Hands-on Lab 3.1: Automating the Deployment of the WCF Service

Table of Contents

Objectives

This HOL demonstrates how to automatically deploy the Windows Communication Foundation (WCF) services that are used in the sample Trey Research application. To enable automatic deployment you must:

- Prepare the configuration files to match the target environments.
- Package the files that are required to deploy and run the WCF services, as well as the configuration files.
- Deploy the WCF services to the target environments.

This HOL is part one of the three-part Automation HOL. Together, the three parts demonstrate how to use Microsoft Visual Studio, Microsoft Team Foundation Server (TFS) and Lab Management to automate the deployment and testing of the following technologies.

- WCF services
- Windows Presentation Foundation (WPF)

Automation is discussed in Chapter 4 of Building a Release Pipeline with Team Foundation Server 2012. This lab shows you how to implement the changes to the pipeline that are discussed in that chapter.

As you go through this lab, you'll notice that there are references to Web Deploy and MSDeploy. MSDeploy is a part of Web Deploy, and is used to package and run deployments. Web Deploy includes MSDeploy as well as other components, such as the Internet Information Server (IIS) agent that communicates with MSDeploy to perform the deployments over IIS.

This lab refers to MSDeploy when it talks about the packaging and deployment tool. It refers to Web Deploy when additional components are involved.

Prerequisites

The only prerequisite for this lab is that you have completed all the preceding labs.

The Trey Research application is in your TreyResearch folder. Visual Studio solutions that are the result of completing all of the tasks in an exercise are in the **Lab03-Automation\Completed-Lab** folder.

You run the exercises for this lab on your local computer and on the computers that act as the environments.

NOTE: You can use either standard environments or SCVMM environments for this lab. Standard environments are much simpler to set up because you do not need to configure the network virtualization but some features, such as snapshots, are not available.

Time

If you are familiar with Visual Studio, Web Deploy, IIS, MSBuild, Windows Install XML (Wix), TFS and Lab Management, you should be able to complete all 3 labs in three to four hours.

This HOL (Lab3.1) takes approximately 60 minutes.

Exercise 1: Deployment Automation: Preparing the Configuration Files

In this exercise you automate the process of preparing each configuration file so that it conforms to a particular environment. When there are multiple environments, it's typical for the configuration file to be different for each of them. For example, database connection strings may vary from one environment to the next as well as endpoints, bindings, and application-specific settings.

One of the objectives of a continuous delivery pipeline is to automate deployment as much as possible. The pipeline should automatically modify each configuration file so that it conforms to an environment's requirements. Modifications should not be done manually.

Later in this lab you use MSDeploy to perform the actual deployments. Consequently, you must ensure that MSDeploy can access the configuration parameters. There are several ways to do this, but the one that best suits the goals of this lab is to prepare a parameters file that sets the specific configuration parameters used during a deployment. You set the MSDeploy **setParamFile** flag so that MSDeploy uses the parameters file during a deployment. (For more information, see Web Deploy Operation Settings.)

This lab generally follows the approach described in Configuring Parameters for Web Package Deployment. The last step of this lab is different from the on-line tutorial because it uses configuration file transforms rather than the MSBuild **XmlPoke** task. Another difference is that this lab automates the step where the correct parameters file is selected for the correct target environment.

Using MSDeploy and a parameters file makes it possible to do deployments in a way that conforms to some best practices for continuous delivery.

- The same binaries are used across all environments because there is no need to have a different build for each environment.
- Deployment is the same for all environments. The pipeline determines which configuration to use.
- Environment-specific information is separate from the actual release.

Task 1: Add an Application-Specific Setting to the Web.config File

In this task you add an application-specific setting to the WCF service's Web.config file. The setting is a key/value pair that stores the name of the environment where the service is running. Although not required to automate a deployment, this setting can be useful. For example, if you want to do A/B testing you can either show or hide the feature you're testing, depending on the environment.

1. In Visual Studio, open **TreyResearch.sln**.
2. Open the Web.config file located in **TreyResearch.WcfService** (this is the WCF Services project).
3. Insert the following code at the end of the file, between the **</system.webServer>** and **</configuration>** tags. Save the file and close it.

XML

```xml
<appSettings>
<add key="Environment" value="Development" />
</appSettings>
```

Task 2: Prepare the SetParameters Files

In this task you add the SetParameters files to the project. These are the files that MSDeploy uses when it deploys to each environment. You need four files. Three are for the three environments. The fourth file stores the base configuration that is transformed for each environment.

Note: There are at least two ways to provide different configuration settings for different environments. One is to have a base configuration file that is transformed before a deployment in order to produce a configuration file for a specific environment. Another, which does not use transforms, is to store complete configuration files for all environments and to pick the right one during a deployment. The transform approach is usually better because you only have to specify and maintain the settings that change from one environment to another. In order to demonstrate this practice, this lab uses the transform approach, even though the actual configuration files are very simple.

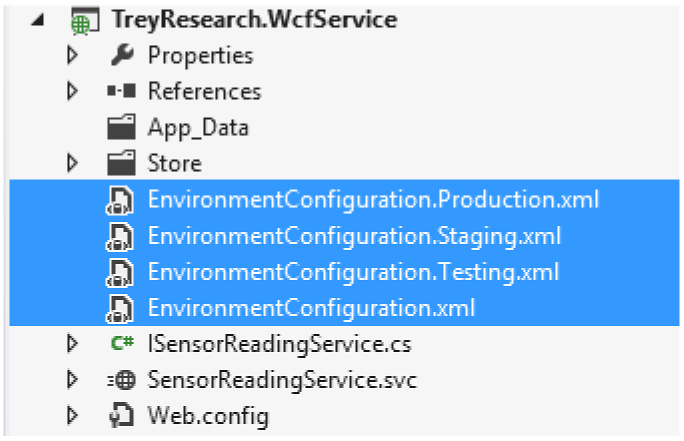
1. Add a new XML file to the project and name it **EnvironmentConfiguration.xml**. This is the base file that will be transformed to generate the configuration for each environment. Add the following code to the file.

XML

```xml
<parameters>
  <!-- AUTOMATION HOL - This parameter defines the name of the target Web Site
where the application is deployed -->
  <setParameter name="IIS Web Application Name" value="IIS Web Application
Name" />
  <!-- AUTOMATION HOL - This parameter defines the name of the environment
where the application is running -->
  <setParameter name="Environment" value="Environment" />
```

```xml
</parameters>
```

2. Add three additional XML files to the project, and name each one after a target environment, using the format **EnvironmentConfiguration.<Environment Name>.xml**. The following screenshot shows the results.



```
TreyResearch.WcfService
    Properties
    References
    App_Data
    Store
    EnvironmentConfiguration.Production.xml
    EnvironmentConfiguration.Staging.xml
    EnvironmentConfiguration.Testing.xml
    EnvironmentConfiguration.xml
    ISensorReadingService.cs
    SensorReadingService.svc
    Web.config
```

3. Add the following XML code to each of the following files. Remember to change the **value** to correspond to each environment. The code that is shown is for the testing environment. Here are the values for the other environments.

   - For the EnvironmentConfiguration.Production.xml file, **value="Production"**.
   - For the EnvironmentConfiguration.Staging.xml file, **value="Staging"**.
   - For the EnvironmentConfiguration.Testing.xml file, **value="Testing"**.

Here is the XML code for the testing environment.

XML

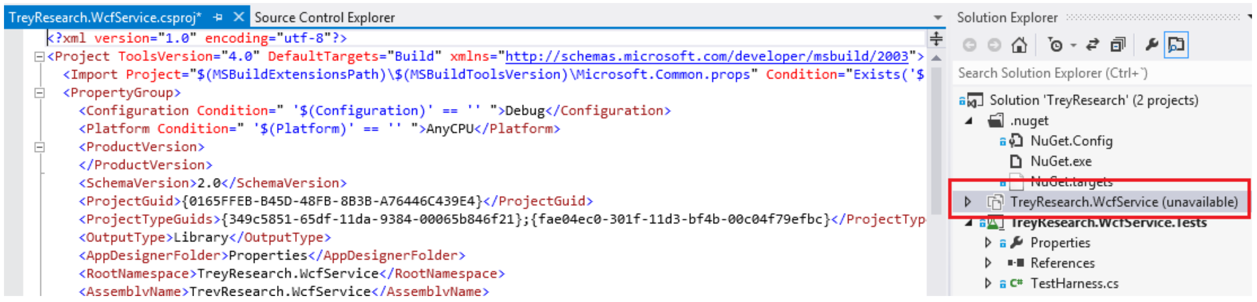
```xml
<parameters xmlns:xdt="http://schemas.microsoft.com/XML-Document-Transform">
  <!-- AUTOMATION HOL - This defines the way the value of the "IIS Web
Application Name" parameter gets transformed for the Testing environment. -->
  <!-- AUTOMATION HOL - In the config file for the Testing environment, the
web application name will be TreyResearchTesting -->
  <setParameter name="IIS Web Application Name"
                value="TreyResearchTesting"
                xdt:Transform="SetAttributes(value)" xdt:Locator="Match(name)"
/>
  <!-- AUTOMATION HOL - This defines the way the value of the "Environment"
parameter gets transformed for the Testing environment. -->
  <!-- AUTOMATION HOL - In the config file for the Testing environment, the
environment name will be Testing -->
  <setParameter name="Environment"
                value="Testing"
                xdt:Transform="SetAttributes(value)" xdt:Locator="Match(name)"
/>
```

```
</parameters>
```

Task 3: Set the Dependency on the Base File

In this task you make all the **EnvironmentConfiguration.<Environment Name>.xml** files dependent on the base configuration file, **EnvironmentConfiguration.xml**. Although this step is not required to perform the transformations, it helps to keep the project organized and understandable.

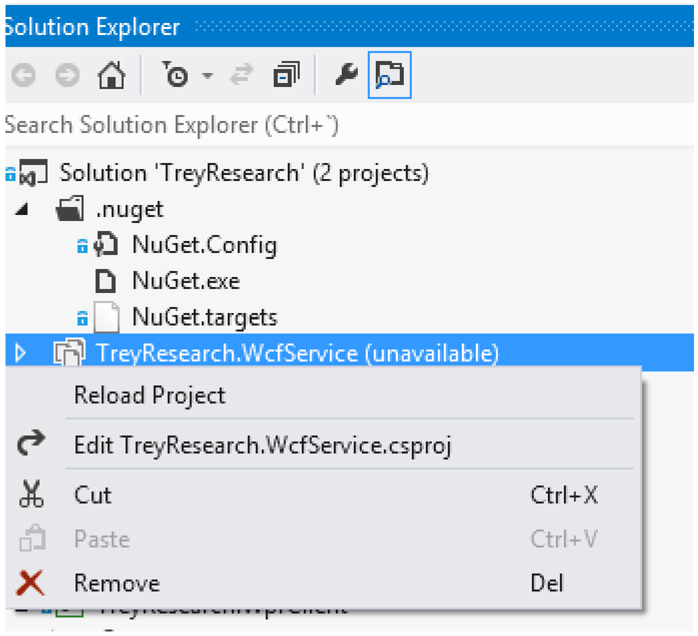
1. Right-click on the **TreyResearch. WcfService** project and select **Unload project**.



```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Common.props" Condition="Exists('$
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProductVersion>
    </ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>{0165FFEB-B45D-48FB-8B3B-A76446C439E4}</ProjectGuid>
    <ProjectTypeGuids>{349c5851-65df-11da-9384-00065b846f21};{fae04ec0-301f-11d3-bf4b-00c04f79efbc}</ProjectTyp
    <OutputType>Library</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>TreyResearch.WcfService</RootNamespace>
    <AssemblyName>TreyResearch.WcfService</AssemblyName>
```

TreyResearch.WcfService.csproj*
Source Control Explorer
Solution Explorer
Search Solution Explorer (Ctrl+`)
Solution 'TreyResearch' (2 projects)
.nuget
NuGet.Config
NuGet.exe
NuGet.targets
TreyResearch.WcfService (unavailable)
TreyResearch.WcfService.Tests
Properties
References
TestHarness.cs

2. Right-click on the project again and select **Edit TreyResearch. WcfService.csproj**. The MSBuild code that makes up the **csproj** project file appears.



Solution Explorer
Search Solution Explorer (Ctrl+`)
Solution 'TreyResearch' (2 projects)
.nuget
NuGet.Config
NuGet.exe
NuGet.targets
TreyResearch.WcfService (unavailable)
Reload Project
Edit TreyResearch.WcfService.csproj
Cut Ctrl+X
Paste Ctrl+V
Remove Del

3. Locate the following XML code.

XML

```
<Content Include="EnvironmentConfiguration.Production.xml" />
<Content Include="EnvironmentConfiguration.Staging.xml" />
```

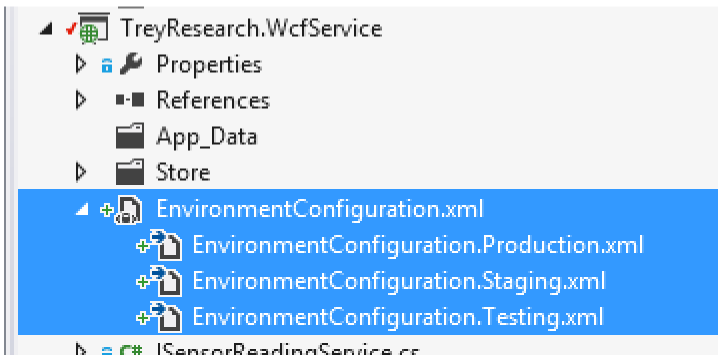
```
<Content Include="EnvironmentConfiguration.Testing.xml" />
<Content Include="EnvironmentConfiguration.xml" />
```

4. Replace this XML code with the following code, which makes the environment-specific files dependent upon the base configuration file.

XML

```
<Content Include="EnvironmentConfiguration.xml" />
<None Include="EnvironmentConfiguration.Testing.xml">
      <DependentUpon>EnvironmentConfiguration.xml</DependentUpon>
</None>
<None Include="EnvironmentConfiguration.Staging.xml">
       <DependentUpon>EnvironmentConfiguration.xml</DependentUpon>
</None>
<None Include="EnvironmentConfiguration.Production.xml">
<DependentUpon>EnvironmentConfiguration.xml</DependentUpon>
</None>
```

5. Save the file. Right-click on the **TreyResearch. WcfService** project and select **Reload Project**. Your project should have the structure shown in the following screenshot. Note that the environment-specific files are nested under the base configuration file.


TreyResearch.WcfService
Properties
References
App_Data
Store
EnvironmentConfiguration.xml
EnvironmentConfiguration.Production.xml
EnvironmentConfiguration.Staging.xml
EnvironmentConfiguration.Testing.xml
ISensorReadingService.cs

Task 4: Set Up the Transformation for the SetParameters Files

This task uses configuration transformations and follows the standard syntax for Web.config transformations in Visual Studio. Note that any XML file can be transformed this way. For more information, see Web.config Transformation Syntax for Web Project Deployment Using Visual Studio.

Transformations occur within the project file, which means that they are performed when the project is built. Consequently, you don't need to add extra steps that direct the commit stage to generate the transformed configuration files after it builds the code.

If you're familiar with Web.config transformations in Visual Studio, you may wonder why the lab uses the **TransformXml** task explicitly. There are two reasons.

One is that, out of the box, Web.config transforms only work with Web.config files. This lab uses MSDeploy SetParams files. There are third-party tools available such as SlowCheetah that address this issue, but they offer only partial solutions. Aside from the inconvenience of adding another tool to the lab, SlowCheetah, for example, also shares some of the same restrictions as Web.config transforms.

Both Web.config transforms and SlowCheetah use Visual Studio build configurations (for example, debug and release) and are triggered when a build occurs. Because the goal of continuous delivery is to use the same binaries across all environments, there must be a single build and the transformations should be based on environments that all use the same build configuration (in the case of this lab, this is the release build configuration). In other words, transformations must be triggered without having to rebuild the source code for each environment.

In this task you use the **TransformXml** MSBuild task to transform the base **EnvironmentConfiguration.xml** parameters file. You use the **EnvironmentConfiguration.<EnvironmentName>.xml** files to define the transformation. When the transformations are complete, you will have the environment-specific parameters file.

1. Unload and edit the **TreyResearch.WcfService.csproj** file, just as you did in Task 3. Near the bottom of the file, just before the closing **</Project>** tag, insert the following code. This code generates the transformed SetParameters files.

XML

```
<!-- AUTOMATION HOL - Referencing the TransformXml task so we can use it -->
  <UsingTask TaskName="TransformXml"
AssemblyFile="$(MSBuildExtensionsPath32)\Microsoft\VisualStudio\v11.0\Web\Micr
osoft.Web.Publishing.Tasks.dll" />
  <!-- AUTOMATION HOL - We are triggering the transformations just after
building -->
  <!-- AUTOMATION HOL - Checking whether we are BuildingInsideVisualStudio
makes that this will be triggered only in the Commit Stage of the pipeline (or
in command-line builds), not while working inside Visual Studio -->
  <!-- AUTOMATION HOL - Checking whether we are building the Release
configuration makes the parameters files available only for Release builds,
the ones being used by the pipeline -->
  <Target Name="AfterBuild" Condition="('$(BuildingInsideVisualStudio)' !=
'true') And ('$(Configuration)' == 'Release')">
    <ItemGroup>
      <TransformationFiles Include="EnvironmentConfiguration.*.xml" />
    </ItemGroup>
  <!-- AUTOMATION HOL - We generate the transformated parameters files in a
subfolder of $(OutDir). That way, they will get copied directly to the
BinariesFolder in the TFS build agent, and in turn, to the Drop folder of the
Commit Stage, without having to explicitly copying them -->
    <MakeDir Directories="$(OutDir)\ConfigFiles\WcfService"
Condition="!Exists('$(OutDir)\ConfigFiles\WcfService')" />
    <!-- AUTOMATION HOL - We call the TransformXml task for all the parameters
files using MSBuild batching (the @() syntax)  -->
    <TransformXml Source="EnvironmentConfiguration.xml"
Transform="@(TransformationFiles)"
```

```
Destination="$(OutDir)\ConfigFiles\WcfService\%(TransformationFiles.Identity)"
/>
  </Target>
```

2. Reload the project. It opens in Solution Explorer.

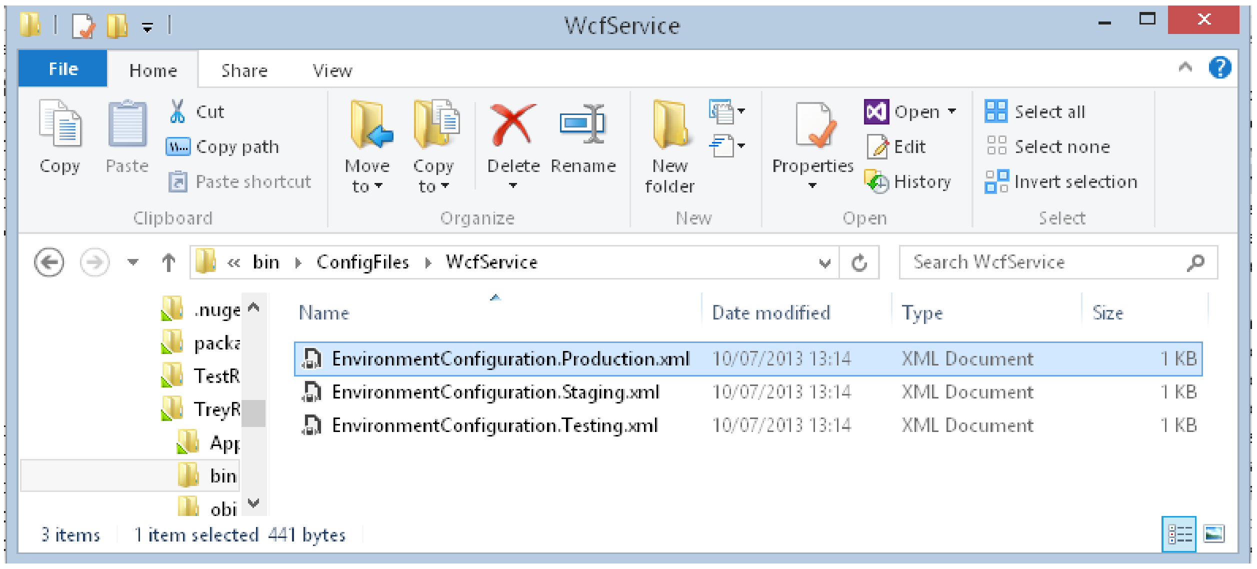
3. From the Windows Start menu (or in Windows 8, the **Start** screen) open a **Developer Command Prompt for VS2012**. Change the directory to where the **TreyResearch. WcfService.csproj** is located.

4. Test to see if the transformations were performed. Run the following command.

CMD

```
msbuild /p:Configuration=Release
```

5. After MSBuild finishes, you will find a new folder named **ConfigFiles\WcfService** under the **bin** subfolder. The folder contains the three transformed parameters files for this project. If you open any of them, you will see that the content has been transformed to match the target environment.



Name	Date modified	Type	Size
EnvironmentConfiguration.Production.xml	10/07/2013 13:14	XML Document	1 KB
EnvironmentConfiguration.Staging.xml	10/07/2013 13:14	XML Document	1 KB
EnvironmentConfiguration.Testing.xml	10/07/2013 13:14	XML Document	1 KB

Exercise 2: Packaging the Deployment Files

In this exercise you package the files that are required for deployment. The package format is the one that MSDeploy generates. The format is based on a zip file that contains the files to be installed on the IIS website. When MSDeploy generates a package, it also provides a script that performs the actual deployment and provides a sample parameters file. You will replace the sample file with the parameter files that correspond to the environments.

NOTE: Besides packaging, there are other MSDeploy publication methods that can deploy directly to the target web server. This lab uses packaging for three reasons. The first reason is that the application should be deployed to a stage only if the previous stages were successful. The second reason is that by using a package that is generated only in the commit stage, you ensure that all the stages use the same binaries for the deployment. The third reason is that Lab Management orchestrates all the stages in the pipeline except the commit stage. In Lab Management environments, a deployment agent uses a script that runs locally on the target machine to deploy to the target environment. A package is a good way to work with MSDeploy when the deployment is done locally, from the same machine that hosts the IIS site.

Task 1: Configure the Packaging Process

In this task you configure the packaging process so that the Web.config parameters are available and can be changed. To do this, you add a parameters.xml file to the project. This file is used by MSBuild when it generates the zip package. The file tells MSBuild which elements in the Web.config file should be exposed as parameters during deployment. MSBuild replaces the elements in the final Web.config file with the corresponding values contained in the parameters files for each environment. Aside from the tutorial mentioned in Exercise 1, you can also refer to How to: Use Web Deploy Parameters in a Web Deployment Package. Read the section named "Using Deployment Parameters for Web.Config File Settings."

1. Open the **TreyResearch. WcfService.csproj** project.
2. Add a new XML file to the project and name it **parameters.xml**. Add the following XML code to the file.

XML

```xml
<parameters>
  <!-- AUTOMATION HOL - This parameter section tells MSBuild to expose the Web.config configuration parameter named Environment, so ti can be changed on the fly during deployment -->
  <parameter name="Environment"
             description="Environment where the services are running"
             defaultValue="Development">
    <!-- AUTOMATION HOL - This parameterEntry section tells MSBuild how to locate the parameter inside the Web.config file, so it can expose it -->
    <parameterEntry kind="XmlFile" scope="Web.config"

match="/configuration/appSettings/add[@key='Environment']/@value" />
  </parameter>
</parameters>
```

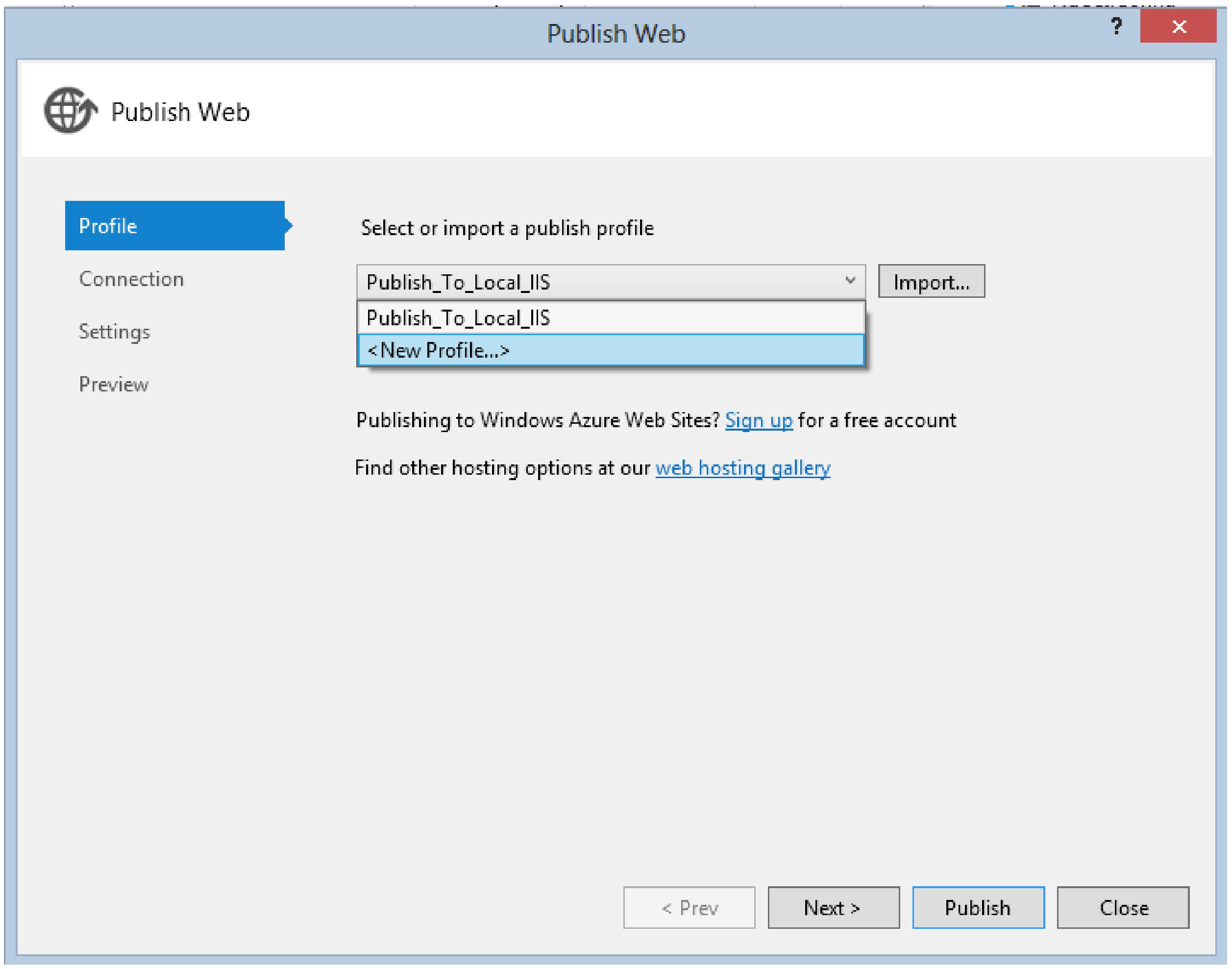
3. Save the file and close it.

Task 2: Create a Publication Profile

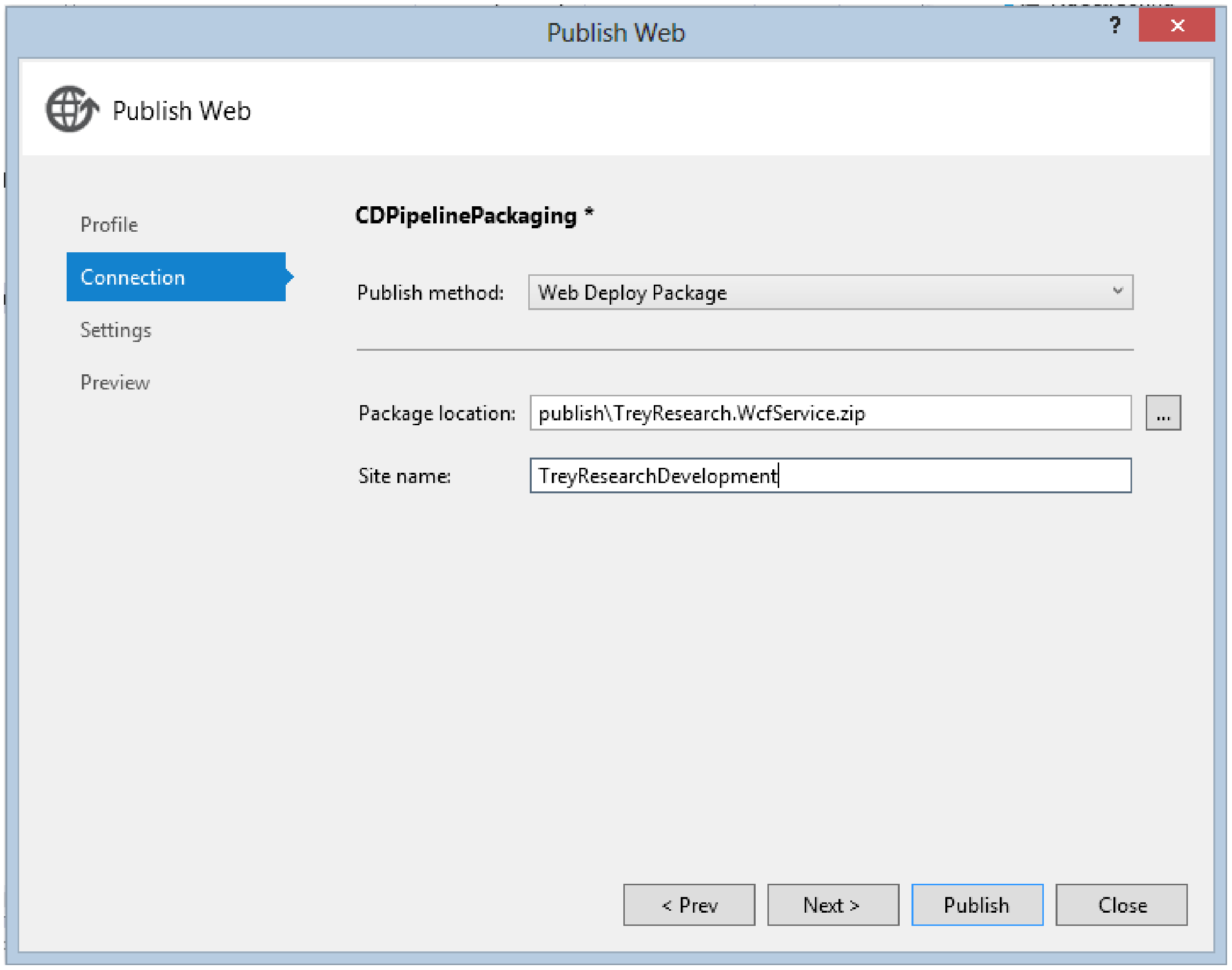
In this task you create a publication profile that defines how the package is created.

1. Right-click on the **TreyResearch. WcfService. csproj** project.

2. Select **Publish**. The **Publish Web** wizard opens. The **Profile** tab should be selected.
3. Select **<New Profile…>** from the **Select or import a publish profile** drop-down list. Click **Next**. The **New Profile** dialog box appears.


Publish Web
?
×
Publish Web
Profile
Connection
Settings
Preview
Select or import a publish profile
Publish_To_Local_IIS
Import...
Publish_To_Local_IIS
<New Profile...>
Publishing to Windows Azure Web Sites? Sign up for a free account
Find other hosting options at our web hosting gallery
< Prev
Next >
Publish
Close

4. Enter the name **CDPipelinePackaging** in the **New Profile** dialog box. Click **OK**. This will be the profile used to package the web services in the commit stage. Click **Next**.
5. The Publish Web wizard advances to the **Connection** tab. Select **Web Deploy Package** from the **Publish method** drop-down list. With this method, MSDeploy generates a zip file that contains the required files for the deployment.
6. In the **Package location** box, enter the package location. In the **Site name** box, enter the site name. These values will be overwritten by the values in the environment-specific parameters files, but remain in the base configuration file, and may be be useful for the development environment. Click **Next**.



Publish Web

Profile
Connection
Settings
Preview

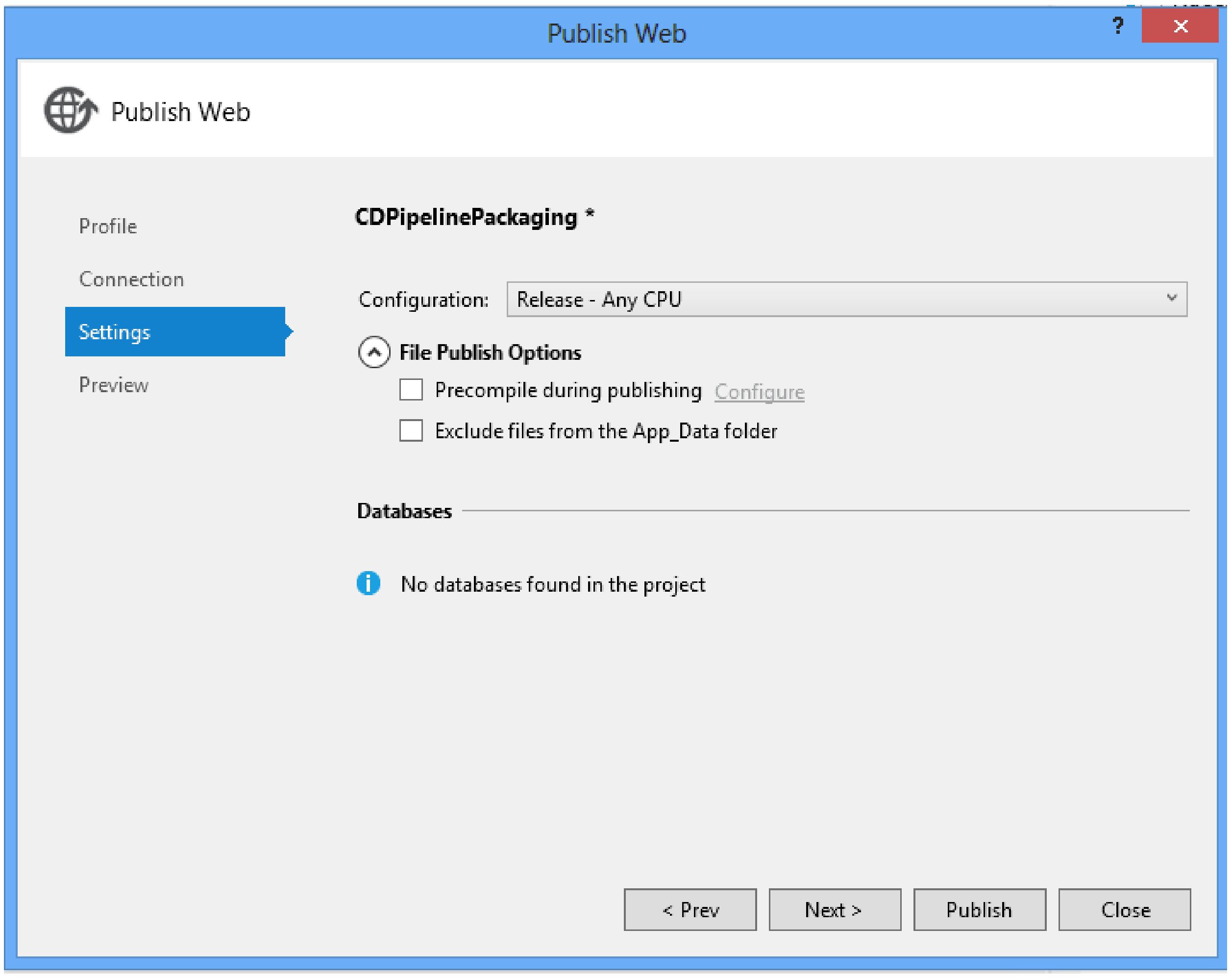
CDPipelinePackaging *

Publish method: Web Deploy Package

Package location: publish\TreyResearch.WcfService.zip

Site name: TreyResearchDevelopment

7. The Publish Web wizard advances to the **Settings** tab. In the **Configuration** box, select **Release – Any CPU** from the drop-down list. The pipeline only uses the release configuration.


Publish Web

Publish Web

CDPipelinePackaging *

Configuration: Release - Any CPU

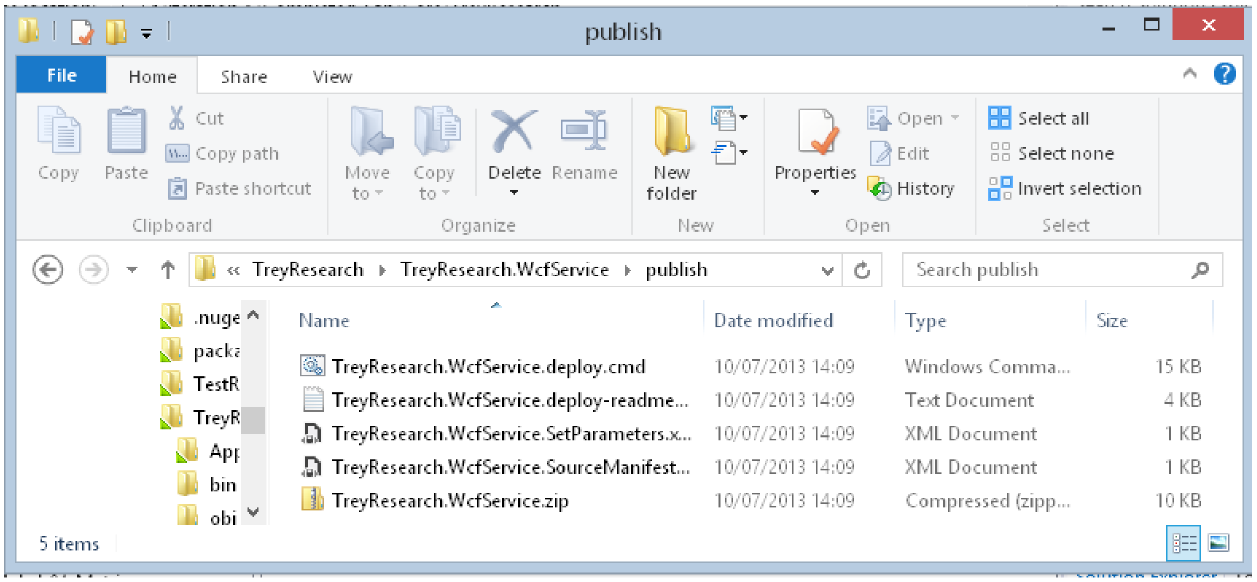
File Publish Options

- [ ] Precompile during publishing Configure
- [ ] Exclude files from the App_Data folder

Databases

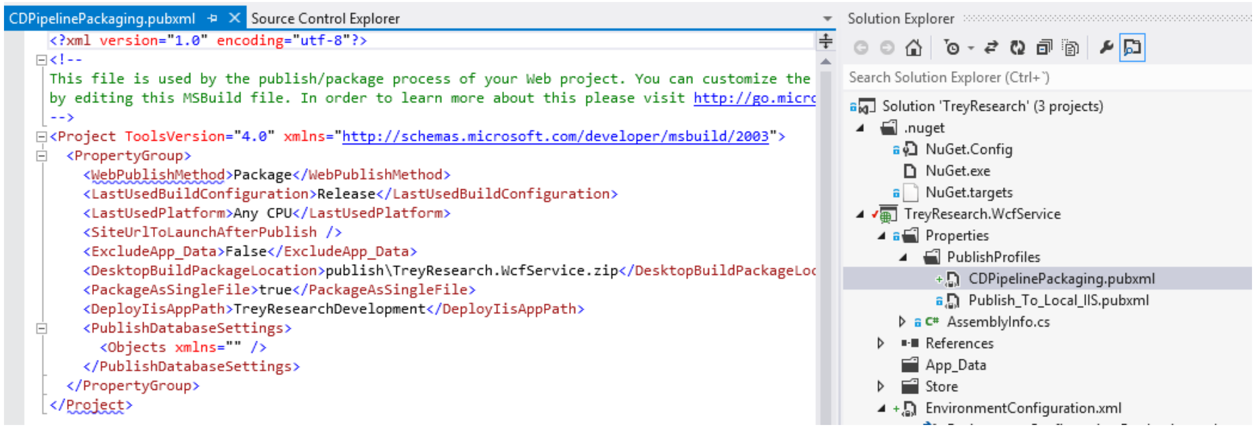
No databases found in the project

8. Click **Publish**. The web services project is built and packaged. The resulting zip file, script and parameters file are copied to the publish subdirectory under the project folder. The following screenshot shows an example.



Name	Date modified	Type	Size
TreyResearch.WcfService.deploy.cmd	10/07/2013 14:09	Windows Comma...	15 KB
TreyResearch.WcfService.deploy-readme...	10/07/2013 14:09	Text Document	4 KB
TreyResearch.WcfService.SetParameters.x...	10/07/2013 14:09	XML Document	1 KB
TreyResearch.WcfService.SourceManifest...	10/07/2013 14:09	XML Document	1 KB
TreyResearch.WcfService.zip	10/07/2013 14:09	Compressed (zipp...	10 KB

The artifact of primary interest is the CDPipelinePackaging.pubxml file that is located in the Properties/PublishProfiles folder. The goal is to have the pipeline generate the package automatically each time the commit stage runs. In the next task, you use this file to tell the pipeline how to do this. The following screenshot shows an example of the CDPipelinePackaging.pubxml file.



```
<?xml version="1.0" encoding="utf-8"?>
<!--
This file is used by the publish/package process of your Web project. You can customize the
by editing this MSBuild file. In order to learn more about this please visit http://go.micrc
-->
<Project ToolsVersion="4.0" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <WebPublishMethod>Package</WebPublishMethod>
    <LastUsedBuildConfiguration>Release</LastUsedBuildConfiguration>
    <LastUsedPlatform>Any CPU</LastUsedPlatform>
    <SiteUrlToLaunchAfterPublish />
    <ExcludeApp_Data>False</ExcludeApp_Data>
    <DesktopBuildPackageLocation>publish\TreyResearch.WcfService.zip</DesktopBuildPackageLoc
    <PackageAsSingleFile>true</PackageAsSingleFile>
    <DeployIisAppPath>TreyResearchDevelopment</DeployIisAppPath>
    <PublishDatabaseSettings>
      <Objects xmlns="" />
    </PublishDatabaseSettings>
  </PropertyGroup>
</Project>
```

```
Solution 'TreyResearch' (3 projects)
  .nuget
    NuGet.Config
    NuGet.exe
    NuGet.targets
  TreyResearch.WcfService
    Properties
      PublishProfiles
        CDPipelinePackaging.pubxml
        Publish_To_Local_IIS.pubxml
      AssemblyInfo.cs
    References
    App_Data
    Store
    EnvironmentConfiguration.xml
```

Task 3: Configuring the Commit Stage to Generate the Package

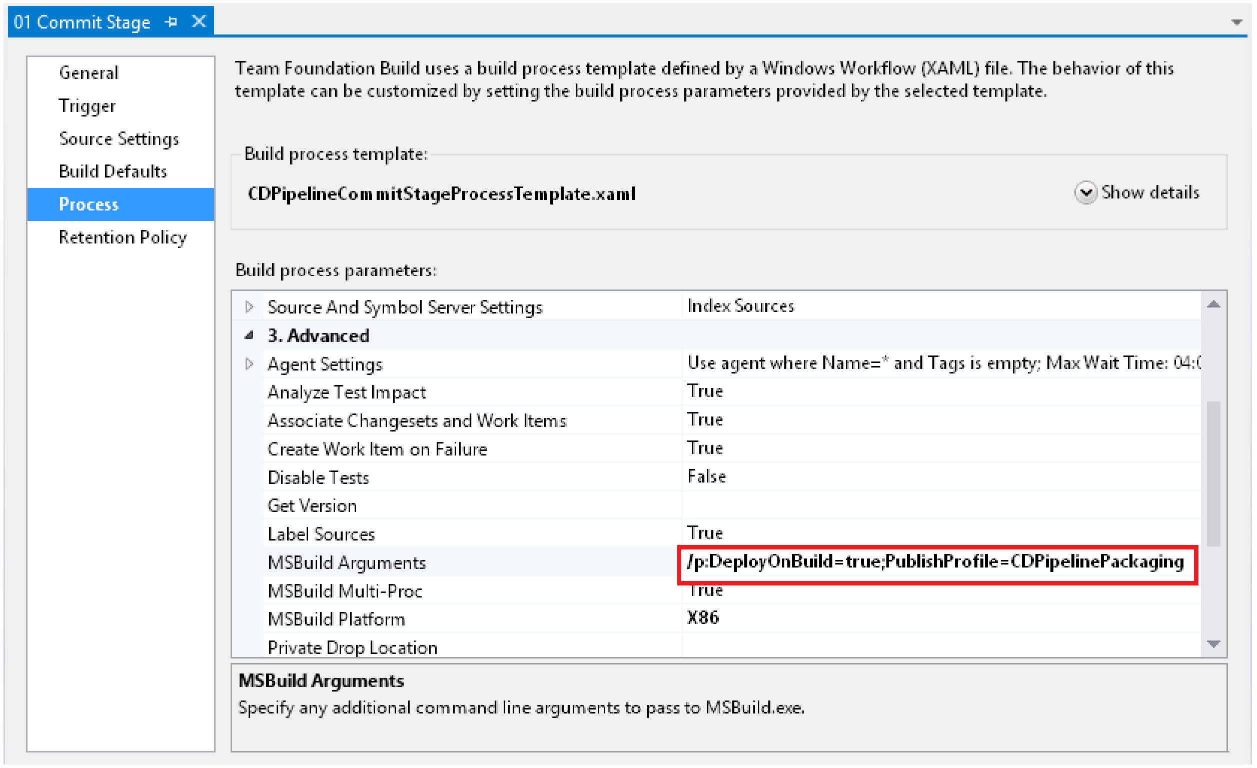
In this task you configure the commit stage to package the web services by editing the **01 Commit Stage** build definition.

1. Open the **01 Commit Stage** build definition.
2. Go to the **Process** tab. In the **Advanced** section add the following value to the **MSBuild Arguments** parameter.

Visual Basic

```
/p:DeployOnBuild=true;PublishProfile=CDPipelinePackaging
```

The following screenshot shows the modified build definition.



Team Foundation Build uses a build process template defined by a Windows Workflow (XAML) file. The behavior of this template can be customized by setting the build process parameters provided by the selected template.

Build process template:

CDPipelineCommitStageProcessTemplate.xaml

Build process parameters:

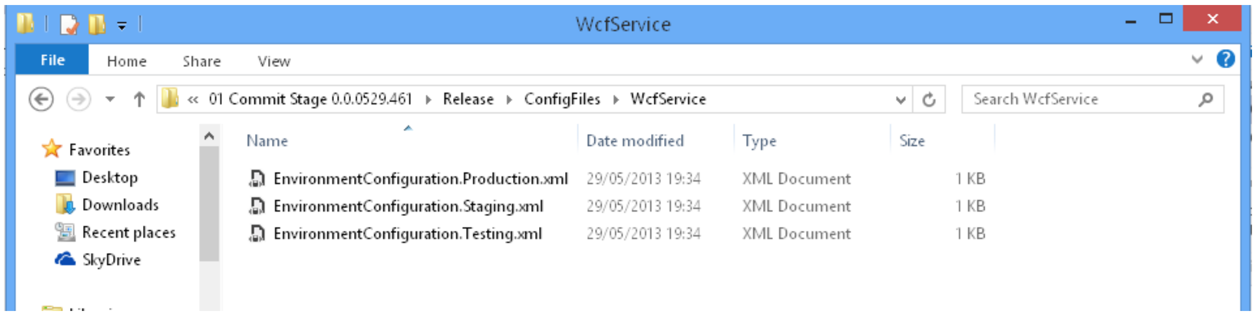
Source And Symbol Server Settings	Index Sources
3. Advanced	
Agent Settings	Use agent where Name=* and Tags is empty; Max Wait Time: 04:(
Analyze Test Impact	True
Associate Changesets and Work Items	True
Create Work Item on Failure	True
Disable Tests	False
Get Version	
Label Sources	True
MSBuild Arguments	/p:DeployOnBuild=true;PublishProfile=CDPipelinePackaging
MSBuild Multi-Proc	True
MSBuild Platform	X86
Private Drop Location	

MSBuild Arguments
Specify any additional command line arguments to pass to MSBuild.exe.

3. The pipeline is now configured to prepare the parameters files and to package the web services. Remember to save the build configuration. Check in all the pending changes. The commit stage should be triggered automatically.

   After the commit stage successfully finishes, go to the drop folder. Inside the **Release** subfolder, you should see:

   - A **_PublishedWebsites\TreyResearch.WcfService_Package** subfolder that contains the .zip package and the .cmd script to deploy it. (There are other files as well, but they aren't relevant).
   - A **ConfigFiles\WcfService** subfolder that contains all the environment specific parameters files. The following screenshot shows an example.



Name	Date modified	Type	Size
EnvironmentConfiguration.Production.xml	29/05/2013 19:34	XML Document	1 KB
EnvironmentConfiguration.Staging.xml	29/05/2013 19:34	XML Document	1 KB
EnvironmentConfiguration.Testing.xml	29/05/2013 19:34	XML Document	1 KB

NOTE: You may have noticed that MSDeploy.exe was not run explicitly during this exercise. This is because MSDeploy is triggered by MSBuild in the context of the Web Publishing Pipeline (WPP). Do not confuse the WPP with the continuous delivery pipeline you are building. These are two different concepts. WPP is used by the continuous delivery pipeline.

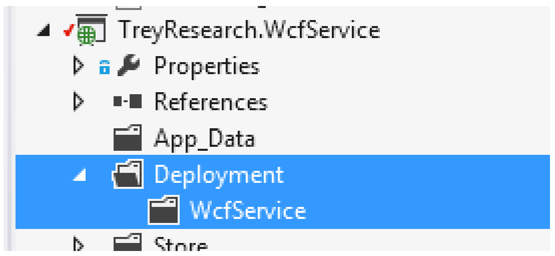
Exercise 3: Performing the Automated Deployment

In this exercise you set up the pipeline so that it can automatically deploy the packages to the environments.

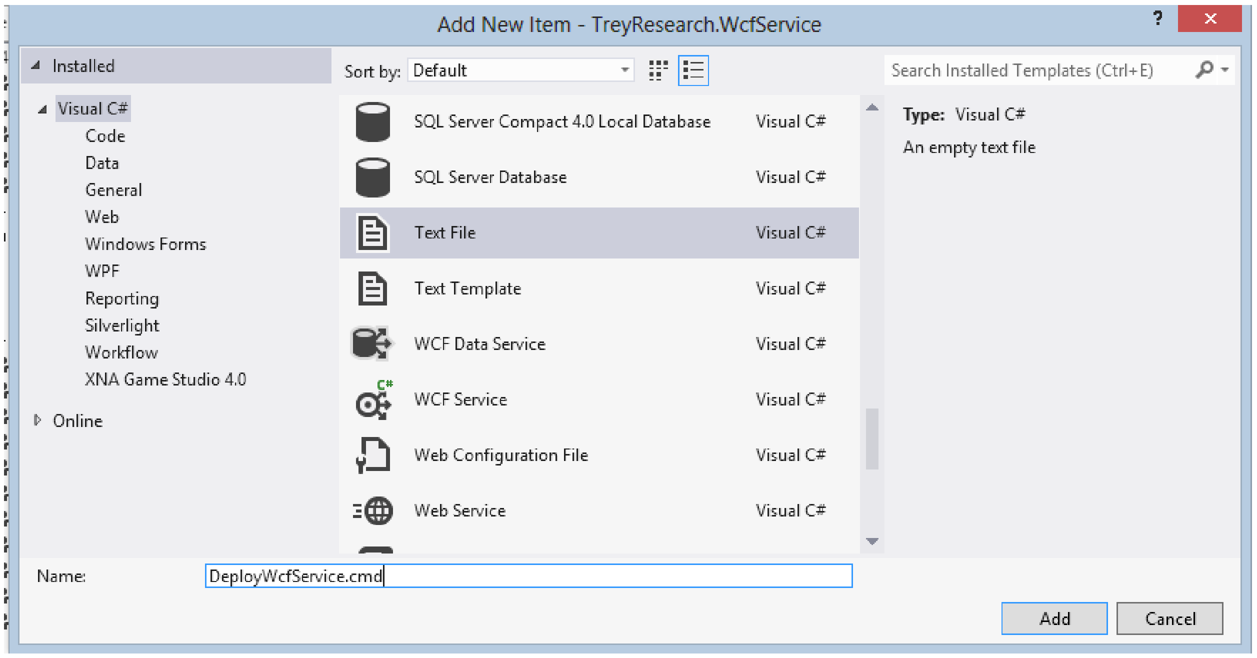
Task 1: Create the Deployment Script

In this task you create the deployment script for the WCF service. Lab Management deployment scripts are batch files that are executed by the Windows command interpreter locally, on the target computer, by the deployment agent.

1. In Solution Explorer, create a new folder named **Deployment** under the **TreyResearch. WcfService** project. This is where you will store the deployment script.
2. Under the **Deployment** folder, create a subfolder named **WcfService**. This folder helps to organize the scripts when they are copied to the drop folder and to the target environments. The following screenshot shows the folder hierarchy.

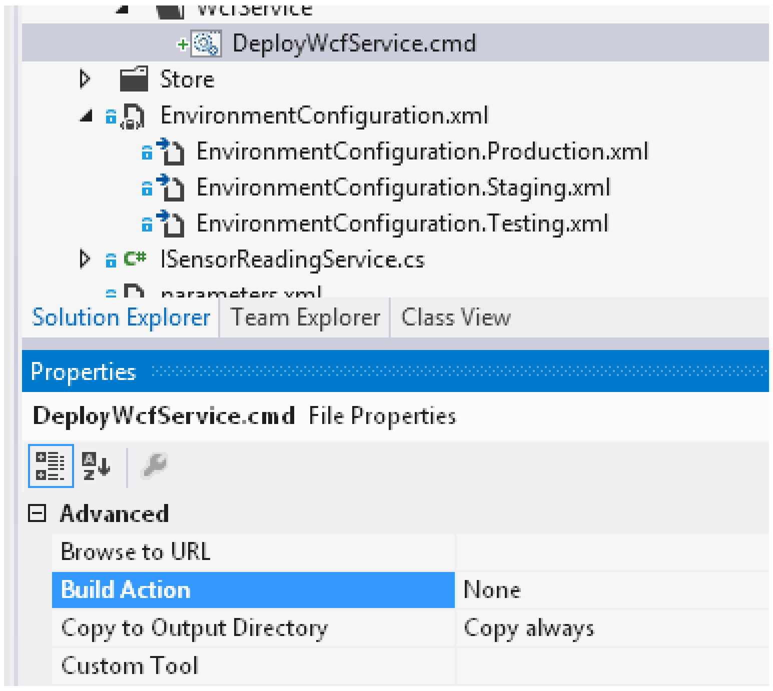


3. Add a new script file to the **WcfService** subfolder. Right-click the **WcfService** folder. Add a new item of type **Text File**. Change the name of the file to **DeployWcfService.cmd**.


Add New Item - TreyResearch.WcfService

Name: DeployWcfService.cmd

4. In the **DeployWcfService.cmd** file properties, set the **Build Action** to **None**. Set **Copy to Output Directory** to **Copy always**. These settings copy the script to the output directory when the project is built, and to the drop location during the commit stage. The deployment agent retrieves the script from the drop location and runs it. The following screenshot shows the **DeployWcfService.cmd** property settings.


DeployWcfService.cmd File Properties

Browse to URL	
Build Action	None
Copy to Output Directory	Copy always
Custom Tool	

WcfService
DeployWcfService.cmd
Store
EnvironmentConfiguration.xml
EnvironmentConfiguration.Production.xml
EnvironmentConfiguration.Staging.xml
EnvironmentConfiguration.Testing.xml
ISensorReadingService.cs
parameters.xml
Solution Explorer
Team Explorer
Class View
Properties
Advanced

5. Paste the following code into the file.

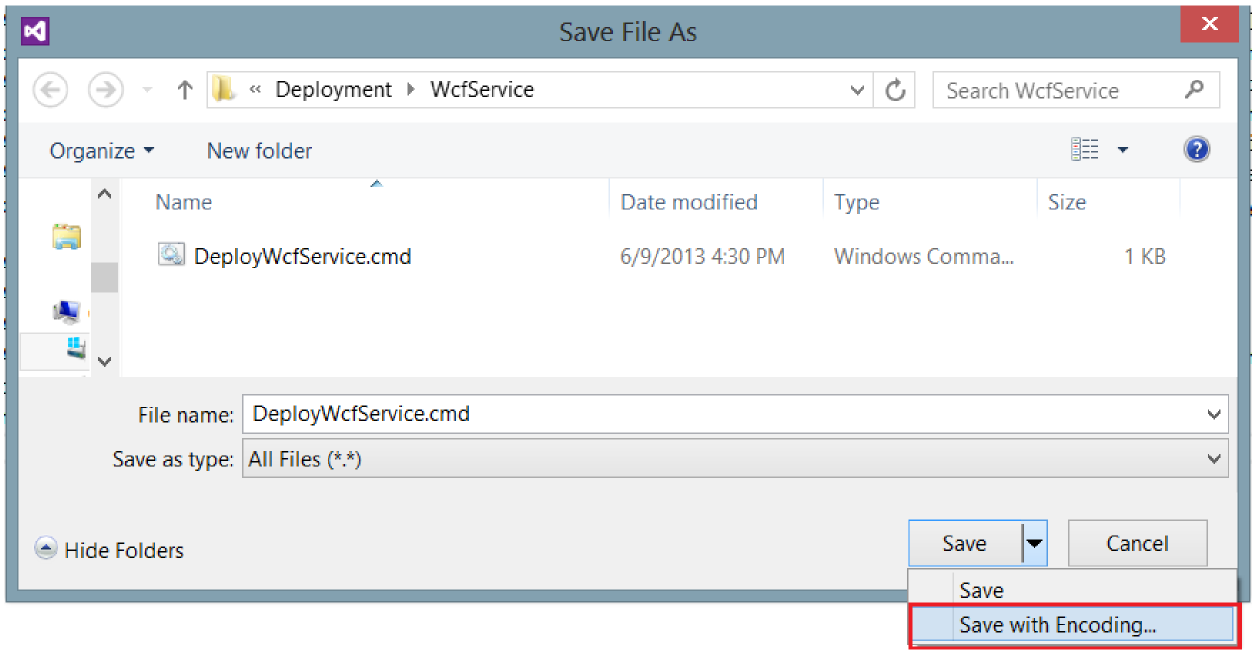
CMD Script

```
REM AUTOMATION HOL - This script copies the needed files from the pipeline
Drop folder to the target machine and runs the MSDeploy deployment script

REM AUTOMATION HOL - Path where the deployment package is stored (the Drop
location for the pipeline instance)
set packageLocation=%~1

REM AUTOMATION HOL - Name of the environment where the deployment is made.
This is used to pick the corresponding parameters file from the available ones
set environment=%~2
REM AUTOMATION HOL - Name of the temporal folder to use in the target computer
set tempfolder=%~3

REM AUTOMATION HOL - Preparing the local temp directory to copy the files and
run the deployment
set deploymentLocation="%tempfolder%\WcfService\%environment%"
if exist "%deploymentLocation%" rd /s /q "%deploymentLocation%"
mkdir "%deploymentLocation%"

REM AUTOMATION HOL - Copying the zip package generated by the Commit Stage,
from the Drop location to the local deployment folder
copy
"%packageLocation%\Release\_PublishedWebsites\TreyResearch.WcfService_Package\
TreyResearch.WcfService.zip" "%deploymentLocation%"
REM AUTOMATION HOL - Copying the MSDeploy deployment script that was auto-
generated by the publish operation in the Commit Stage, from the Drop location
to the local deployment folder
copy
"%packageLocation%\Release\_PublishedWebsites\TreyResearch.WcfService_Package\
TreyResearch.WcfService.deploy.cmd" "%deploymentLocation%"
REM AUTOMATION HOL - Copying the parameters file that tells the MSDeploy
deployment script what to change in the Web.config file during deployment
REM AUTOMATION HOL - We pick the one corresponding to the environment where
the deployment is being done, and rename it to match the name that the
MSDeploy recognizes as a parameters file
copy
"%packageLocation%\Release\ConfigFiles\WcfService\EnvironmentConfiguration.%en
vironment%.xml"
"%deploymentLocation%\TreyResearch.WcfService.SetParameters.xml"

REM AUTOMATION HOL - Running the deployment over the Web Server
REM AUTOMATION HOL - The /Y switch tells the script to actually perform the
deployment
REM AUTOMATION HOL - The script deploys the services over the configured IIS
Website
```

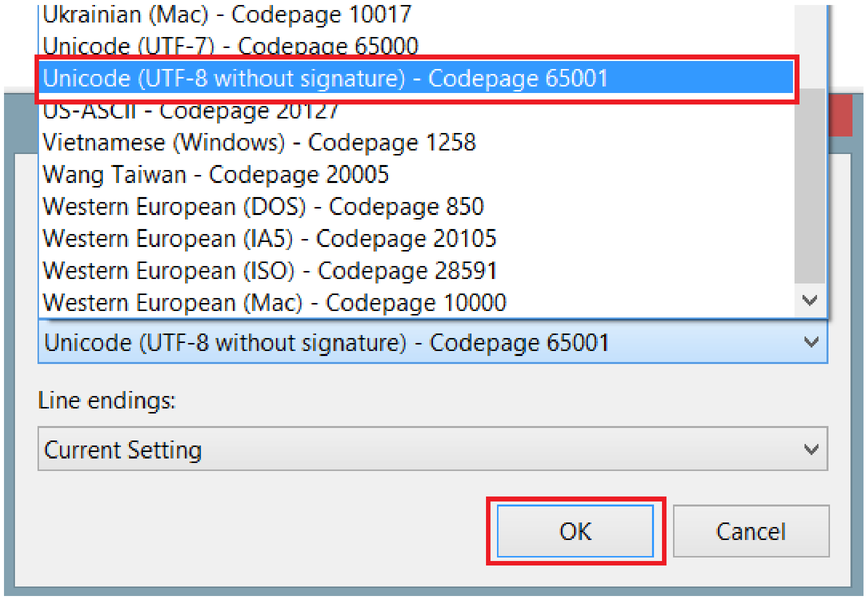
```
REM AUTOMATION HOL - The script will also transform the final Web.config with
any parameter contained in
TreyResearch.Cloud.WcfService.WebRole.SetParameters.xml
cd "%deploymentLocation%"
TreyResearch.WcfService.deploy.cmd /Y
```

This script copies the deployment package, the MSDeploy deployment script, and the parameters file corresponding to the target environment to the web server. The script runs the MSDeploy deployment script on the server.

6. Save the **DeployWcfService.cmd** file using the encoding **UTF-8 without signature**. If you don't do this, there will be errors when the script runs. In Visual Studio, select the **Save As** option from the **File** menu. The **Save File As** dialog box appears. Select the **Save with Encoding** option from the drop-down menu. The following screenshot shows the **Save with Encoding** option.



7. The **Advanced Save Options** dialog box opens. Select **Unicode** (**UTF-8 without signature)**. Click **OK**. Ignore the source control warning. The following screenshot shows the completed dialog box.


Ukrainian (Mac) - Codepage 10017
Unicode (UTF-7) - Codepage 65000
Unicode (UTF-8 without signature) - Codepage 65001
US-ASCII - Codepage 20127
Vietnamese (Windows) - Codepage 1258
Wang Taiwan - Codepage 20005
Western European (DOS) - Codepage 850
Western European (IA5) - Codepage 20105
Western European (ISO) - Codepage 28591
Western European (Mac) - Codepage 10000

Unicode (UTF-8 without signature) - Codepage 65001

Line endings:

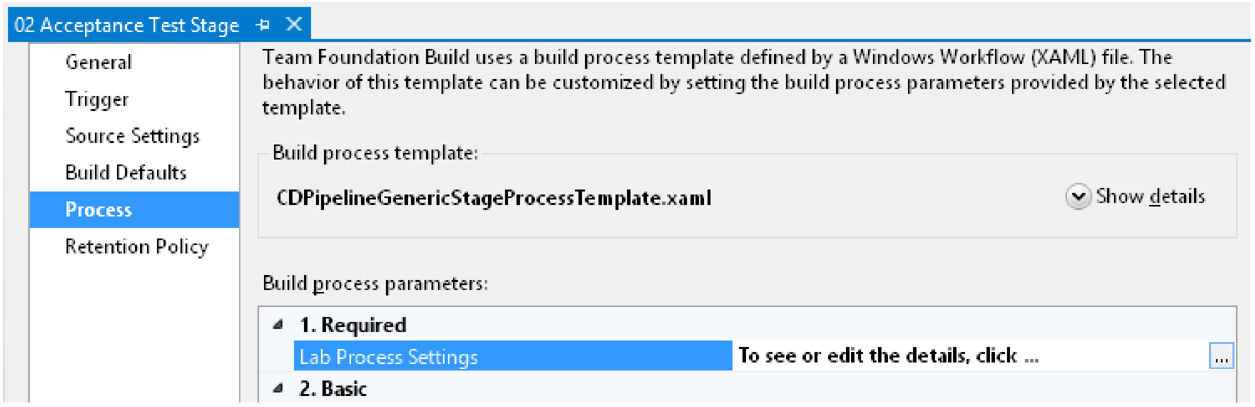
Current Setting

8. Check-in the changes. The script will be copied to the drop location during the commit stage, and be used by subsequent stages to deploy the web services.

Task 2: Configure the Pipeline Stages to Run the Deployment Script

In this task you configure the remaining stages of the pipeline to run the deployment script.

1. Edit the **02 Acceptance Test Stage** build definition so that the stage deploys to the test environment. Under the **Process** tab, select the **Lab Process Settings** parameter by clicking the **ellipses (…)**. The **Lab Workflow Parameters** dialog box opens.



Team Foundation Build uses a build process template defined by a Windows Workflow (XAML) file. The behavior of this template can be customized by setting the build process parameters provided by the selected template.

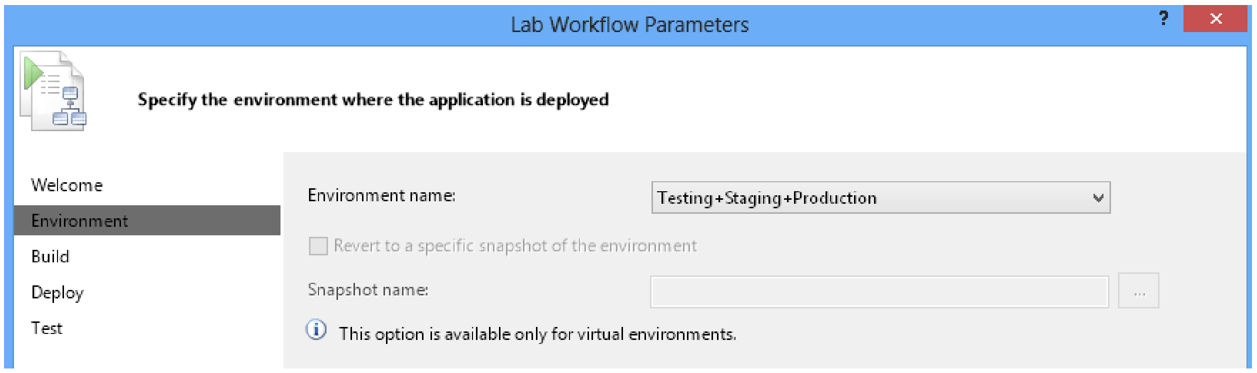
Build process template:

CDPipelineGenericStageProcessTemplate.xaml

Build process parameters:

1. Required
Lab Process Settings — To see or edit the details, click ...
2. Basic

2. Under the **Environment** tab, select the environment you created in Starting Point Lab 01 – Starting Point.


Lab Workflow Parameters

Specify the environment where the application is deployed

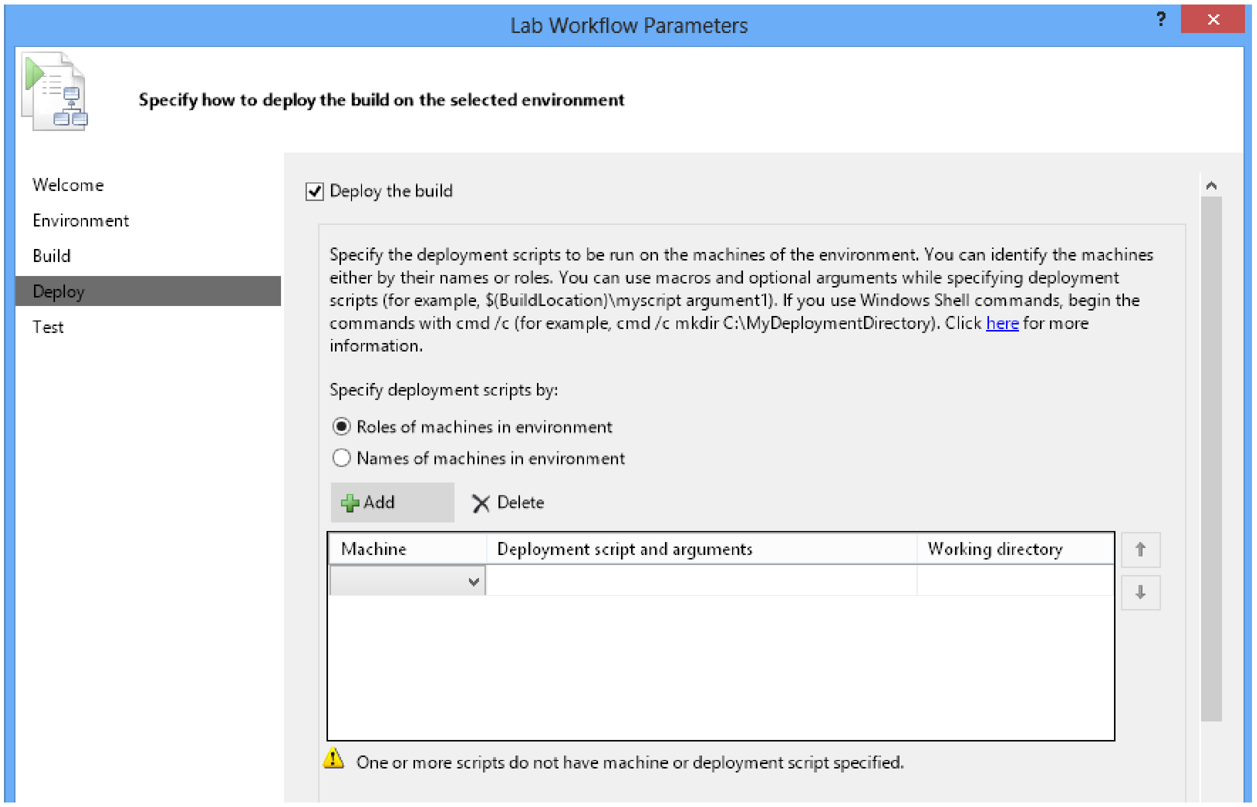
Environment name: Testing+Staging+Production

☐ Revert to a specific snapshot of the environment

Snapshot name:

This option is available only for virtual environments.

3. Select the **Deploy** tab. Select **Deploy the build**. Click **Add**. This incorporates the deployment script into the build definition.


Lab Workflow Parameters

Specify how to deploy the build on the selected environment

☑ Deploy the build

Specify the deployment scripts to be run on the machines of the environment. You can identify the machines either by their names or roles. You can use macros and optional arguments while specifying deployment scripts (for example, $(BuildLocation)\myscript argument1). If you use Windows Shell commands, begin the commands with cmd /c (for example, cmd /c mkdir C:\MyDeploymentDirectory). Click here for more information.

Specify deployment scripts by:

● Roles of machines in environment
○ Names of machines in environment

Machine	Deployment script and arguments	Working directory

One or more scripts do not have machine or deployment script specified.

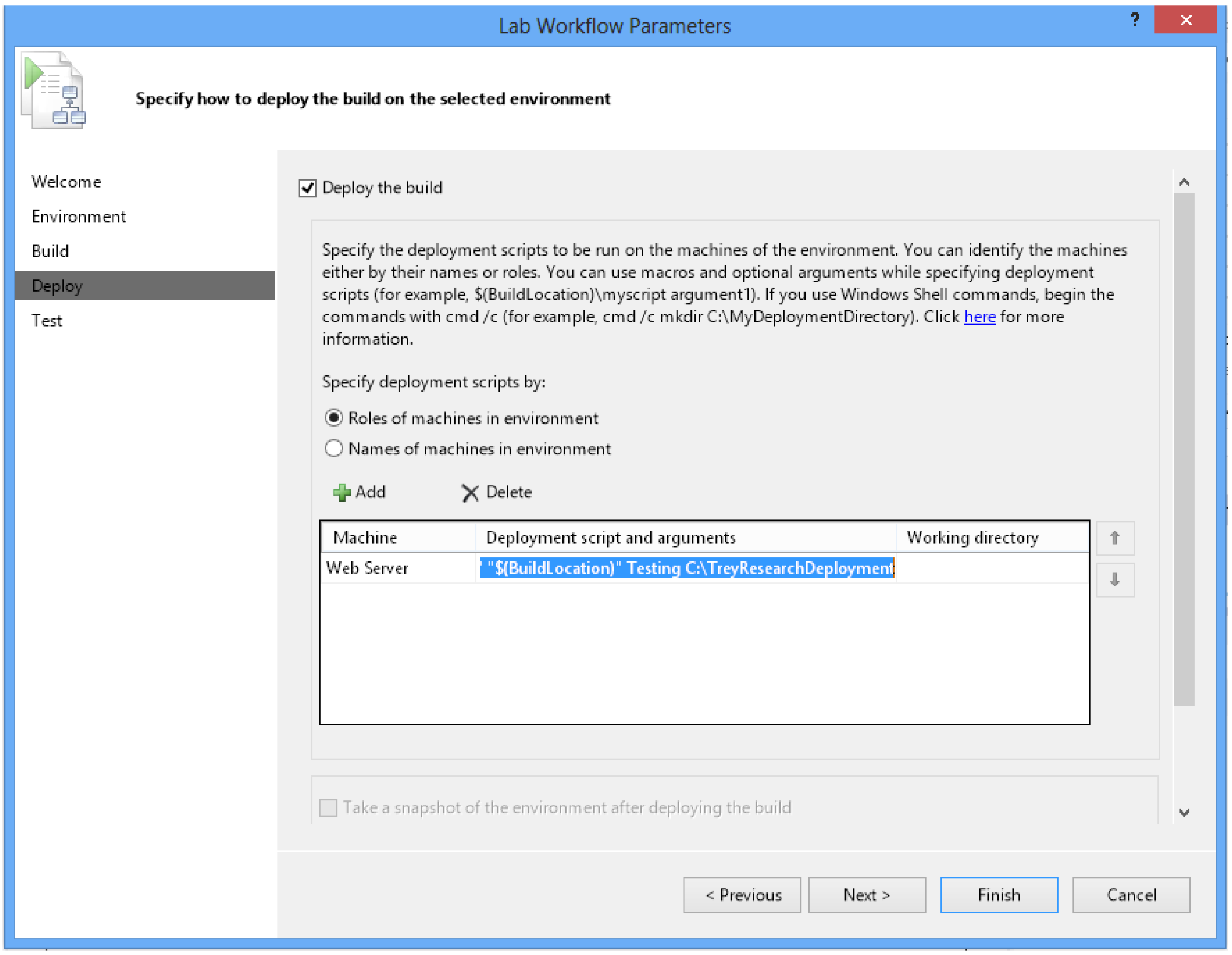
4. Select **Web Server** from the **Machine** drop-down list. This setting ensures that the deployment will be done on all the web servers in the environment (in this lab, there is a single web server).
5. In the **Deployment script and arguments** column, add the following command line, which causes the deployment script to execute.

Note: The following command is a single line.

CMD

```
"$(BuildLocation)\Release\Deployment\WcfService\DeployWcfService.cmd"
"$(BuildLocation)" Testing C:\TreyResearchDeployment
```

The deployment agent retrieves the script directly from the drop location, where it is placed by the specific instance of the pipeline. Use the **$(BuildLocation)** built-in variable to compose the path. The first parameter is the drop location. The second parameter is the name of the target environment. The third parameter is the location of the temporary folder on the target computer that is used by the deployment script. The following screenshot shows the completed dialog box, with a portion of the command.



[✓] Deploy the build

Specify the deployment scripts to be run on the machines of the environment. You can identify the machines either by their names or roles. You can use macros and optional arguments while specifying deployment scripts (for example, $(BuildLocation)\myscript argument1). If you use Windows Shell commands, begin the commands with cmd /c (for example, cmd /c mkdir C:\MyDeploymentDirectory). Click here for more information.

Specify deployment scripts by:

● Roles of machines in environment

○ Names of machines in environment

Machine	Deployment script and arguments	Working directory
Web Server	"$(BuildLocation)" Testing C:\TreyResearchDeployment	

☐ Take a snapshot of the environment after deploying the build

6. Click **Finish** and save the changes.

7. Repeat steps 1 through 6 for the **03a Release Stage** build definition. In the fifth step, the second parameter is **Production**, because the release stage deploys to the production environment. Here is the code.

CMD

```
"$(BuildLocation)\Release\Deployment\WcfService\DeployWcfService.cmd"
"$(BuildLocation)" Production C:\TreyResearchDeployment
```

8. Repeat steps 1 through 6 for the **03b UAT Stage** build definition. In the fifth step, the second parameter is **Staging**, because the UAT stage deploys to the staging environment. Here is the code.

CMD

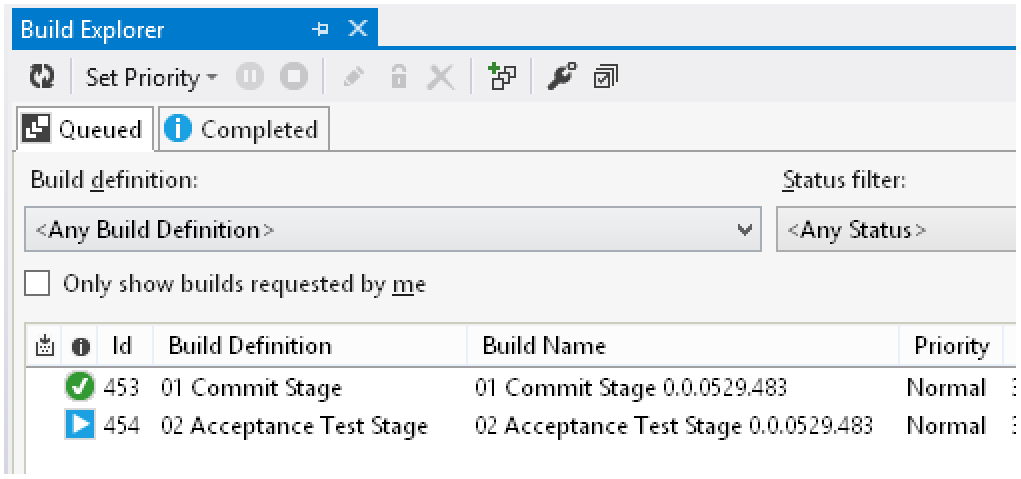
```
"$(BuildLocation)\Release\Deployment\WcfService\DeployWcfService.cmd"
"$(BuildLocation)" Staging C:\TreyResearchDeployment
```

Exercise 4: Testing the Automated Deployment

In this exercise you test to see if the automated deployment works correctly.

1. Create a new instance of the pipeline by running the **01 Commit Stage**. You can either make some changes in the code and check them in, or, in Team Explorer, queue the build definition.

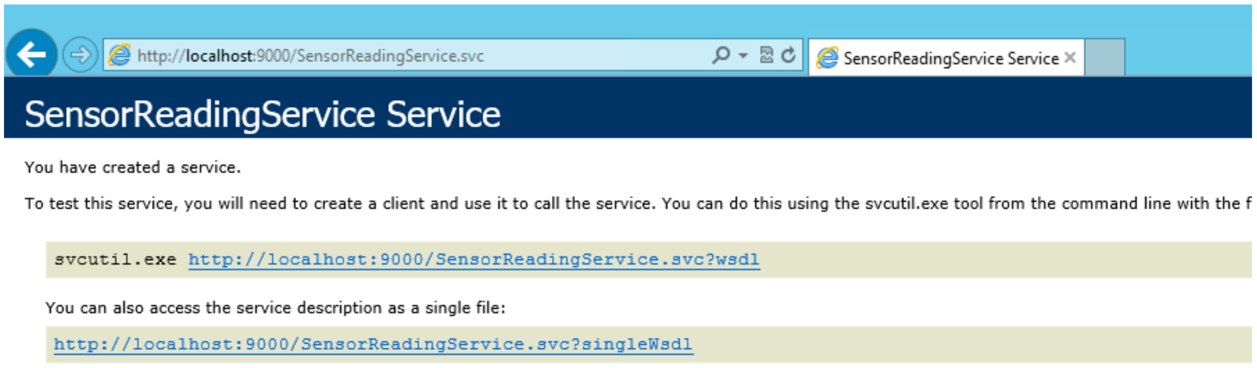
2. After the commit stage finishes successfully, the changes are propagated through the pipeline, so **02 Acceptance Test Stage** is triggered automatically. You can monitor the progress from the Build Explorer window.



Build definition: <Any Build Definition>
Status filter: <Any Status>
Only show builds requested by me

		Id	Build Definition	Build Name	Priority
	✓	453	01 Commit Stage	01 Commit Stage 0.0.0529.483	Normal
	▶	454	02 Acceptance Test Stage	02 Acceptance Test Stage 0.0.0529.483	Normal

3. After the acceptance stage finishes successfully, the web services should be deployed to the test environment. In the browser, navigate to the URL for the web service to verify this.


http://localhost:9000/SensorReadingService.svc
SensorReadingService Service

SensorReadingService Service

You have created a service.

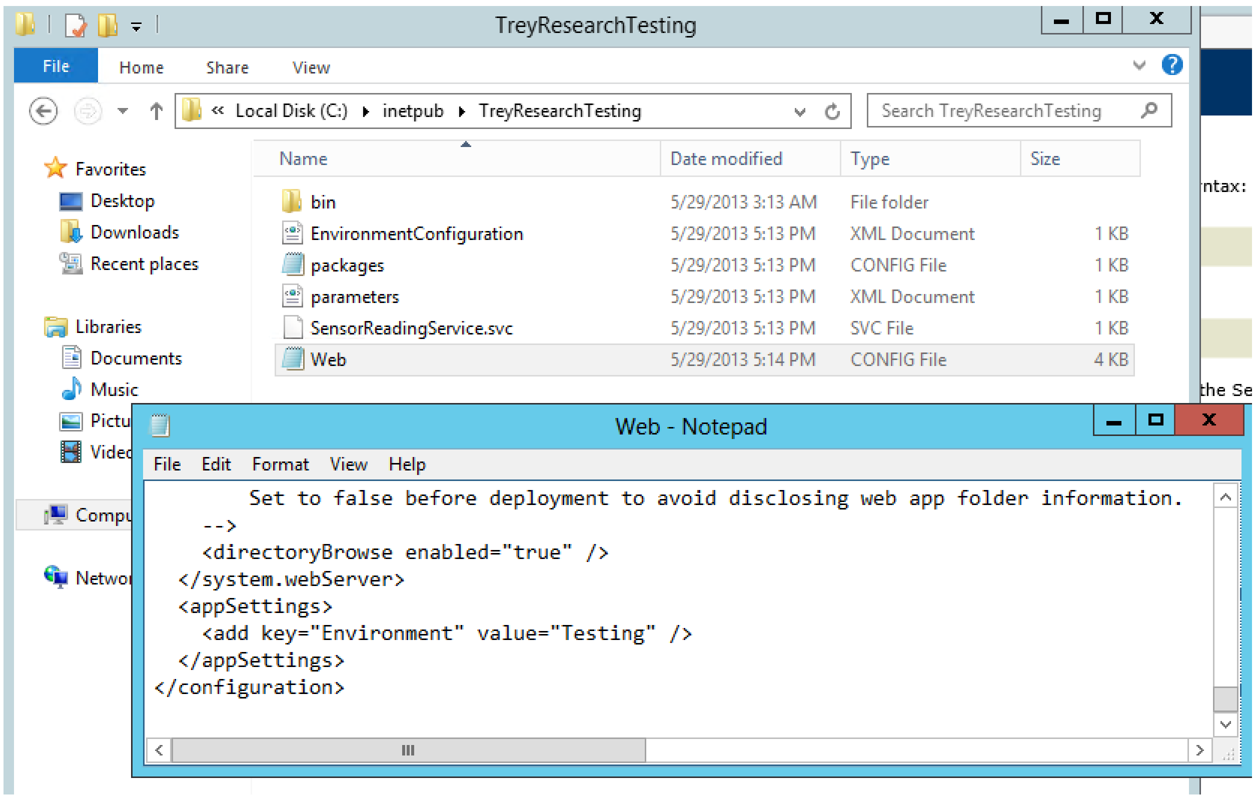
To test this service, you will need to create a client and use it to call the service. You can do this using the svcutil.exe tool from the command line with the f

```
svcutil.exe http://localhost:9000/SensorReadingService.svc?wsdl
```

You can also access the service description as a single file:

```
http://localhost:9000/SensorReadingService.svc?singleWsdl
```

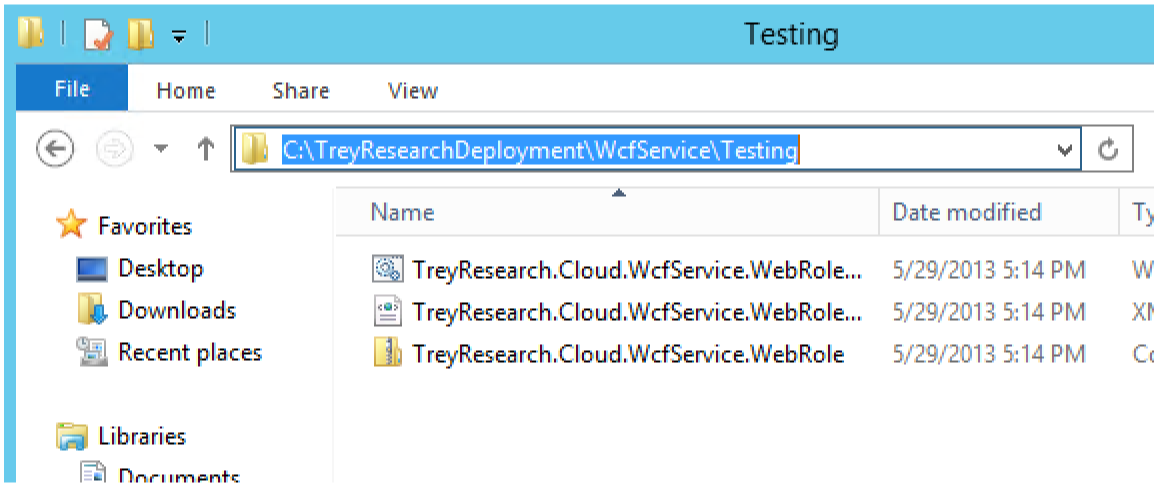
4. You can also open the Web.config file that was deployed and check that the configuration settings match the test environment.



Name	Date modified	Type	Size
bin	5/29/2013 3:13 AM	File folder	
EnvironmentConfiguration	5/29/2013 5:13 PM	XML Document	1 KB
packages	5/29/2013 5:13 PM	CONFIG File	1 KB
parameters	5/29/2013 5:13 PM	XML Document	1 KB
SensorReadingService.svc	5/29/2013 5:13 PM	SVC File	1 KB
Web	5/29/2013 5:14 PM	CONFIG File	4 KB

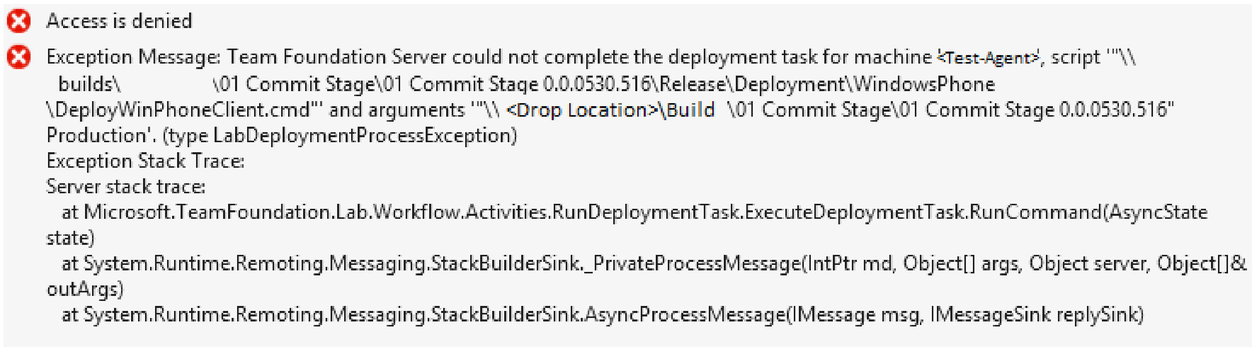
```
        Set to false before deployment to avoid disclosing web app folder information.
    -->
    <directoryBrowse enabled="true" />
  </system.webServer>
  <appSettings>
    <add key="Environment" value="Testing" />
  </appSettings>
</configuration>
```

5. Finally, you can verify that the deployment folder was created at **C:\TreyResearchDeployment** and that it contains the files that are used for the deployment.



Name	Date modified	Ty
TreyResearch.Cloud.WcfService.WebRole...	5/29/2013 5:14 PM	W
TreyResearch.Cloud.WcfService.WebRole...	5/29/2013 5:14 PM	XI
TreyResearch.Cloud.WcfService.WebRole	5/29/2013 5:14 PM	C

NOTE: Depending on the configuration of your environment and network, you may get an **Access is denied** error when the Lab Management agent tries to run the deployment script from the drop location (see the screenshot below this note for an example). This is because the Visual Studio Lab Agent service is configured by default to run under the Local System account, and it is not being impersonated by the test controller that uses the Lab service account and that accesses the drop location. The easiest workaround is to grant read permissions to the computer account where the agent is running (DOMAIN\COMPUTERNAME$) both for the network share and for the folder used as the drop location.



```
Access is denied
Exception Message: Team Foundation Server could not complete the deployment task for machine '<Test-Agent>', script '"\\
  builds\            \01 Commit Stage\01 Commit Stage 0.0.0530.516\Release\Deployment\WindowsPhone
\DeployWinPhoneClient.cmd"' and arguments '"\\ <Drop Location>\Build  \01 Commit Stage\01 Commit Stage 0.0.0530.516"
Production'. (type LabDeploymentProcessException)
Exception Stack Trace:
Server stack trace:
   at Microsoft.TeamFoundation.Lab.Workflow.Activities.RunDeploymentTask.ExecuteDeploymentTask.RunCommand(AsyncState
state)
   at System.Runtime.Remoting.Messaging.StackBuilderSink._PrivateProcessMessage(IntPtr md, Object[] args, Object server, Object[]&
outArgs)
   at System.Runtime.Remoting.Messaging.StackBuilderSink.AsyncProcessMessage(IMessage msg, IMessageSink replySink)
```

6. At this point, if you want to perform UAT, queue the **03b UAT Stage** build definition and provide the values for the pipeline instance and the drop location. The services will be automatically deployed to the staging environment.

Note: These values are required. If you don't supply them an error occurs.

7. You can perform the same verifications that you did in steps 3 and 4 for the acceptance test stage.
8. When you ready to release to production, manually trigger **03a Release Stage**.

Summary

In this lab you automated the deployment of the WCF web service. To do this, you prepared the configuration files that set up the environments. There is a base configuration file and three other files that store the differences between the base configuration and the environments. You made those three files dependent on the base file. Finally, you added the code that performs the transformations.

You packaged all the files required for the deployment, using the MSDeploy package format. You configured the packaging process so that the Web.config parameters were available and could be changed. You then created a publication profile, and configured the pipeline's commit stage to generate the package.

You then configured the pipeline to perform the deployment automatically by creating a Lab Management deployment script. Then, you edited the build definitions for the pipeline stages (other than the commit stage) to deploy to the correct environment. Finally, you tested the automated deployment.

Copyright