabaseToLatestVersion` initializer. This initializer automatically

updates the database to the latest version when the application accesses the database for the first time after deployment.

Configure publish profile transforms

1. Click **Close**, and then click **Yes** when you are asked if you want to save changes.
2. In **Solution Explorer**, expand **Properties**, expand **PublishProfiles**.
3. Rright-click *Test.pubxml,* and then click **Add Config Transform**.



Visual Studio creates the *Web.Test.config* transform file and opens it.

4. In the *Web.Test.config* transform file, insert the following code immediately after the opening configuration tag.

```
<appSettings>
    <add key="Environment" value="Test" xdt:Transform="SetAttributes"
```

```
xdt:Locator="Match(key)"/>
</appSettings>
```

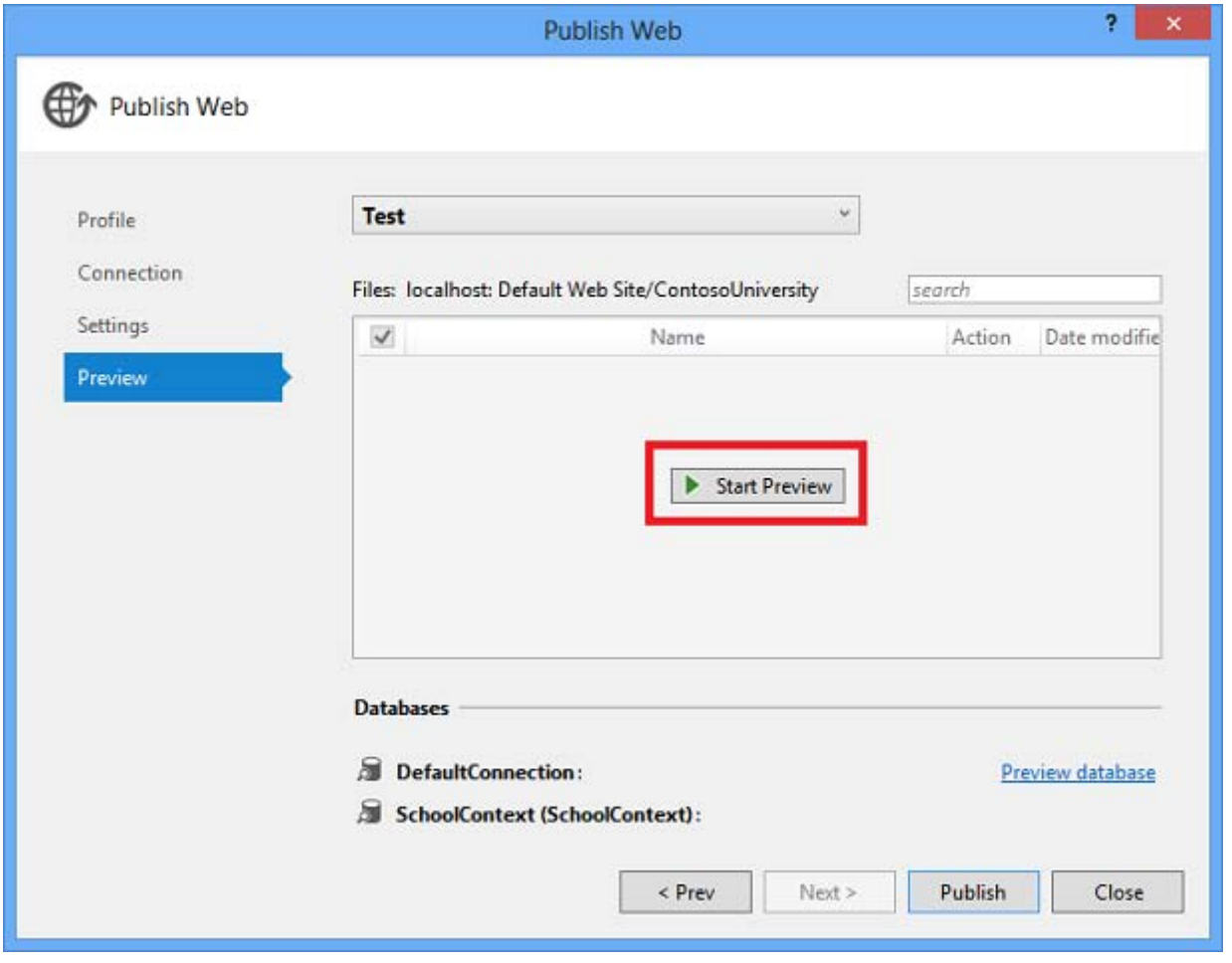
When you use the Test publish profile, this transform sets the environment indicator to "Test". In the deployed site you'll see "(Test)" after the "Contoso University" H1 heading.

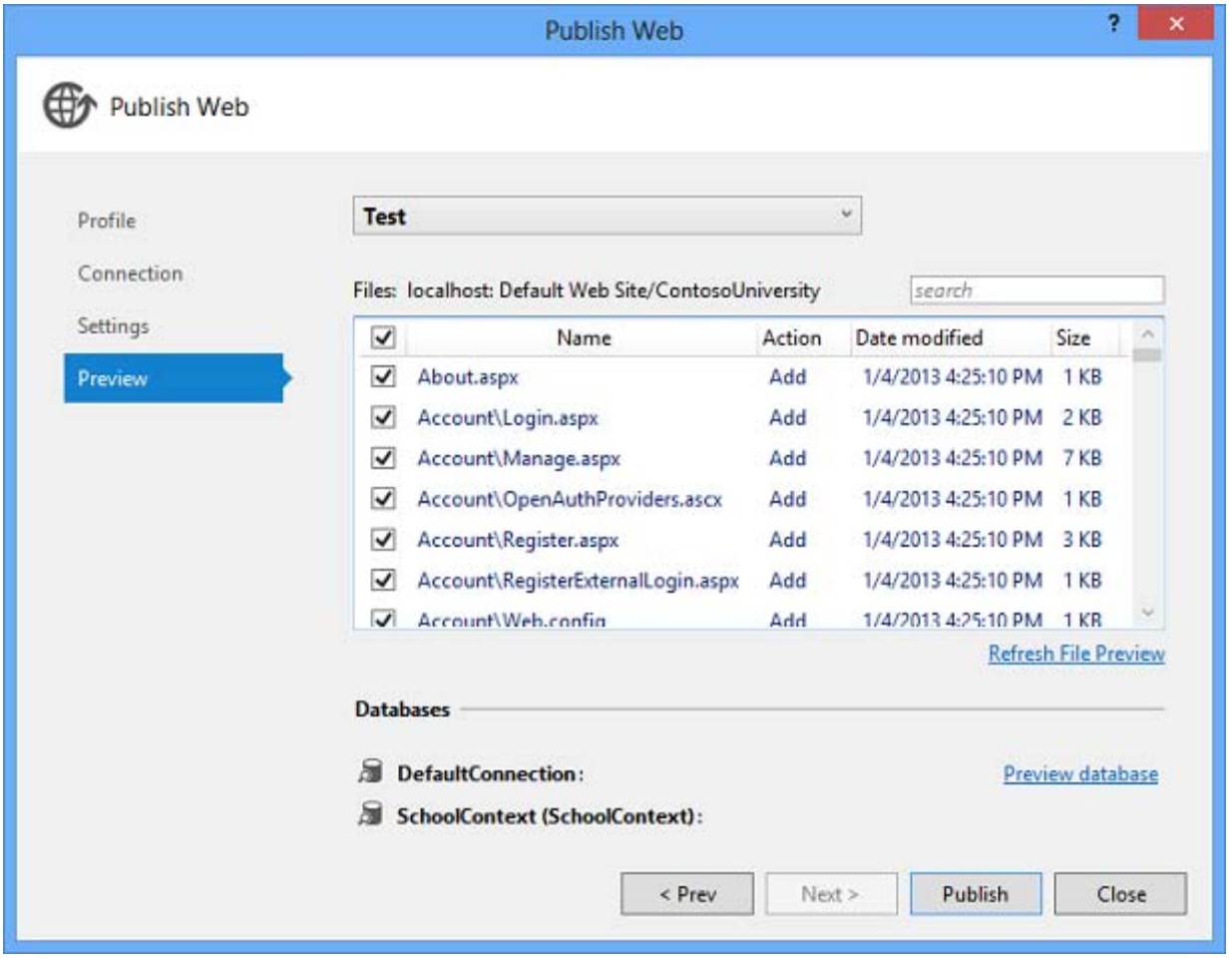
5. Save and close the file.
6. Right-click the *Web.Test.config* file and click **Preview Transform** to make sure that the transform you coded produces the expected changes.

   The **Web.config Preview** window shows the result of applying both the *Web.Release.config* transforms and the *Web.Test.config* transforms.

Preview the deployment updates

1. Open the **Publish Web** wizard again (right-click the ContosoUniversity project and click **Publish**).
2. In the **Preview** tab, make sure that the **Test** profile is still selected, and then click **Start Preview** to see a list of the files that will be copied.



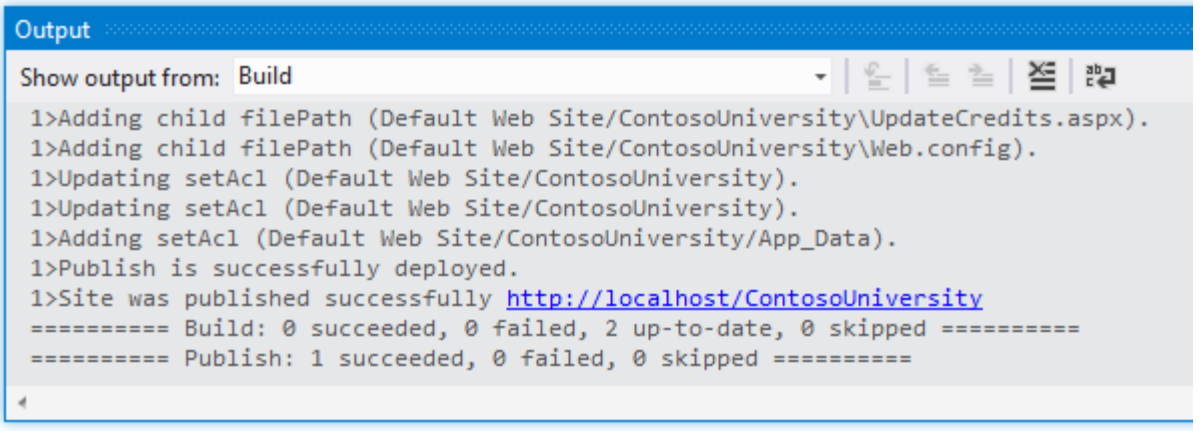


You can also click the **Preview database** link to see the scripts that will run in the membership database. (No scripts are run for Code First Migrations deployment, so there is nothing to preview for the application database.)

3. Click **Publish**.

   If Visual Studio is not in administrator mode, you might get an error message that indicates a permissions error. In that case, close Visual Studio, open it in administrator mode, and try to publish again.

   If Visual Studio is in administrator mode, the **Output** window reports successful build and publish.



If you entered the URL in the **Destination URL** box on the publish profile **Connection** tab, the browser automatically opens to the Contoso University Home page running in IIS on the local computer.

Test in the test environment

Notice that the environment indicator shows "(Test)" instead of "(Dev)", which shows that the *Web.config* transformation for the environment indicator was successful.

Run the **Instructors** page to verify that Code First seeded the database with instructor data. When you select this page, it may take a few minutes to load because Code First creates the database and then runs the `Seed` method. (It didn't do that when you were on the home page because the application didn't try to access the database yet.)

Click the **Students** tab to verify that the deployed database has no students.

Select **Add Students** from the **Students** menu, add a student, and then view the new student in the **Students** page to verify that you can successfully write to the database.

From the **Courses** menu, select **Update Credits**. The **Update Credits** page requires administrator permissions, so the **Log In** page is displayed. Enter the administrator account credentials that you created earlier ("admin" and "devpwd"). The **Update Credits** page is displayed, which verifies that the administrator account that you created in the previous tutorial was correctly deployed to the test environment.

Verify that an *Elmah* folder exists in the *c:\inetpub\wwwroot\ContosoUniversity* folder with only the placeholder file in it.

Review the automatic Web.config changes for Code First Migrations

Open the *Web.config* file in the deployed application at *C:\inetpub\wwwroot\ContosoUniversity* and you can see where the deployment process configured Code First Migrations to automatically update the database to the latest version.

```
<entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.SqlCeConnectionFactor
        <parameters>
            <parameter value="System.Data.SqlServerCe.4.0" />
        </parameters>
    </defaultConnectionFactory>
    <contexts>
        <context type="ContosoUniversity.DAL.SchoolContext, ContosoUniversity.DAL">
            <databaseInitializer type="System.Data.Entity.MigrateDatabaseToLatestVersion`2[
                <parameters>
                    <parameter value="SchoolContext_DatabasePublish" />
                </parameters>
            </databaseInitializer>
        </context>
    </contexts>
</entityFramework>
```

The deployment process also created a new connection string for Code First Migrations to use exclusively for updating the database schema:

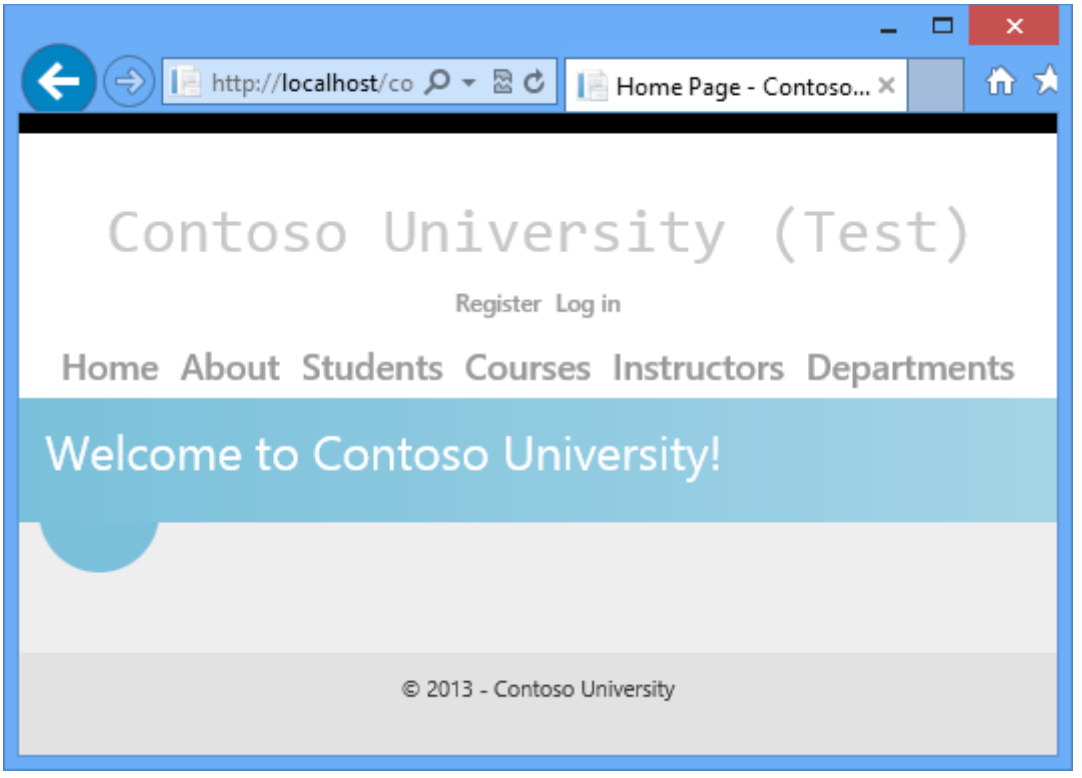
```
<connectionStrings>
  <add name="DefaultConnection" providerName="System.Data.SqlClient" connectionString="Data
  <add name="SchoolContext" providerName="System.Data.SqlClient" connectionString="Data Sou
  <add name="SchoolContext_DatabasePublish" connectionString="Data Source=.\SQLExpress;Init
</connectionStrings>
```

This additional connection string enables you to specify one user account for database schema updates, and a different user account for application data access. For example, you could assign the **db_owner** role to Code First Migrations, and **db_datareader** and **db_datawriter** roles to the application. This is a common defense-in-depth pattern that prevents potentially malicious code in the application from changing the database schema. (For example, this might happen in a successful SQL injection attack.) This pattern is not used by these tutorials. If you want to implement this pattern in your scenario, you can do it by performing the following steps:

1. In the **Settings** tab of the **Publish Web** wizard, enter the connection string that specifies a user with full database schema update permissions, and clear the **Use this connection string at runtime** check box. In the deployed Web.config file, this becomes the `DatabasePublish` connection string.
2. Create a Web.config file transformation for the connection string that you want the application to use at run time.

Summary

You have now deployed your application to IIS on your development computer and tested it there.



This verifies that the deployment process copied the application's content to the right location (excluding the files that you did not want to deploy), and also that Web Deploy configured IIS correctly during deployment. In the next tutorial, you'll run one more test that finds a deployment task that has not yet been done: setting folder permissions on the *Elmah* folder.

More information

For information about running IIS or IIS Express in Visual Studio, see the following resources:

- IIS Express Overview on the IIS.net site.
- Introducing IIS Express on Scott Guthrie's blog.
- Web Servers in Visual Studio for ASP.NET Web Projects.
- Core Differences Between IIS and the ASP.NET Development Server on the ASP.NET site.

For information about what issues might arise when your application runs in medium trust, see Hosting ASP.NET Applications in Medium Trust on the 4 Guys from Rolla site.

Setting Folder Permissions

Overview

In this tutorial, you set folder permissions for the *Elmah* folder in the deployed web site so that the application can create log files in that folder.

When you test a web application in Visual Studio using the Visual Studio Development Server (Cassini) or IIS Express, the application runs under your identity. You are most likely an administrator on your development computer and have full authority to do anything to any file in any folder. But when an application runs under IIS, it runs under the identity defined for the application pool that the site is assigned to. This is typically a system-defined account that has limited permissions. By default it has read and execute permissions on your web application's files and folders, but it doesn't have write access.

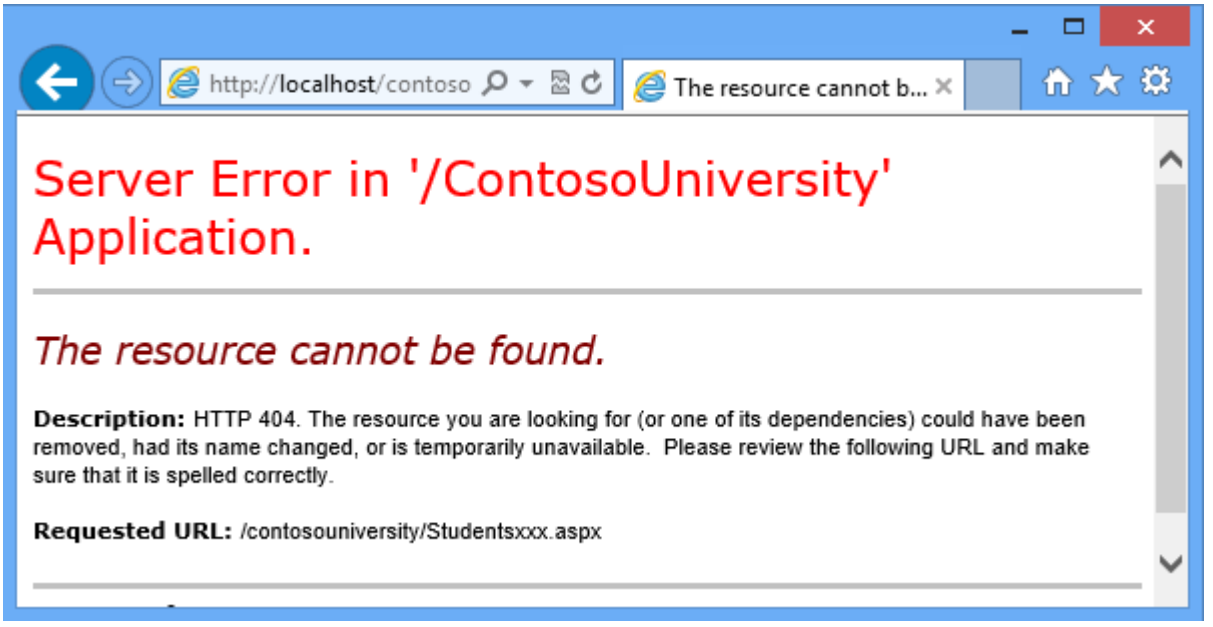
This becomes an issue if your application creates or updates files, which is a common need in web applications. In the Contoso University application, Elmah creates XML files in the *Elmah* folder in order to save details about errors. Even if you don't use something like Elmah, your site might let users upload files or perform other tasks that write data to a folder in your site.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

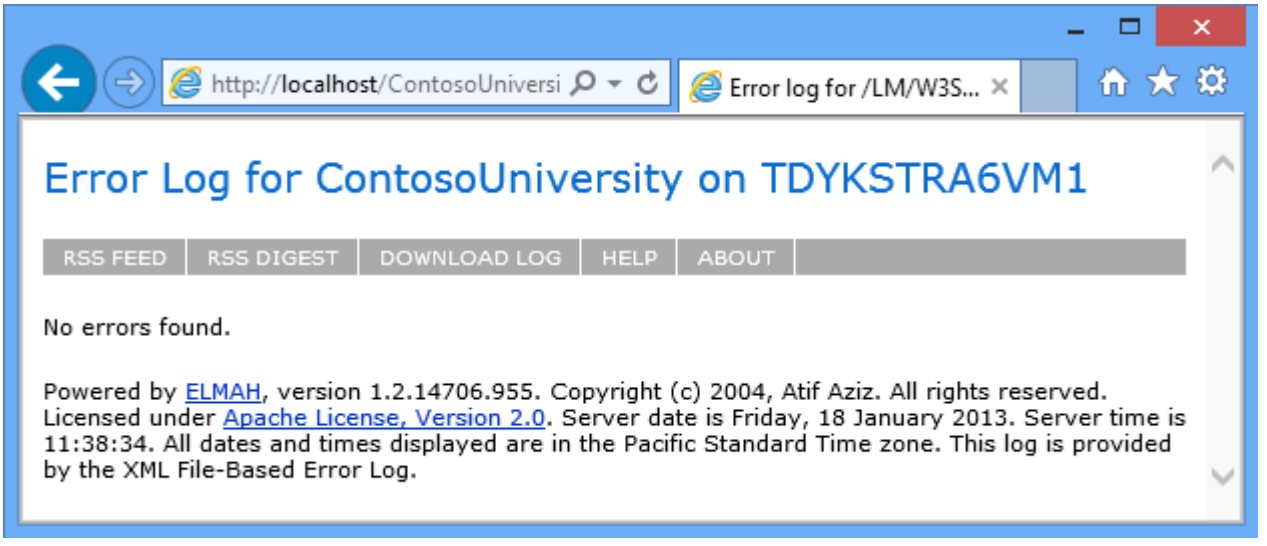
Test error logging and reporting

To see how the application doesn't work correctly in IIS (although it did when you tested it in Visual Studio), you can cause an error that would normally be logged by Elmah, and then open the Elmah error log to see the details. If Elmah was unable to create an XML file and store the error details, you see an empty error report.

Open a browser and go to *http://localhost/ContosoUniversity*, and then request an invalid URL like *Studentsxxx.aspx*. You see a system-generated error page instead of the *GenericErrorPage.aspx* page because the `customErrors` setting in the Web.config file is "RemoteOnly" and you are running IIS locally:



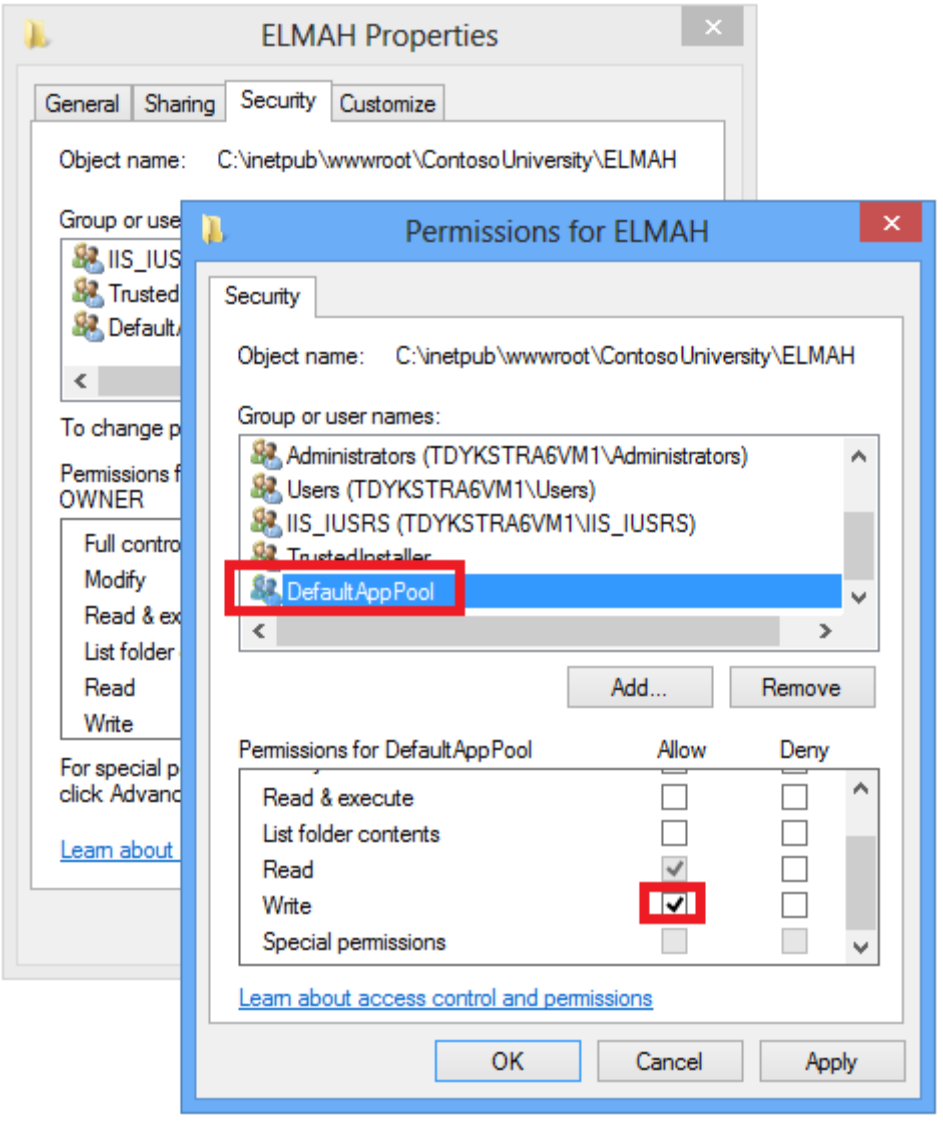
Now run *Elmah.axd* to see the error report. After you log in with the administrator account credentials ("admin" and "devpwd"), you see an empty error log page because Elmah was unable to create an XML file in the *Elmah* folder:



Set write permission on the Elmah folder

You can set folder permissions manually or you can make it an automatic part of the deployment process. Making it automatic requires complex MSBuild code, and since you only have to do this the first time you deploy, the following steps how to do it manually. (For information about how to make this part of the deployment process, see Setting Folder Permissions on Web Publish on Sayed Hashimi's blog.)

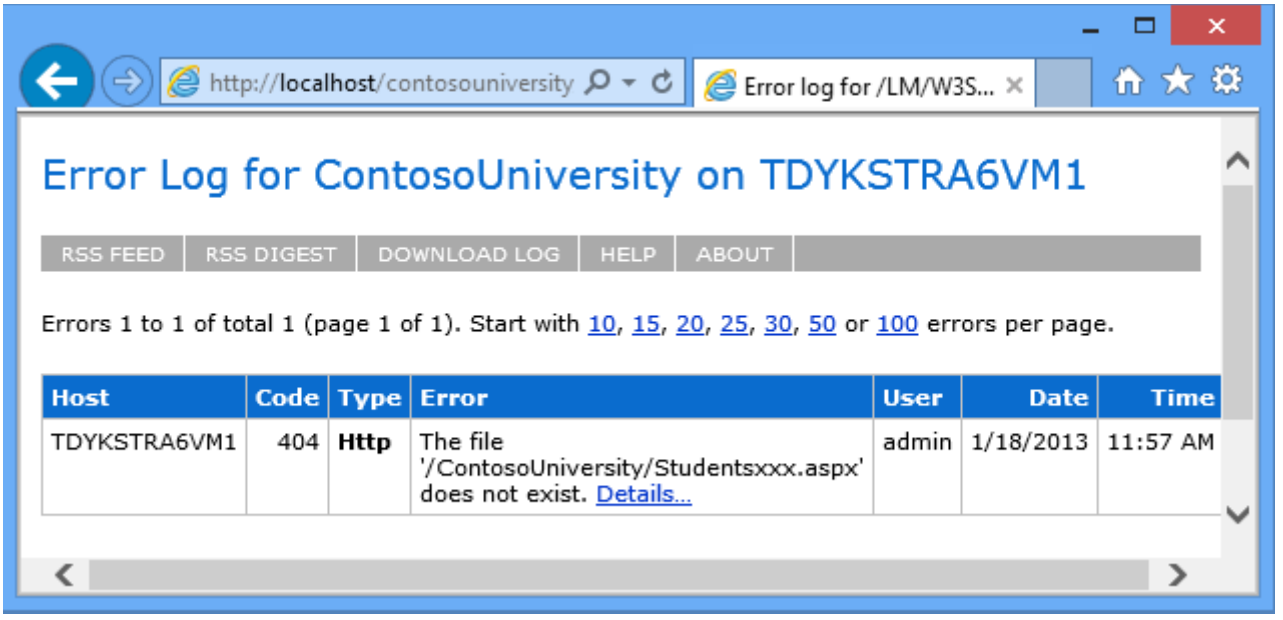
1. In **File Explorer**, navigate to *C:\inetpub\wwwroot\ContosoUniversity*. Right-click the *Elmah* folder, select **Properties**, and then select the **Security** tab.
2. Click **Edit**.
3. In the **Permissions for Elmah** dialog box, select **DefaultAppPool**, and then select the **Write** check box in the **Allow** column.



(If you don't see **DefaultAppPool** in the **Group or user names** list, you probably used some other method than the one specified in this tutorial to set up IIS and ASP.NET 4 on your computer. In that case, find out what identity is used by the application pool assigned to the Contoso University application, and grant write permission to that identity. See the links about application pool identities at the end of this tutorial.)Click **OK** in both dialog boxes.

Retest error logging and reporting

Test by causing an error again in the same way (request a bad URL) and run the **Error Log** page. This time the error appears on the page.



Summary

You have now completed all of the tasks necessary to get Contoso University working correctly in IIS on your local computer. In the next tutorial, you will make the site publicly available by deploying it to Windows Azure.

More information

In this example, the reason why Elmah was unable to save log files was fairly obvious. You can use IIS tracing in cases where the cause of the problem is not so obvious; see Troubleshooting Failed Requests Using Tracing in IIS 7 on the IIS.net site.

For more information about how to grant permissions to application pool identities, see Application Pool Identities and Secure Content in IIS Through File System ACLs on the IIS.net site.

Deploying to Production

Overview

In this tutorial, you set up a Windows Azure account, create staging and production environments, and deploy your ASP.NET web application to the staging and production environments by using the Visual Studio one-click publish feature.

If you prefer, you can deploy to a third-party hosting provider. Most of the procedures described in this tutorial are the same for a hosting provider or for Windows Azure, except that each provider has its own user interface for account and web site management. You can find a hosting provider in the gallery of providers on the Microsoft.com/web site.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Get a Windows Azure account

If you don't already have a Windows Azure account, you can create a free trial account in just a couple of minutes. For details, see Windows Azure Free Trial. After you create the account, or if you already have an account, enable the Windows Azure Web Site preview; see Enable Windows Azure preview features.

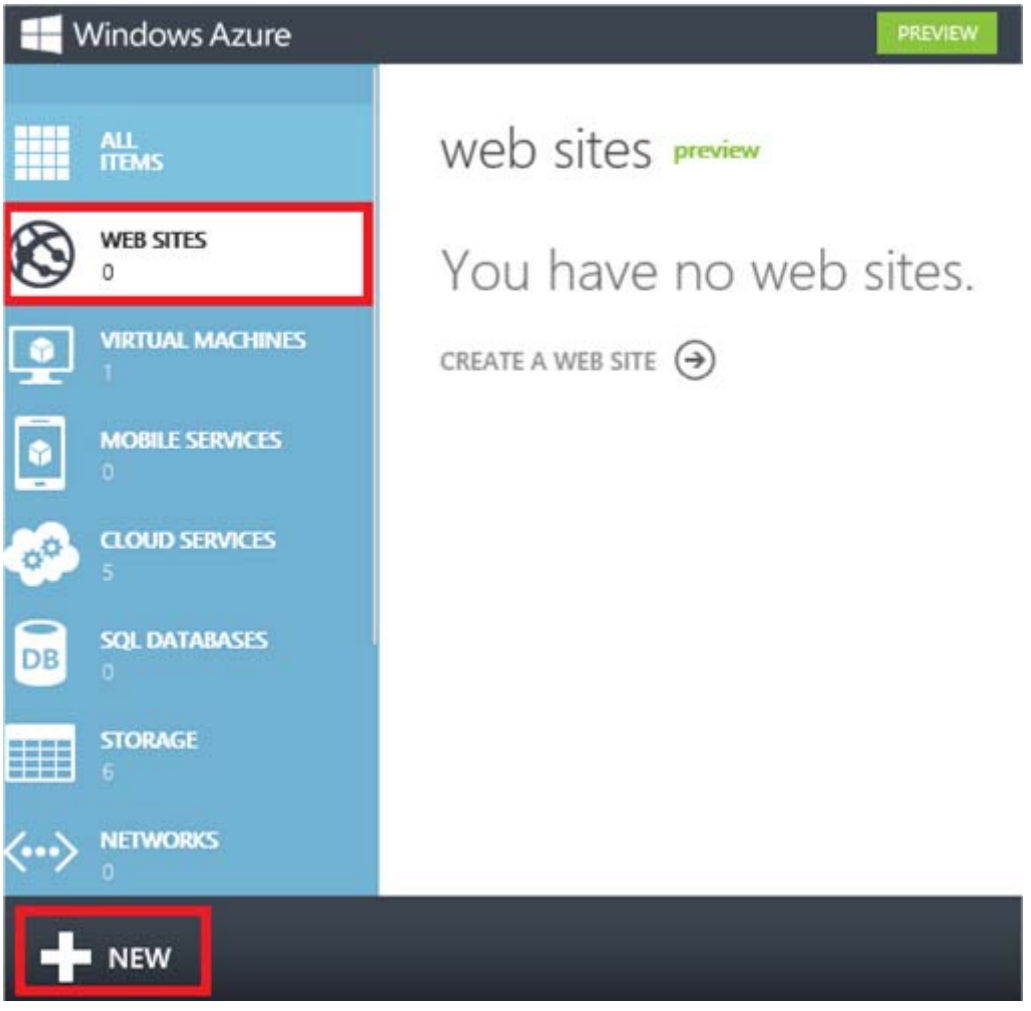
Create a staging environment

As explained in the Deploy to the Test Environment tutorial, the most reliable test environment is a web site at the hosting provider that's just like the production web site. At many hosting providers you would have to weigh the benefits of this against significant additional cost, but in Windows Azure you can create an additional free web site as your staging site. You also need a database, and the additional expense for that over the expense of your production database will be either none or minimal. In Windows Azure you pay for the amount of database storage you use rather than for each database, and the amount of additional storage you'll use in staging will be minimal.

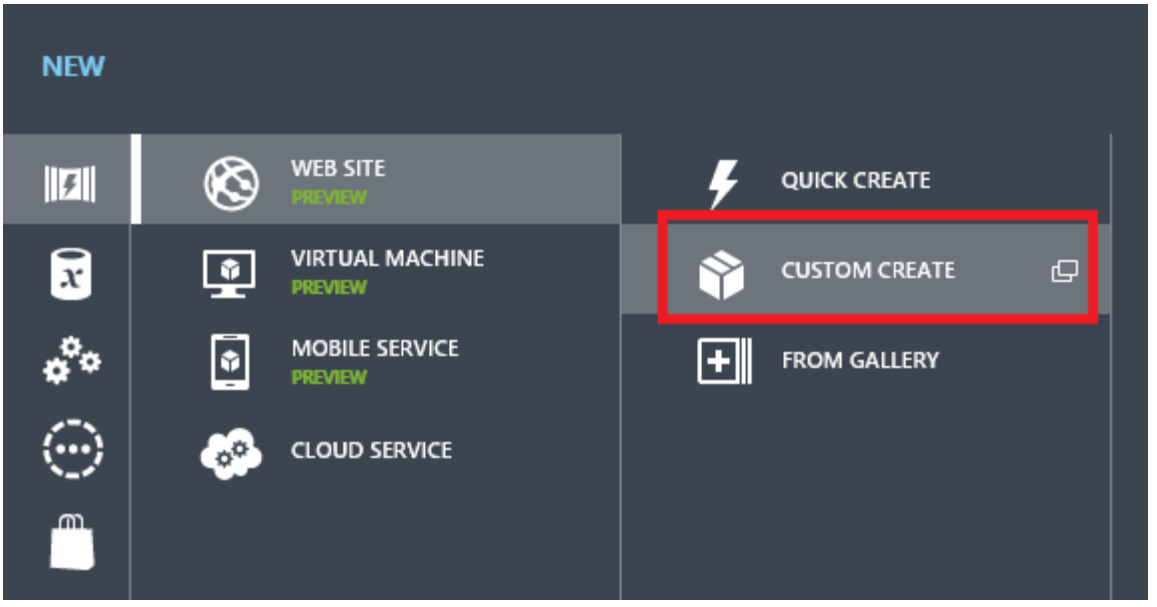
As explained in the Deploy to the Test Environment tutorial, in staging and production you're going to deploy your two databases into one database. If you wanted to keep them separate, the process would be the same except that you'd create an additional database for each environment and you would select the correct destination string for each database when you create the publish profile.

In this section of the tutorial you'll create a web site and database to use for the staging environment, and you'll deploy to staging and test there before creating and deploying to the production environment.

1. In the Windows Azure Management Portal, click **Web Sites**, and then click **New**.



2. Click **Web Site**, and then click **Custom Create**.



The **New Web Site - Custom Create** wizard opens. The **Custom Create** wizard enables you to create a web site and a database at the same time.

3. In the **Create Web Site** step of the wizard, enter a string in the **URL** box to use as the unique URL for your application's staging environment. For example, enter ContosoUniversity-staging123 (including random numbers at the end to make it unique in case ContosoUniversity-staging is taken).

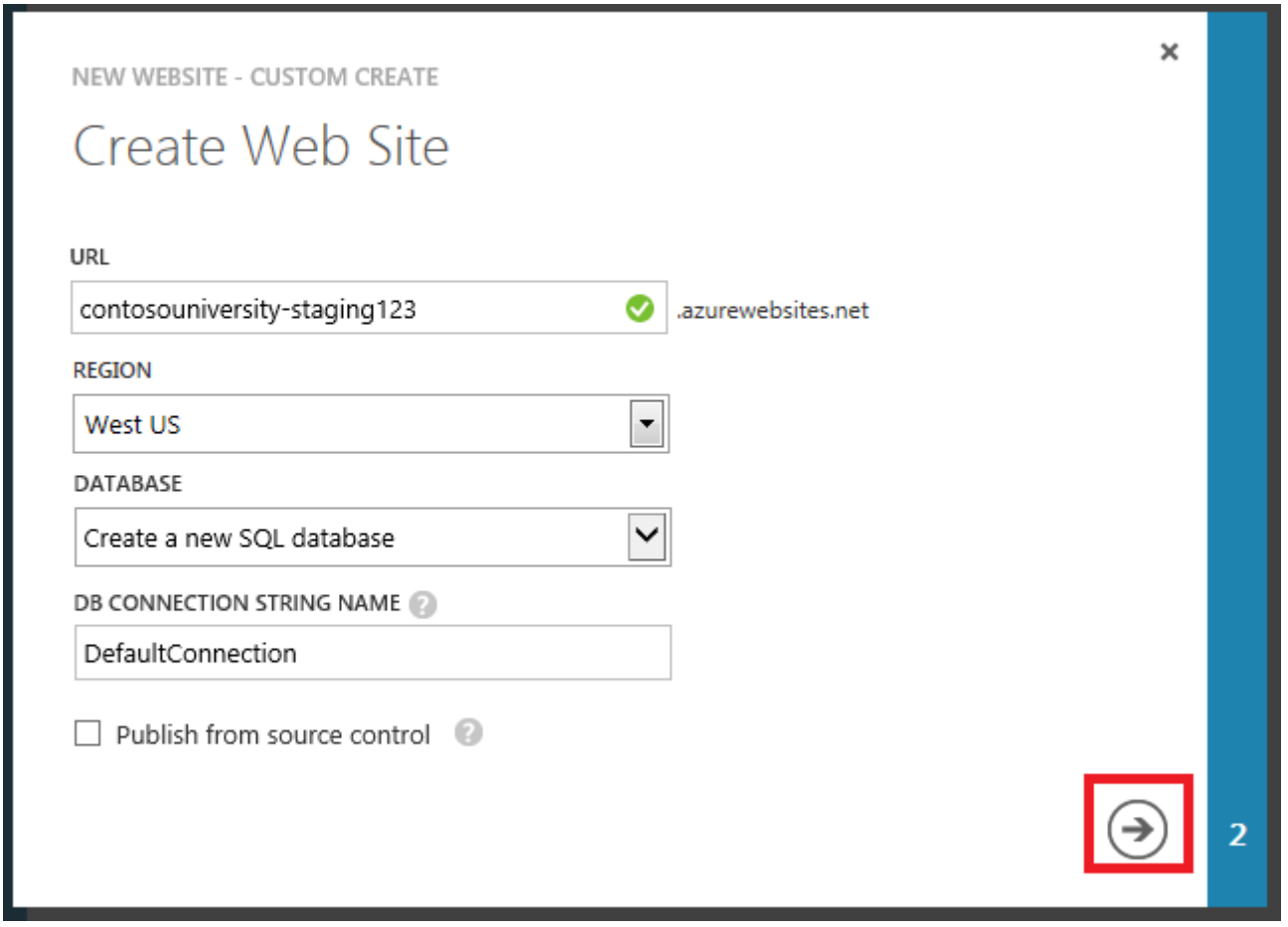
   The complete URL will consist of what you enter here plus the suffix that you see next to the text box.

4. In the **Region** drop-down list, choose the region that is closest to you.

   This setting specifies which data center your web site will run in.

5. In the **Database** drop-down list, choose **Create a new SQL database**.
6. In the **DB Connection String Name** box, leave the default value, *DefaultConnection*.
7. Click the arrow that points to the right at the bottom of the box.

   The following illustration shows the **Create Web Site** dialog with sample values in it. The URL and Region that you have entered will be different.



The wizard advances to the **Specify database settings** step.

8. In the **Name** box, enter *ContosoUniversity* plus a random number to make it unique, for example *ContosoUniversity123*.
9. In the **Server** box, select **New SQL Database Server**.
10. Enter an administrator name and password.

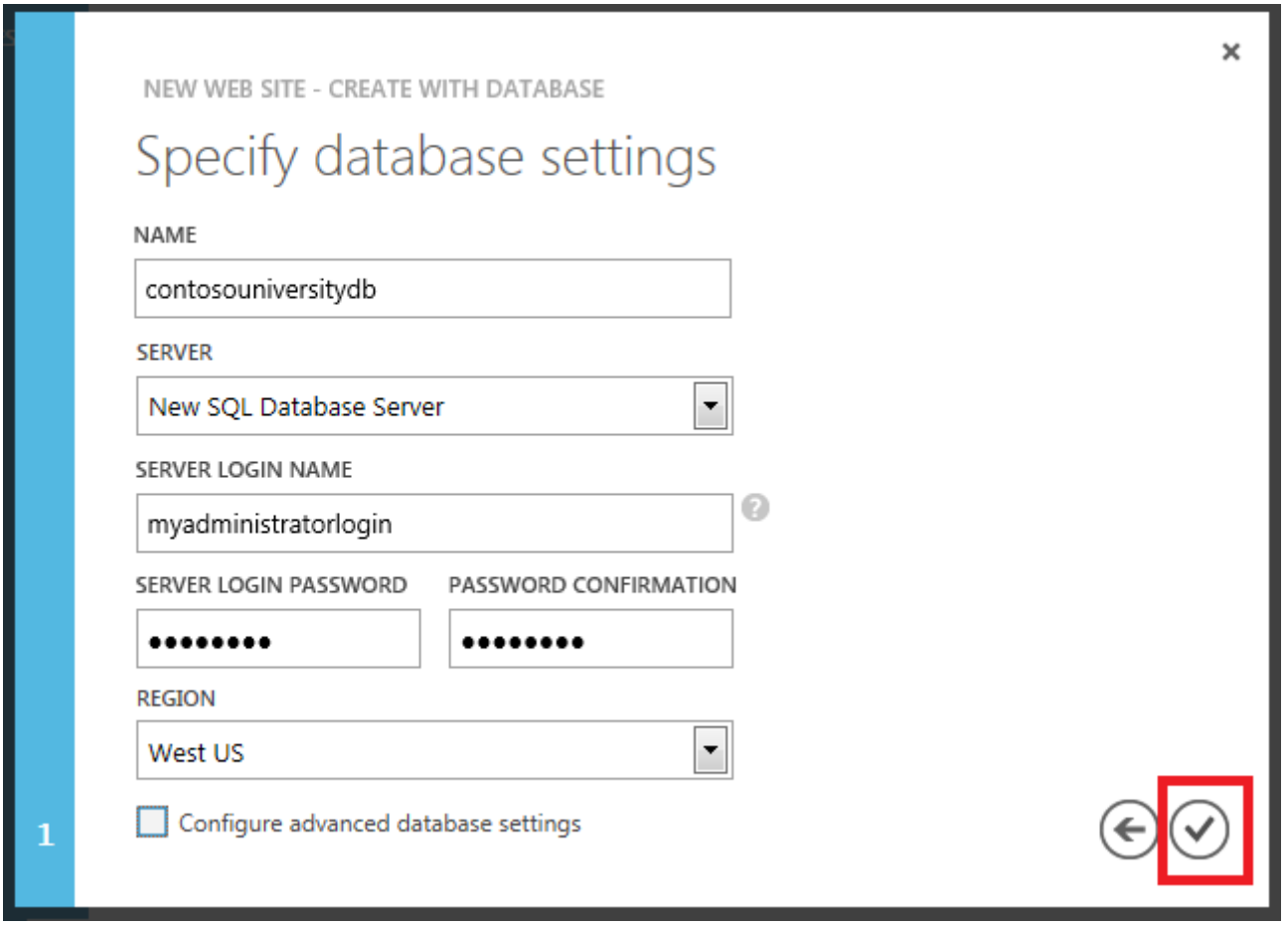
    You aren't entering an existing name and password here. You're entering a new name and password that you're defining now to use later when you access the database.

11. In the **Region** box, choose the same region that you chose for the web site.

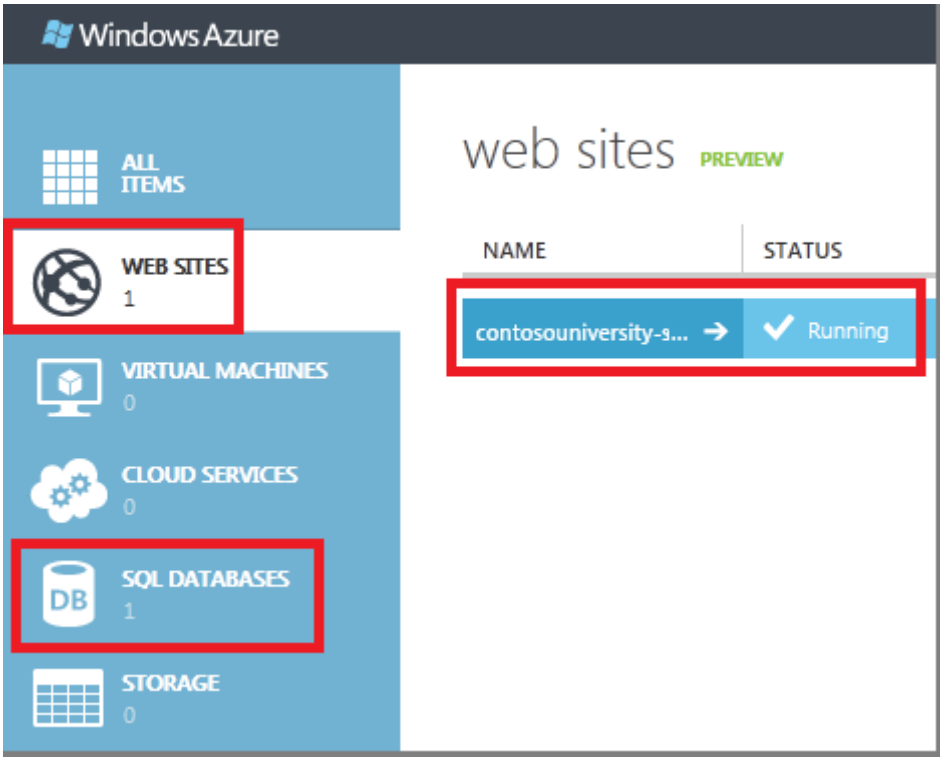
    Keeping the web server and the database server in the same region gives you the best performance and minimizes expenses.

12. Click the check mark at the bottom of the box to indicate that you're finished.

    The following illustration shows the **Specify database settings** dialog with sample values in it. The values you have entered may be different.



The Management Portal returns to the Web Sites page, and the **Status** column shows that the site is being created. After a while (typically less than a minute), the **Status** column shows that the site was successfully created. In the navigation bar at the left, the number of sites you have in your account appears next to the **Web Sites** icon, and the number of databases appears next to the **SQL Databases** icon.



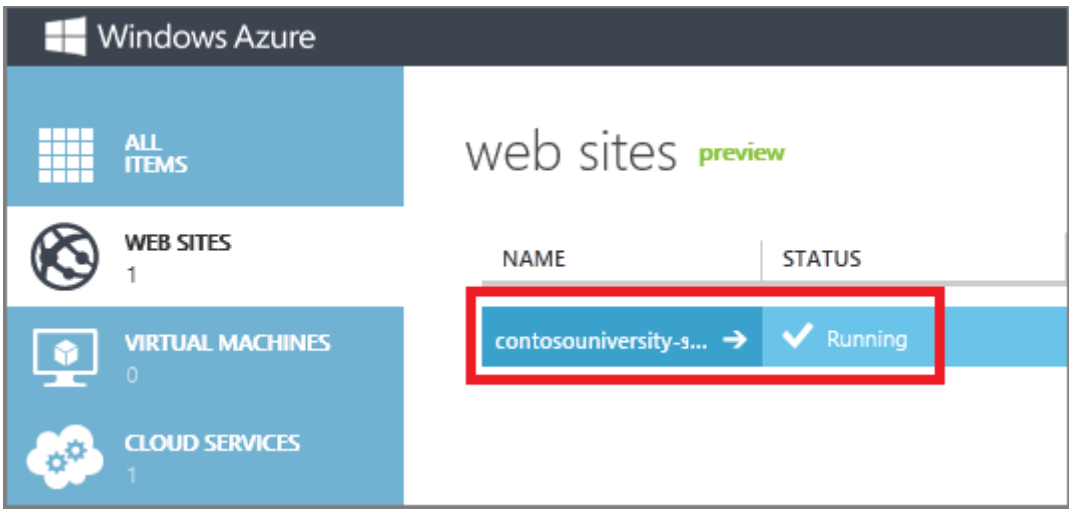
Your web site name will be different from the example web site in the illustration.

Deploy the application to staging

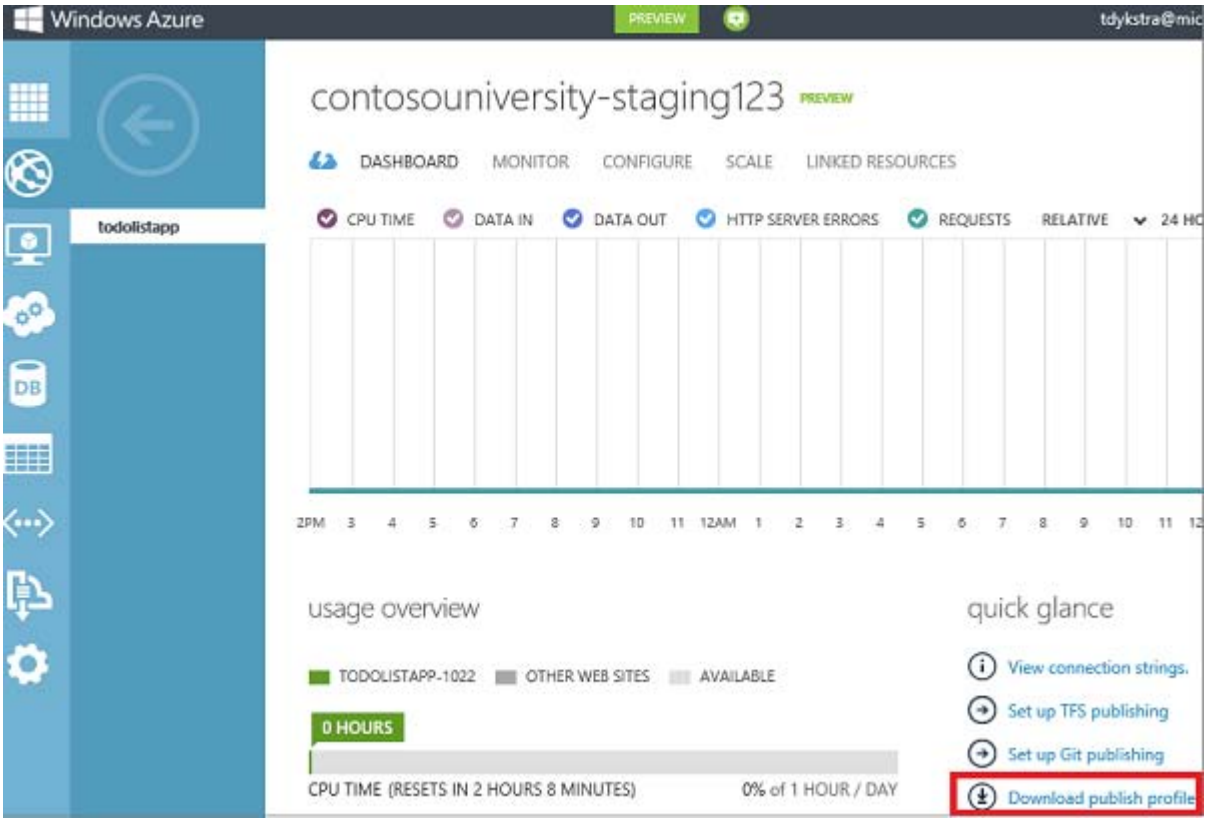
Now that you have created a web site and database for the staging environment, you can deploy the project to it.

Download the .publishsettings file

1. Click the name of the site that you just created.



2. Under **Quick glance** in the **Dashboard** tab, click **Download publish profile**.



This step downloads a file that contains all of the settings that you need in order to deploy an application to your web site. You'll import this file into Visual Studio so you don't have to enter this information manually.

3. Save the *.publishsettings* file in a folder that you can access from Visual Studio.

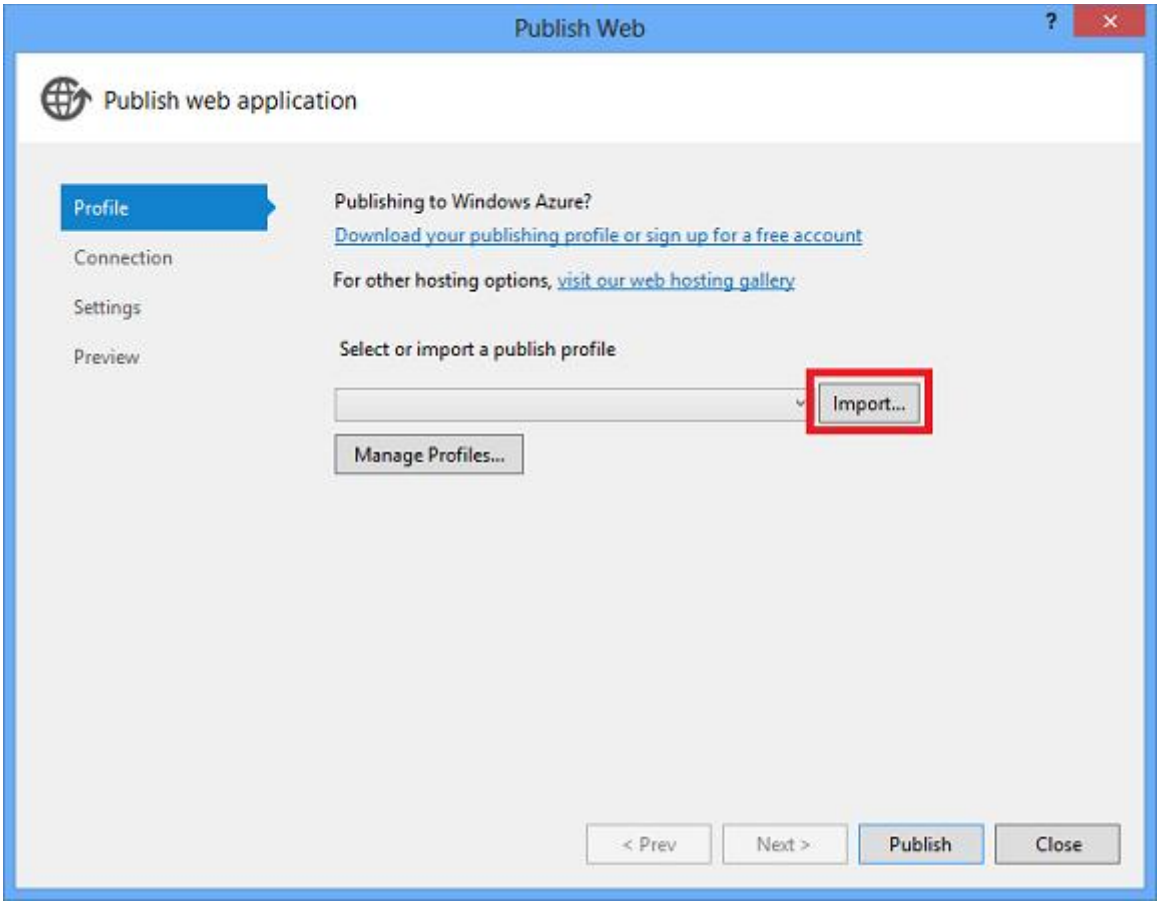
Note: The *.publishsettings* file contains credentials that can be used to publish to your web site. Keep it in a secure location.

Create a publish profile

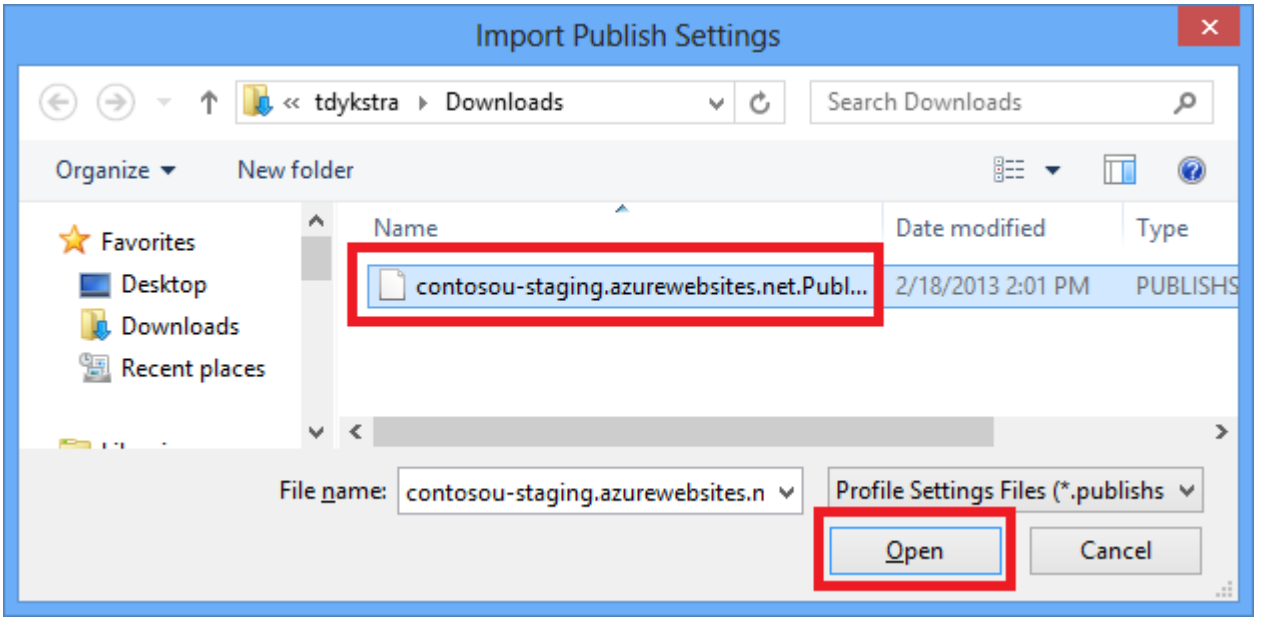
1. In Visual Studio, right-click the ContosoUniversity project in **Solution Explorer** and select **Publish** from the context menu.

   The **Publish Web** wizard opens.

2. Click the **Profile** tab.
3. Click **Import**.



4. Navigate to the *.publishsettings* file you downloaded earlier, and then click **Open**.

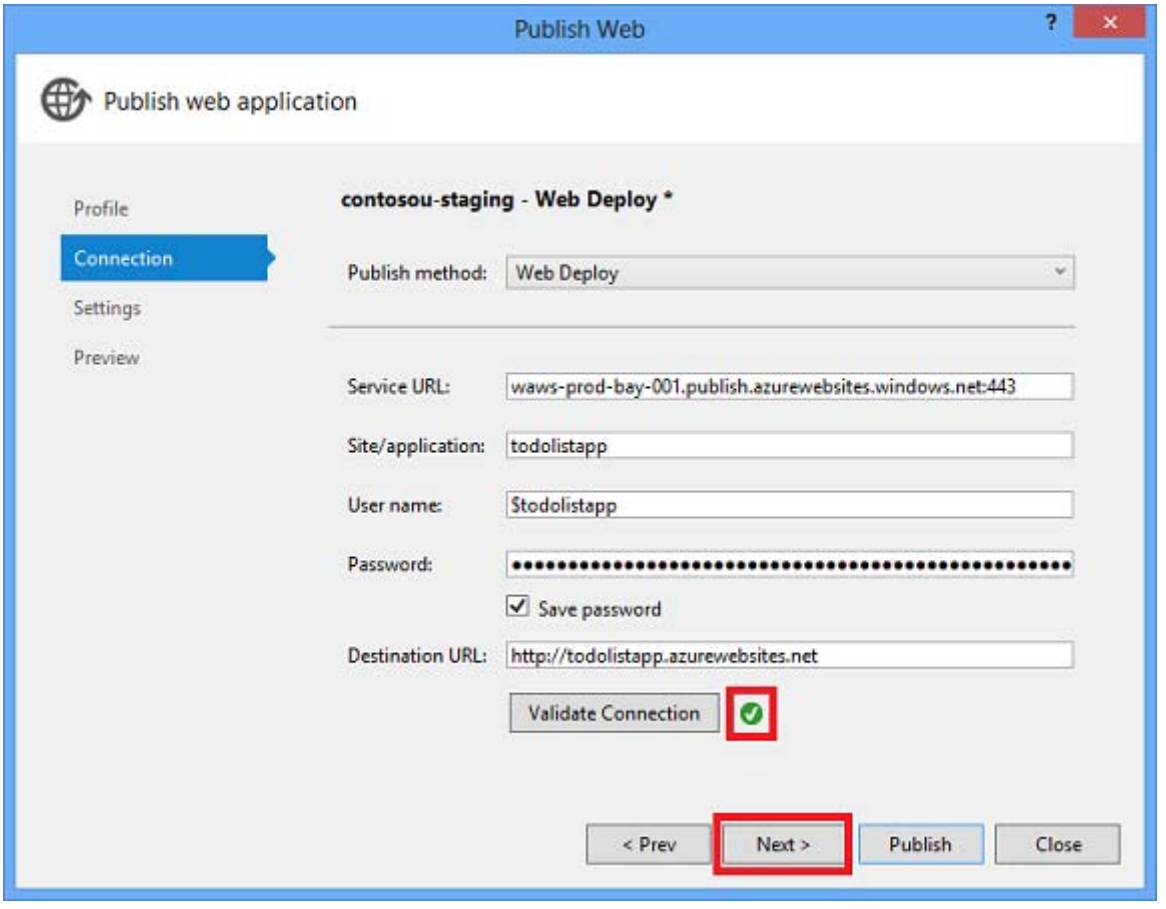


5. In the **Connection** tab, click **Validate Connection** to make sure that the settings are correct.

   When the connection has been validated, a green check mark is shown next to the **Validate Connection** button.

   For some hosting providers, when you click **Validate Connection**, you might see a **Certificate Error** dialog box. If you do, verify that the server name is what you expect. If the server name is correct, select **Save this certificate for future sessions of Visual Studio** and click **Accept**. (This error means that the hosting provider has chosen to avoid the expense of purchasing an SSL certificate for the URL that you are deploying to. If you prefer to establish a secure connection by using a valid certificate, contact your hosting provider.)

6. Click **Next**.



7. In the **Settings** tab, expand **File Publish Options**, and then select **Exclude files from the App_Data folder**.

   For information about the other options under **File Publish Options**, see the deploying to IIS tutorial. The screen shot that shows the result of this step and the following database configuration steps is at the end of the database configuration steps.

8. Under **DefaultConnection** in the **Databases** section, configure database deployment for the membership database.
   1. Select **Update database**.

      The **Remote connection string** box directly below **DefaultConnection** is filled in with the connection string from the .publishsettings file.The connection string includes SQL Server credentials, which are stored in plain text in the *.pubxml* file. If you prefer not to store them permanently there, you can remove them from the publish profile after the database is deployed and store them in Windows Azure instead. For more information, see How to keep your ASP.NET database connection strings secure when deploying to Azure from Source on Scott Hanselman's blog.

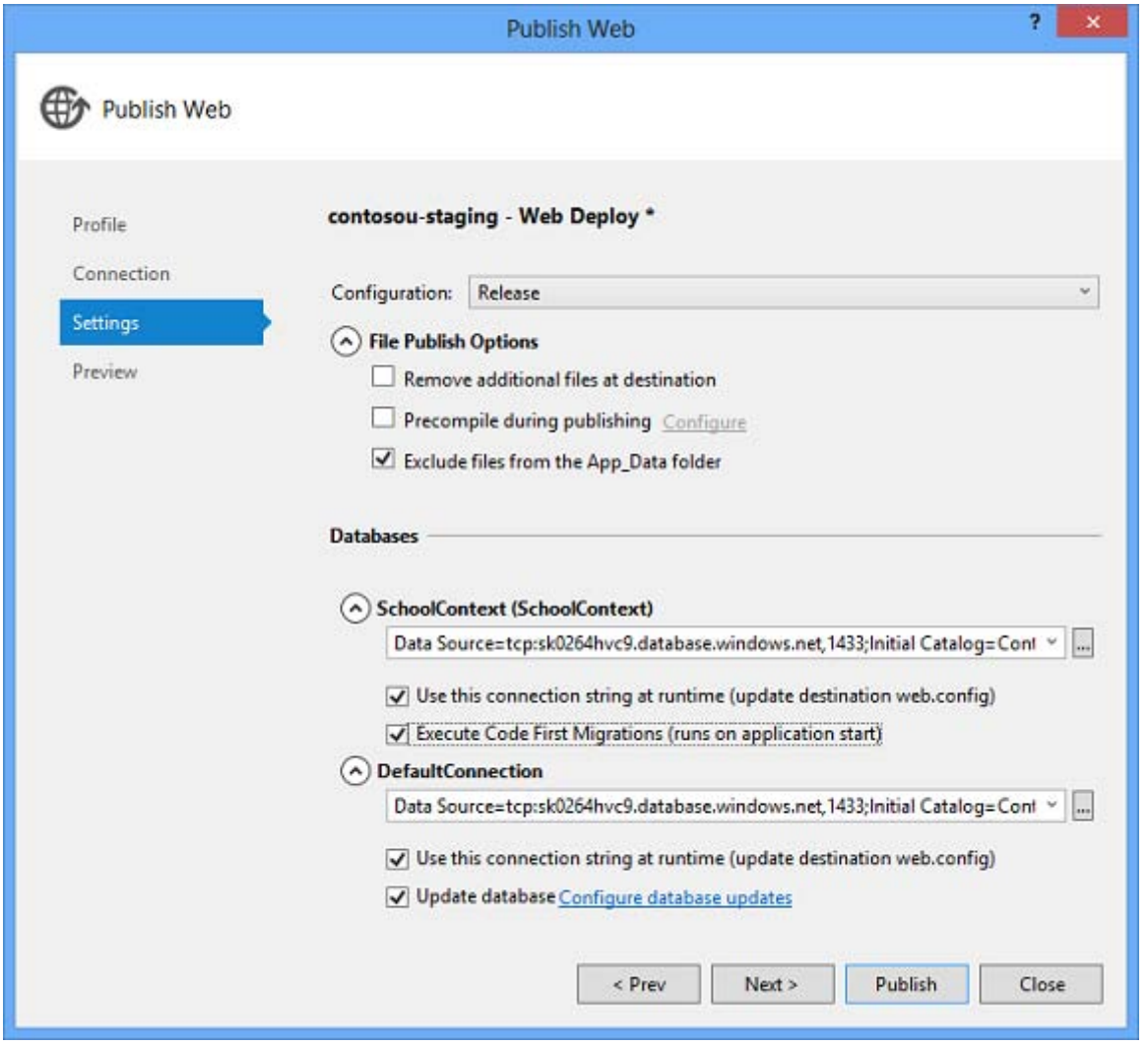
   2. Click **Configure database updates**.

3. In the **Configure Database Updates** dialog box, click **Add SQL Script**.
4. In the **Add SQL Script** box, navigate to the *aspnet-data-prod.sql* script that you saved earlier in the solution folder, and then click **Open**.
5. Close the **Configure Database Updates** dialog box.

9. Under **SchoolContext** in the **Databases** section, select **Execute Code First Migrations (runs on application start)**.

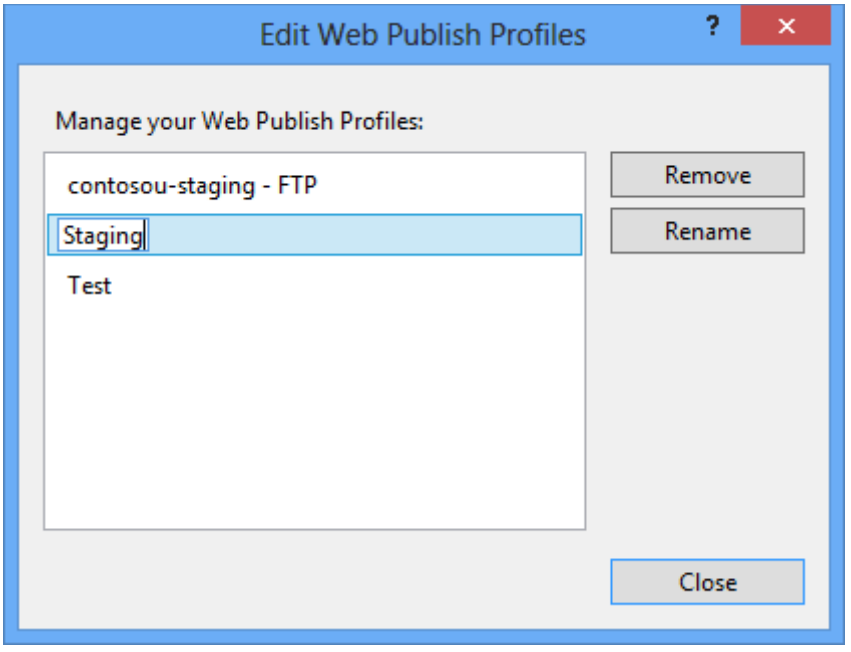
   Visual Studio displays **Execute Code First Migrations** instead of **Update Database** for `DbContext` classes. If you want to use the dbDacFx provider instead of Migrations to deploy a database that you access by using a `DbContext` class, see How do I deploy a Code First database without Migrations? in the Web Deployment FAQ for Visual Studio and ASP.NET on MSDN.

   The **Settings** tab now looks like the following example:



10. Perform the following steps to save the profile and rename it to *Staging*:
    1. Click the **Profile** tab, and then click **Manage Profiles**.

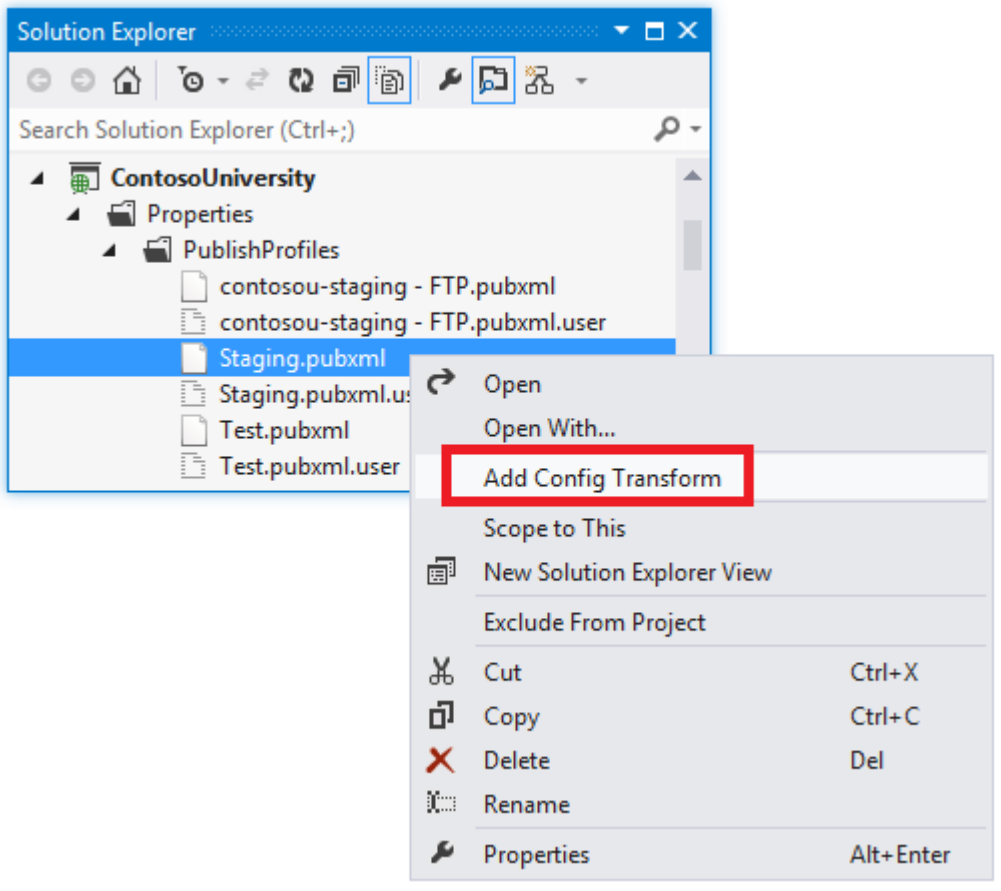
2. The import created two new profiles, one for FTP and one for Web Deploy. You configured the Web Deploy profile: rename this profile to *Staging*.



3. Close the **Edit Web Publish Profiles** dialog box.
4. Close the **Publish Web** wizard.

Configure a publish profile transform for the environment indicator

1. In **Solution Explorer**, expand **Properties**, and then expand **PublishProfiles**.
2. Right-click *Staging.pubxml*, and then click **Add Config Transform**.



Visual Studio creates the *Web.Staging.config* transform file and opens it.

3. In the *Web.Staging.config* transform file, insert the following code immediately after the opening `configuration` tag.

```
<appSettings>
    <add key="Environment" value="Prod" xdt:Transform="SetAttributes"
xdt:Locator="Match(key)"/>
</appSettings>
```

When you use the Staging publish profile, this transform sets the environment indicator to "Prod". In the deployed site you won't see any suffix such as "(Dev)" or "(Test)" after the "Contoso University" H1 heading.

4. Right-click the *Web.Staging.config* file and click **Preview Transform** to make sure that the transform you coded produces the expected changes.

The **Web.config Preview** window shows the result of applying both the *Web.Release.config* transforms and the *Web.Staging.config* transforms.

Prevent public use of the test site

An important consideration for the staging site is that it will be live on the Internet, but you don't want the public to use it. To minimize the likelihood that people will find and use it, you can use one or more of the following methods:

- Set firewall rules that allow access to the staging site only from IP addresses that you use to test staging.
- Use an obfuscated URL that would be impossible to guess.
- Create a *robots.txt* file to ensure that search engines will not crawl the test site and report links to it in search results.

The first of these methods is the most effective but is not covered in this tutorial because it would require that you deploy to a Windows Azure Cloud Service instead of a Windows Azure Web Site. For more information about Cloud Services and IP restrictions in Windows Azure, see Windows Azure Execution Models and How to Block Specific IP Addresses from Accessing a Web Role. If you are deploying to a third-party hosting provider, contact the provider to find out how to implement IP restrictions.

For this tutorial, you'll create a *robots.txt* file.

1. In **Solution Explorer**, right-click the ContosoUniversity project and click **Add New Item**.
2. Create a new **Text File** named *robots.txt*, and put the following text in it:

   ```
   User-agent: *
   Disallow: /
   ```

   The `User-agent` line tells search engines that the rules in the file apply to all search engine web crawlers (robots), and the `Disallow` line specifies that no pages on the site should be crawled.

   You do want search engines to catalog your production site, so you need to exclude this file from production deployment. To do that, you'll configure a setting in the Production publish profile when you create it.

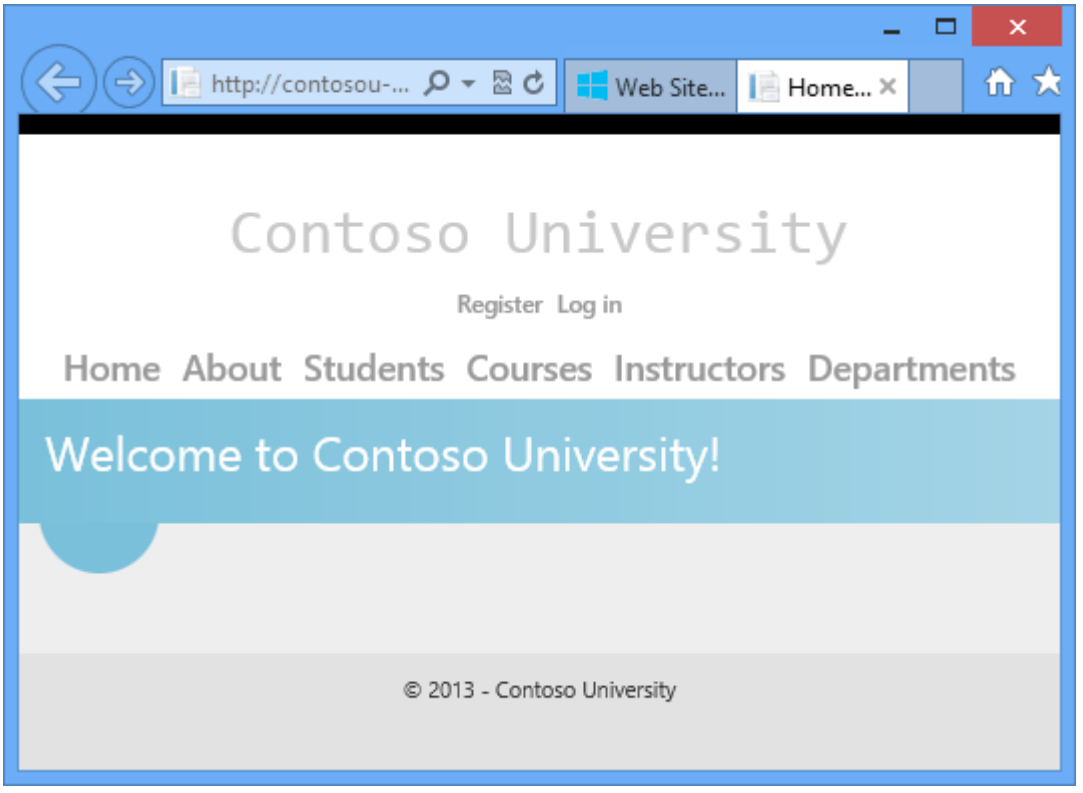
Deploy to staging

1. Open the **Publish Web** wizard by right-clicking the Contoso University project and clicking **Publish**.
2. Make sure that the **Staging** profile is selected.
3. Click **Publish**.

   The **Output** window shows what deployment actions were taken and reports successful completion of the deployment. The default browser automatically opens to the URL of the deployed site.

Test in the staging environment

Notice that the environment indicator is absent (there is no "(Test)" or "(Dev)" after the H1 heading, which shows that the *Web.config* transformation for the environment indicator was successful.



Run the **Students** page to verify that the deployed database has no students.

Run the **Instructors** page to verify that Code First seeded the database with instructor data:

Select **Add Students** from the **Students** menu, add a student, and then view the new student in the **Students** page to verify that you can successfully write to the database.

From the **Courses** page, click **Update Credits**. The **Update Credits** page requires administrator permissions, so the **Log In** page is displayed. Enter the administrator account credentials that you created earlier ("admin" and "prodpwd"). The **Update Credits** page is displayed, which verifies that the administrator account that you created in the previous tutorial was correctly deployed to the test environment.

Request an invalid URL to cause an error that ELMAH will track, and then request the ELMAH error report. If you are deploying to a third-party hosting provider, you will probably find that the report is empty for the same reason that it was empty in the previous tutorial. You will have to use the hosting provider's account management tools to configure folder permissions to enable ELMAH to write to the log folder.

The application that you created is now running in the cloud in a web site that is just like what you will use for production. Since everything is working correctly, the next step is to deploy to production.

Deploy to production

The process for creating a production web site and deploying to production is the same as for staging, except that you need to exclude the *robots.txt* from deployment. To do that you'll edit the publish profile file.

Create the production environment and the production publish profile

1. Create the production web site and database in Windows Azure, following the same procedure that you used for staging.

   When you create the database, you can choose to put it on the same server you created earlier, or create a new server.

2. Download the *.publishsettings* file.
3. Create the publish profile by importing the production *.publishsettings* file, following the same procedure that you used for staging.

   Don't forget to configure the data deployment script under **DefaultConnection** in the **Databases** section of the **Settings** tab.

4. Rename the publish profile to *Production*.
5. Configure a publish profile transform for the environment indicator, following the same procedure that you used for staging..

Edit the .pubxml file to exclude robots.txt

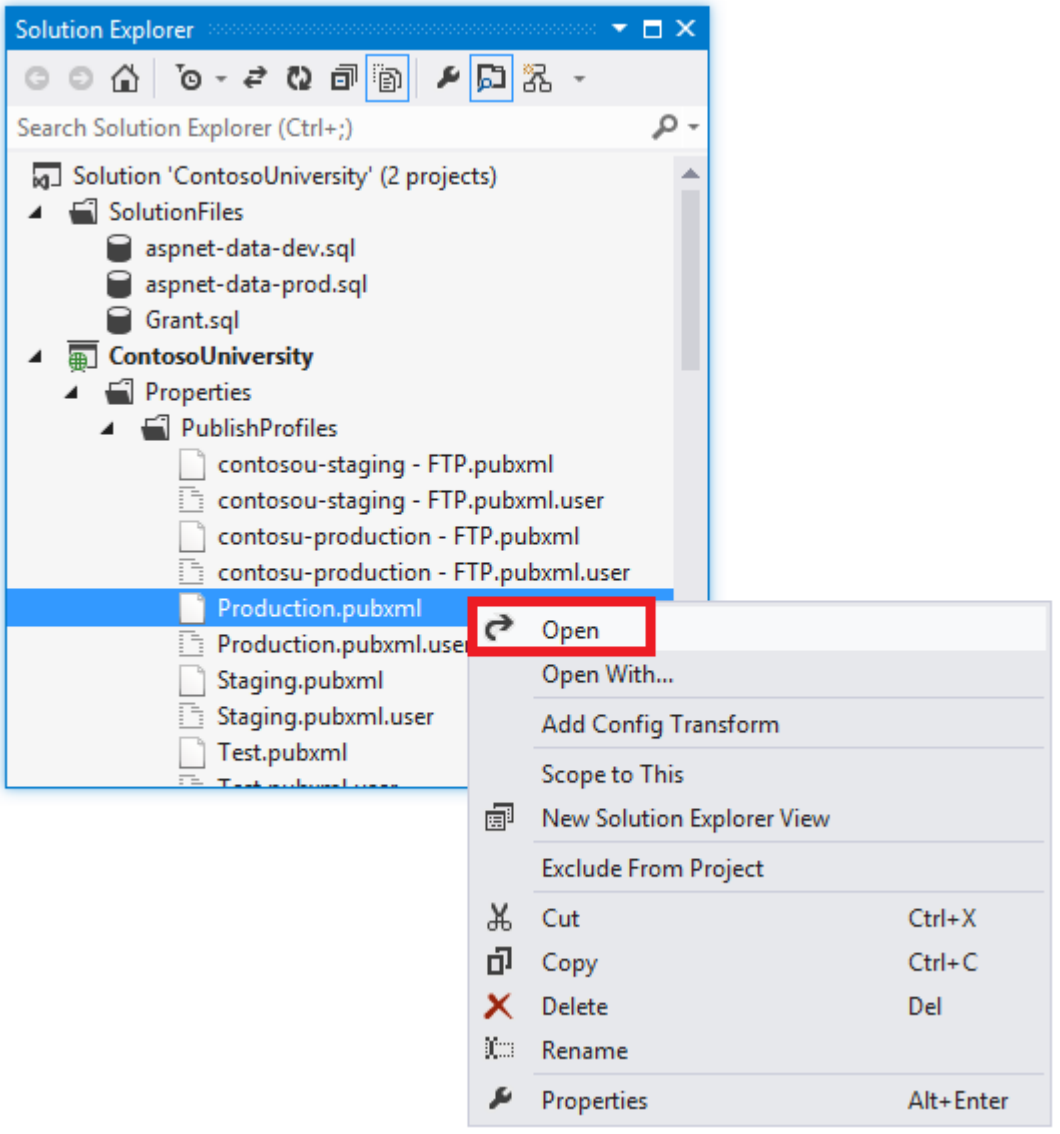
Publish profile files are named <profilename>.*pubxml* and are located in the *PublishProfiles* folder. The *PublishProfiles* folder is under the *Properties* folder in a C# web application project, under the *My Project* folder in a VB web application project, or under the *App_Data* folder in a web site project. Each *.pubxml* file contains settings that apply to one publish profile. The values you enter in the Publish Web wizard are stored in these files, and you can edit them to create or change settings that aren't made available in the Visual Studio UI.

By default, *.pubxml* files are included in the project when you create a publish profile, but you can exclude them from the project and Visual Studio will still use them. Visual Studio looks in the *PublishProfiles* folder for *.pubxml* files, regardless of whether they are included in the project.

For each *.pubxml* file there is a *.pubxml.user* file. The *.pubxml.user* file contains the encrypted password if you selected the **Save password** option, and by default it is excluded from the project.

A *.pubxml* file contains the settings that pertain to a specific publish profile. If you want to configure settings that apply to all profiles, you can create a *.wpp.targets* file. The build process imports these files into the *.csproj* or *.vbproj* project file, so most settings that you can configure in the project file can be configured in these files. For more information about *.pubxml* files and *.wpp.targets* files, see How to: Edit Deployment Settings in Publish Profile (.pubxml) Files and the .wpp.targets File in Visual Studio Web Projects.

1. In **Solution Explorer**, expand **Properties** and expand **PublishProfiles**.
2. Right-click *Production.pubxml* and click **Open**.



3. Right-click *Production.pubxml* and click **Open**.
4. Add the following lines immediately before the closing `PropertyGroup` element:

```
<ExcludeFilesFromDeployment>
    robots.txt
</ExcludeFilesFromDeployment>
```

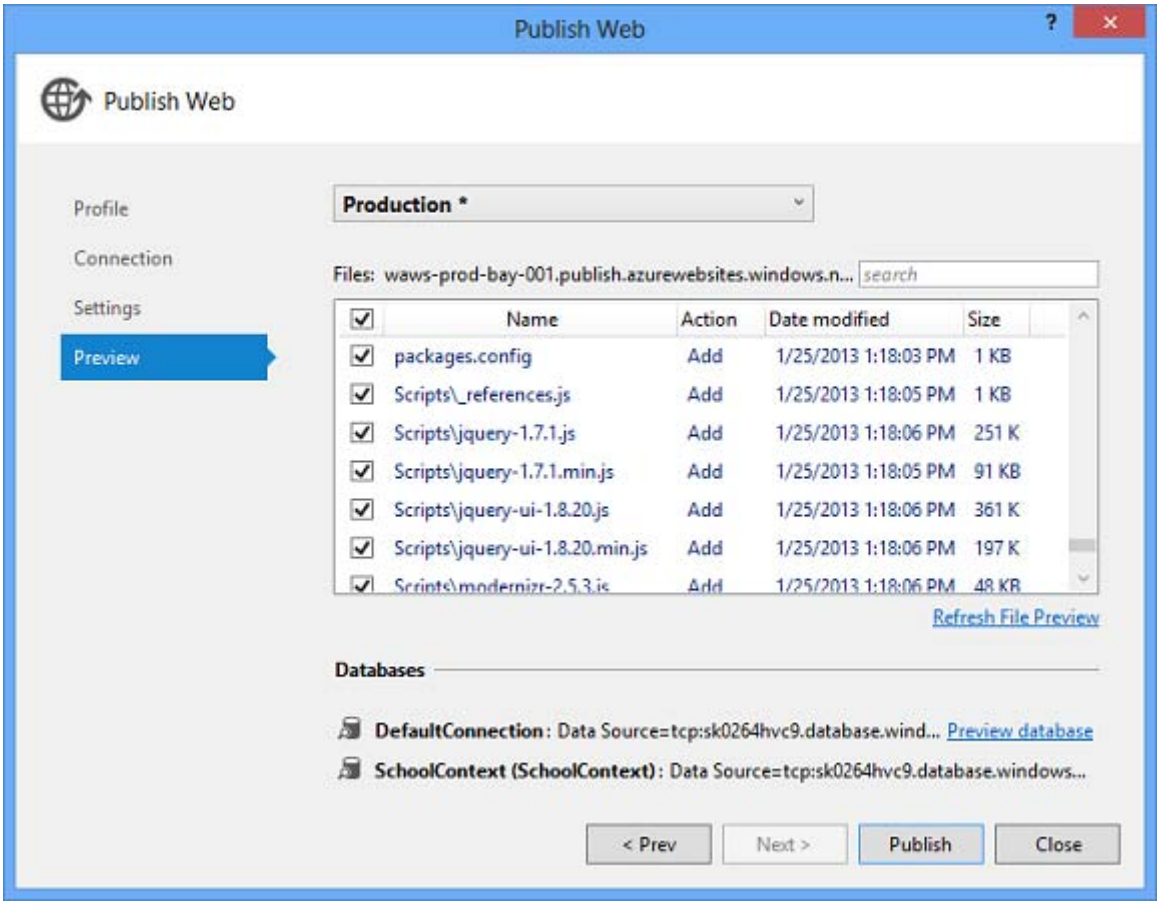
The .pubxml file now looks like the following example:

```
<Project ToolsVersion="4.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <WebPublishMethod>MSDeploy</WebPublishMethod>
    <LastUsedBuildConfiguration>Release</LastUsedBuildConfiguration>
    <LastUsedPlatform>Any CPU</LastUsedPlatform>
    <SiteUrlToLaunchAfterPublish>http://contosou-
staging.azurewebsites.net</SiteUrlToLaunchAfterPublish>
    <ExcludeApp_Data>True</ExcludeApp_Data>
    <MSDeployServiceURL>waws-prod-bay-
001.publish.azurewebsites.windows.net:443</MSDeployServiceURL>
    <DeployIisAppPath>contosou-staging</DeployIisAppPath>
    <RemoteSitePhysicalPath />
    <SkipExtraFilesOnServer>True</SkipExtraFilesOnServer>
    <MSDeployPublishMethod>WMSVC</MSDeployPublishMethod>
    <UserName>[username]</UserName>
    <_SavePWD>True</_SavePWD>
    <PublishDatabaseSettings>
      <!-- database settings removed -->
    </PublishDatabaseSettings>
    <ExcludeFilesFromDeployment>
        robots.txt
    </ExcludeFilesFromDeployment>
  </PropertyGroup>
  <ItemGroup>
    <MSDeployParameterValue
Include="$(DeployParameterPrefix)DefaultConnection-Web.config
Connection String">
      <ParameterValue>Data
Source=tcp:sk0264hvc9.database.windows.net,1433;Initial
Catalog=ContosoUniversity-staging;User Id=CU-staging-
admin@sk0264hvc9;Password=Pas$w0rd</ParameterValue>
    </MSDeployParameterValue>
    <MSDeployParameterValue
Include="$(DeployParameterPrefix)SchoolContext-Web.config Connection
String">
      <ParameterValue>Data
Source=tcp:sk0264hvc9.database.windows.net,1433;Initial
Catalog=ContosoUniversity-staging;User Id=CU-staging-
admin@sk0264hvc9;Password=Pas$w0rd</ParameterValue>
    </MSDeployParameterValue>
  </ItemGroup>
</Project>
```

For more information about how to exclude files and folders, see Can I exclude specific files or folders from deployment? in the **Web Deployment FAQ for Visual Studio and ASP.NET** on MSDN.

Deploy to production

1. Open the **Publish Web** wizard make sure that the **Production** publish profile is selected, and then click **Start Preview** on the **Preview** tab to verify that the *robots.txt* file will not be copied to the production site.



Review the list of files that will be copied. You'll see that all of the *.cs* files, including *.aspx.cs*, *.aspx.designer.cs*, *Master.cs*, and *Master.designer.cs* files are omitted. All of this code has been compiled into the *ContosoUniversity.dll* and *ContosUniversity.pdb* files that you'll find in the *bin* folder. Because only the *.dll* is needed to run the application, and you specified earlier that only files needed to run the application should be deployed, no *.cs* files were copied to the destination environment. The *obj* folder and the *ContosoUniversity.csproj* and *.csproj.user* files are omitted for the same reason.

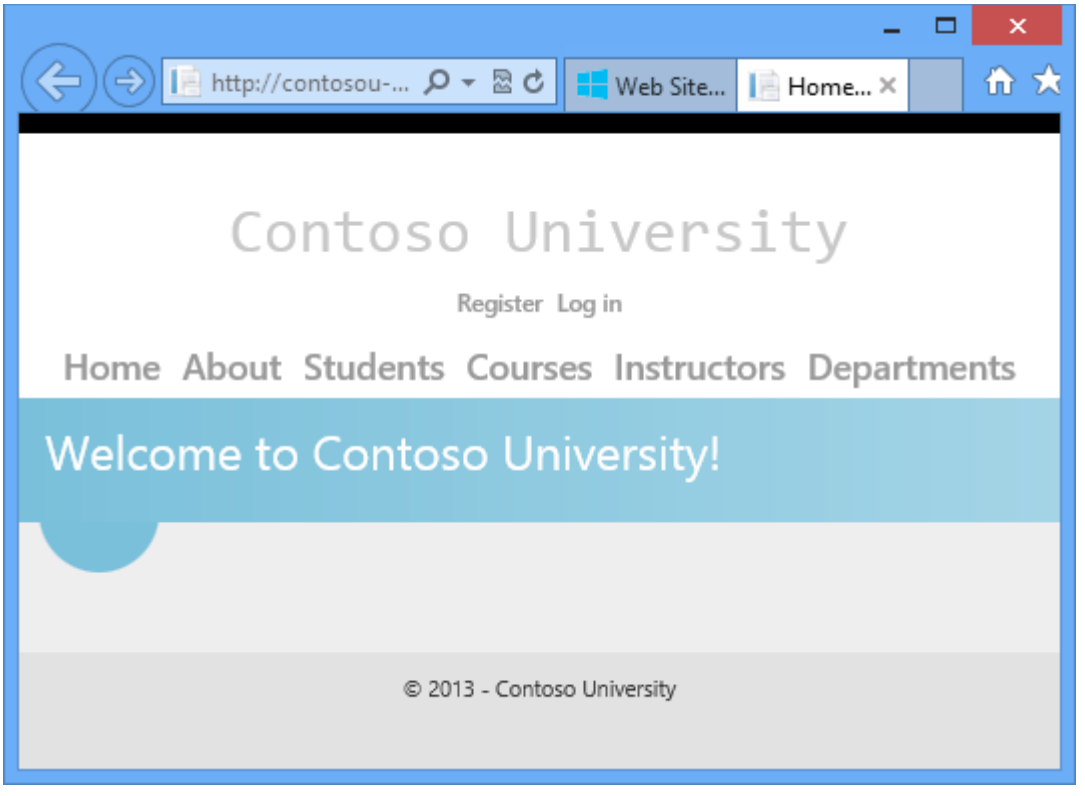
Click **Publish** to deploy to the production environment.

2. Test in production, following the same procedure you used for staging.

   Everything is identical to staging except for the URL and the absence of the *robots.txt* file.

Summary

You have now successfully deployed and tested your site and it is available publicly over the Internet.



In the next tutorial, you'll update application code and deploy the change to the test, staging, and production environments.

Note: While your application is in use in the production environment you should be implementing a recovery plan. That is, you should be periodically backing up your databases from the production site to a secure storage location, and you should be keeping several generations of such backups. When you update the database, you should make a backup copy from immediately before the change. Then, if you make a mistake and don't discover it until after you have deployed it to production, you will still be able to recover the database to the state it was in before it became corrupted. For more information, see Windows Azure SQL Database Backup and Restore.

Note: In this tutorial the SQL Server edition that you are deploying to is Windows Azure SQL Database. While the deployment process is similar to other editions of SQL Server, a real production application might require special code for Windows Azure SQL Database in some scenarios. For more information, see Working with Windows Azure SQL Database and Choosing between SQL Server and Windows Azure SQL Database.

Deploying a Code Update

Overview

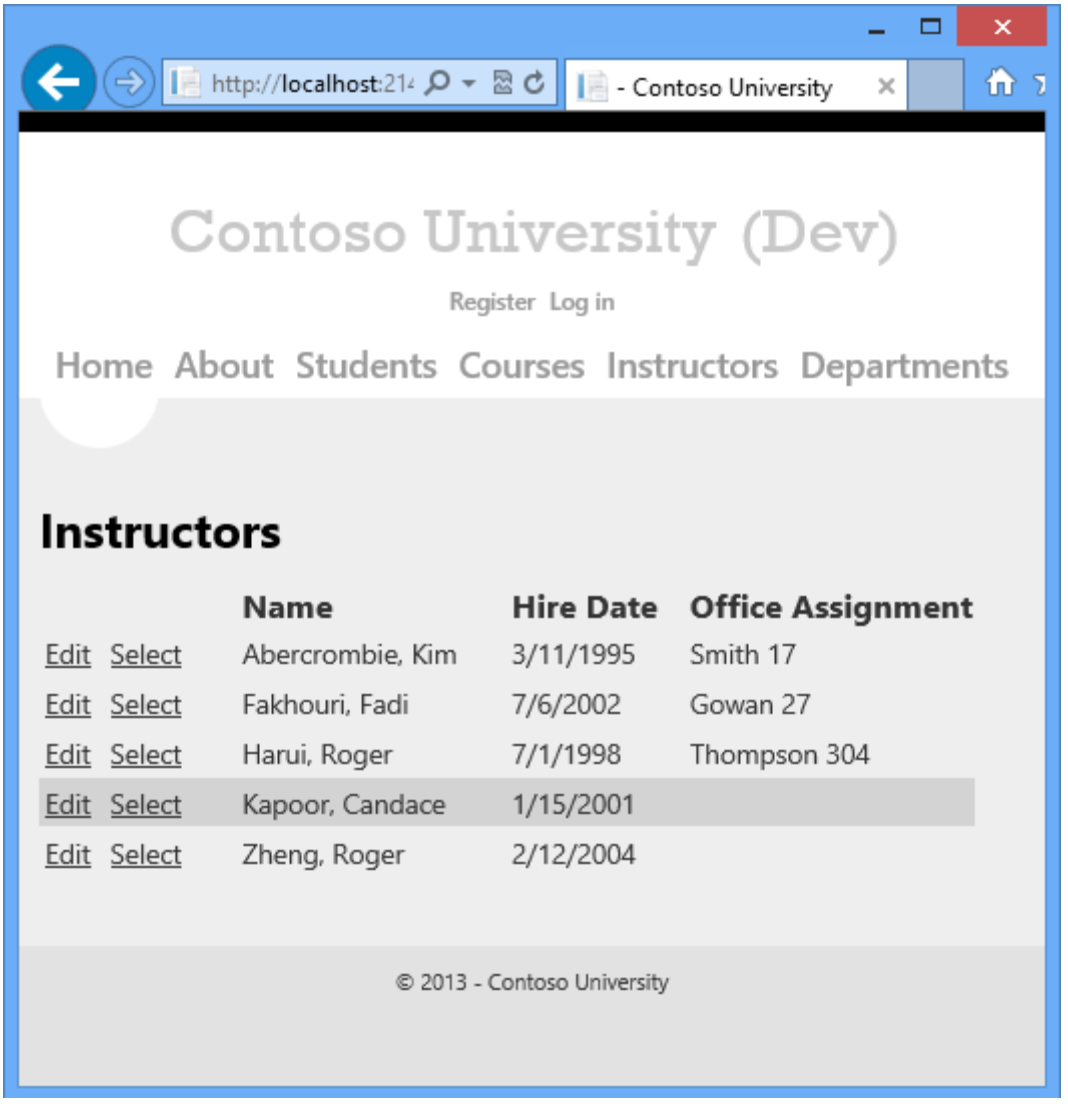
After the initial deployment, your work of maintaining and developing your web site continues, and before long you will want to deploy an update. This tutorial takes you through the process of deploying an update to your application code. The update that you implement and deploy in this tutorial does not involve a database change; you'll see what's different about deploying a database change in the next tutorial.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Make a code change

As a simple example of an update to your application, you'll add to the **Instructors** page a list of courses taught by the selected instructor.

If you run the **Instructors** page, you'll notice that there are **Select** links in the grid, but they don't do anything other than make the row background turn gray.



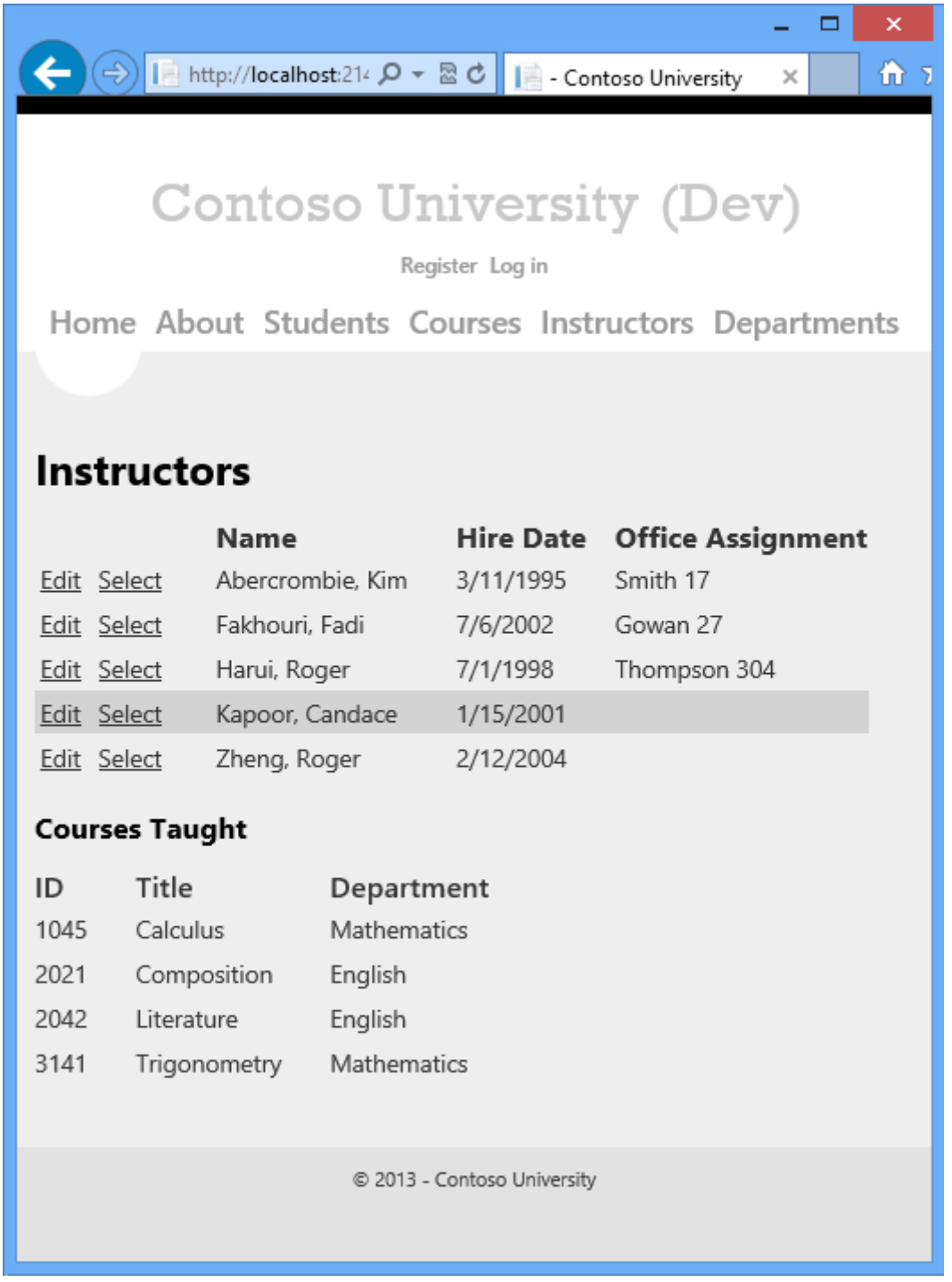
Now you'll add code that runs when the **Select** link is clicked and displays a list of courses taught by the selected instructor .

1. In *Instructors.aspx*, add the following markup immediately after the **ErrorMessageLabel** `Label` control:

```
<h3>Courses Taught</h3>
<asp:ObjectDataSource ID="CoursesObjectDataSource" runat="server"
TypeName="ContosoUniversity.BLL.SchoolBL"
    DataObjectTypeName="ContosoUniversity.DAL.Course"
SelectMethod="GetCoursesByInstructor">
    <SelectParameters>
        <asp:ControlParameter ControlID="InstructorsGridView"
Name="PersonID" PropertyName="SelectedDataKey.Value"
            Type="Int32" />
    </SelectParameters>
</asp:ObjectDataSource>
<asp:GridView ID="CoursesGridView" runat="server"
```

```
DataSourceID="CoursesObjectDataSource"
    AllowSorting="True" AutoGenerateColumns="False" SelectedRowStyle-
BackColor="LightGray"
    DataKeyNames="CourseID">
    <EmptyDataTemplate>
        <p>No courses found.</p>
    </EmptyDataTemplate>
    <Columns>
        <asp:BoundField DataField="CourseID" HeaderText="ID"
ReadOnly="True" SortExpression="CourseID" />
        <asp:BoundField DataField="Title" HeaderText="Title"
SortExpression="Title" />
        <asp:TemplateField HeaderText="Department"
SortExpression="DepartmentID">
            <ItemTemplate>
                <asp:Label ID="GridViewDepartmentLabel" runat="server"
Text='<%# Eval("Department.Name") %>'></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

2. Run the page and select an instructor. You see a list of courses taught by that instructor.



3. Close the browser.

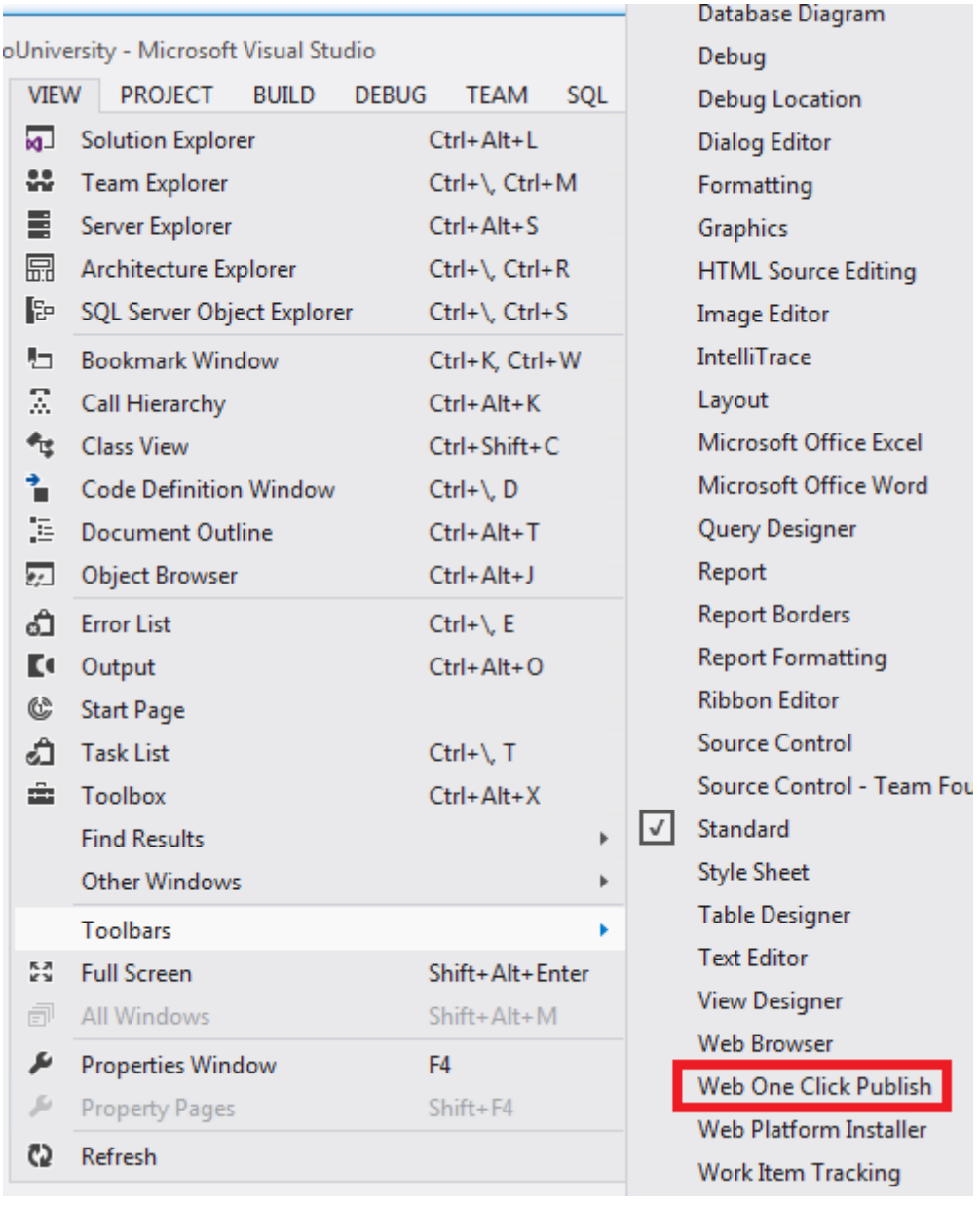
Deploy the code update to the test environment

Before you can use your publish profiles to deploy to test, staging, and production, you need to change database publishing options. You no longer need to run the grant and data deployment scripts for the membership database.

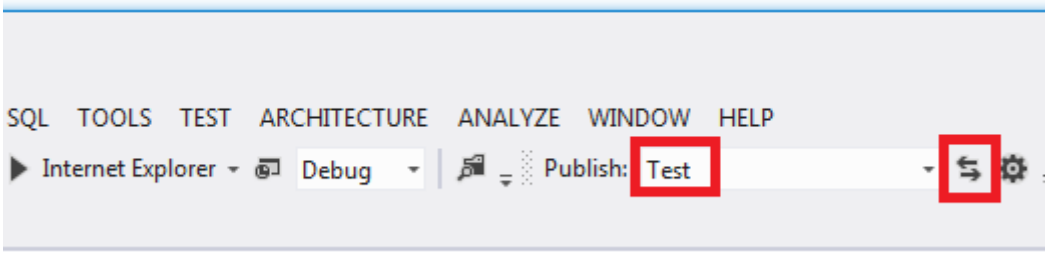
1. Open the **Publish Web** wizard by right-clicking the ContosoUniversity project and clicking **Publish**.
2. Click the **Test** profile in the **Profile** drop-down list.
3. Click the **Settings** tab.
4. Under **DefaultConnection** in the **Databases** section, clear the **Update database** check box.
5. Click the **Profile** tab, and then click the **Staging** profile in the **Profile** drop-down list.
6. When you are asked if you want to save the changes made to the **Test** profile, click **Yes**.
7. Make the same change in the Staging profile.
8. Repeat the process to make the same change in the Production profile.
9. Close the **Publish Web** wizard.

Deploying to the test environment is now a simple matter of running one-click publish again. To make this process quicker, you can use the **Web One Click Publish** toolbar.

1. In the **View** menu, choose **Toolbars** and then select **Web One Click Publish**.



2. In **Solution Explorer**, select the ContosoUniversity project.
3. the **Web One Click Publish** toolbar, choose the **Test** publish profile and then click **Publish Web** (the icon with arrows pointing left and right).



4. Visual Studio deploys the updated application, and the browser automatically opens to the home page.
5. Run the Instructors page and select an instructor to verify that the update was successfully deployed.

You would normally also do regression testing (that is, test the rest of the site to make sure that the new change didn't break any existing functionality). But for this tutorial you'll skip that step and proceed to deploy the update to staging and production.

Take the application offline during deployment

The change you're deploying now is a simple change to a single page. But sometimes you deploy larger changes, or you deploy both code and database changes, and the site might behave incorrectly if a user requests a page before deployment is finished. To prevent users from accessing the site while deployment is in progress, you can use an *app_offline.htm* file. When you put a file named *app_offline.htm* in the root folder of your application, IIS automatically displays that file instead of running your application. So to prevent access during deployment, you put *app_offline.htm* in the root folder, run the deployment process, and then remove *app_offline.htm* after successful deployment.

Using *app_offline.htm* in the staging site isn't required, because you don't have users accessing the staging site. But it's a good practice to use staging to test everything the way you plan to deploy in production.

Create app_offline.htm

1. In **Solution Explorer**, right-click the solution and click **Add**, and then click **New Item**.
2. Create an **HTML Page** named *app_offline.htm* (delete the final "l" in the *.html* extension that Visual Studio creates by default).
3. Replace the template markup with the following markup:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Contoso University - Under Construction</title>
</head>
<body>
  <h1>Contoso University</h1>
  <h2>Under Construction</h2>
  <p>The Contoso University site is temporarily unavailable while we
upgrade it. Please try again later.</p>
</body>
</html>
```

4. Save and close the file.

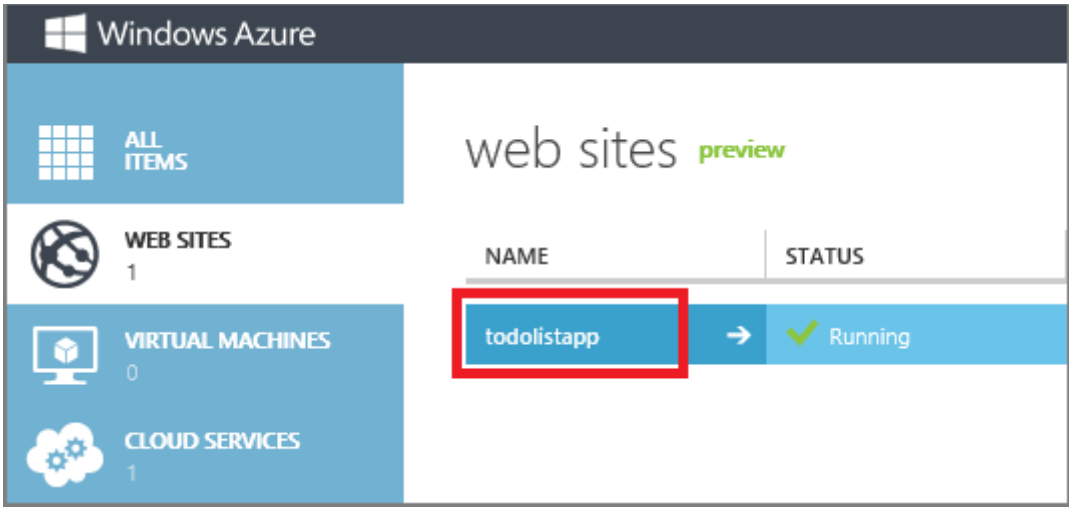
Copy app_offline.htm to the root folder of the web site

You can use any FTP tool to copy files to the web site. FileZilla is a popular FTP tool and is shown in the screen shots.

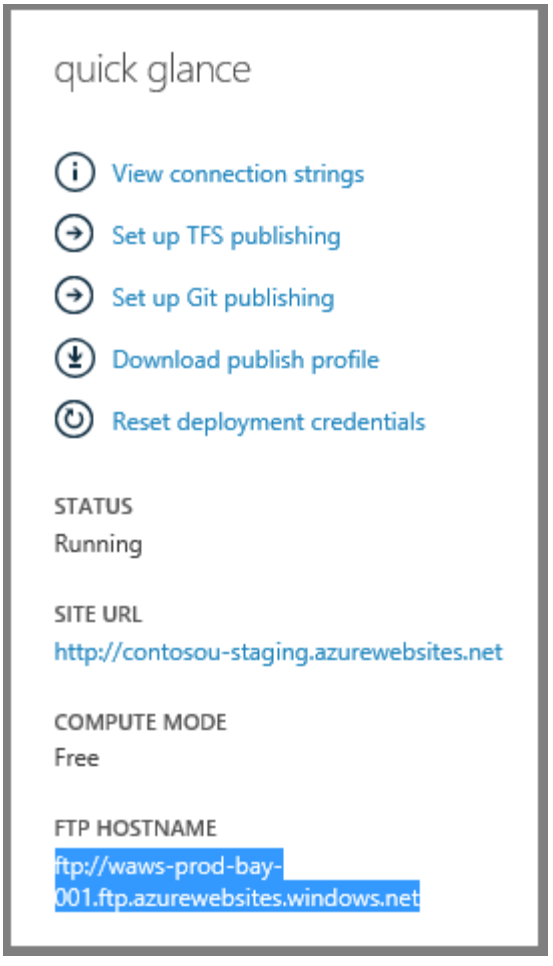
To use an FTP tool, you need three things: the FTP URL, the user name, and the password.

The URL is shown on the web site's dashboard page in the Windows Azure Management Portal, and the user name and password for FTP can be found in the *.publishsettings* file that you downloaded earlier. The following steps show how to get these values.

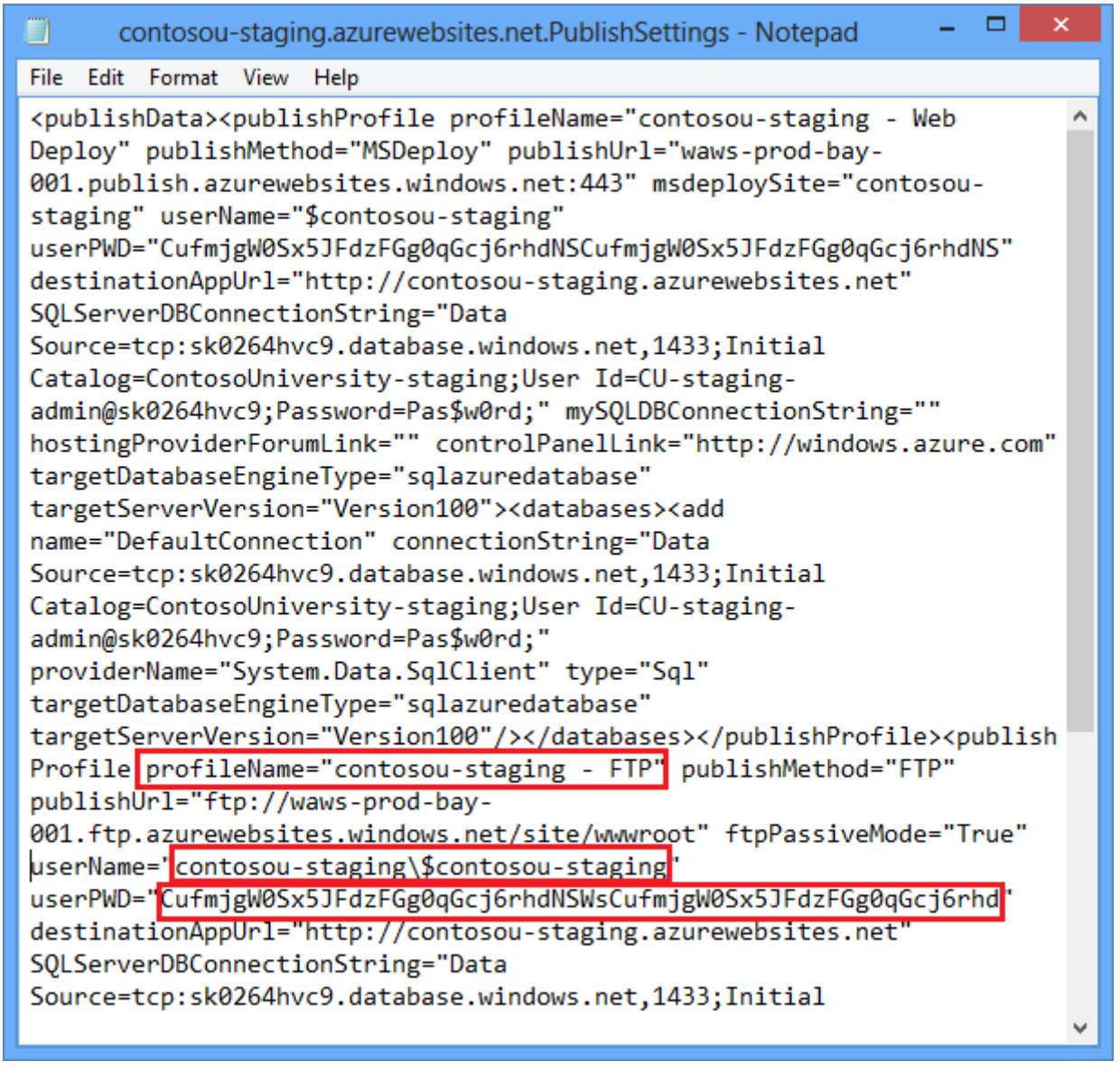
1. Click **Web Sites** tab and then click the staging web site.



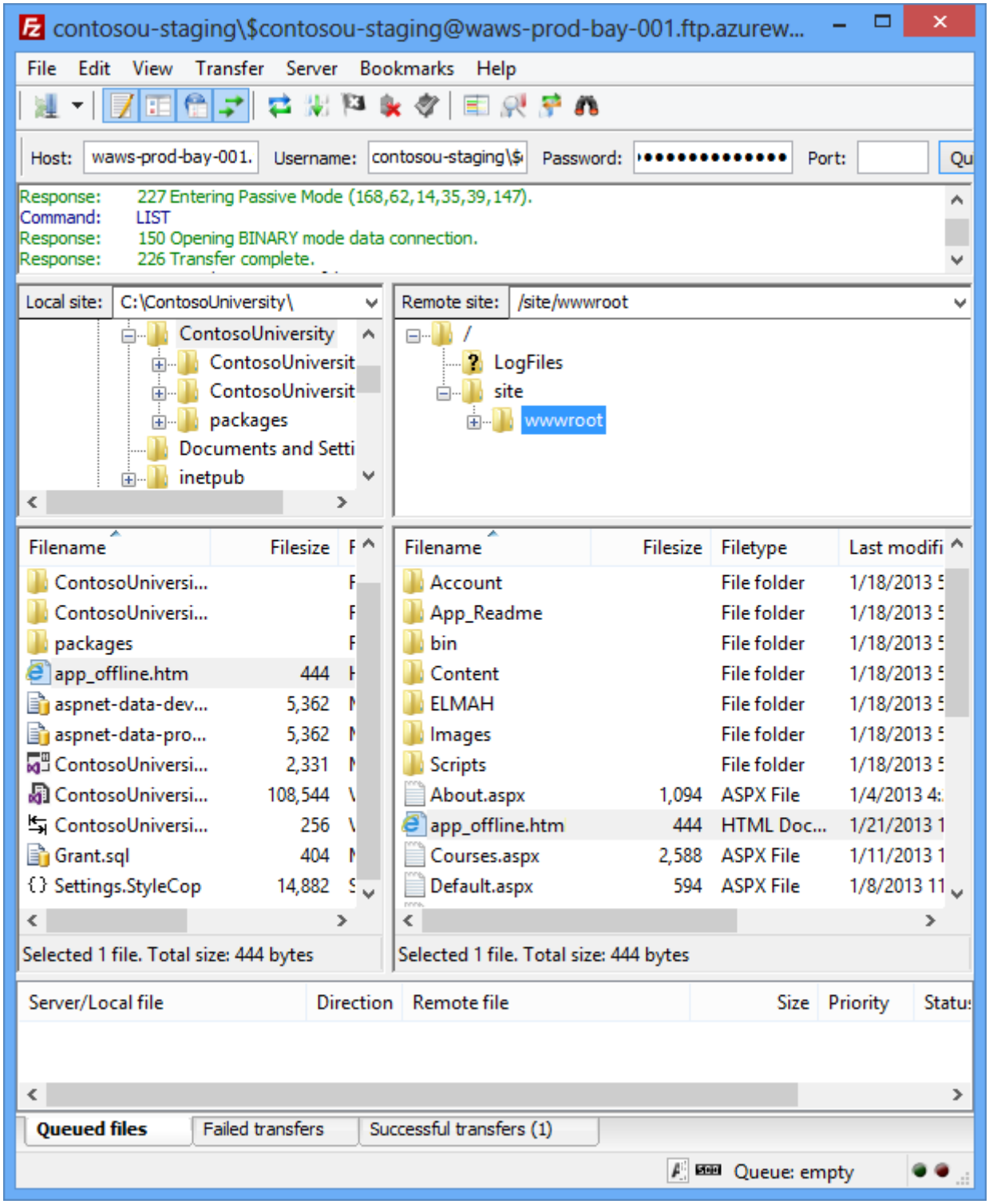
2. On the **Dashboard** page, scroll down to find the FTP host name in the **Quick Glance** section.



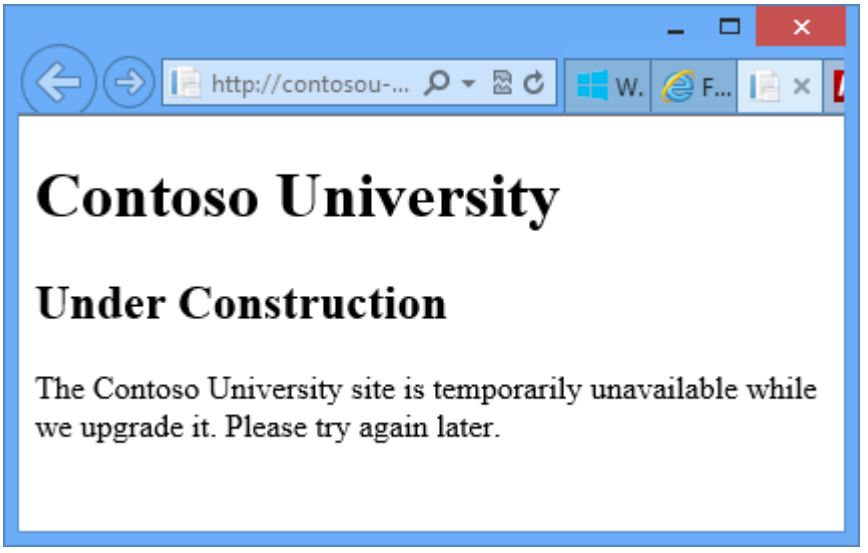
3. Open the staging *.publishsettings* file in Notepad or another text editor.
4. Find the `publishProfile` element for the FTP profile.
5. Copy the `userName` and `userPWD` values.



6. Open your FTP tool and log on to the FTP URL.
7. Copy *app_offline.htm* from the solution folder to the */site/wwwroot* folder in the staging site.



8. Browse to your staging site's URL. You see that the *app_offline.htm* page is now displayed instead of your home page.



You are now ready to deploy to staging.

Deploy the code update to staging and production

1. In the **Web One Click Publish** toolbar, choose the **Staging** publish profile and then click **Publish Web**.

   Visual Studio deploys the updated application and opens the browser to the site's home page. The *app_offline.htm* file is displayed. Before you can test to verify successful deployment, you must remove the *app_offline.htm* file.

2. Return to your FTP tool, and delete **app_offline.htm** from the staging site.
3. In the browser, open the Instructors page in the staging site, and select an instructor to verify that the update was successfully deployed.
4. Follow the same procedure for production as you did for staging.

Reviewing Changes and Deploying Specific Files

Visual Studio 2012 also gives you the ability to deploy individual files. For a selected file you can view differences between the local version and the deployed version, deploy the file to the destination environment, or copy the file from the destination environment to the local project. In this section of the tutorial you see how to use these features.

Make a change to deploy

1. Open *Content/Site.css*, and find the block for the `body` tag.
2. Change the value for `background-color` from `#fff` to `darkblue`.

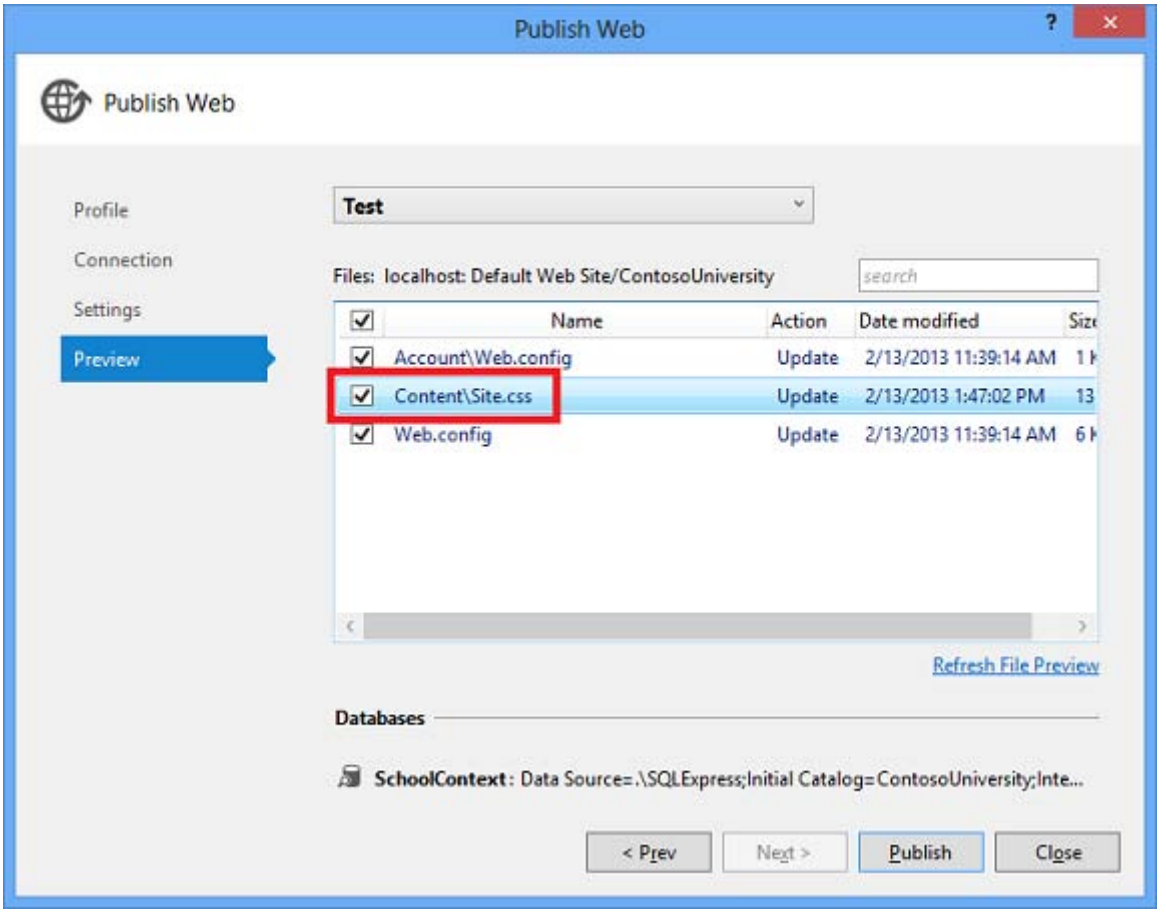
```
body {
    background-color: darkblue;
```

```
    border-top: solid 10px #000;
    color: #333;
    font-size: .85em;
    font-family: "Segoe UI", Verdana, Helvetica, Sans-Serif;
    margin: 0;
    padding: 0;
}
```

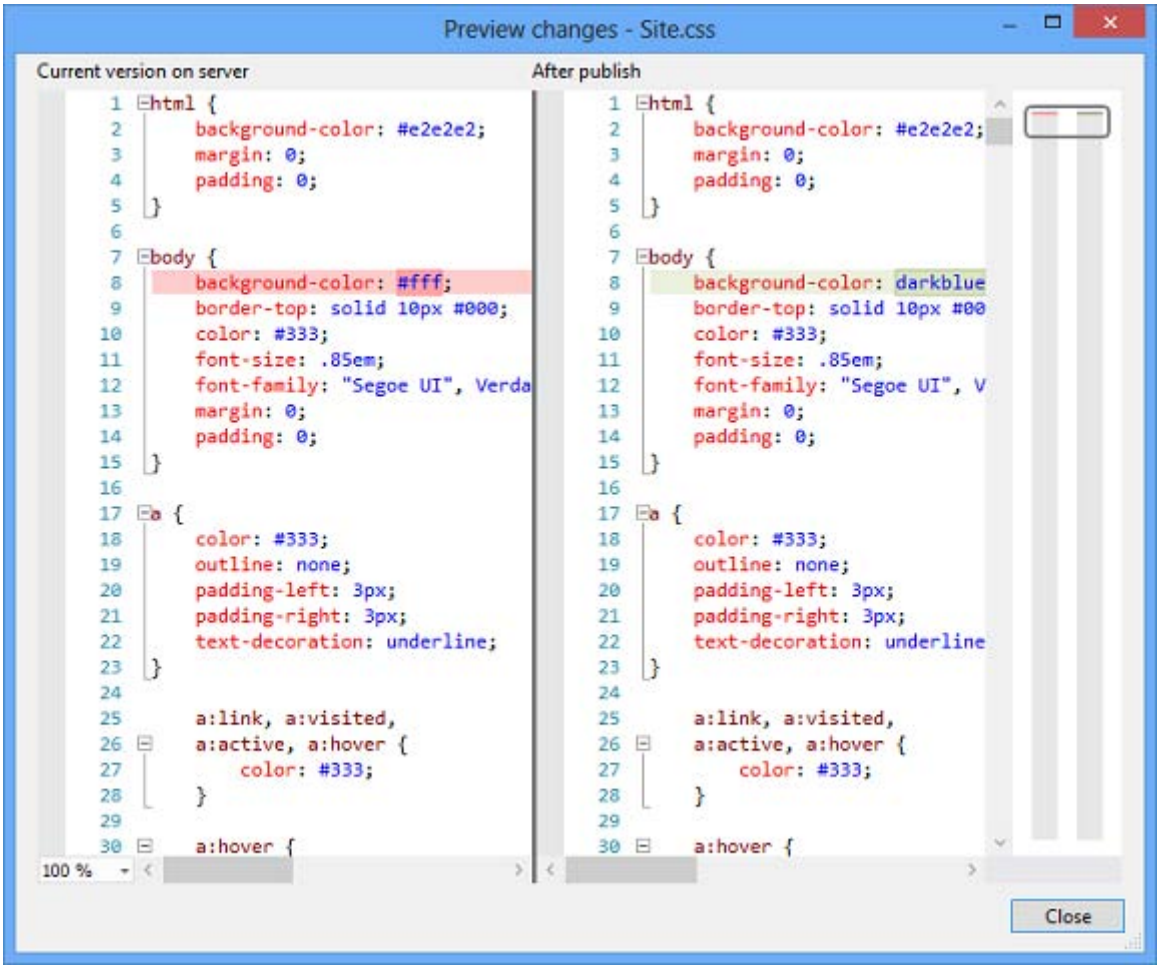
View the change in the Publish Preview window

When you use the **Publish Web** wizard to publish the project, you can see what changes are going to be published by double-clicking the file in the **Preview** window.

1. Right-click the ContosoUniversity project and click **Publish**.
2. From the **Profile** drop-down list, select the **Test** publish profile.
3. Click **Preview**, and then click **Start Preview**.
4. In the **Preview** pane, double-click **Site.css**.

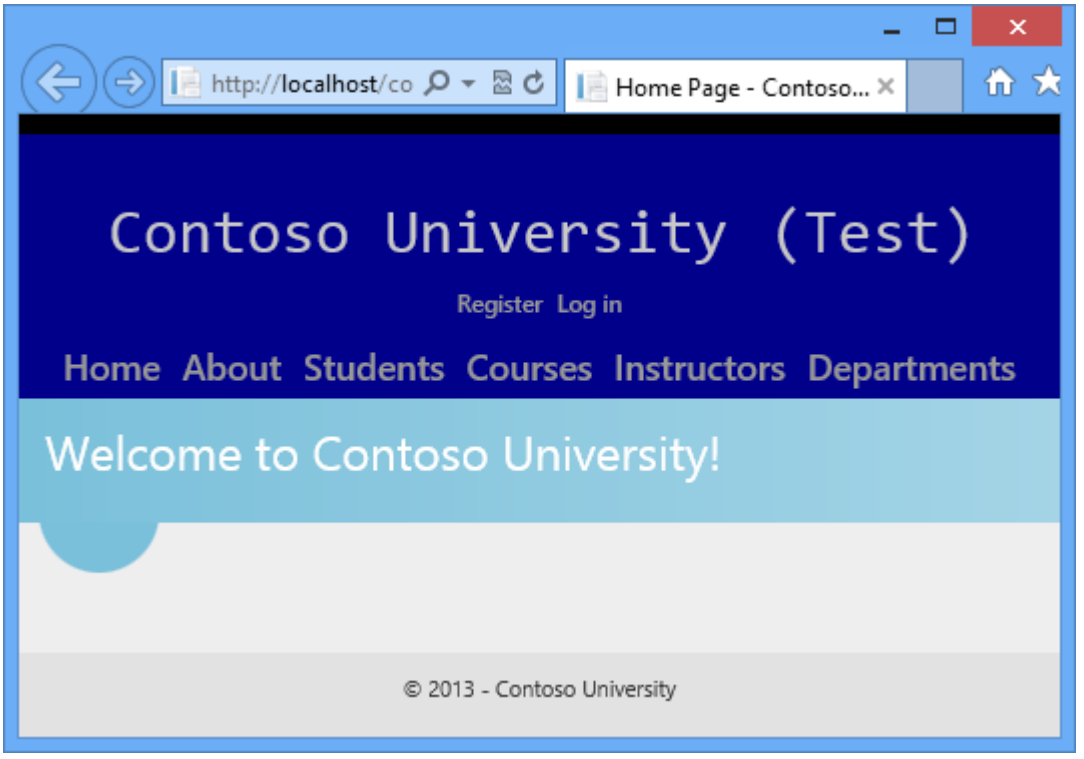


The **Preview changes** dialog shows a preview of the changes that will be deployed.



If you double-click the *Web.config* file, the **Preview changes** dialog shows the effect of your build configuration transformations and publish profile transformations. At this point you have not done anything that would cause the *Web.config* file on the server to change, so you expect to see no changes. However, the **Preview changes** window incorrectly shows two changes. It looks like two XML elements will be removed. These elements are added by the publish process when you select **Execute Code First Migrations on application start** for a Code First context class. The comparison is done before the publish process adds those elements, so it looks like they are being removed although they will not be removed. This error will be corrected in a future release.

5. Click **Close**.
6. Click **Publish**.
7. When the browser opens to the home page of the Test site, press CTRL+F5 to cause a hard refresh in order to see the effect of the CSS change.



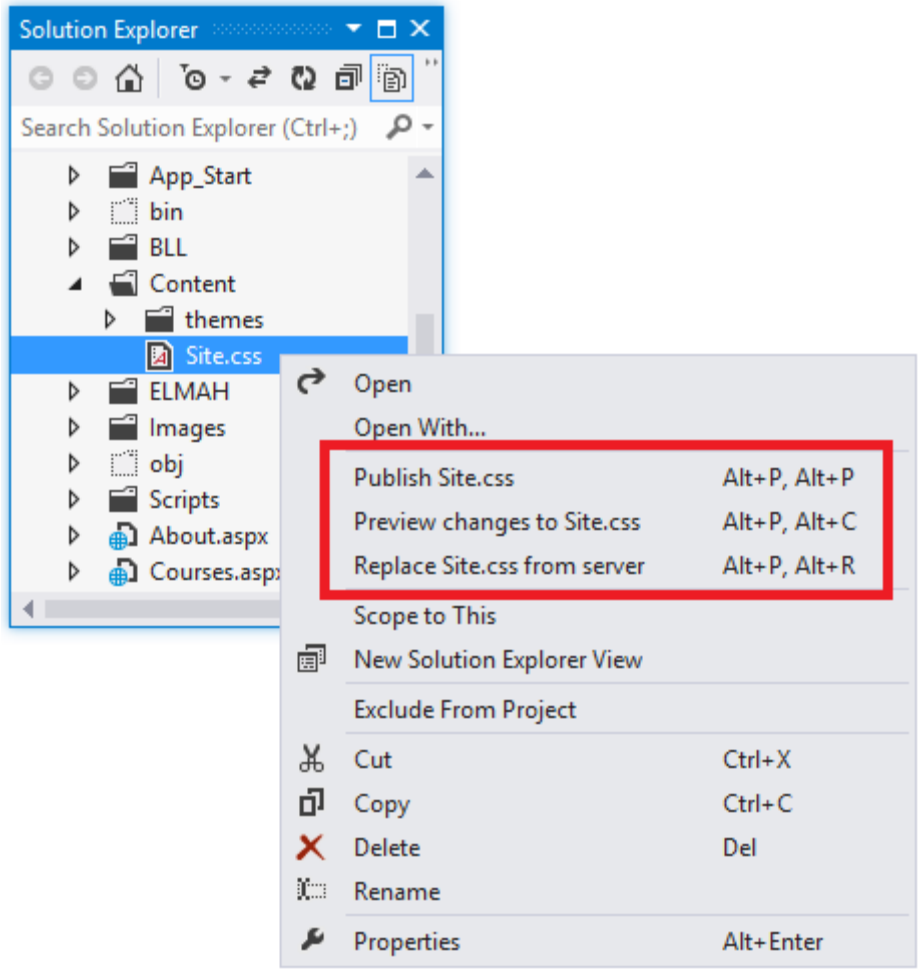
8. Close the browser.

Publish specific files from Solution Explorer

Suppose you don't like the blue background and want to revert to the original color. In this section you'll restore the original settings by publishing a specific file directly from **Solution Explorer**.

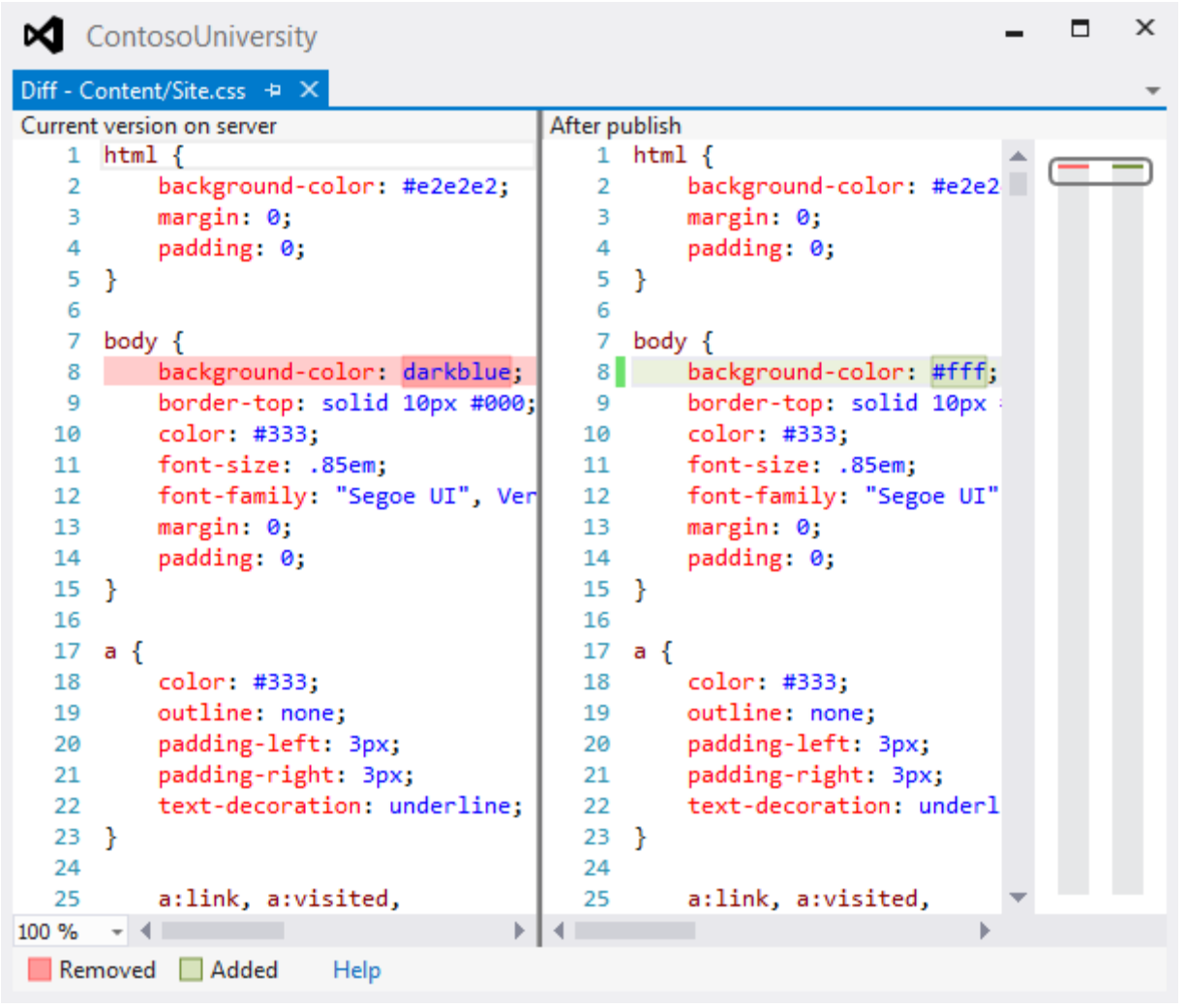
1. Open *Content/Site.css* and restore the `background-color` setting to `#fff`.
2. In **Solution Explorer**, right-click the *Content/Site.css* file.

   The context menu shows three publish options.



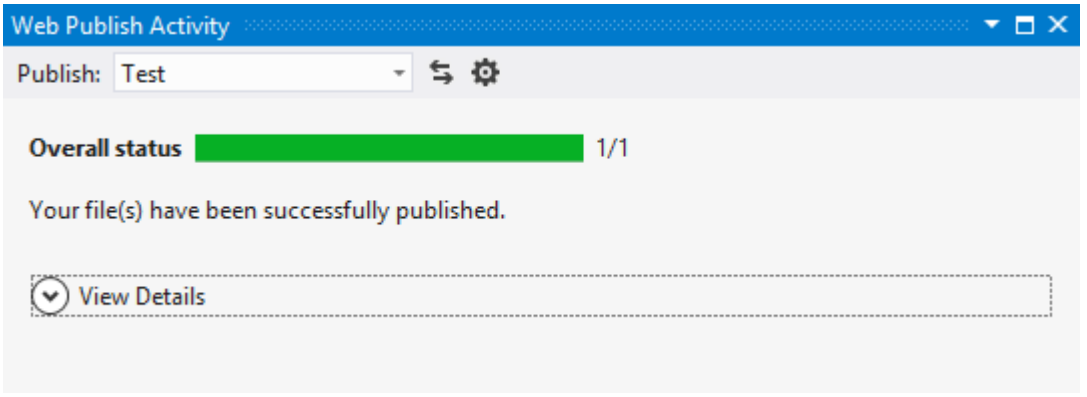
3. Click **Preview changes to Site.css**.

   A window opens to show the differences between the local file and the version of it in the destination environment.

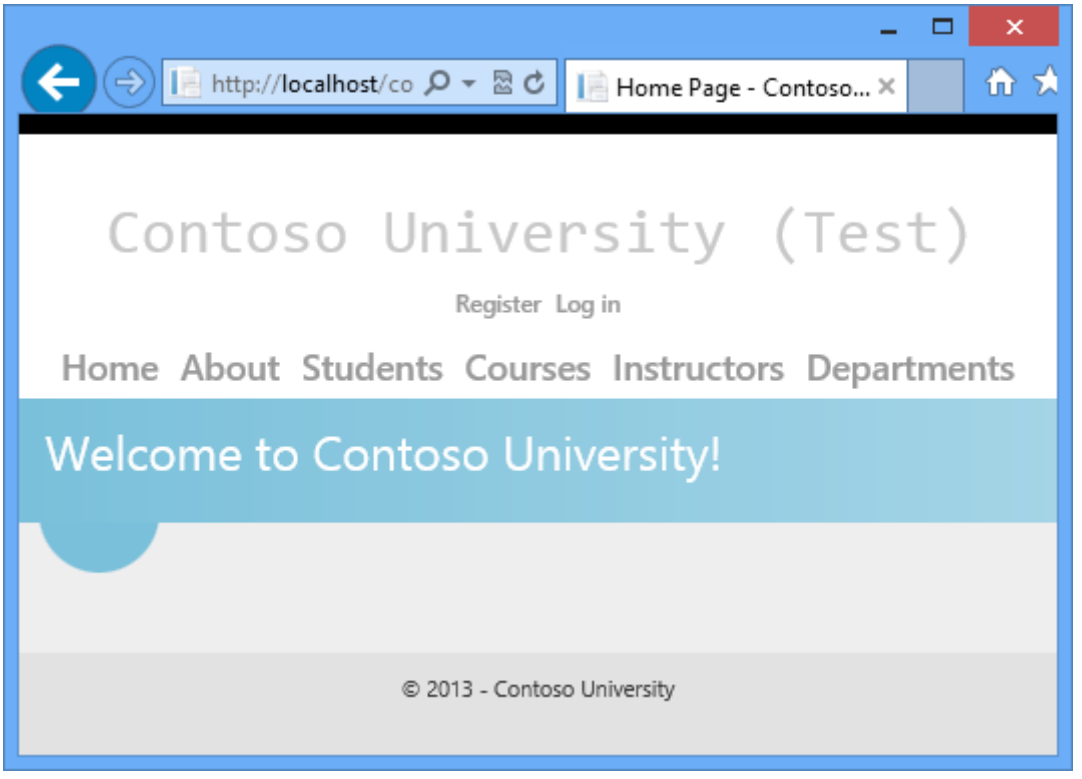


4. In **Solution Explorer**, right-click **Site.css** again and click **Publish Instructors.aspx**.

   The **Web Publish Activity** window shows that the file has been published.



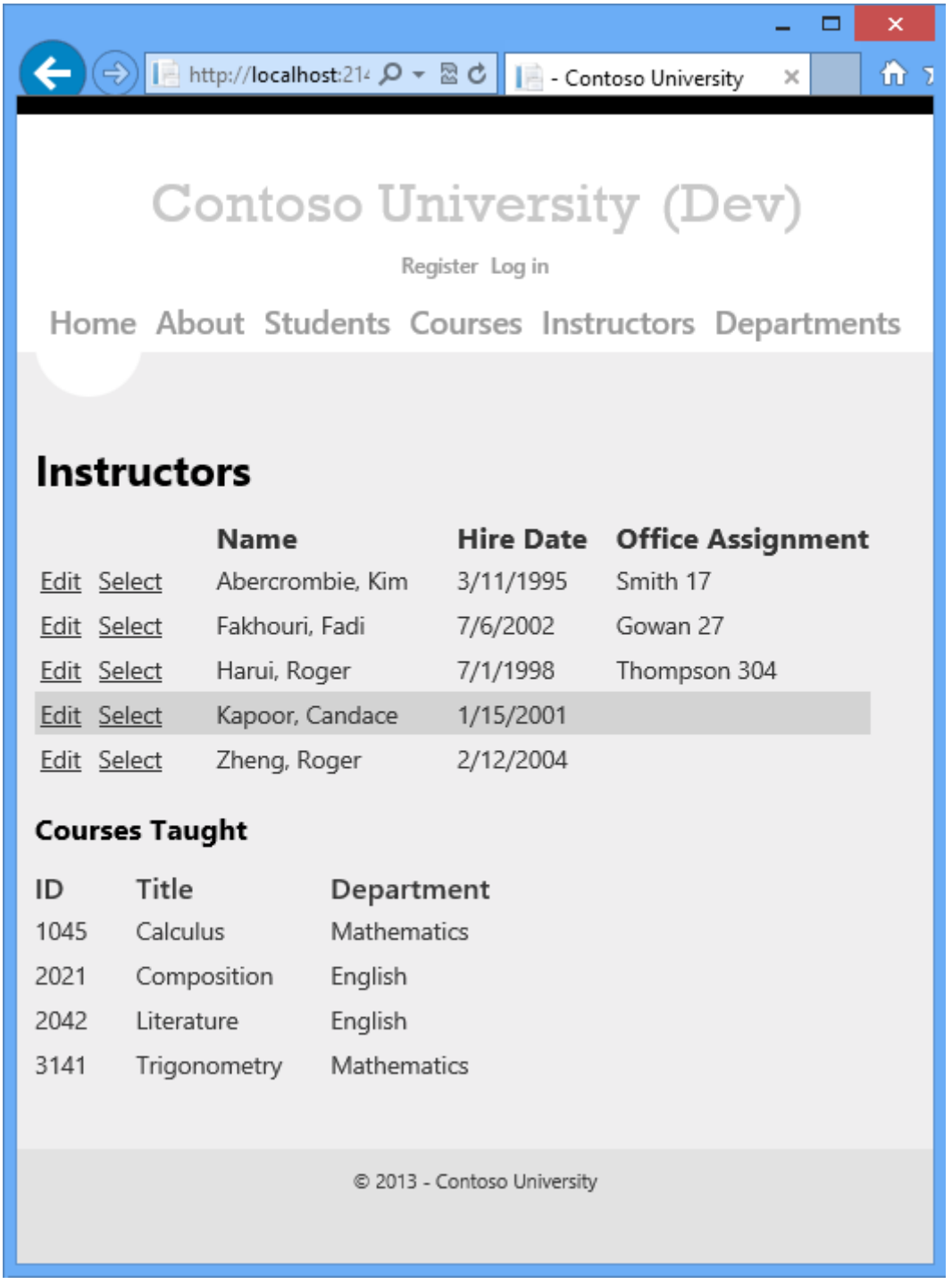
5. Open a browser to the http://localhost/contosouniversity URL, and then press CTRL+F5 to cause a hard refresh in order to see the effect of the CSS change.



6. Close the browser.

Summary

You've now seen several ways to deploy an application update that does not involve a database change, and you've seen how to preview the changes to verify that what will be updated is what you expect. The Instructors page now has a **Courses Taught** section.



The next tutorial shows you how to deploy a database change: you'll add a birthdate field to the database and to the Instructors page.

Deploying a Database Update

Overview

In this tutorial, you make a database change and related code changes, test the changes in Visual Studio, then deploy the update to the test, staging, and production environments.

The tutorial first shows how to update a database that is managed by Code First Migrations, and then later it shows how to update a database by using the dbDacFx provider.

Reminder: If you get an error message or something doesn't work as you go through the tutorial, be sure to check the troubleshooting page.

Deploy a database update by using Code First Migrations

In this section, you add a birth date column to the `Person` base class for the `Student` and `Instructor` entities. Then you update the page that displays instructor data so that it displays the new column. Finally, you deploy the changes to test, staging, and production.

Add a column to a table in the application database

1. In the *ContosoUniversity.DAL* project, open *Person.cs* and add the following property at the end of the `Person` class (there should be two closing curly braces following it):

   ```
   [DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode =
   true)]
   [Display(Name = "Birth Date")]
   public DateTime? BirthDate { get; set; }
   ```

   Next, update the `Seed` method so that it provides a value for the new column. Open *Migrations\Configuration.cs* and replace the code block that begins `var instructors = new List<Instructor>` with the following code block which includes birth date information:

   ```
   var instructors = new List<Instructor>
   {
       new Instructor { FirstMidName = "Kim",     LastName =
   "Abercrombie", HireDate = DateTime.Parse("1995-03-11"), BirthDate =
   DateTime.Parse("1918-08-12"), OfficeAssignment = new OfficeAssignment {
   Location = "Smith 17" } },
       new Instructor { FirstMidName = "Fadi",    LastName =
   "Fakhouri",    HireDate = DateTime.Parse("2002-07-06"), BirthDate =
   DateTime.Parse("1960-03-15"), OfficeAssignment = new OfficeAssignment {
   Location = "Gowan 27" } },
       new Instructor { FirstMidName = "Roger",   LastName =
   "Harui",       HireDate = DateTime.Parse("1998-07-01"), BirthDate =
   DateTime.Parse("1970-01-11"), OfficeAssignment = new OfficeAssignment {
   ```

```
Location = "Thompson 304" } },
    new Instructor { FirstMidName = "Candace", LastName =
"Kapoor",      HireDate = DateTime.Parse("2001-01-15"), BirthDate =
DateTime.Parse("1975-04-11") },
    new Instructor { FirstMidName = "Roger",   LastName =
"Zheng",       HireDate = DateTime.Parse("2004-02-12"), BirthDate =
DateTime.Parse("1957-10-12") }
};
```

2. Build the solution, and then open the **Package Manager Console** window. Make sure that ContosoUniversity.DAL is still selected as the **Default project**.
3. In the **Package Manager Console** window, select **ContosoUniversity.DAL** as the **Default project**, and then enter the following command:

   ```
   add-migration AddBirthDate
   ```

   When this command finishes, Visual Studio opens the class file that defines the new `DbMIgration` class, and in the `Up` method you can see the code that creates the new column. The `Up` method creates the column when you are implementing the change, and the `Down` method deletes the column when you are rolling back the change.

   ```
   public partial class AddBirthDate : DbMigration
   {
       public override void Up()
       {
           AddColumn("dbo.Person", "BirthDate", c => c.DateTime());
       }

       public override void Down()
       {
           DropColumn("dbo.Person", "BirthDate");
       }
   }
   ```

4. Build the solution, and then enter the following command in the **Package Manager Console** window (make sure the ContosoUniversity.DAL project is still selected):

   ```
   update-database
   ```

   The Entity Framework runs the `Up` method and then runs the `Seed` method.

Display the new column in the Instructors page

1. In the ContosoUniversity project, open *Instructors.aspx* and add a new template field to display the birth date. Add it between the ones for hire date and office assignment:

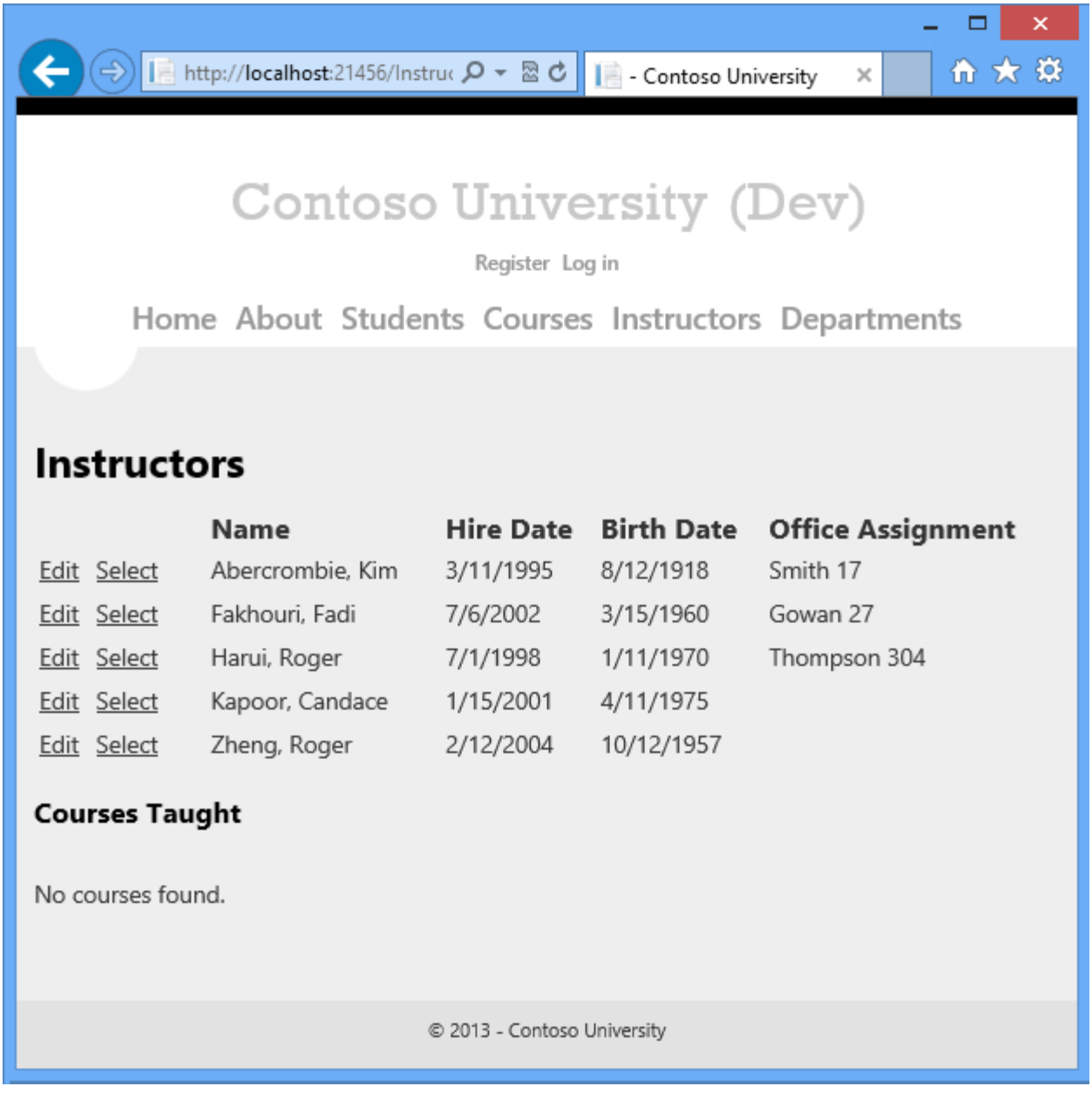
   ```
   <asp:TemplateField HeaderText="Hire Date" SortExpression="HireDate">
       <ItemTemplate>
           <asp:Label ID="InstructorHireDateLabel" runat="server"
   Text='<%# Eval("HireDate", "{0:d}") %>'></asp:Label>
   ```

```
    </ItemTemplate>
    <EditItemTemplate>
        <asp:TextBox ID="InstructorHireDateTextBox" runat="server"
Text='<%# Bind("HireDate", "{0:d}") %>' Width="7em"></asp:TextBox>
    </EditItemTemplate>
</asp:TemplateField>
<asp:TemplateField HeaderText="Birth Date" SortExpression="BirthDate">
    <ItemTemplate>
        <asp:Label ID="InstructorBirthDateLabel" runat="server"
Text='<%# Eval("BirthDate", "{0:d}") %>'></asp:Label>
    </ItemTemplate>
    <EditItemTemplate>
        <asp:TextBox ID="InstructorBirthDateTextBox" runat="server"
Text='<%# Bind("BirthDate", "{0:d}") %>'
            Width="7em"></asp:TextBox>
    </EditItemTemplate>
</asp:TemplateField>
<asp:TemplateField HeaderText="Office Assignment"
SortExpression="OfficeAssignment.Location">
    <ItemTemplate>
        <asp:Label ID="InstructorOfficeLabel" runat="server" Text='<%#
Eval("OfficeAssignment.Location") %>'></asp:Label>
    </ItemTemplate>
    <EditItemTemplate>
        <asp:TextBox ID="InstructorOfficeTextBox" runat="server"
            Text='<%# Eval("OfficeAssignment.Location") %>' Width="7em"
            OnInit="InstructorOfficeTextBox_Init"></asp:TextBox>
    </EditItemTemplate>
</asp:TemplateField>
```

(If code indentation gets out of sync, you can press CTRL-K and then CTRL-D to automatically reformat the file.)

2. Run the application and click the **Instructors** link.

   When the page loads, you see that it has the new birth date field.



3. Close the browser.

Deploy the database update

1. In **Solution Explorer** select the ContosoUniversity project.
2. In the **Web One Click Publish** toolbar, click the **Test** publish profile, and then click **Publish Web**. (If the toolbar is disabled, select the ContosoUniversity project in **Solution Explorer**.)

   Visual Studio deploys the updated application, and the browser opens to the home page.

3. Run the **Instructors** page to verify that the update was successfully deployed.

When the application tries to access the database for this page, Code First updates the database schema and runs the `Seed` method. When the page displays, you see the expected **Birth Date** column with dates in it.

4. In the **Web One Click Publish** toolbar, click the **Staging** publish profile, and then click **Publish Web**.
5. Run the **Instructors** page in staging to verify that the update was successfully deployed.
6. In the **Web One Click Publish** toolbar, click the **Production** publish profile, and then click **Publish Web**.
7. Run the **Instructors** page in production to verify that the update was successfully deployed.

   For a a real production application update that includes a database change you would also typically take the application offline during deployment by using *app_offline.htm*, as you saw in the previous tutorial.

Deploy a database update by using the dbDacFx provider

In this section, you add a *Comments* column to the *User* table in the membership database and create a page that lets you display and edit comments for each user. Then you deploy the changes to test, staging, and production.

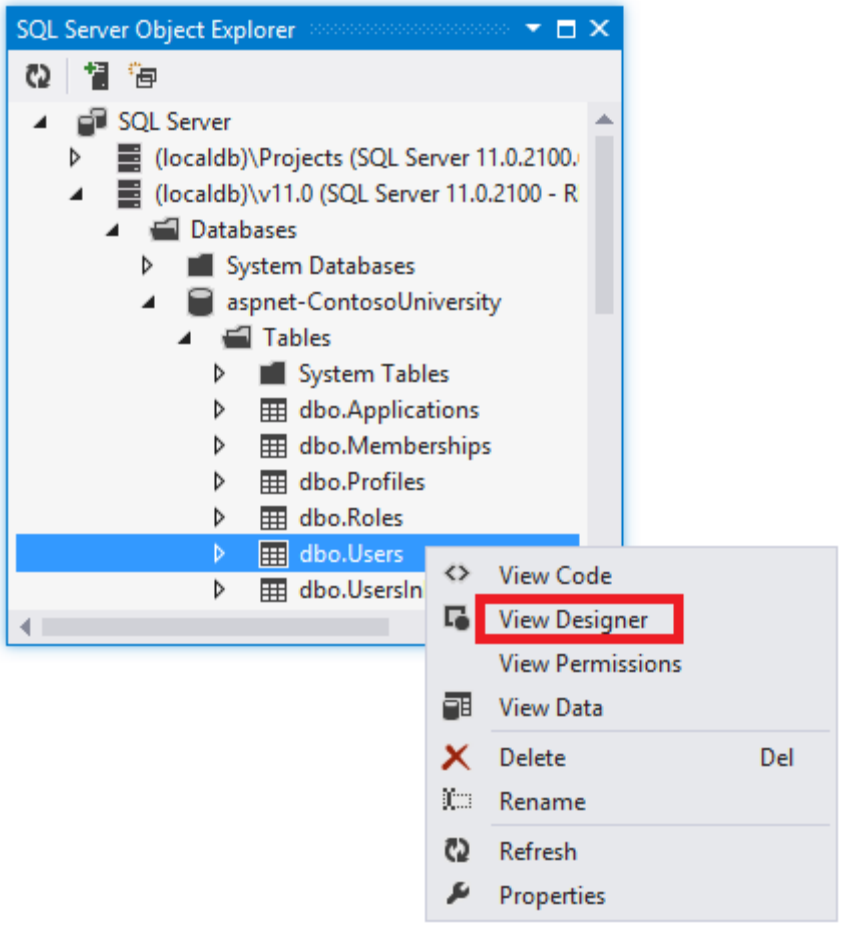
Add a column to a table in the membership database

1. In Visual Studio, open **SQL Server Object Explorer**.
2. Expand **(localdb)\v11.0**, expand **Databases**, expand **aspnet-ContosoUniversity** (not **aspnet-ContosoUniversity-Prod**) and then expand **Tables**.

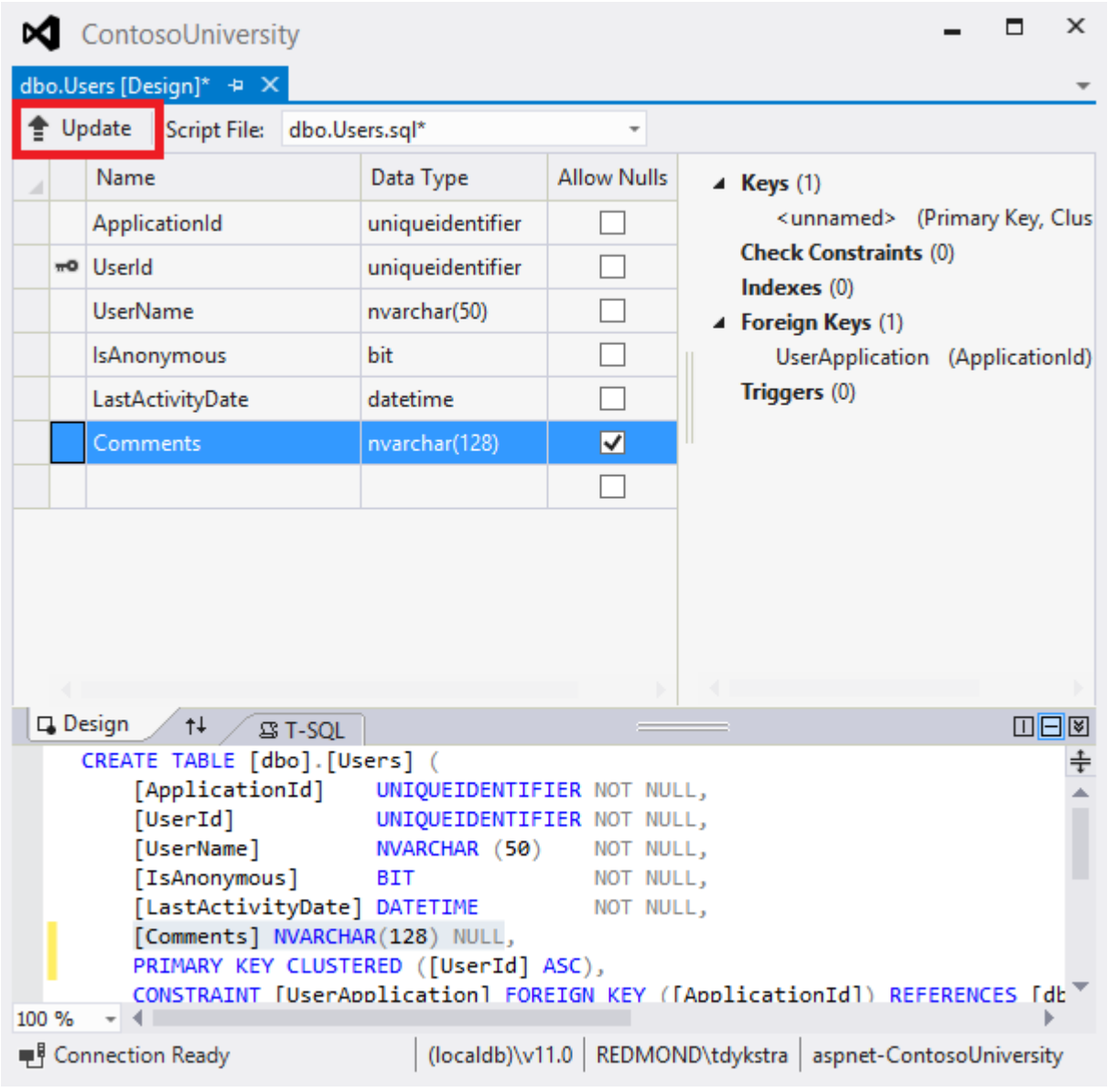
   If you don't see **(localdb)\v11.0** under the **SQL Server** node, right-click the **SQL Server** node and click **Add SQL Server**. In the **Connect to Server** dialog box enter *(localdb)\v11.0* as the **Server name**, and then click **Connect**.

   If you don't see **aspnet-ContosoUniversity**, run the project and log in using the *admin* credentials (password is *devpwd*), and then refresh the **SQL Server Object Explorer** window.

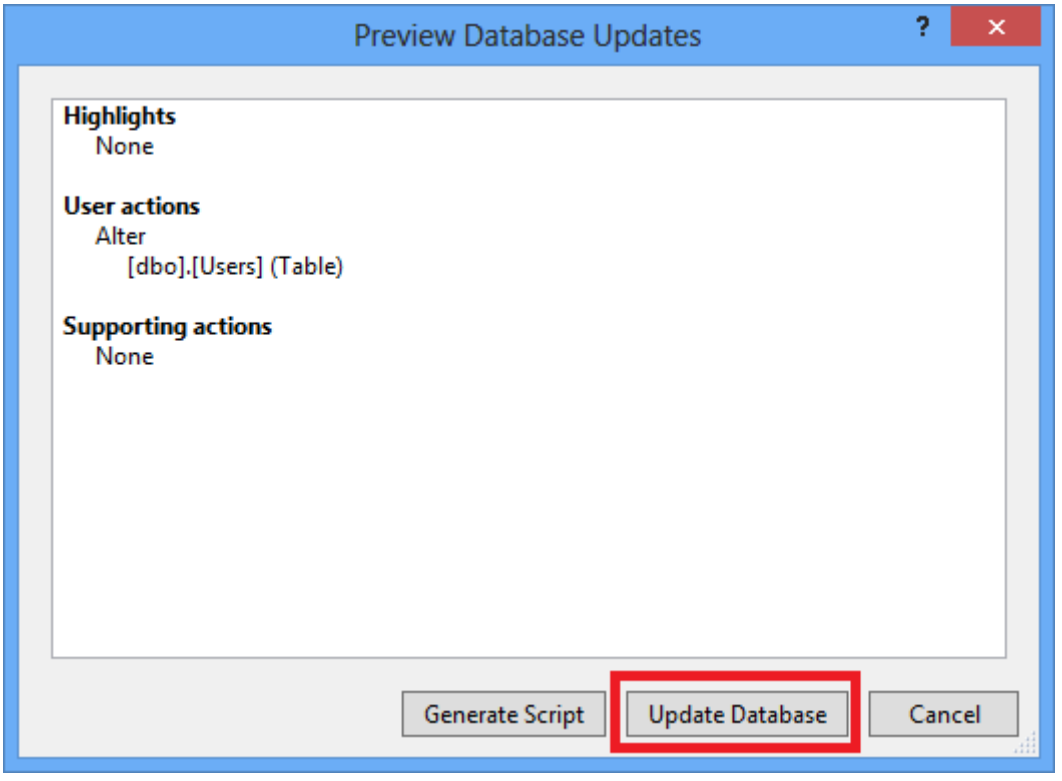
3. Right-click the **Users** table, and then click **View Designer**.



4. In the designer, add a *Comments* column and make it *nvarchar(128)* and nullable, and then click **Update**.



5. In the **Preview Database Updates** box, click **Update Database**.



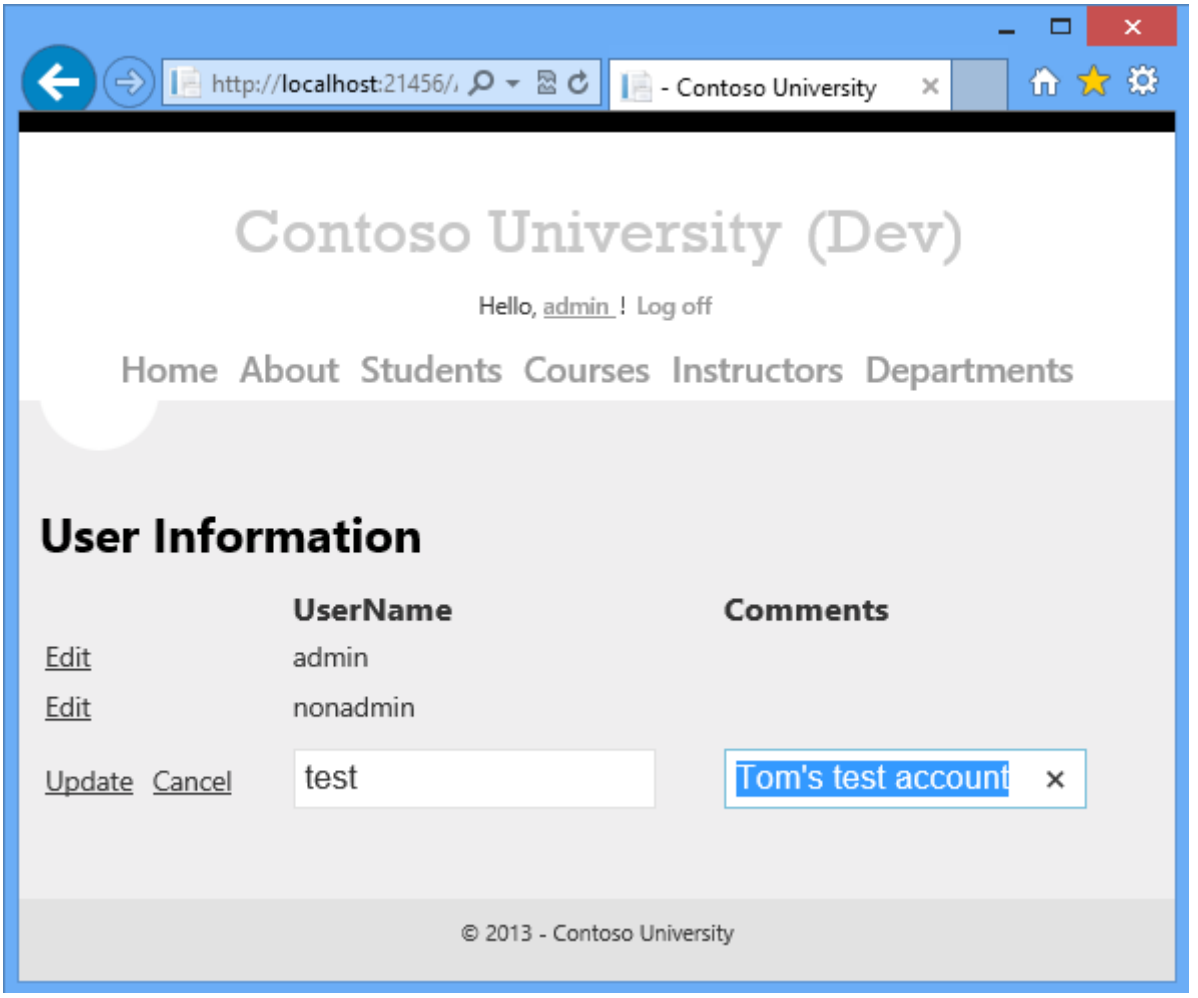
Create a page to display and edit the new column

1. In **Solution Explorer**, right-click the **Account** folder in the ContosoUniversity project, click **Add**, and then click **New Item**.
2. Create a new **Web Form Using Master Page** and name it *UserInfo.aspx*. Accept the default *Site.Master* file as the master page.
3. Copy the following markup into the `MainContent Content` element (the last of the 3 `Content` elements):

```
    <h2>User Information</h2>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString="<%$ ConnectionStrings:DefaultConnection %>"
        SelectCommand="SELECT UserId, UserName, Comments FROM [Users]"
        UpdateCommand="UPDATE [Users] SET [UserName] = @UserName,
[Comments] = @Comments WHERE [UserId] = @UserId">
        <DeleteParameters>
            <asp:Parameter Name="UserId" Type="Object" />
        </DeleteParameters>
        <UpdateParameters>
            <asp:Parameter Name="UserId" Type="Object" />
            <asp:Parameter Name="UserName" Type="String" />
            <asp:Parameter Name="Comments" Type="String" />
        </UpdateParameters>
    </asp:SqlDataSource>

    <asp:GridView ID="GridView1" runat="server"
AutoGenerateColumns="False" DataKeyNames="UserId"
DataSourceID="SqlDataSource1">
```

```
        <Columns>
            <asp:CommandField ShowEditButton="True" />
            <asp:BoundField DataField="UserName" HeaderText="UserName"
SortExpression="UserName" />
            <asp:BoundField DataField="Comments" HeaderText="Comments"
SortExpression="Comments" />
        </Columns>
    </asp:GridView>
```

4. Right-click the *UserInfo.aspx* page and click **View in Browser**.
5. Log in with your *admin* user credentials (password is *devpwd*) and add some comments to a user to verify that the page works correctly.



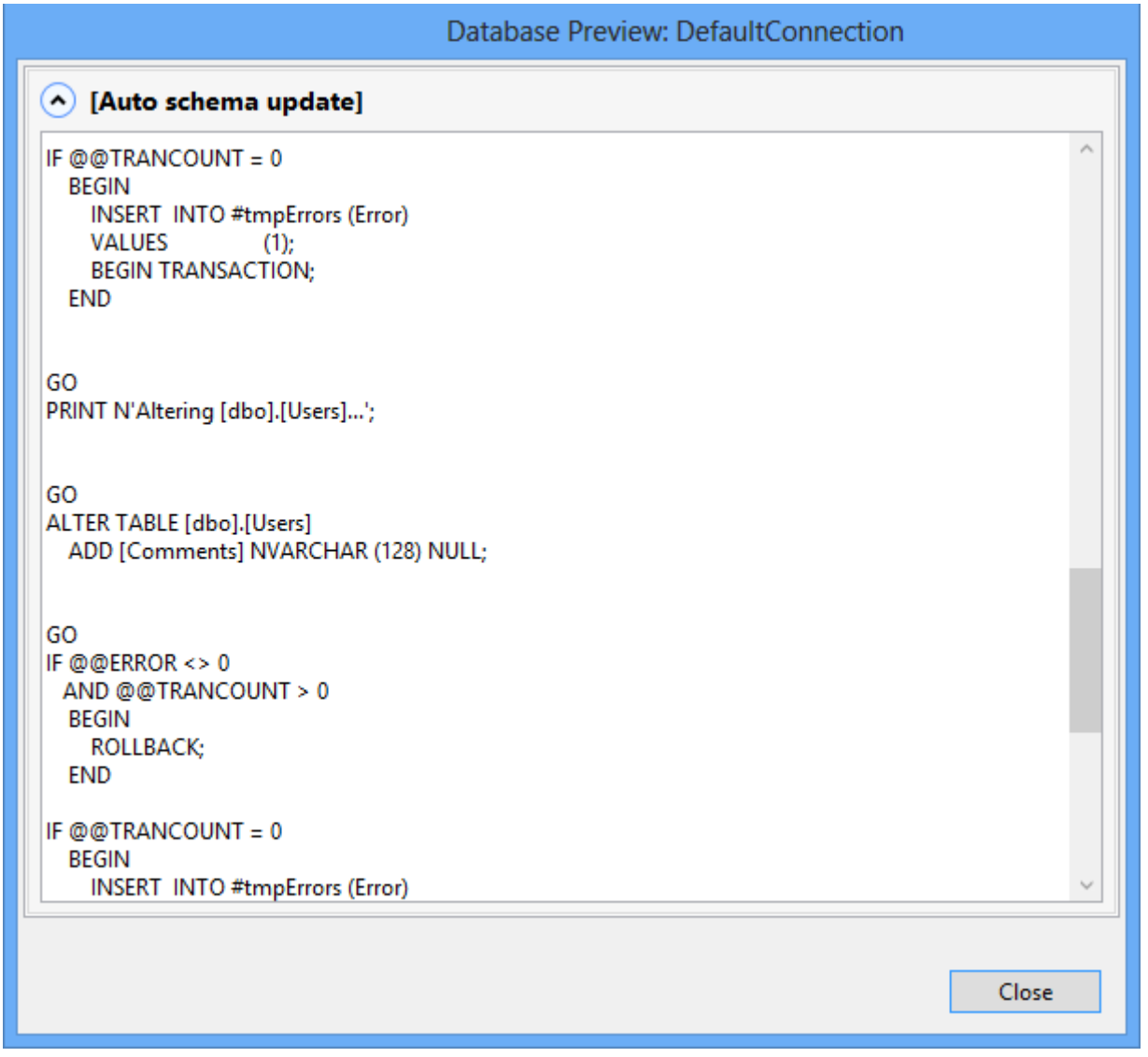
6. Close the browser.

Deploy the database update

To deploy by using the dbDacFx provider, you just need to select the **Update database** option in the publish profile. However, for the initial deployment when you used this option you also

configured some additional SQL scripts to run: those are still in the profile and you'll have to prevent them from running again.

1. Open the **Publish Web** wizard by right-clicking the ContosoUniversity project and clicking **Publish**.
2. Select the **Test** profile.
3. Click the **Settings** tab.
4. Under **DefaultConnection**, select **Update database**.
5. Disable the additional scripts that you configured to run for the initial deployment:
    1. Click **Configure database updates**.
    2. In the **Configure Database Updates** dialog box, clear the check boxes next to *Grant.sql* and *aspnet-data-dev.sql*.
    3. Click **Close**.
6. Click the **Preview** tab.
7. Under **Databases** and to the right of **DefaultConnection**, click the **Preview database** link.



The preview window shows the script that will be run in the destination database to make that database schema match the schema of the source database. The script includes an ALTER TABLE command that adds the new column.

8. Close the **Database Preview** dialog box, and then click **Publish**.

   Visual Studio deploys the updated application, and the browser opens to the home page.

9. Run the UserInfo page (add *Account/UserInfo.aspx* to the home page URL) to verify that the update was successfully deployed. You'll have to log in by entering *admin* and *devpwd*.

   Data in tables is not deployed by default, and you didn't configure a data deployment script to run, so you won't find the comment that you added in development. You can

add a new comment now in staging to verify that the change was deployed to the database and the page works correctly.

10. Follow the same procedure to deploy to staging and production.

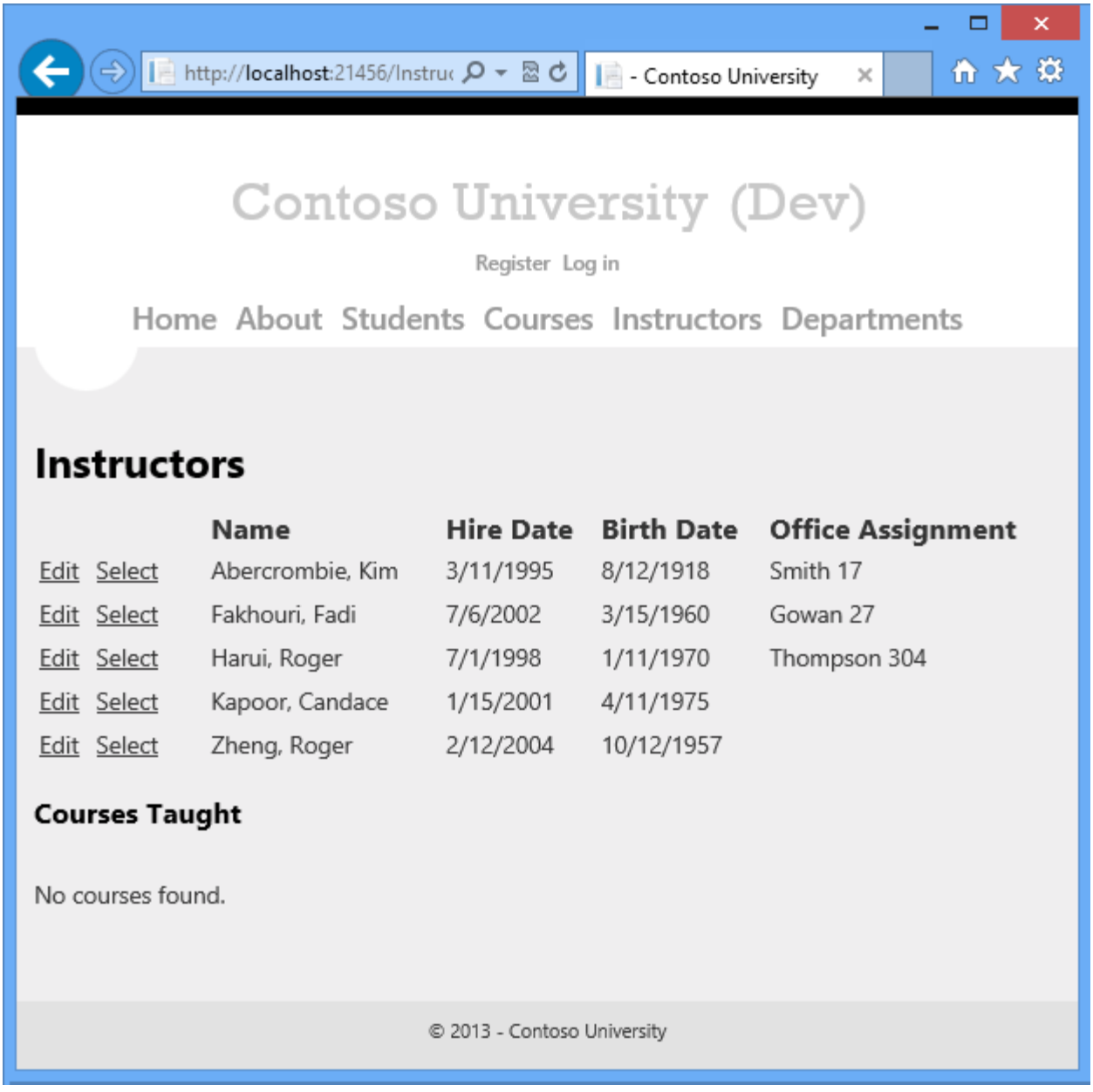
    Don't forget to disable the extra scripts. The only difference compared to the Test profile is that you will disable only one script in the Staging and Production profiles because they were configured to run only *aspnet-prod-data.sql*.

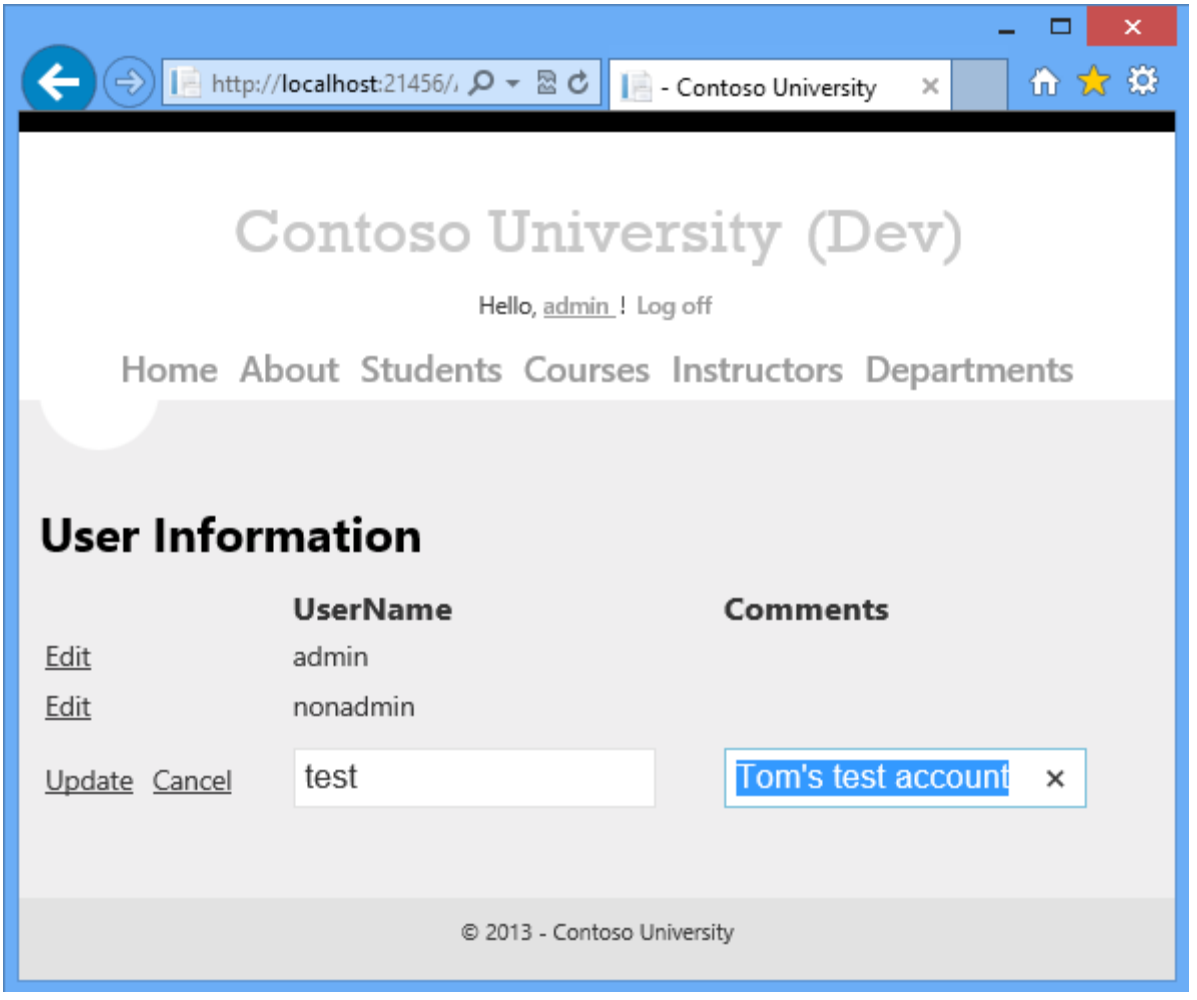
    The credentials for staging and production are admin and prodpwd.

    For a real production application update that includes a database change you would also typically take the application offline during deployment by uploading *app_offline.htm* before publishing and deleting it afterward, as you saw in the previous tutorial.

Summary

You've now deployed an application update that included a database change using both Code First Migrations and the dbDacFx provider.


http://localhost:21456/Instruc
- Contoso University
Contoso University (Dev)
Register Log in
Home About Students Courses Instructors Departments
Instructors
Name Hire Date Birth Date Office Assignment
Edit Select Abercrombie, Kim 3/11/1995 8/12/1918 Smith 17
Edit Select Fakhouri, Fadi 7/6/2002 3/15/1960 Gowan 27
Edit Select Harui, Roger 7/1/1998 1/11/1970 Thompson 304
Edit Select Kapoor, Candace 1/15/2001 4/11/1975
Edit Select Zheng, Roger 2/12/2004 10/12/1957
Courses Taught
No courses found.
© 2013 - Contoso University



The next tutorial shows you how to execute deployments by using the command line.

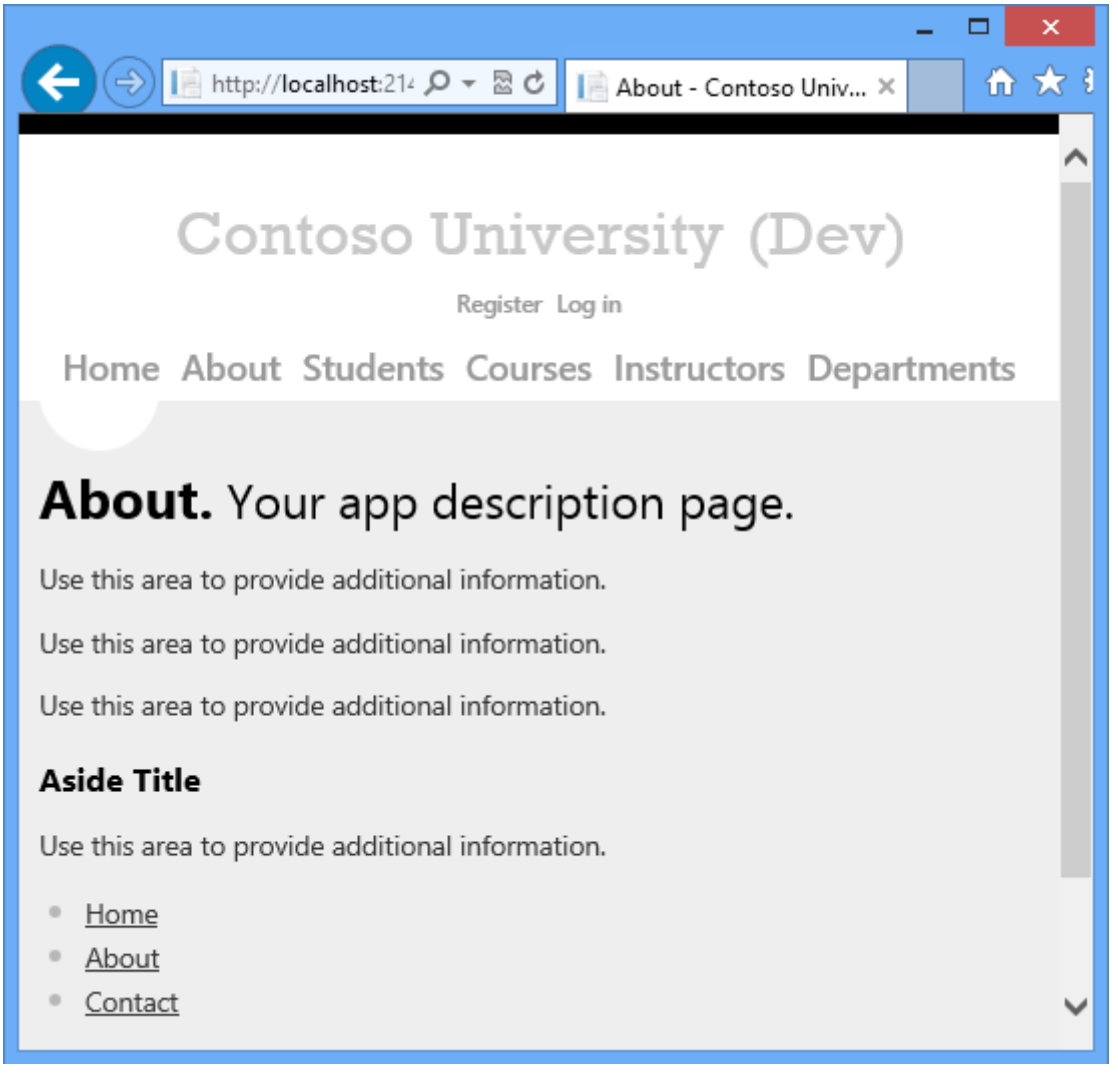
Command Line Deployment

Overview

This tutorial shows you how to invoke the Visual Studio web publish pipeline from the command line. This is useful for scenarios where you want to automate the deployment process instead of doing it manually in Visual Studio, typically by using a source code version control system.

Make a change to deploy

Currently the About page displays the template code.

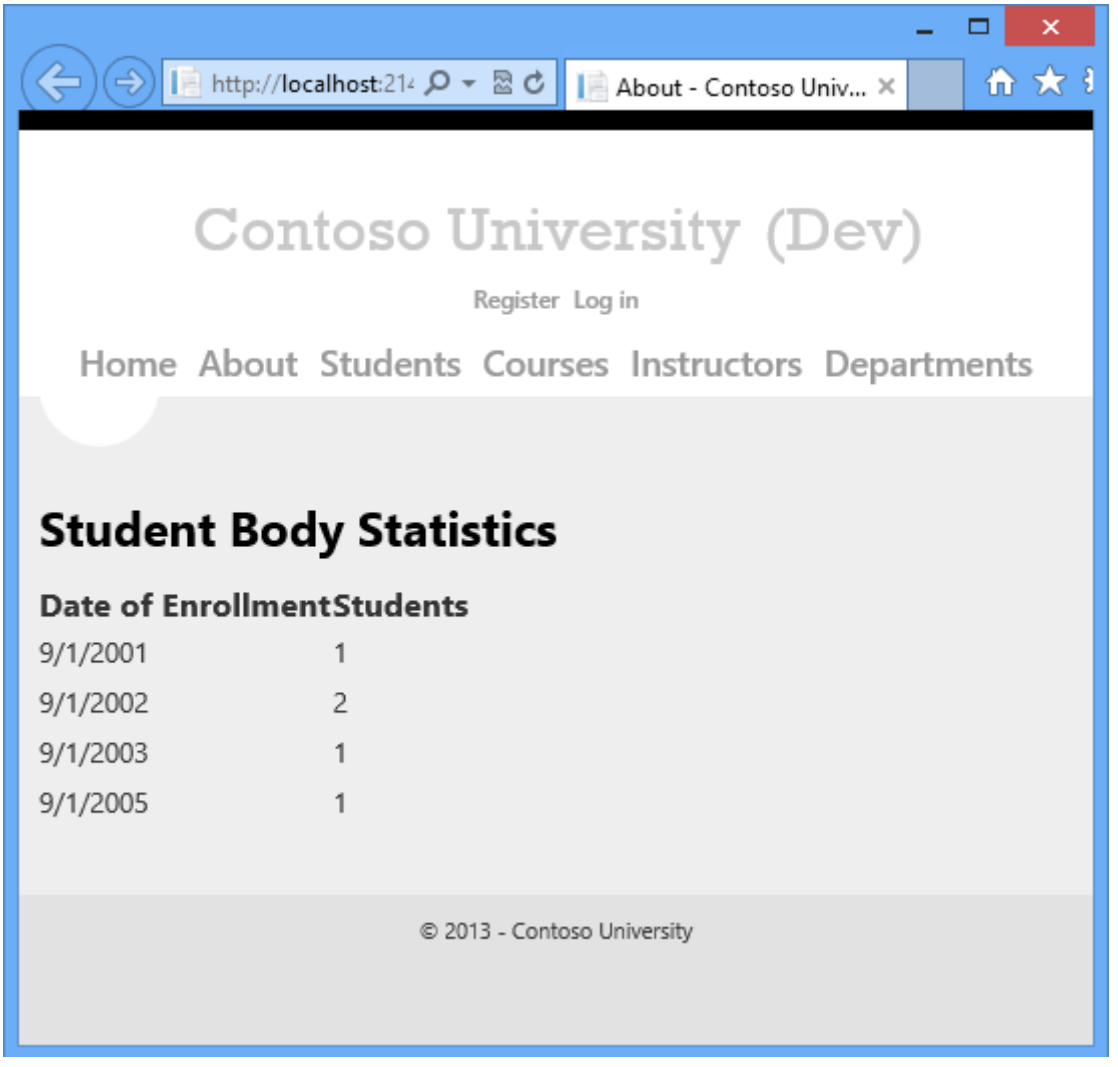


You'll replace that with code that displays a summary of student enrollment.

Open the *About.aspx* page, delete all of the markup inside the `MainContent Content` element, and insert the following markup in its place:

```
    <h2>Student Body Statistics</h2>
    <asp:ObjectDataSource ID="StudentStatisticsObjectDataSource"
runat="server" TypeName="ContosoUniversity.BLL.SchoolBL"
        SelectMethod="GetStudentStatistics"
DataObjectTypeName="ContosoUniversity.DAL.EnrollmentDateGroup">
    </asp:ObjectDataSource>
    <asp:GridView ID="StudentStatisticsGridView" runat="server"
AutoGenerateColumns="False"
        DataSourceID="StudentStatisticsObjectDataSource">
        <Columns>
            <asp:BoundField DataField="EnrollmentDate"
DataFormatString="{0:d}" HeaderText="Date of Enrollment"
                ReadOnly="True" SortExpression="EnrollmentDate" />
            <asp:BoundField DataField="StudentCount" HeaderText="Students"
ReadOnly="True"
                SortExpression="StudentCount" />
        </Columns>
    </asp:GridView>
```

Run the project and select the **About** page.



Deploy to Test by using the command line

You won't be deploying another database change, so disable dbDacFx database deployment for the aspnet-ContosoUniversity database. Open the **Publish Web** wizard, and in each of the three publish profiles, clear the **Update Database** check box on the **Settings** tab.

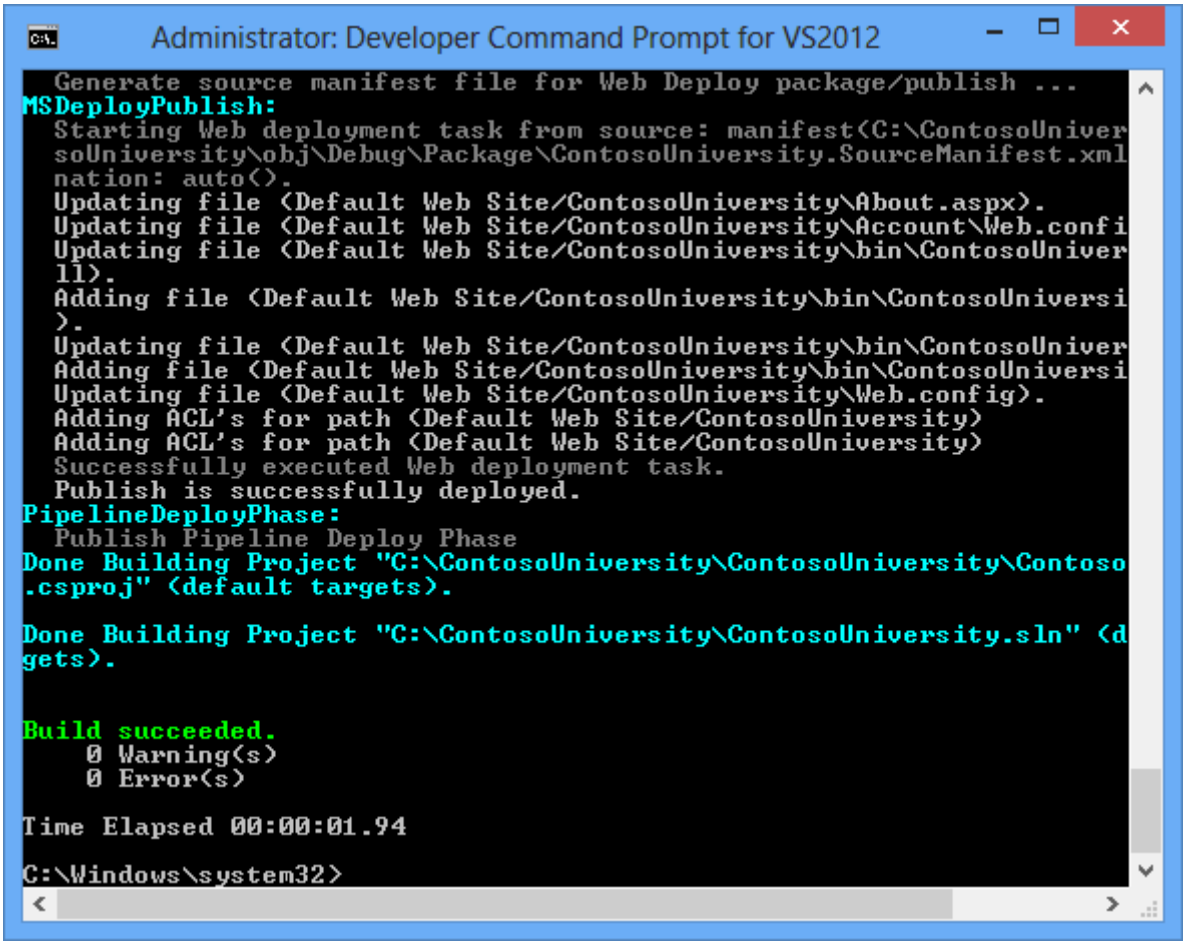
In the Windows 8 Start page, search for **Developer Command Prompt for VS2012**.

Right-click the icon for **Developer Command Prompt for VS2012** and click **Run as administrator**.

Enter the following command at the command prompt, replacing the path to the solution file with the path to your solution file:

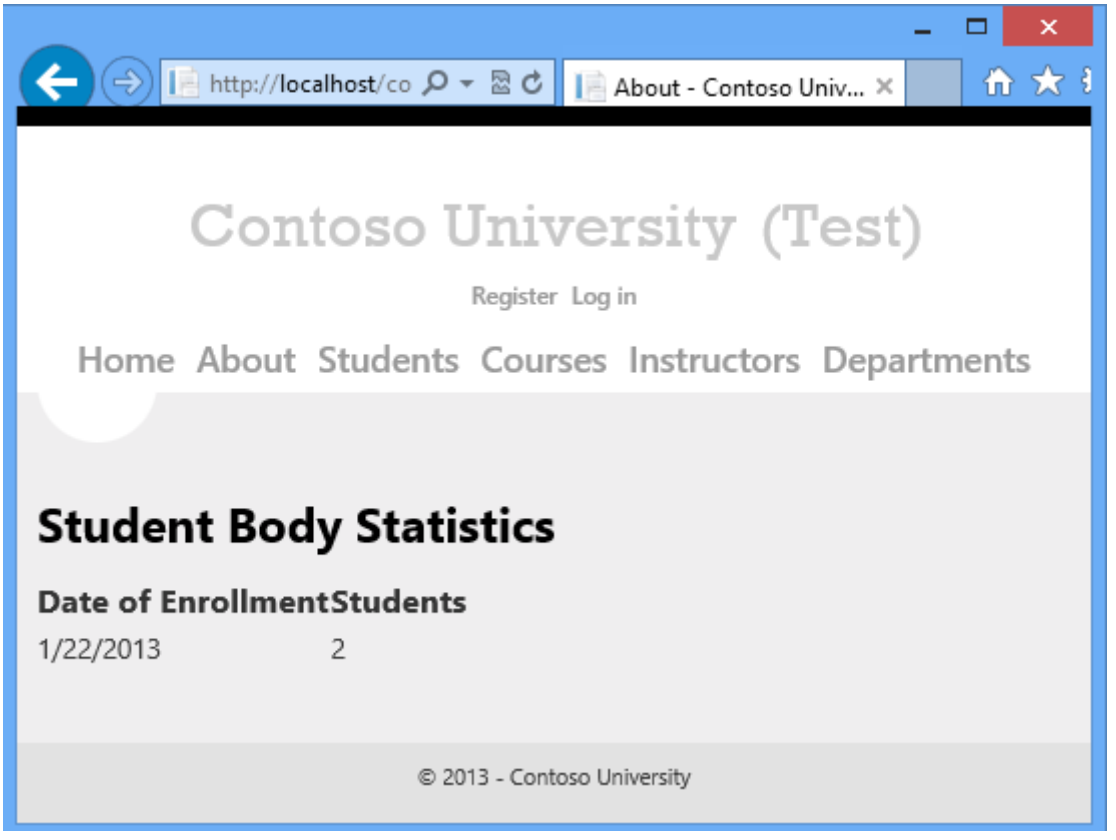
```
msbuild C:\ContosoUniversity\ContosoUniversity.sln /p:DeployOnBuild=true
/p:PublishProfile=Test
```

MSBuild builds the solution and deploys it to the test environment.



Open a browser and go to *http://localhost/ContosoUniversity*, then click the **About** page to verify that the deployment was successful.

If you haven't created any students in test, you'll see an empty page under the **Student Body Statistics** heading. Go to the **Students** page, click **Add Student**, and add some students, and then return to the **About** page to see student statistics.



Key command line options

The command that you entered passed the solution file path and two properties to MSBuild:

```
msbuild C:\ContosoUniversity\ContosoUniversity.sln /p:DeployOnBuild=true
/p:PublishProfile=Test
```

Deploying the solution versus deploying individual projects

Specifying the solution file causes all projects in the solution to be built. If you have multiple web projects in the solution, the following MSBuild behavior applies:

- The properties that you specify on the command line are passed to every project. Therefore, each web project must have a publish profile with the name that you specify. If you specify `/p:PublishProfile=Test`, each web project must have a publish profile named *Test*.
- You might successfully publish one project when another one doesn't even build. For more information, see the stackoverflow thread MSBuild fails with two packages.

If you specify an individual project instead of a solution, you have to add a parameter that specifies the Visual Studio version. If you are using Visual Studio 2012 the command line would be similar to the following example:

```
msbuild C:\ContosoUniversity\ContosoUniversity\ContosoUniversity.csproj
/p:DeployOnBuild=true /p:PublishProfile=Test /p:VisualStudioVersion=11.0
```

The version number for Visual Studio 2010 is 10.0. For more information, see Visual Studio project compatability and VisualStudioVersion on Sayed Hashimi's blog.

Specifying the publish profile

You can specify the publish profile by name or by the full path to the *.pubxml* file, as shown in the following example:

```
msbuild C:\ContosoUniversity\ContosoUniversity.sln /p:DeployOnBuild=true
/p:PublishProfile=C:\ContosoUniversity\ContosoUniversity\Properties\PublishPr
ofiles\Test.pubxml
```

Web publish methods supported for command-line publishing

Three publish methods are supported for command line publishing:

- `MSDeploy` - Publish by using Web Deploy.
- `Package` - Publish by creating a Web Deploy Package. You have to install the package separately from the MSBuild command that creates it.
- `FileSystem` - Publish by copying files to a specified folder.

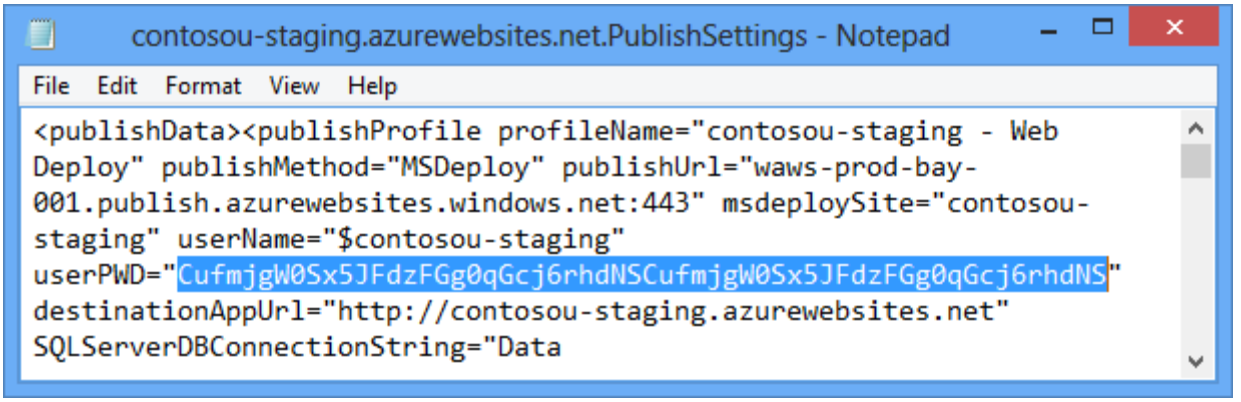
Specifying the build configuration and platform

The build configuration and platform must be set in Visual Studio or on the command line. The publish profiles include properties that are named `LastUsedBuildConfiguration` and `LastUsedPlatform`, but you can't set these properties in order to determine how the project is built. For more information, see MSBuild: how to set the configuration property on Sayed Hashimi's blog.

Deploy to staging

To deploy to Windows Azure you have to add the password to the command line. If you saved the password in the publish profile in Visual Studio, it was stored in encrypted form in the your *.pubxml.user* file. That file is not accessed by MSBuild when you do a command line deployment, so you have to pass in the password in a command line parameter.

1. Copy the password that you need from the *.publishsettings* file that you downloaded earlier for the staging web site. The password is the value of the `userPWD` attribute for the Web Deploy `publishProfile` element.



2. In the Windows 8 Start page, search for **Developer Command Prompt for VS2012**, and click the icon to open the command prompt. (You don't have to open it as administrator this time because you aren't deploying to IIS on the local computer.)
3. Enter the following command at the command prompt, replacing the path to the solution file with the path to your solution file and the password with your password:

   ```
   msbuild C:\ContosoUniversity\ContosoUniversity.sln
   /p:DeployOnBuild=true /p:PublishProfile=Staging
   /p:Password=hdNSWsbuqno7J5uqnwKafwlfNPt1DSco7J5uqnwKafwlfNPt1DSpKHuYgCc
   o7J5 /p:AllowUntrustedCertificate=true
   ```

   Notice that this command line includes an extra parameter: `/p:AllowUntrustedCertificate=true`. As this tutorial is being written, the `AllowUntrustedCertificate` property must be set when you publish to Windows Azure from the command line. When the fix for this bug is released, you won't need that parameter.

4. Open a browser and go to the URL of your staging site, and then click the **About** page to verify that the deployment was successful.

   As you saw earlier for the test environment, you might have to create some students to see statistics on the **About** page.

Deploy to production

The process for deploying to production is similar to the process for staging.

1. Copy the password that you need from the *.publishsettings* file that you downloaded earlier for the production web site.
2. Open **Developer Command Prompt for VS2012**.
3. Enter the following command at the command prompt, replacing the path to the solution file with the path to your solution file and the password with your password:

   ```
   msbuild C:\ContosoUniversity\ContosoUniversity.sln
   /p:DeployOnBuild=true /p:PublishProfile=Production
   ```

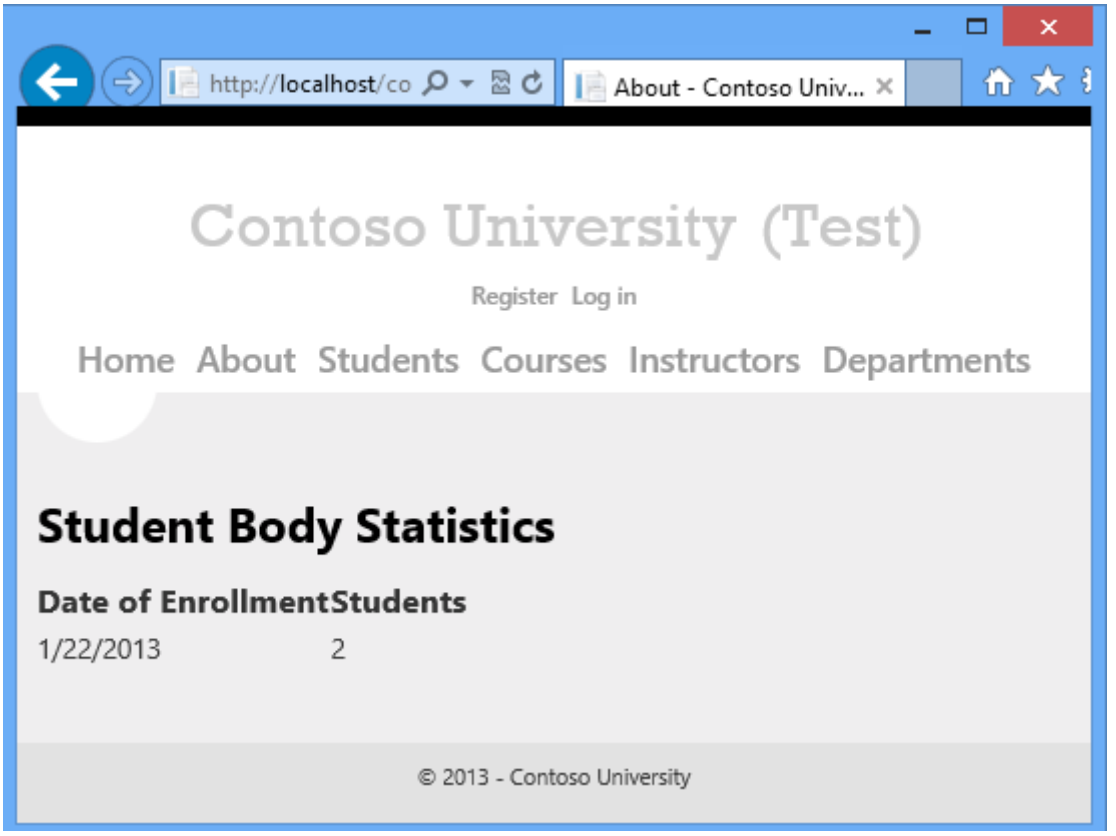
```
/p:Password=hdNSWsbuqnwKafwlo7J5uqnwKafwlfNPt1DSqnwKafwlfNPt1DSpKHuYgCc
o7J5 /p:AllowUntrustedCertificate=true
```

For a real production site, if there was also a database change, you would typically copy the *app_offline.htm* file to the site before deployment and delete it after successful deployment.

4. Open a browser and go to the URL of your staging site, and then click the **About** page to verify that the deployment was successful.

Summary

You have now deployed an application update by using the command line.



In the next tutorial, you will see an example of how to extend the web publish pipeline. The example will show you how to deploy files that are not included in the project.

Deploying Extra Files

Overview

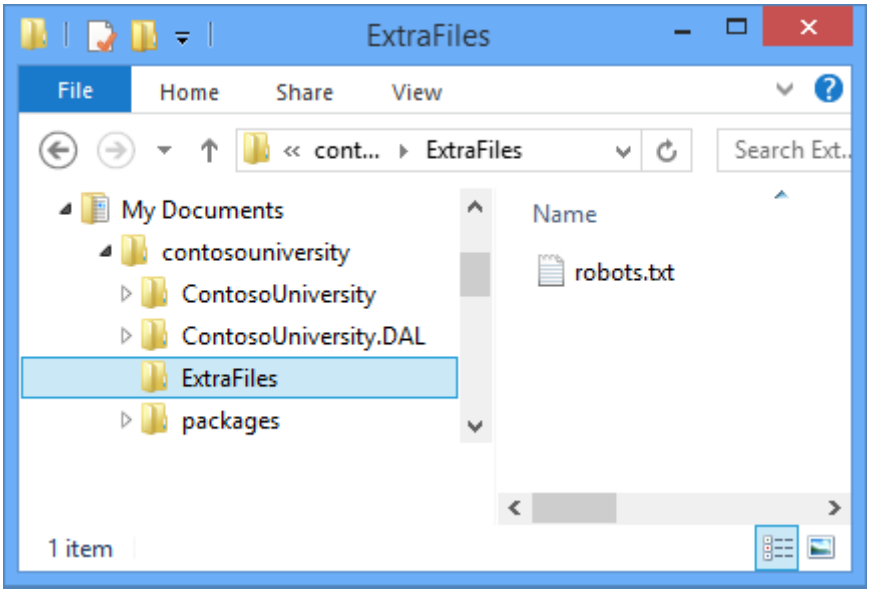
This tutorial shows how to extend the Visual Studio web publish pipeline to do an additional task during deployment. The task is to copy extra files that are not in the project folder to the destination web site.

For this tutorial you'll copy one extra file: *robots.txt*. You want to deploy this file to staging but not to production. In the Deploying to Production tutorial, you added this file to the project and configured the Production publish profile to exclude it. In this tutorial you'll see an alternative method to handle this situation, one that will be useful for any files that you want to deploy but don't want to include in the project.

Move the robots.txt file

To prepare for a different method of handling *robots.txt*, in this section of the tutorial you move the file to a folder that is not included in the project, and you delete *robots.txt* from the staging environment. It is necessary to delete the file from staging so that you can verify that your new method of deploying the file to that environment is working correctly.

1. In **Solution Explorer**, right-click the *robots.txt* file and click **Exclude From Project**.
2. Using Windows File Explorer, create a new folder in the solution folder and name it *ExtraFiles*.
3. Move the *robots.txt* file from the *ContosoUniversity* project folder to the *ExtraFiles* folder.



4. Using your FTP tool, delete the *robots.txt* file from the staging web site.

   As an alternative, you can select **Remove additional files at destination** under **File Publish Options** on the **Settings** tab of the Staging publish profile, and republish to staging.

Update the publish profile file

You only need *robots.txt* in staging, so the only publish profile you need to update in order to deploy it is Staging.

1. In Visual Studio, open *Staging.pubxml*.
2. At the end of the file, before the closing `</Project>` tag, add the following markup:

```
  <Target Name="CustomCollectFiles">
    <ItemGroup>
      <_CustomFiles Include="..\ExtraFiles\**\*" />
      <FilesForPackagingFromProject Include="%(_CustomFiles.Identity)">
        <DestinationRelativePath>%(RecursiveDir)%(Filename)%(Extension)
</DestinationRelativePath>
      </FilesForPackagingFromProject>
    </ItemGroup>
  </Target>
```

   This code creates a new *target* that will collect additional files to be deployed. A target is composed of one or more tasks that MSBuild will execute based on conditions you specify.

   The `Include` attribute specifies that the folder in which to find the files is *ExtraFiles*, located at the same level as the project folder. MSBuild will collect all files from that folder and recursively from any subfolders (the double asterisk specifies recursive subfolders). With this code you could put multiple files, and files in subfolders inside the *ExtraFiles* folder, and all will be deployed.

   The `DestinationRelativePath` element specifies that the folders and files should be copied to the root folder of the destination web site, in the same file and folder structure as they are found in the *ExtraFiles* folder. If you wanted to copy the *ExtraFiles* folder itself, the `DestinationRelativePath` value would be *ExtraFiles\%(RecursiveDir)%(Filename)%(Extension)*.

3. At the end of the file, before the closing `</Project>` tag, add the following markup that specifies when to execute the new target.

```
<PropertyGroup>
    <CopyAllFilesToSingleFolderForPackageDependsOn>
      CustomCollectFiles;
      $(CopyAllFilesToSingleFolderForPackageDependsOn);
    </CopyAllFilesToSingleFolderForPackageDependsOn>
```

```
    <CopyAllFilesToSingleFolderForMsdeployDependsOn>
      CustomCollectFiles;
      $(CopyAllFilesToSingleFolderForPackageDependsOn);
    </CopyAllFilesToSingleFolderForMsdeployDependsOn>
</PropertyGroup>
```

This code causes the new `CustomCollectFiles` target to be executed whenever the target that copies files to the destination folder is executed. There is a separate target for publish versus deployment package creation, and the new target is injected in both targets in case you decide to deploy by using a deployment package instead of publishing.

The *.pubxml* file now looks like the following example:

```
<?xml version="1.0" encoding="utf-8"?>
<!--
This file is used by the publish/package process of your Web project.
You can customize the behavior of this process
by editing this MSBuild file. In order to learn more about this please
visit http://go.microsoft.com/fwlink/?LinkID=208121.
-->
<Project ToolsVersion="4.0"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <WebPublishMethod>MSDeploy</WebPublishMethod>
    <LastUsedBuildConfiguration>Release</LastUsedBuildConfiguration>
    <LastUsedPlatform>Any CPU</LastUsedPlatform>
    <SiteUrlToLaunchAfterPublish>http://contosou-
staging.azurewebsites.net</SiteUrlToLaunchAfterPublish>
    <ExcludeApp_Data>True</ExcludeApp_Data>
    <MSDeployServiceURL>waws-prod-bay-
001.publish.azurewebsites.windows.net:443</MSDeployServiceURL>
    <DeployIisAppPath>contosou-staging</DeployIisAppPath>
    <RemoteSitePhysicalPath />
    <SkipExtraFilesOnServer>False</SkipExtraFilesOnServer>
    <MSDeployPublishMethod>WMSVC</MSDeployPublishMethod>
    <UserName>$contosou-staging</UserName>
    <_SavePWD>True</_SavePWD>
    <PublishDatabaseSettings>
      <Objects xmlns="">
        <ObjectGroup Name="SchoolContext" Order="1" Enabled="True">
          <Destination Path="Data
Source=tcp:sk0264hvc9.database.windows.net,1433;Initial
Catalog=ContosoUniversity-staging;User ID=CU-staging-
admin@sk0264hvc9;Password=Pas$w0rd" Name="Data
Source=tcp:sk0264hvc9.database.windows.net,1433;Initial
Catalog=ContosoUniversity-staging;User Id=CU-staging-
admin@sk0264hvc9;Password=Pas$w0rd" />
          <Object Type="DbCodeFirst">
            <Source Path="DBMigration"
DbContext="ContosoUniversity.DAL.SchoolContext, ContosoUniversity.DAL"
MigrationConfiguration="ContosoUniversity.DAL.Migrations.Configuration,
ContosoUniversity.DAL" Origin="Configuration" />
          </Object>
        </ObjectGroup>
```

```
        <ObjectGroup Name="DefaultConnection" Order="2"
Enabled="False">
          <Destination Path="Data
Source=tcp:sk0264hvc9.database.windows.net,1433;Initial
Catalog=ContosoUniversity-staging;User ID=CU-staging-
admin@sk0264hvc9;Password=Pas$w0rd" Name="Data
Source=tcp:sk0264hvc9.database.windows.net,1433;Initial
Catalog=ContosoUniversity-staging;User Id=CU-staging-
admin@sk0264hvc9;Password=Pas$w0rd" />
          <Object Type="DbDacFx">
            <PreSource Path="Data
Source=(LocalDb)\v11.0;AttachDbFilename=|DataDirectory|\aspnet-
ContosoUniversity.mdf;Initial Catalog=aspnet-
ContosoUniversity;Integrated Security=True" includeData="False" />
            <Source
Path="$(IntermediateOutputPath)AutoScripts\DefaultConnection_Incrementa
lSchemaOnly.dacpac" dacpacAction="Deploy" />
          </Object>
          <UpdateFrom Type="Web.Config">
            <Source MatchValue="Data Source=(LocalDb)\v11.0;Integrated
Security=SSPI;Initial Catalog=aspnet-
ContosoUniversity;AttachDBFilename=|DataDirectory|\aspnet-
ContosoUniversity.mdf"
MatchAttributes="$(UpdateFromConnectionStringAttributes)" />
          </UpdateFrom>
          <Object Type="DbFullSql" Enabled="False">
            <Source Path="..\aspnet-data-prod.sql" Transacted="False"
/>
          </Object>
        </ObjectGroup>
      </Objects>
    </PublishDatabaseSettings>
    <EnableMSDeployBackup>False</EnableMSDeployBackup>
  </PropertyGroup>
  <ItemGroup>
    <MSDeployParameterValue
Include="$(DeployParameterPrefix)DefaultConnection-Web.config
Connection String">
      <ParameterValue>Data
Source=tcp:sk0264hvc9.database.windows.net,1433;Initial
Catalog=ContosoUniversity-staging;User Id=CU-staging-
admin@sk0264hvc9;Password=Pas$w0rd</ParameterValue>
    </MSDeployParameterValue>
    <MSDeployParameterValue
Include="$(DeployParameterPrefix)SchoolContext-Web.config Connection
String">
      <ParameterValue>Data
Source=tcp:sk0264hvc9.database.windows.net,1433;Initial
Catalog=ContosoUniversity-staging;User Id=CU-staging-
admin@sk0264hvc9;Password=Pas$w0rd</ParameterValue>
    </MSDeployParameterValue>
  </ItemGroup>
  <Target Name="CustomCollectFiles">
    <ItemGroup>
      <_CustomFiles Include="..\ExtraFiles\**\*" />
      <FilesForPackagingFromProject Include="%(_CustomFiles.Identity)">
        <DestinationRelativePath>%(RecursiveDir)%(Filename)%(Extension)
```

```
</DestinationRelativePath>
      </FilesForPackagingFromProject>
    </ItemGroup>
  </Target>
  <PropertyGroup>
    <CopyAllFilesToSingleFolderForPackageDependsOn>
      CustomCollectFiles;
      $(CopyAllFilesToSingleFolderForPackageDependsOn);
    </CopyAllFilesToSingleFolderForPackageDependsOn>

    <CopyAllFilesToSingleFolderForMsdeployDependsOn>
      CustomCollectFiles;
      $(CopyAllFilesToSingleFolderForPackageDependsOn);
    </CopyAllFilesToSingleFolderForMsdeployDependsOn>
  </PropertyGroup>
</Project>
```

4. Save and close the *Staging.pubxml* file.

Publish to staging

Using one-click publish or the command line, publish the application by using the Staging profile.

If you use one-click publish, you can verify in the **Preview** window that *robots.txt* will be copied. Otherwise, use your FTP tool to verify that the *robots.txt* file is in the root folder of the web site after deployment.

Summary

This completes this series of tutorials on deploying an ASP.NET web application to a third-party hosting provider. For more information about any of the topics covered in these tutorials, see the ASP.NET Deployment Content Map.

More information

If you know how to work with MSBuild files, you can automate many other deployment tasks by writing code in *.pubxml* files (for profile-specific tasks) or the project *.wpp.targets* file (for tasks that apply to all profiles). For more information about *.pubxml* and *.wpp.targets* files, see How to: Edit Deployment Settings in Publish Profile (.pubxml) Files and the .wpp.targets File in Visual Studio Web Projects . For a basic introduction to MSBuild code, see **The Anatomy of a Project File** in Enterprise Deployment Series: Understanding the Project File. To learn how to work with MSBuild files to perform tasks for your own scenarios, see this book: Inside the Microsoft Build Engine: Using MSBuild and Team Foundation Build by Sayed Ibraham Hashimi and William Bartholomew.

Acknowledgements

I would like to thank the following people who made significant contributions to the content of this tutorial series:

- Alberto Poblacion, MVP & MCT, Spain
- Jarod Ferguson, Data Platform Development MVP, United States
- Harsh Mittal, Microsoft
- Kristina Olson, Microsoft
- Mike Pope, Microsoft
- Mohit Srivastava, Microsoft
- Raffaele Rialdi, Italy
- Rick Anderson, Microsoft
- Sayed Hashimi, Microsoft (twitter: @sayedihashimi)
- Scott Hanselman (twitter: @shanselman)
- Scott Hunter, Microsoft (twitter: @coolcsh)
- Srđan Božović, Serbia
- Vishal Joshi, Microsoft (twitter: @vishalrjoshi)

Troubleshooting

This page describes some common problems that may arise when you deploy an ASP.NET web application by using Visual Studio. For each one, one or more possible causes and corresponding solutions are provided.

Server Error in '/' Application - Current Custom Error Settings Prevent Details of the Error from Being Viewed Remotely

Scenario

After deploying a site to a remote host, you get an error message that mentions the customErrors setting in the Web.config file but doesn't indicate what the actual cause of the error was:

```
Server Error in '/' Application.
Runtime Error

Description: An application error occurred on the server. The current custom
error settings
for this application prevent the details of the application error from being
viewed remotely
(for security reasons). It could, however, be viewed by browsers running on
the local server
machine.

Details: To enable the details of this specific error message to be viewable
on remote machines,
please create a <customErrors> tag within a "web.config" configuration file
located in the
root directory of the current web application. This <customErrors> tag should
then have its
"mode" attribute set to "Off".
```

Possible Cause and Solution

By default, ASP.NET shows detailed error information only when your web application is running on the local computer. Generally you don't want to display detailed error information when your web application is publicly available over the Internet, because hackers may be able to use this information to find vulnerabilities in the application. However, when you are deploying a site or updates to a site, sometimes something will go wrong and you need to get the actual error message.

To enable the application to display detailed error messages when it runs on the remote host, edit the Web.config file to set `customErrors` mode off, redeploy the application, and run the application again:

1. If the application Web.config file has a `customErrors` element in the `system.web` element, change the `mode` attribute to "off". Otherwise add a `customErrors` element in the `system.web` element with the `mode` attribute set to "off", as shown in the following example:

```
<configuration>
  <system.web>
    <customErrors mode="off"/>
  </system.web>
</configuration>
```

2. Deploy the application.
3. Run the application and repeat whatever you did earlier that caused the error to occur. Now you can see what the actual error message is.
4. When you have resolved the error, restore the original `customErrors` setting and redeploy the application.

Cannot create/shadow copy 'ContosoUniversity' when that file already exists.

Scenario

When you try to run a project in Visual Studio you get an error page with a message like the following example:

Server Error in '/' Application. Cannot create/shadow copy 'ContosoUniversity' when that file already exists.

Possible Cause and Solution

Wait a minute and refresh the browser, or recompile the site and try running it again.

Access is Denied in a Web Page that Uses SQL Server Compact

Scenario

When you deploy a site that uses SQL Server Compact and you run a page in the deployed site that accesses the database, you see the following error message:

```
Access is denied. (Exception from HRESULT: 0x80070005 (E_ACCESSDENIED))
```

Possible Cause and Solution

The NETWORK SERVICE account on the server needs to be able to read SQL Service Compact native binaries that are in the *bin\amd64* or *bin\x86* folder, but it does not have read permissions for those folders. Set read permission for NETWORK SERVICE on the *bin* folder, making sure to extend the permissions to subfolders.

Cannot Read Configuration File Due to Insufficient Permissions

Scenario

When you click the Visual Studio publish button to deploy an application to IIS on your local machine, publishing fails and the **Output** window shows an error message similar to this:

```
An error occurred when reading the IIS Configuration File
'MACHINE/REDIRECTION'. The identity performing this operation was ... Error:
Cannot read configuration file due to insufficient permissions.
```

Possible Cause and Solution

To use one-click publish to IIS on your local machine, you must be running Visual Studio with administrator permissions. Close Visual Studio and restart it with administrator permissions.

Could Not Connect to the Destination Computer ... Using the Specified Process

Scenario

When you click the Visual Studio publish button to deploy an application, publishing fails and the **Output** window shows an error message similar to this:

```
Web deployment task failed.(Could not connect to the destination computer
("<server URL>") using the specified process
("The Web Management Service"). This can happen if a proxy server is
interrupting communication with the destination server.
Disable the proxy server and try again.) ... The remote server returned an
error: (502) Bad Gateway.
```

Possible Cause and Solution

A proxy server is interrupting communication with the destination server. From the Windows Control Panel or in Internet Explorer, select **Internet Options** and select the **Connections** tab. In the **Internet Properties** dialog box, click **LAN Settings**. In the **Local Area Network (LAN) Settings** dialog box, clear the **Automatically detect settings** checkbox. Then click the publish button again.

If the problem persists, contact your system administrator to determine what can be done with proxy or firewall settings. The problem happens because Web Deploy uses a non-standard port for Web Management Service deployment (8172); for other connections, Web Deploy uses port 80. When you are deploying to a third-party hosting provider, you are typically using the Web Management Service.

Default .NET 4.0 Application Pool Does Not Exist

Scenario

When you deploy an application that requires the .NET Framework 4, you see the following error message:

```
The default .NET 4.0 application pool does not exist or the application could
not be added. Please verify that ASP.NET 4.0 is installed on this machine.
```

Possible Cause and Solution

ASP.NET 4 is not installed in IIS. If the server you are deploying to is your development computer and has Visual Studio 2010 installed on it, ASP.NET 4 is installed on the computer but might not be installed in IIS. On the server that you are deploying to, open an elevated command prompt and install ASP.NET 4 in IIS by running the following commands:

```
cd %windir%\Microsoft.NET\Framework\v4.0.30319
aspnet_regiis.exe –iru
```

You might also need to manually set the .NET Framework version of the default application pool. For more information, see the Deploying to IIS as a Test Environment tutorial.

Format of the initialization string does not conform to specification starting at index 0.

Scenario

After you deploy an application using one-click publish, when you run a page that accesses the database you get the following error message:

```
Format of the initialization string does not conform to specification
starting at index 0.
```

Possible Cause and Solution

Open the *Web.config* file in the deployed site and check to see whether the connection string values begin with `$(ReplacableToken_`, as in the following example:

```
 <connectionStrings>
  <add name="DefaultConnection"
connectionString="$(ReplacableToken_DefaultConnection-Web.config Connection
String_0)" providerName="System.Data.SqlServerCe.4.0" />
  <add name="SchoolContext"
connectionString="$(ReplacableToken_SchoolContext-Web.config Connection
String_0)" providerName="System.Data.SqlServerCe.4.0" />
</connectionStrings>
```

If the connection strings look like this example, edit the project file and add the following property to the `PropertyGroup` element that is for all build configurations:

```
<AutoParameterizationWebConfigConnectionStrings>False</AutoParameterizationWe
bConfigConnectionStrings>
```

Then redeploy the application.

HTTP 500 Internal Server Error

Scenario

When you run the deployed site, you see the following error message without specific information indicating the cause of the error:

```
HTTP Error 500 - Internal Server Error.
```

Possible Cause and Solution

There are many causes of 500 errors, but one possible cause if you are following these tutorials is that you put an XML element in the wrong place in one of the Web.config transformation files. For example, you would get this error if you put the transformation that inserts a `<location>` element under `<system.web>` instead of directly under `<configuration>`. You can use the Web.config transform preview feature to verify that transformations are working as intended. The solution if you find a transform that was coded incorrectly is to correct the transformation file and redeploy. If an error isn't obvious, try commenting out transforms and redeploying to see which one is causing the 500 error.

HTTP 500.21 Internal Server Error

Scenario

When you run the deployed site, you see the following error message:

```
HTTP Error 500.21 - Internal Server Error. Handler "PageHandlerFactory-
Integrated" has a bad module "ManagedPipelineHandler" in its module list.
```

Possible Cause and Solution

The site you have deployed targets ASP.NET 4, but ASP.NET 4 is not registered in IIS on the server. On the server open an elevated command prompt and register ASP.NET 4 by running the following commands:

```
cd %windir%\Microsoft.NET\Framework\v4.0.30319
aspnet_regiis.exe -iru
```

You might also need to manually set the .NET Framework version of the default application pool. For more information, see the Deploying to IIS as a Test Environment tutorial.

Login Failed Opening SQL Server Express Database in App_Data

Scenario

You updated the *Web.config* file connection string to point to a SQL Server Express database as an *.mdf* file in your *App_Data* folder, and the first time you run the application you see the following error message:

```
System.Data.SqlClient.SqlException: Cannot open database "DatabaseName"
requested by the login. The login failed.
```

Possible Cause and Solution

The name of the *.mdf* file cannot match the name of any SQL Server Express database that has ever existed on your computer, even if you deleted the *.mdf* file of the previously existing database. Change the name of the *.mdf* file to a name that has never been used as a database name and change the *Web.config* file to use the new name. As an alternative, you can use SQL Server Management Studio Express to delete previously existing SQL Server Express databases.

Model Compatibility Cannot be Checked

Scenario

You updated the *Web.config* file connection string to point to a new SQL Server Express database, and the first time you run the application you see the following error message:

```
Model compatibility cannot be checked because the database does not contain
model metadata. Ensure that IncludeMetadataConvention has been added to the
DbModelBuilder conventions.
```

Possible Cause and Solution

If the database name you put in the Web.config file was ever used before on your computer, a database might already exist with some tables in it. Select a new name that has not been used on your computer before and change the *Web.config* file to point to use this new database name. As

an alternative, you can use SQL Server Express Utility or SQL Server Management Studio Express to delete the existing database.

SQL Error When a Script Attempts to Create Users or Roles

Scenario

You are using database deployment configured on the **Package/Publish SQL** tab, SQL scripts that run during deployment include Create User or Create Role commands, and script execution fails when those commands are executed. You might see more detailed messages, such as the following:

```
The approximate location of the error was between lines '1' and '3' of the
script.
The verbose log may have more information about the error. The command
started with:
CREATE USER [user2] FOR LOGIN [user2] WITH DEFAULT
Error: User does not have permission to perform this action.
```

If this error occurs when you have configured database deployment in the **Publish Web** wizard rather than the **Package/Publish SQL** tab, create a thread in the Configuration and Deployment forum, and the solution will be added to this troubleshooting page.

Possible Cause and Solution

The user account you are using to perform deployment does not have permission to create users or roles. For example, the hosting company might assign the `db_datareader`, `db_datawriter`, and `db_ddladmin` roles to the user account that it sets up for you. These are sufficient for creating most database objects, but not for creating users or roles. One way to avoid the error is by excluding users and roles from database deployment. You can do this by editing the `PreSource` element for the database's automatically generated script so that it includes the following attributes:

```
CopyAllUsers=false, CopyAllRoles=false
```

For information about how to edit the `PreSource` element in the project file, see How to: Edit Deployment Settings in the Project File. If the users or roles in your development database need to be in the destination database, contact your hosting provider for assistance.

SQL Server Timeout Error When Running Custom Scripts During Deployment

Scenario

You have specified custom SQL scripts to run during deployment, and when Web Deploy runs them, they time out.

Possible Cause and Solution

Running multiple scripts that have different transaction modes can cause time-out errors. By default, automatically generated scripts run in a transaction, but custom scripts do not. If you select the **Pull data and/or schema from an existing database** option on the **Package/Publish SQL** tab, and if you add a custom SQL script, you must change transaction settings on some scripts so that all scripts use the same transaction settings. For more information, see How to: Deploy a Database With a Web Application Project.

If you have configured transaction settings so that all are the same but still get this error, a possible workaround is to run the scripts separately. In the **Database Scripts** grid in the **Package/Publish** SQL tab, clear the **Include** check box for the script that causes the timeout error, then publish the project. Then go back into the **Database Scripts** grid, select that script's **Include** check box, and clear the **Include** check boxes for the other scripts. Then publish the project again. This time when you publish, only the selected custom script runs.

Stream Data of Site Manifest Is Not Yet Available

Scenario

When you are installing a package using the *deploy.cmd* file with the `t` (test) option, you see the following error message:

```
Error: The stream data of
'sitemanifest/dbFullSql[@path='C:\TEMP\AdventureWorksGrant.sql']/sqlScript'
is not yet available.
```

Possible Cause and Solution

The error message means that the command cannot produce a test report. However, the command might run if you use the `y` (actual installation) option. The message indicates only that there is a problem with running the command in test mode.

This Application Requires ManagedRuntimeVersion v4.0

Scenario

When you attempt to deploy, you see the following error message:

```
The application pool that you are trying to use has the
'managedRuntimeVersion' property set to 'v2.0'. This application requires
'v4.0'.
```

Possible Cause and Solution

ASP.NET 4 is not installed in IIS. If the server you are deploying to is your development computer and has Visual Studio 2010 installed on it, ASP.NET 4 is installed on the computer but might not be installed in IIS. On the server that you are deploying to, open an elevated command prompt and install ASP.NET 4 in IIS by running the following commands:

```
cd %windir%\Microsoft.NET\Framework\v4.0.30319
aspnet_regiis.exe –i
```

Unable to cast Microsoft.Web.Deployment.DeploymentProviderOptions

Scenario

When you are deploying a package, you see the following error message:

```
Unable to cast object of type
'Microsoft.Web.Deployment.DeploymentProviderOptions' to
'Microsoft.Web.Deployment.DeploymentProviderOptions'.
```

Possible Cause and Solution

You are trying to deploy from IIS Manager using the Web Deploy 1.1 UI to a server that has Web Deploy 2.0 installed. If you are using the IIS Remote Administration Tool to deploy by importing a package, check the **New Features Available** dialog box when you establish the connection. (This dialog box might only be shown once when the connection is first established. To clear the connection and start over, close IIS Manager and start it up again by entering `inetmgr /reset` at the command prompt.) If one of the features listed is **Web Deploy UI**, and it has a version number lower than 8, the server you are deploying to might have both 1.1 and 2.0 versions of Web Deploy installed. To deploy from a client that has 2.0 installed, the server must have only Web Deploy 2.0 installed. You will have to contact your hosting provider to resolve this problem.

Unable to load the native components of SQL Server Compact

Scenario

When you run the deployed site, you see the following error message:

```
Unable to load the native components of SQL Server Compact corresponding to
the ADO.NET provider of version 8482. Install the correct version of SQL
Server Compact. Refer to KB article 974247 for more details.
```

Possible Cause and Solution

The deployed site does not have *amd64* and *x86* subfolders with the native assemblies in them under the application's *bin* folder. On a computer that has SQL Server Compact installed, the native assemblies are located in *C:\Program Files\Microsoft SQL Server Compact Edition\v4.0\Private*. The best way to get the correct files into the correct folders in a Visual Studio project is to install the NuGet SqlServerCompact package. Package installation adds a post-build script to copy the native assemblies into *amd64* and *x86*. In order for these to be deployed, however, you have to manually include them in the project. For more information, see the Deploying SQL Server Compact tutorial.

"Path is not valid" error after deploying an Entity Framework Code First application

Scenario

You deploy an application that uses Entity Framework Code First Migrations and a DBMS such as SQL Server Compact which stores its database in a file in the App_Data folder. You have Code First Migrations configured to create the database after your first deployment. When you run the application you get an error message like the following example:

```
The path is not valid. Check the directory for the database. [Path =
c:\inetpub\wwwroot\App_Data\DatabaseName.sdf ]
```

Possible Cause and Solution

Code First is attempting to create the database but the App_Data folder does not exist. Either you didn't have any files in the *App_Data* folder when you deployed, or you selected **Exclude App_Data** on the **Package/Publish Web** tab of the **Project Properties** window. The deployment process won't create a folder on the server if there are no files in the folder to be copied to the server. If you already had the database set up in the site, the deployment process will delete the files and the *App_Data* folder itself if you selected **Remove additional files at destination** in the publish profile. To solve the problem, put a placeholder file such as a .txt file in the *App_Data* folder, make sure you do not have **Exclude App_Data** selected, and redeploy.

"COM object that has been separated from its underlying RCW cannot be used."

Scenario

You have been successfully using one-click publish to deploy your application and then you start getting this error:

```
Web deployment task failed. (Could not complete the request to remote agent
URL 'https://serverurl.com/msdeploy.axd?site=sitename'.)
Could not complete the request to remote agent URL
'https://url/msdeploy.axd?site=sitename'.
```

```
The request was aborted: The request was canceled.
COM object that has been separated from its underlying RCW cannot be used.
```

Possible Cause and Solution

Closing and restarting Visual Studio is usually all that is required to resolve this error.

Deployment Fails Because User Credentials Used for Publishing Don't Have setACL Authority

Scenario

Publishing fails with an error that indicates you don't have authority to set folder permissions (the user account you are using doesn't have setACL authority).

Possible Cause and Solution

By default, Visual Studio sets read permissions on the root folder of the site and write permissions on the App_Data folder. If you know that the default permissions on site folders are correct and do not need to be set, you disable this behavior by adding **<IncludeSetACLProviderOn Destination>False</IncludeSetACLProviderOnDestination>** to the publish profile file (to affect a single profile) or to the wpp.targets file (to affect all profiles). For information about how to edit these files, see How to: Edit Deployment Settings in Profile (.pubxml) Files.

Access Denied Errors when the Application Tries to Write to an Application Folder

Scenario

Your application errors when it tries to create or edit a file in one of the application folders, because it does not have write authority for that folder.

Possible Cause and Solution

By default, Visual Studio sets read permissions on the root folder of the site and write permissions on the App_Data folder. If your application needs write access to a sub-folder, you can set permissions for that folder as shown in the Setting Folder Permissions and Deploying to the Production Environment tutorials. If your application needs write access to the root folder of the site, you have to prevent it from setting read-only access on the root folder by adding **<IncludeSetACLProviderOn Destination>False</IncludeSetACLProviderOnDestination>** to the publish profile file (to affect a single profile) or to the wpp.targets file (to affect all profiles). For information about how to edit these files, see How to: Edit Deployment Settings in Profile (.pubxml) Files.

Configuration Error - targetFramework attribute references a version that is later than the installed version of the .NET Framework

Scenario

You successfully published a web project that targets ASP.NET 4.5, but when you run the application (with the `customErrors` mode set to "off" in the Web.config file) you get the following error:

```
The 'targetFramework' attribute in the <compilation> element of the
Web.config file is used only to target version 4.0 and later of the .NET
Framework (for example, '<compilation targetFramework="4.0">'). The
'targetFramework' attribute currently references a version that is later than
the installed version of the .NET Framework. Specify a valid target version
of the .NET Framework, or install the required version of the .NET Framework.
```

The Source Error box of the error page highlights the following line from Web.config as the cause of the error:

```
<compilation targetFramework="4.5" />
```

Possible Cause and Solution

The server does not support ASP.NET 4.5. Contact the hosting provider to determine when and if support for ASP.NET 4.5 can be added. If upgrading the server is not an option, you have to deploy a web project that targets ASP.NET 4 or earlier instead.

If you deploy an ASP.NET 4 or earlier web project to the same destination, select the **Remove additional files at destination** check box on the **Settings** tab of the **Publish Web** wizard. If you don't select **Remove additional files at destination**, you will continue to get the Configuration Error page.

The project **Properties** windows includes a Target framework drop-down list, but you can't resolve this problem by just changing that from **.NET Framework 4.5** to **.NET Framework 4**. If you change the target framework to an earlier framework version, the project will still have references to the later framework version's assemblies and will not run. You have to manually change those references or create a new project that targets .NET Framework 4 or earlier. For more information, see .NET Framework Targeting for Web Sites.

Medium Trust Errors

Scenario

When you run your application in production, it gets an error related to medium trust.

Possible Cause and Solution

Many third-party hosting providers run your web site in medium trust, which means that there are some things it isn't allowed to do. For example, application code can't access the Windows registry and can't read or write files that are outside of your application's folder hierarchy. By default your application runs in *full trust* on your local computer, which means that the application might be able to do things that would fail when you deploy it to production.

You can configure the application to run in medium trust in the local IIS environment in order to troubleshoot. To do that, open the application *Web.config* file, and add a **trust** element in the **system.web** element, as shown in this example.

```
<configuration>
  <!-- Settings -->
  <system.web>
    <trust level="Medium" />
    <!-- Settings -->
  </system.web>
</configuration>
```

The application will now run in medium trust in IIS even on your local computer.

Don't do this if you are deploying to Windows Azure Web Sites, because Windows Azure does not require medium trust. At the time this tutorial is being written in February, 2012, using this method to make your application run in medium trust will cause an error in Windows Azure.

If you are using Entity Framework Code First Migrations and you are deploying to a hosting provider that runs your application in medium trust, make sure that you have version 5.0 or later installed. In Entity Framework version 4.3, Migrations requires full trust in order to update the database schema.

HTTP 404.17 Not Found Error

Scenario

When you run the deployed site on your development computer in IIS, you see the following error message reporting that the server can't process Default.aspx:

```
HTTP Error 404.17 - Not Found

The requested content appears to be script and will not be served by the
static file handler.
```

Possible Cause and Solution

ASP.NET 4.5 might not be installed on your computer. See the steps in the deploying to IIS tutorial that explain how to install ASP.NET 4.5.