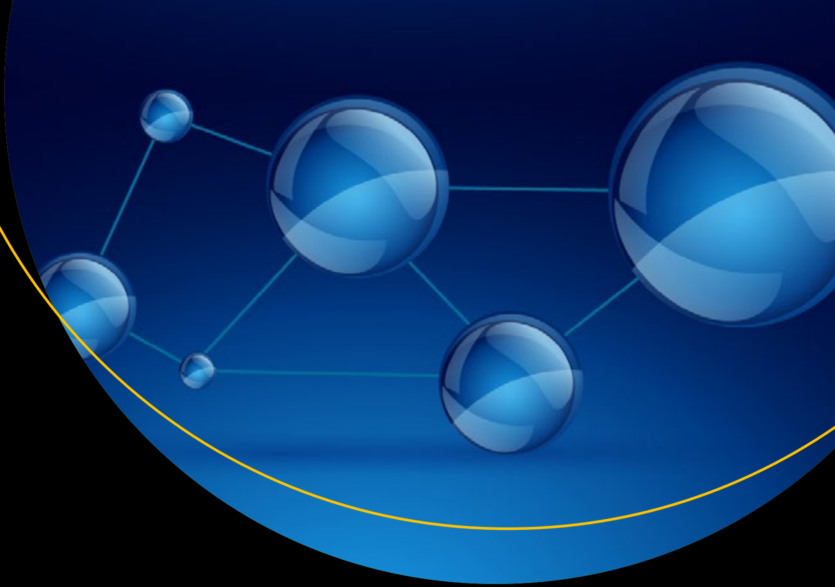


DevOps for Azure Applications

Deploy Web Applications on Azure

Suren Machiraju
Suraj Gaurav

Apress®

Table of Contents

CHAPTER 1

DevOps for Azure

DevOps is all about automating the application deployment process. It addresses the drawbacks associated with manual application deployment. The application deployment process contains several steps from writing code to deploying the created release to the target environment, i.e., Microsoft Azure Cloud. This chapter discusses the need for DevOps, the DevOps functions, the application deployment process, and the DevOps tools.

The Need for DevOps

Traditionally, the software development lifecycle warranted siloed teams taking on specific tasks, i.e., the development team and the operations team. The developers were responsible for writing code, checking in source code into source control, testing code, QA of code, and staging for deployment. The Operations/Production team was responsible for deploying the code to servers and thereafter coordinating with customers and providing feedback to developers. Such siloed efforts were mostly manual processes with a small degree of siloed application/software deployment work. This manual process had several drawbacks, some of which are as follows:

- The communication gap between different teams results in resentment and blame, which in turn delays fixing errors.
- e entire process took a long time to complete.

' Suren Machiraju, Suraj Gaurav 2018
S. Machiraju and S. Gaurav, DevOps for Azure Applications,
https://doi.org/10.1007/978-1-4842-3643-7_1

e nal product did not meet all required criteria.

Some tools could not be implemented on the production server for security reasons.

The communication barriers slowed down performance and added to inefficiency.

To cope with these drawbacks, a push for automation arose, leading to the development of DevOps. DevOps is a combination of two terms and two teams namely Developers and Operations. As the name indicates, it integrates the functionality of both of these teams (Developers and Operations/Production) in the application development and deployment process.

Describing the Functions of DevOps

The basic functions of DevOps are as follows:

Automates the entire process of application deployment. As a result, the entire process is straightforward and streamlined.

Allows multiple developers to check in and check out code simultaneously in/from the Source repository.

Provides a Continuous Integration (CI) server that pools the code from the Source repository and prepares the build by running and passing the unit tests and functional tests automatically.

Automates testing, integration, deployment, and monitoring tasks.

Automates work ows and infrastructure.

Enhances productivity and collaboration through continuous measurement of application performance.

Allows for rapid and reliable build, test, and release operations of the entire software development process.

DevOps Application Deployment Process

The entire application deployment process is shown in Figure 1-1.

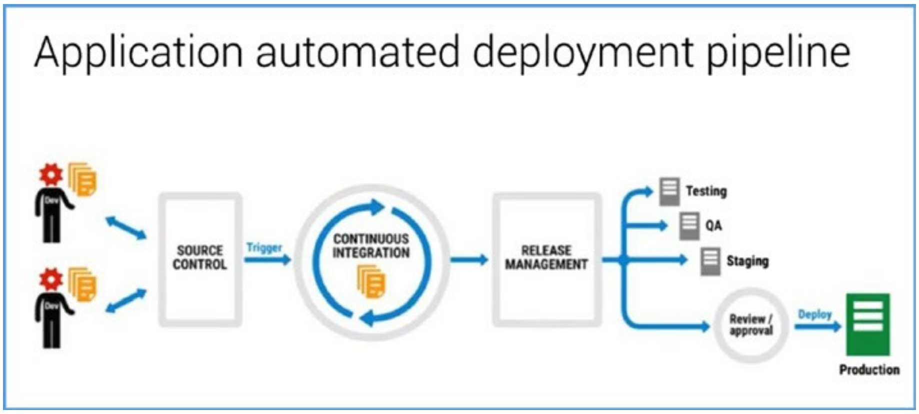


Figure 1-1. The application deployment process

Let s now review the various steps in the application deployment process:

1. Developers write code.
2. Code is checked in to the source control/Source repository.

The development team must first fix the bug and check in the code again. The code goes through the same process of generating the build and release until the code passes all tests.

Figure 1-2 shows the release management process.

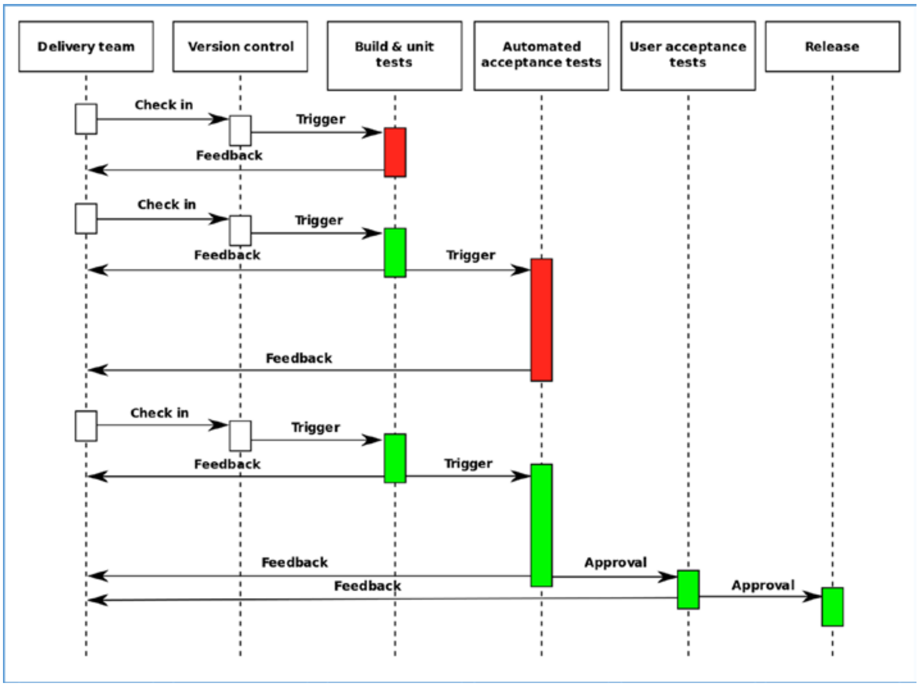


Figure 1-2. Release management process

6. e last step in the process is deploying the created release to the target environment Microsoft Azure Cloud (`https://azure.microsoft.com`). Once the deployment is complete, all changes in the code are live for users of the target environment in Azure.

Understanding DevOps Tools

There are several DevOps tools available that can help you develop an effective automated environment. You can also use separate tools for performing specific operations in DevOps. A list of tools, based on the broad level functionality, follows. Note that to demonstrate the DevOps principles, we selected a set of tools to use as an example.

Build automation tools: These tools automate the process of creating a software build, compiling source code, and packaging the code. Some build automation tools are:

- Apache Ant (`https://ant.apache.org/bindownload.cgi`)
- Apache Maven (`https://maven.apache.org/download.cgi`)
- Boot (`http://boot-clj.com/`)
- Gradle (`https://gradle.org/`)
- Grunt (`https://gruntjs.com/`)
- MSBuild (`https://www.microsoft.com/en-in/download/details.aspx?id=48159`)
- Waf (`https://waf.io/`)

Continuous Integration tools: ese tools create builds and run tests automatically when the code changes are checked in to the central repository. Some CI tools are:

- Bamboo (`https://www.atlassian.com/software/bamboo/download`)
- Buildbot (`https://buildbot.net/`)

Microsoft Visual Studio Team Services (VSTS) (`https://www.visualstudio.com/team-services/`). We focus on this tool in this book.

AWS CodePipeline (`https://aws.amazon.com/codepipeline/getting-started/`)

With a basic understanding of the fundamentals, you re ready to move forward and dive deeper into the specifics. We start by discussing stand-alone tools, and thereafter discuss an all-in-one integrated platform.

Summary

This chapter discussed the importance of DevOps over the manual process of application deployment. DevOps integrates the functionality of both teams (Developers and Operations/Production) in the application development and deployment process. This chapter provided information about the basic functions of DevOps. The entire process of application deployment was discussed. Toward the end of the chapter, a list of DevOps tools was provided.

CHAPTER 2

Deployment via TeamCity and Octopus Deploy

As discussed in the previous chapter, application deployment in DevOps requires a Continuous Integration (CI) tool and Continuous Delivery (CD) tool/release management software to automate the entire process. Currently, there are several tools available in the market. This chapter discusses three best-of-breed tools TeamCity as a CI tool, Octopus Deploy as a release management tool, and CD software to deploy the package on the Azure web application. Since different vendors deliver these best-of-breed tools, there is some complexity involved in integrating them into a single solution.

Introduction to Microsoft Public Cloud, Azure

Before we delve into the DevOps tools, let s recap the deployment environment. As a reminder, we are focusing on Microsoft Azure. However, be assured that information from this chapter can be applied to other public cloud solutions.

' Suren Machiraju, Suraj Gaurav 2018
S. Machiraju and S. Gaurav, DevOps for Azure Applications,
https://doi.org/10.1007/978-1-4842-3643-7_2

Azure has the capability to host applications. These applications can be further integrated with other applications and services on the Azure platform rather easily. Azure s integration features provide customers with enhanced business agility and efficiency. They help users deploy the source code to multiple Azure websites.

Understanding TeamCity

TeamCity is a CI server for developers and is powered by JetBrains. It provides several relevant features:

- Supports different platforms/tools/languages
- Automates the build and deployment processes
- Enhances quality and standards across teams
- Works as an artifact and NuGet repository
- Provides a reporting and statistics feature

Definition According to Martin Fowler, Continuous Integration is a software development practice in which developers commit code changes into a shared repository several times a day. Each commit is followed by an automated build to ensure that new changes integrate well into the existing code base and to detect problems early.

Basic Concepts of TeamCity

Here are the basic concepts of TeamCity:

- Project: Refers to a set of build configurations.
- Build con guration: Refers to a collection of settings (VCS roots, build steps, and build triggers) that de ne a build procedure.

Configuring a Build in TeamCity

In this section, we configure arguments for the PowerShell script in TeamCity. This will enable TeamCity to execute the PowerShell script. For this scenario, we created a PowerShell script named `[string]App.Ps1`.

The build configuration uses a step-oriented approach, which is outlined in the following sections.

Step 1: Creating a Project

To configure a build in TeamCity, first create a project. There are several options available for this task, as follows:

- Manually
- Pointing to a repository URL
- Pointing to a GitHub.com repository
- Pointing to a Bitbucket Cloud repository

Perform the following steps to create a standard project:

1. Click the Administration link in the top-right corner of the Administration area.
2. Click the down arrow button beside the Create Project button. A drop-down list appears.
3. Select the Manually option from the drop-down list to create a project manually. After you click the Manual option, the Create New Project page appears.
4. Enter the desired name of the project in the Name text box.
5. Enter the desired ID of the project in the Project ID text box.

6. Enter the desired description of the project in the Description text box.

7. Click the Create button to create the project.

Now, the project has been created.

Step 2: Creating a Build Configuration

Build configurations describe the process by which a project s sources are fetched and built. Once the project is created, TeamCity prompts you to create build configurations. Alternatives to create build configurations are as follows:

Manually

Pointing to a repository URL

Pointing to a GitHub.com repository

Pointing to a Bitbucket Cloud repository

Perform the following steps to create a build configuration manually:

1. Click the down arrow button beside the Create Build Configuration button. A drop-down list appears.

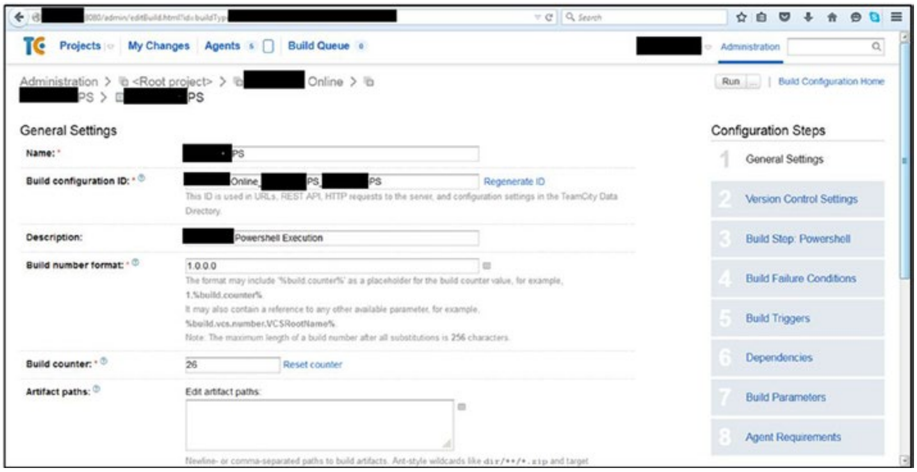
2. Select the Manual option from the drop-down list to create the build con guration manually.

3. Specify the name of the build con guration in the Name text box.

4. Specify the build con guration ID in the Build Con guration ID text box.

5. Specify the desired description in the Description text box.

6. Click the Save button. Figure 2-1 shows the General Settings page.



Administration > <Root project> > [redacted] Online > [redacted] PS > [redacted] PS

General Settings

Name: * [redacted] PS

Build configuration ID: * [redacted] Online_[redacted] PS_[redacted] PS Regenerate ID
This ID is used in URLs, REST API, HTTP requests to the server, and configuration settings in the TeamCity Data Directory.

Description: [redacted] Powershell Execution

Build number format: * 1.0.0.0
The format may include '%build.counter%' as a placeholder for the build counter value, for example, 1.%build.counter%
It may also contain a reference to any other available parameter, for example, %build.vcs.number.VCSRootName%.
Note: The maximum length of a build number after all substitutions is 256 characters.

Build counter: * 26 Reset counter

Artifact paths: Edit artifact paths:
Newline- or comma-separated paths to build artifacts. Ant-style wildcards like dir/**/*.zip and target

Configuration Steps
1 General Settings
2 Version Control Settings
3 Build Step: Powershell
4 Build Failure Conditions
5 Build Triggers
6 Dependencies
7 Build Parameters
8 Agent Requirements

Figure 2-1. General Settings page

Step 3: Configuring the Version Control Settings

In this step, we provide settings related to the VCS root. The VCS root describes a connection to a version control system, and there are several settings associated with it. These settings allow VCS to communicate with TeamCity. They define the way changes are monitored and sources are specified for a build. Perform the following steps to configure the version control settings:

1. Select the Version Control Settings tab.
2. Click the Attach VCS Root button. e New VCS Root page appears.
3. Select the desired type of VCS from the Type of VCS drop-down list. We selected Subversion.
4. Specify a unique VCS root name in the VCS Root Name text box.
5. Specify a unique VCS root ID in the VCS Root ID text box.

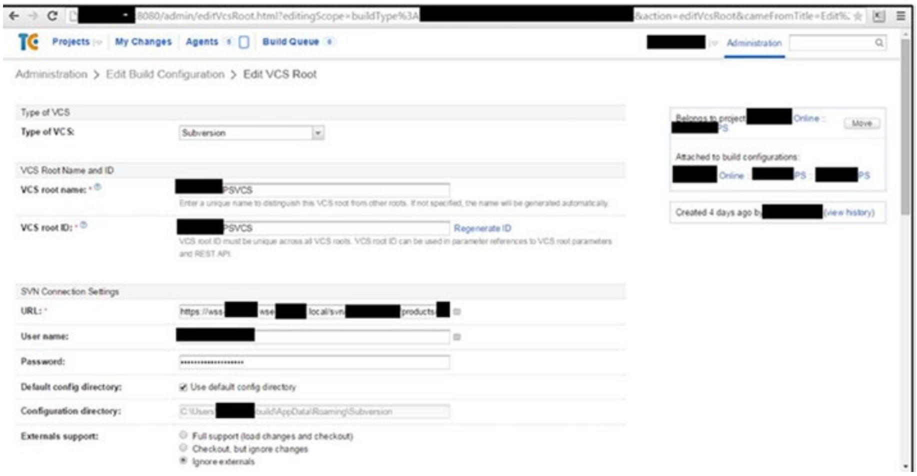
The connection settings appear on the page depending on the type of VCS selected. In our case, the SVN Connection Settings section appears.

6. Specify the repository URL in the URL text box.

7. To allow TeamCity to communicate with the Source repository, specify the username and password in the Username and Password text boxes, respectively.

8. Click the Test Connection button to test the connection. is validates that TeamCity can communicate with the repository. A Test Connection message box appears with the `Connection Successful` message. If the connection shows failure, check the speci ed URL and the credentials.

9. Click the Create button. Figure 2-2 shows the settings for the New VCS Root page.


[redacted]8080/admin/editVcsRoot.html?editingScope=buildType%3A[redacted]&action=editVcsRoot&cameFromTitle=Edit%

Projects | My Changes | Agents | Build Queue
[redacted] Administration

Administration > Edit Build Configuration > Edit VCS Root

Type of VCS
Type of VCS: Subversion

VCS Root Name and ID
VCS root name: * [redacted]PSVCS
Enter a unique name to distinguish this VCS root from other roots. If not specified, the name will be generated automatically.
VCS root ID: * [redacted]PSVCS Regenerate ID
VCS root ID must be unique across all VCS roots. VCS root ID can be used in parameter references to VCS root parameters and REST API.

SVN Connection Settings
URL: * https://wss[redacted]wse[redacted]local/svn[redacted]products[redacted]
User name: [redacted]
Password: ••••••••••••••••••
Default config directory: ✓ Use default config directory
Configuration directory: C:\Users\[redacted]build\AppData\Roaming\Subversion
Externals support: ○ Full support (load changes and checkout)
○ Checkout, but ignore changes
● Ignore externals

Belongs to project [redacted] Online : [redacted]PS Move
Attached to build configurations:
[redacted] Online : [redacted]PS : [redacted]PS
Created 4 days ago by[redacted](view history)

Figure 2-2. Settings for the New VCS Root page

Step 4: Configuring the Build Steps

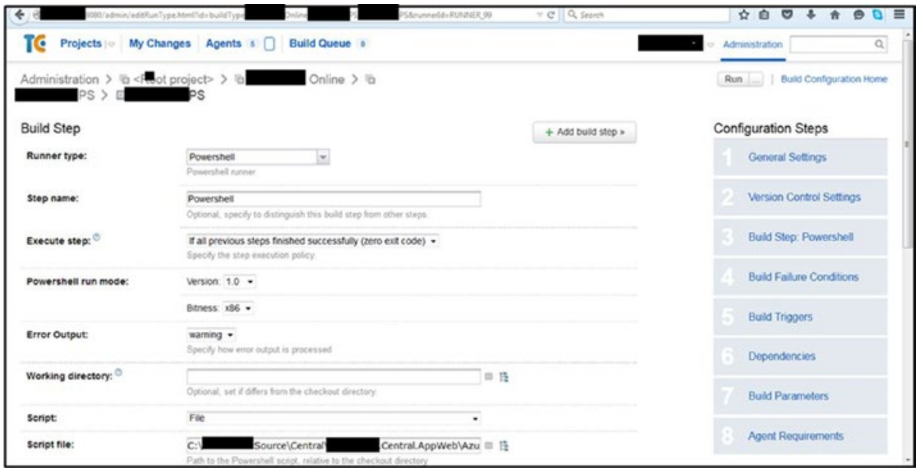
Once the VCS root is created, we can configure the build steps. Perform the following steps to add a build step:

1. Select the Build Steps tab.
2. Click the Add Build Step button. e Build Step page appears.
3. Select the PowerShell option from the Runner Type drop-down list.

Note In this example, we use the PowerShell script file named `[string]App.ps1`. This file compiles the source code.

---

4. Specify the desired step name in the Step Name text box.
5. Select the desired step execution policy from the Execute Step drop-down list.
6. Select the File option from the Script drop-down list.
7. Specify the path to the PowerShell script in the Script File box. This field contains the physical path mapped to the `[string]App.ps1` script, which is located on the build agent, as shown in Figure 2-3.


[redacted]080/admin/editRunType.html?id=buildType[redacted]Online[redacted]PS[redacted]PS&runnerId=RUNNER_99

Administration > <R[redacted]ot project> > [redacted] Online > [redacted]PS > [redacted]PS

Build Step

Runner type:	Powershell Powershell runner
Step name:	Powershell Optional, specify to distinguish this build step from other steps.
Execute step:	If all previous steps finished successfully (zero exit code) Specify the step execution policy
Powershell run mode:	Version: 1.0 Bitness: x86
Error Output:	warning Specify how error output is processed
Working directory:	Optional, set if differs from the checkout directory
Script:	File
Script file:	C:\[redacted]Source\Central\[redacted]Central.AppWeb\Azu Path to the Powershell script, relative to the checkout directory

Configuration Steps
1 General Settings
2 Version Control Settings
3 Build Step: Powershell
4 Build Failure Conditions
5 Build Triggers
6 Dependencies
7 Build Parameters
8 Agent Requirements

Figure 2-3. Creating a build step

8. Specify the PowerShell script execution mode in the Script Execution Mode option.

9. Enter script arguments in the Script Arguments section. We entered ve arguments that will be passed to the `[string]App.ps1` script during execution by TeamCity.

ARGUMENTS PASSED TO THE POWERSHELL SCRIPT

All arguments should be explained in terms of their relative paths. Descriptions of all the arguments passed to the PowerShell script follow:

`..\Workflow`: Allows the PowerShell script to access the contents of the `Workflow` folder.

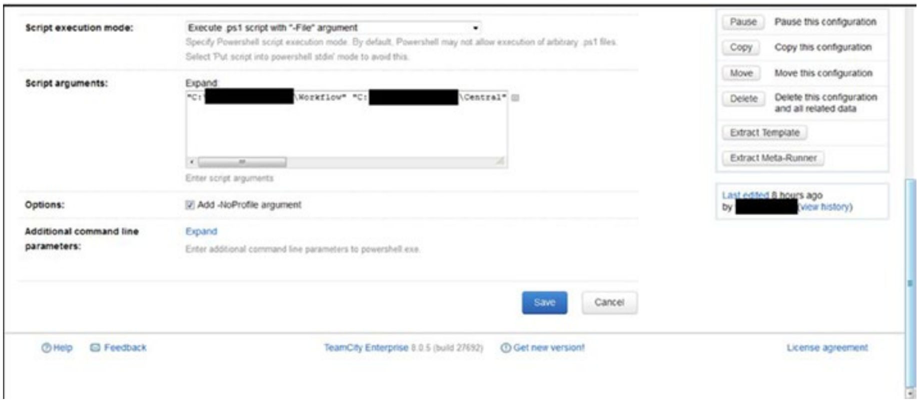
`..\Central`: Allows the PowerShell script to access the contents of the `Central` folder.

`..\Server`: Allows the PowerShell script to access the contents of the `Server` folder.

`Nuget.exe`: Allows the PowerShell script to load the `Nuget.exe` file, which is located on the build agent.

v. `Targetfolder`: Specifies the path of a folder on the build agent where the compiled code is placed.

10. Click the Save button, as shown in Figure 2-4.



Script execution mode: Execute .ps1 script with "-File" argument

Specify Powershell script execution mode. By default, Powershell may not allow execution of arbitrary .ps1 files. Select 'Put script into powershell stdin' mode to avoid this.

Script arguments: Expand

`"C:\[redacted]\Workflow" "C:[redacted]\Central"`

Enter script arguments

Options: ☑ Add -NoProfile argument

Additional command line parameters: Expand

Enter additional command line parameters to powershell.exe.

Pause — Pause this configuration

Copy — Copy this configuration

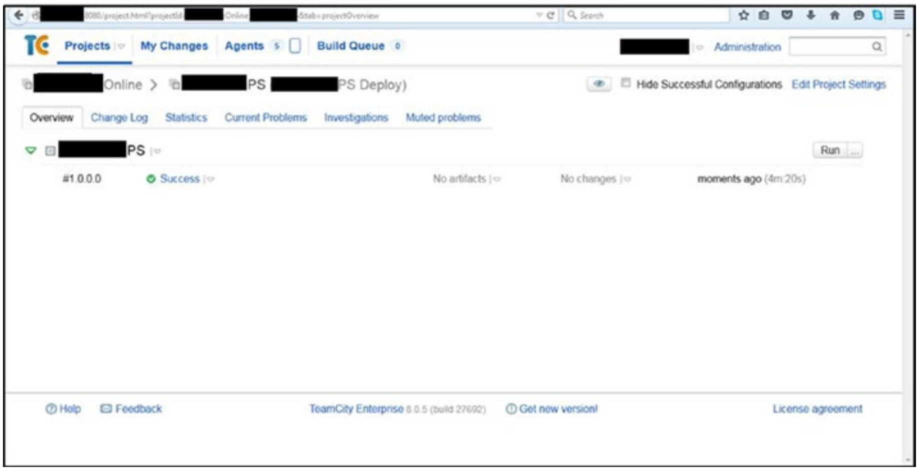
Move — Move this configuration

Delete — Delete this configuration and all related data

Last edited 8 hours ago by [redacted] (view history)

Figure 2-4. Saving the build step

A successful build is created in TeamCity, which is executable through the PowerShell script `[string]App.ps1`, as shown in Figure 2-5.

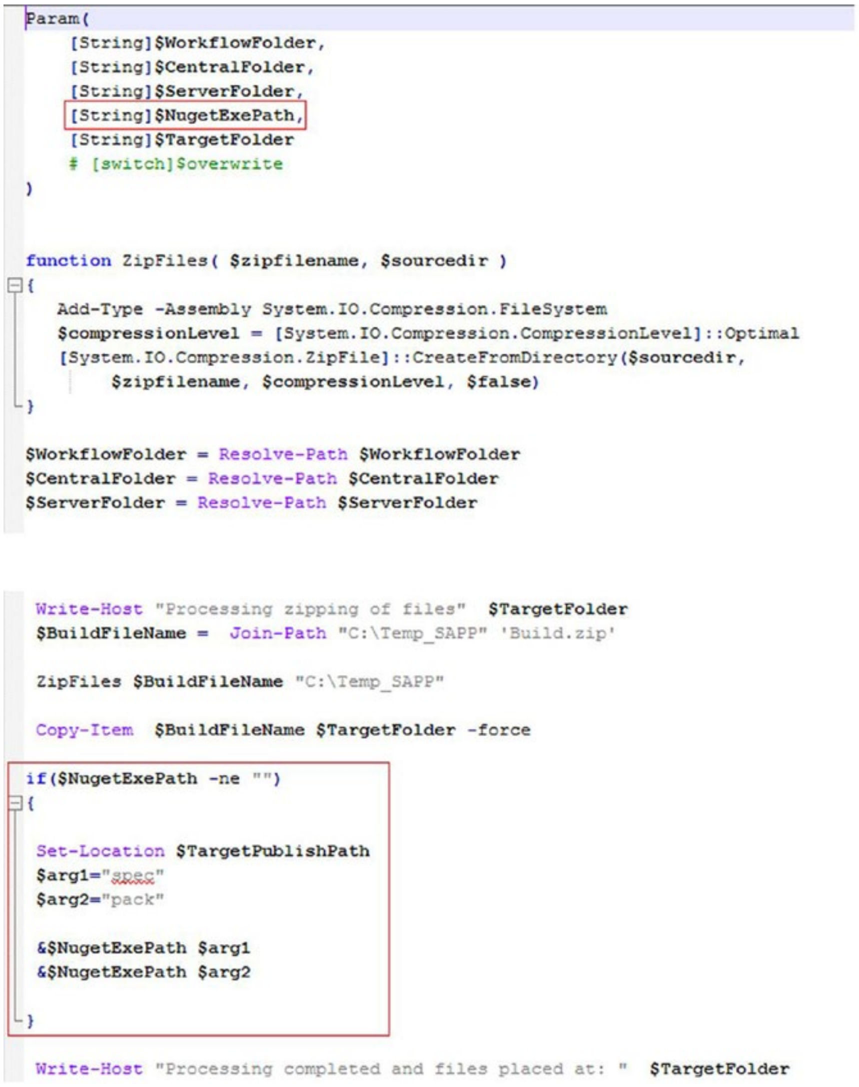


#1.0.0.0 Success No artifacts No changes moments ago (4m:20s)

Figure 2-5. Successful build message

Creating a Package

Once TeamCity creates a successful build, changes may need to be made to the PowerShell script (`[string]App.ps1`). For example, we may need to make changes to `NugetExePath` to accept a new argument, as shown in Figure 2-6.

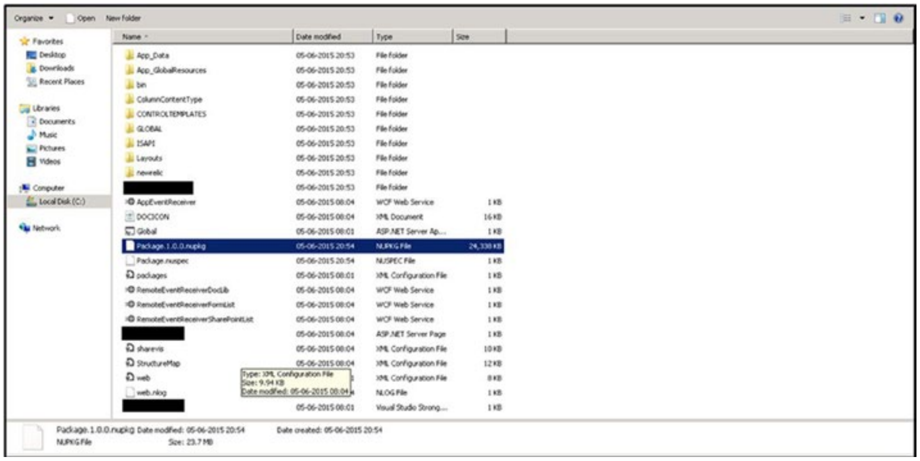


```
Param(
    [String]$WorkflowFolder,
    [String]$CentralFolder,
    [String]$ServerFolder,
    [String]$NugetExePath,
    [String]$TargetFolder
    # [switch]$overwrite
)


function ZipFiles( $zipfilename, $sourcedir )
{
   Add-Type -Assembly System.IO.Compression.FileSystem
   $compressionLevel = [System.IO.Compression.CompressionLevel]::Optimal
   [System.IO.Compression.ZipFile]::CreateFromDirectory($sourcedir,
        $zipfilename, $compressionLevel, $false)
}

$WorkflowFolder = Resolve-Path $WorkflowFolder
$CentralFolder = Resolve-Path $CentralFolder
$ServerFolder = Resolve-Path $ServerFolder


 Write-Host "Processing zipping of files"  $TargetFolder
 $BuildFileName =  Join-Path "C:\Temp_SAPP" 'Build.zip'

 ZipFiles $BuildFileName "C:\Temp_SAPP"

 Copy-Item  $BuildFileName $TargetFolder -force

if($NugetExePath -ne "")
{

 Set-Location $TargetPublishPath
 $arg1="spec"
 $arg2="pack"

 &$NugetExePath $arg1
 &$NugetExePath $arg2

}

 Write-Host "Processing completed and files placed at: "  $TargetFolder
```

Figure 2-6. Making changes to NugetExePath

The changes made to the PowerShell script create a package in the target folder.

Figure 2-7 shows the created NuGet package.



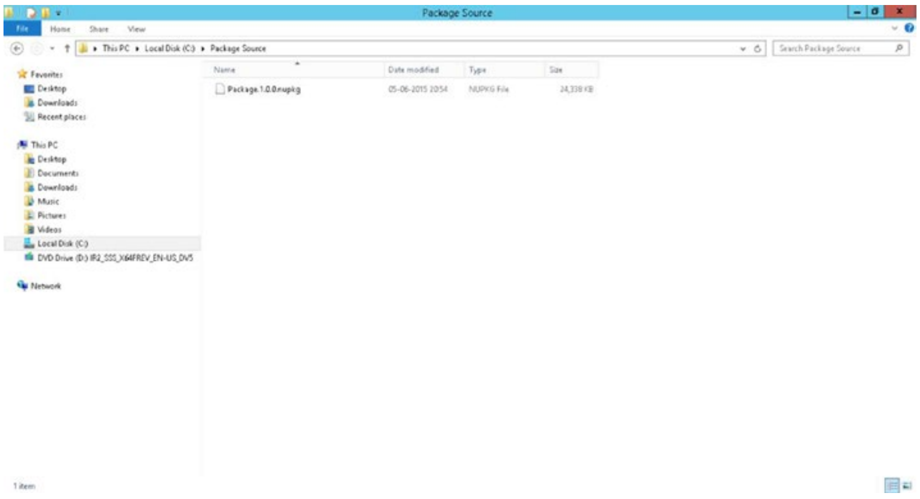
Name	Date modified	Type	Size
App_Data	05-06-2015 20:53	File Folder	
App_GlobalResources	05-06-2015 20:53	File Folder	
bin	05-06-2015 20:53	File Folder	
ColumnContentType	05-06-2015 20:53	File Folder	
CONTROLTEMPLATES	05-06-2015 20:53	File Folder	
GLOBAL	05-06-2015 20:53	File Folder	
ISAPI	05-06-2015 20:53	File Folder	
Layouts	05-06-2015 20:53	File Folder	
newrelic	05-06-2015 20:53	File Folder	
[redacted]	05-06-2015 20:53	File Folder	
AppEventReceiver	05-06-2015 08:04	WCF Web Service	1 KB
DOCICON	05-06-2015 08:04	XML Document	16 KB
Global	05-06-2015 08:01	ASP.NET Server Ap...	1 KB
Package.1.0.0.nupkg	05-06-2015 20:54	NUPKG File	24,338 KB
Package.nuspec	05-06-2015 20:54	NUSPEC File	1 KB
packages	05-06-2015 08:01	XML Configuration File	1 KB
RemoteEventReceiverDocLib	05-06-2015 08:04	WCF Web Service	1 KB
RemoteEventReceiverFormList	05-06-2015 08:04	WCF Web Service	1 KB
RemoteEventReceiverSharePointList	05-06-2015 08:04	WCF Web Service	1 KB
[redacted]	05-06-2015 08:04	ASP.NET Server Page	1 KB
sharevis	05-06-2015 08:04	XML Configuration File	10 KB
StructureMap	05-06-2015 08:04	XML Configuration File	12 KB
web	[illegible]	XML Configuration File	8 KB
web.nlog	[illegible]	NLOG File	1 KB
[redacted]	05-06-2015 08:01	Visual Studio Strong...	1 KB

Type: XML Configuration File
Size: 9.94 KB
Date modified: 05-06-2015 08:04

Package.1.0.0.nupkg Date modified: 05-06-2015 20:54 Date created: 05-06-2015 20:54
NUPKG File Size: 23.7 MB

Figure 2-7. The NuGet package

Copy this NuGet package from the build agent to where it will be imported into the Octopus server for deployment purposes, as shown in Figure 2-8.



Name	Date modified	Type	Size
Package.1.0.0.nupkg	05-06-2015 20:54	NUPKG File	24,338 KB

Figure 2-8. NuGet package ready for deployment

Using Octopus Deploy

Octopus Deploy is a deployment server (or release management software) that automates the deployment of different applications into different environments. It makes this process effortless.

Octopus Deploy automates the deployment of:

- ASP.NET web applications
- Java applications
- Database updates
- NodeJS applications
- Custom scripts

Octopus Deploy supports the following environments:

- Development
- Test
- Production

Octopus Deploy provides a consistent deployment process to support the deployment needs of team members; an Octopus user can define a process for deploying the software. The Octopus user can specify different environments for different applications and can set privileges for different team members to deploy to different environments. For example, a team member can be authorized to deploy to a test environment while also being restricted to the production deployment.

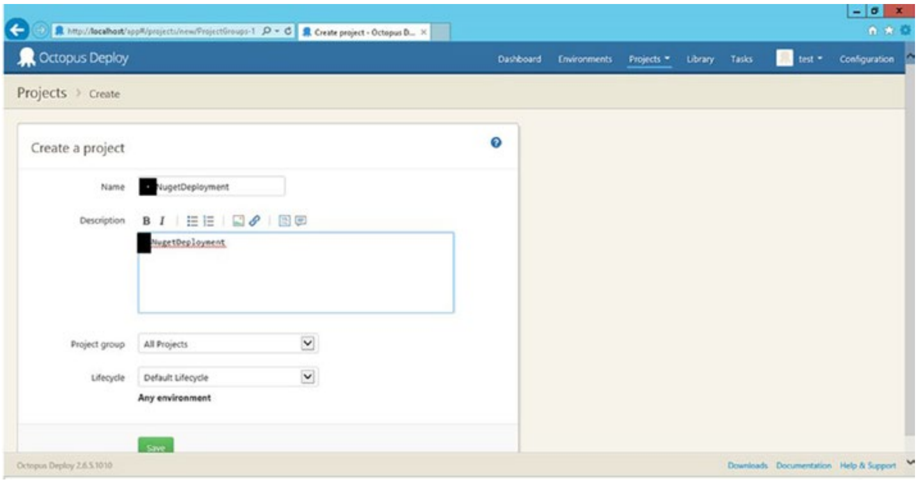
Note The latest MSI of Octopus Deploy can be downloaded at `https://octopus.com/downloads`.

Creating a Project

Octopus Deploy allows users to create projects. In Octopus Deploy, a project is a set of deliverable components, including websites and database scripts. A project is created within Octopus Deploy to manage multiple software projects across different environments. For instance, if there are six developers working on the same business project, we need to create a single project in Octopus Deploy.

Perform the following steps to create a project:

1. Navigate to the Projects area.
2. Click the Add Project button. e Create Project page opens.
3. Specify a relevant name for the project in the Name text box.
4. Specify a relevant description for the project in the Description text area.
5. Select the desired option from the Project Group drop-down list.
6. Select the desired lifecycle from the Lifecycle drop-down list.
7. Click the Save button, as shown in Figure 2-9.



Create a project

Name: [redacted]NugetDeployment

Description: [redacted]NugetDeployment

Project group: All Projects

Lifecycle: Default Lifecycle

Any environment

Octopus Deploy 2.6.5.1010

Figure 2-9. Steps to create a project

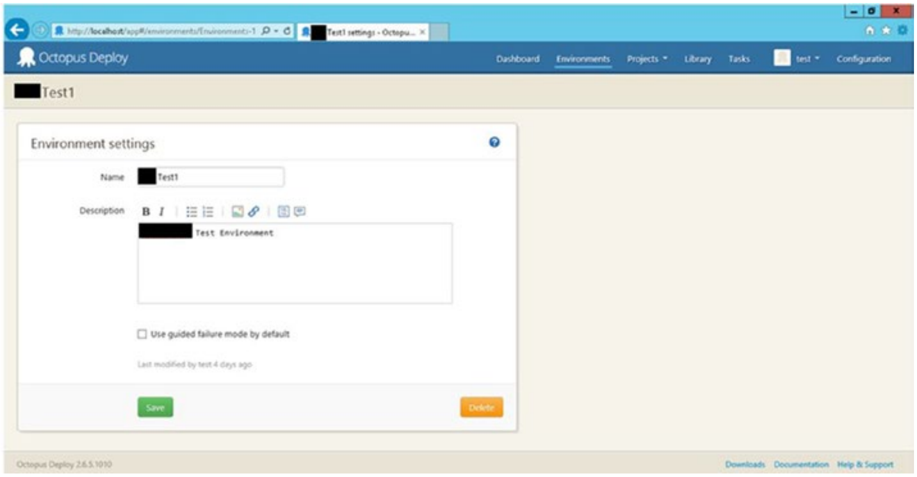
Note A lifecycle is used to replicate deployments between environments automatically.

Creating an Environment

An environment is a group of machines to which the software is deployed simultaneously. Common environments in the Octopus Deploy are Test, Acceptance, Staging, and Production. In other words, an environment can be defined as a group of deployment targets (Windows servers, Linux servers, Microsoft Azure, etc.). For the current scenario, we are creating two environments so that we can deploy to two websites. Each environment represents a single tenant.

Perform the following steps to create an environment:

1. Navigate to the Environments area.
2. Click the Add Environment button to add an environment. e Environment Settings page opens.
3. Enter a relevant name for the environment in the Name text box. In this case, we entered `Test1`.
4. Enter a relevant description of the environment in the Description text box.
5. Click the Save button, as shown in Figure 2-10.


http://localhost/app#/environments/Environments-1
[redacted] Test1 settings - Octopu...
Octopus Deploy
Dashboard
Environments
Projects
Library
Tasks
test
Configuration

[redacted] Test1

Environment settings

Name: [redacted] Test1

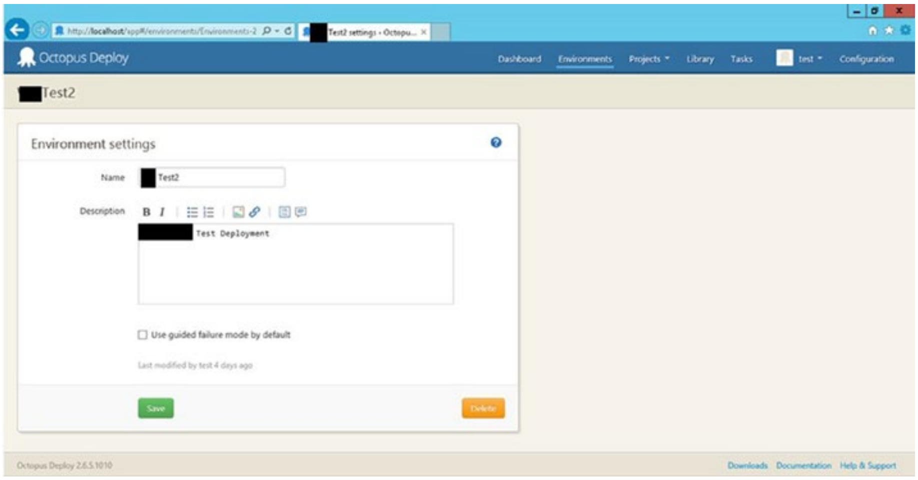
Description: [redacted] Test Environment

☐ Use guided failure mode by default

Last modified by test 4 days ago

Figure 2-10. Steps to create an environment

Similarly, create another environment with the name `Test2`, as shown in Figure 2-11.



[redacted] Test2

Environment settings

Name [redacted] Test2

Description [redacted] Test Deployment

☐ Use guided failure mode by default

Last modified by test 4 days ago

Figure 2-11. Steps to create another environment

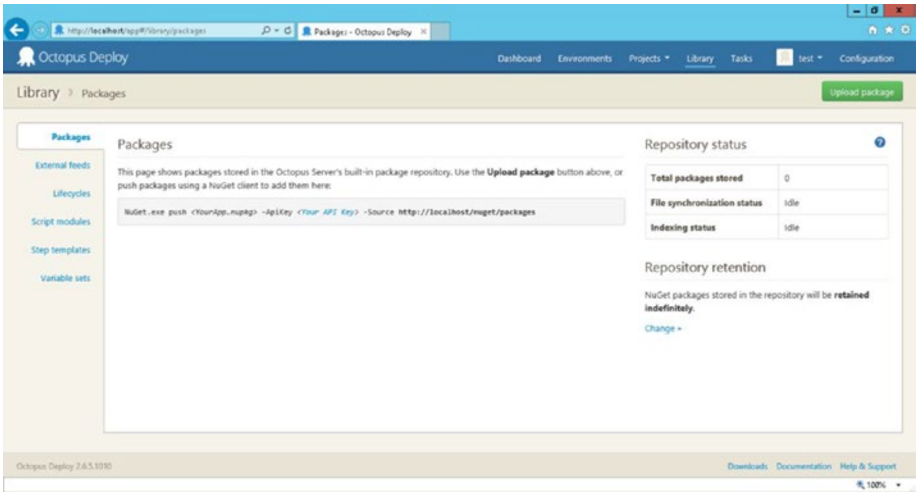
Uploading NuGet Package to Octopus Deploy

We can now upload the NuGet package, which we created earlier using the PowerShell script in TeamCity, on Octopus Deploy.

Perform the following steps to upload the NuGet package:

1. Navigate to Library, then Packages, in the Octopus Deploy interface.

2. Click the Upload Package button, as shown in Figure 2-12.



Packages

This page shows packages stored in the Octopus Server's built-in package repository. Use the **Upload package** button above, or push packages using a NuGet client to add them here:

```
NuGet.exe push <YourApp.nupkg> -ApiKey <Your API Key> -Source http://localhost/nuget/packages
```

Repository status

Total packages stored	0
File synchronization status	Idle
Indexing status	Idle

Repository retention

NuGet packages stored in the repository will be **retained indefinitely**.

Figure 2-12. Clicking the Upload Package button

The Upload a NuGet Package page appears.

3. Click the Browse button beside the NUPKG File option. e Choose File to Upload dialog box appears.

4. Navigate to the package s location. As discussed earlier, we copied the package to the `Package Source` folder.

5. Select the package.

6. Click the Open button, as shown in Figure 2-13.

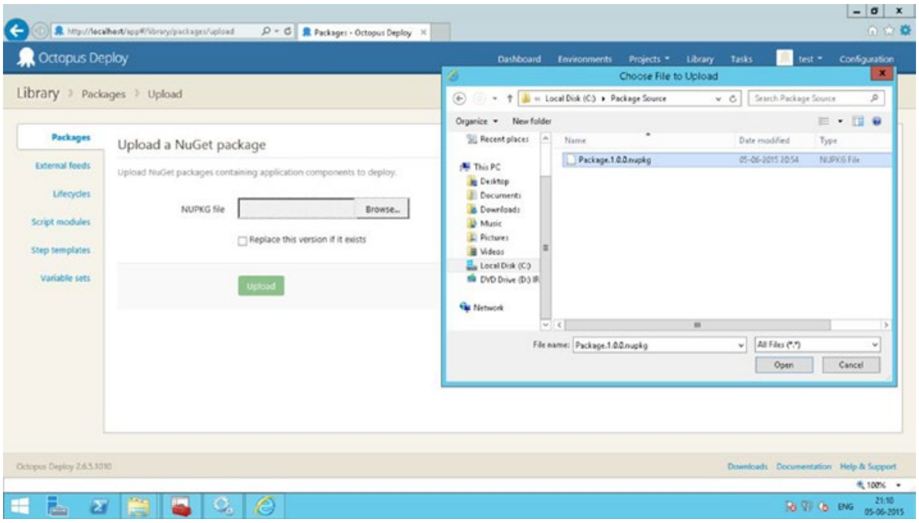


Figure 2-13. Uploading a NuGet package

The name of the selected package file with its complete path appears in the NUPKG File box.

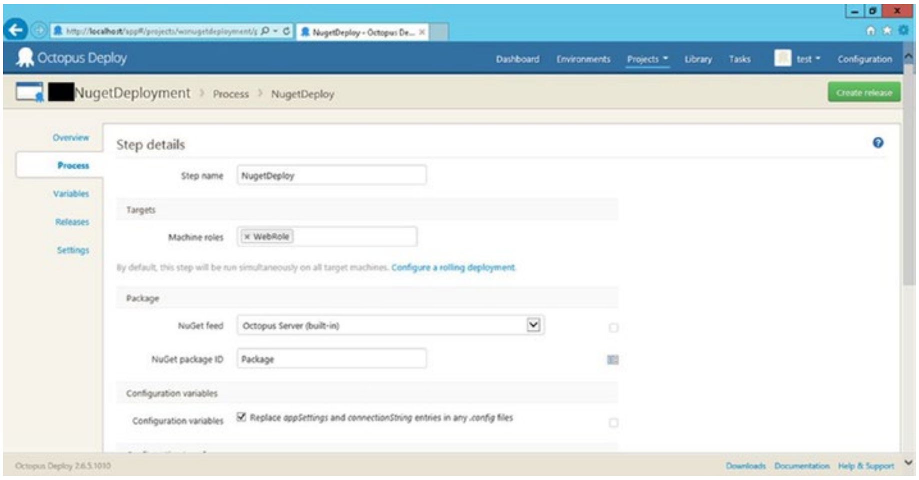
7. Click the Upload button.

After clicking the Upload button, the package file starts uploading.

Creating Steps for the Deployment Process

As discussed earlier, Octopus Deploy allows users to define the deployment process for their project easily. Users can add steps to the deployment process using templates, including built-in step templates, custom step templates, and community contributed step templates.

Users can also select the Add Step button to display a list of templates and then select the desired step. The built-in steps can be used to handle common deployment scenarios.



[redacted]NugetDeployment › Process › NugetDeploy

Step details

Step name: NugetDeploy

Targets

Machine roles: × WebRole

By default, this step will be run simultaneously on all target machines. Configure a rolling deployment.

Package

NuGet feed: Octopus Server (built-in)

NuGet package ID: Package

Configuration variables

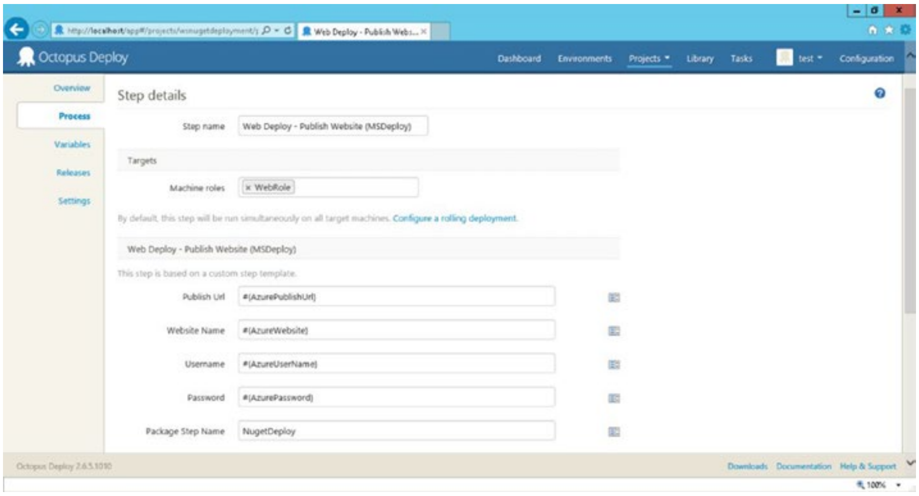
Configuration variables: ☑ Replace *appSettings* and *connectionString* entries in any *.config* files

Octopus Deploy 2.6.5.1010

Figure 2-14. The NugetDeploy step

In Figure 2-14, we see that the NuGet Package ID field contains the name of the NuGet package that was uploaded earlier.

Similarly, we can add a step using the custom step template with the name `Web Deploy-Publish Website (MSDeploy)`, as shown in Figure 2-15.



Step details

Step name: Web Deploy - Publish Website (MSDeploy)

Targets

Machine roles: × WebRole

By default, this step will be run simultaneously on all target machines. Configure a rolling deployment.

Web Deploy - Publish Website (MSDeploy)

This step is based on a custom step template.

Publish Url: #{AzurePublishUrl}

Website Name: #{AzureWebsite}

Username: #{AzureUserName}

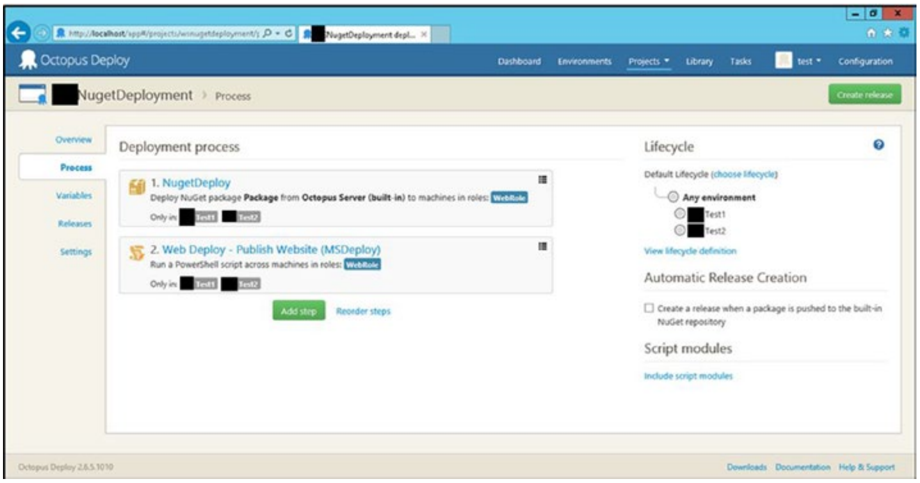
Password: #{AzurePassword}

Package Step Name: NugetDeploy

Octopus Deploy 2.6.5.1010

Figure 2-15. Adding a step

We can look at the created steps by selecting the Process tab of the created project, as shown in Figure 2-16.



[redacted]NugetDeployment › Process

Deployment process

1. NugetDeploy
Deploy NuGet package **Package** from **Octopus Server (built-in)** to machines in roles: WebRole
Only in: [redacted] Test1 [redacted] Test2

2. Web Deploy - Publish Website (MSDeploy)
Run a PowerShell script across machines in roles: WebRole
Only in: [redacted] Test1 [redacted] Test2

Lifecycle

Default Lifecycle (choose lifecycle)
Any environment
[redacted]Test1
[redacted]Test2
View lifecycle definition

Automatic Release Creation

☐ Create a release when a package is pushed to the built-in NuGet repository

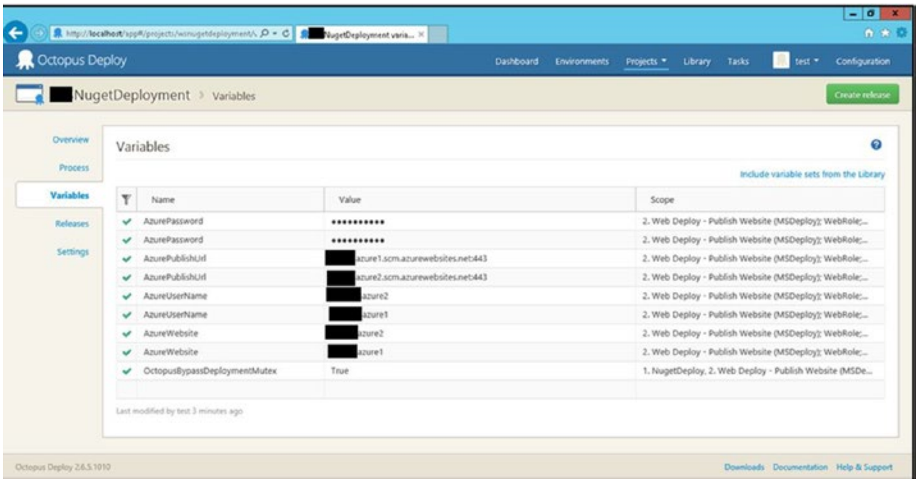
Script modules

Include script modules

Figure 2-16. Displaying the created steps

Using Variables

Variables are required for eliminating the need for hard-coding the configuration values to support different environments easily. They are required while deploying packages to Azure websites. As a NuGet package is shared between two sites, we used the `OctopusBypassDeploymentMutex` variable to avoid resource locking of the NuGet package, as shown in Figure 2-17.


http://localhost/app#/projects/wsnugetdeployment/\
[redacted]NugetDeployment varia...
Octopus Deploy
Dashboard
Environments
Projects
Library
Tasks
test
Configuration
[redacted]NugetDeployment › Variables
Create release
Overview
Process
Variables
Releases
Settings
Include variable sets from the Library
Octopus Deploy 2.6.5.1010
Downloads
Documentation
Help & Support

Variables

Name	Value	Scope
AzurePassword	••••••••••	2. Web Deploy - Publish Website (MSDeploy); WebRole;...
AzurePassword	••••••••••	2. Web Deploy - Publish Website (MSDeploy); WebRole;...
AzurePublishUrl	[redacted]azure1.scm.azurewebsites.net:443	2. Web Deploy - Publish Website (MSDeploy); WebRole;...
AzurePublishUrl	[redacted]azure2.scm.azurewebsites.net:443	2. Web Deploy - Publish Website (MSDeploy); WebRole;...
AzureUserName	[redacted]azure2	2. Web Deploy - Publish Website (MSDeploy); WebRole;...
AzureUserName	[redacted]azure1	2. Web Deploy - Publish Website (MSDeploy); WebRole;...
AzureWebsite	[redacted]azure2	2. Web Deploy - Publish Website (MSDeploy); WebRole;...
AzureWebsite	[redacted]azure1	2. Web Deploy - Publish Website (MSDeploy); WebRole;...
OctopusBypassDeploymentMutex	True	1. NugetDeploy, 2. Web Deploy - Publish Website (MSDe...

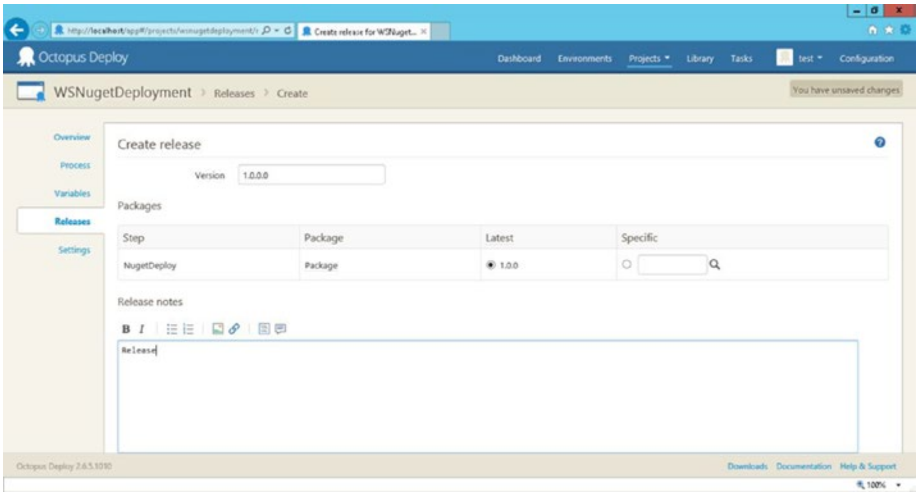
Last modified by test 3 minutes ago

Figure 2-17. The OctopusBypassDeploymentMutex variable

Creating and Deploying a Release

A release contains all details of the project and package so that it can be deployed to different environments as per requirements. Perform the following steps to create a release:

1. Navigate to the Overview page, which displays all details of the project.
2. Click the Create Release button. e Create page appears.
3. Enter the desired release version in the Version text box.
4. Select the desired package from the Package column.
5. Enter the desired release notes in the Release Notes text area.
6. Click the Save button. Figure 2-18 shows the process of creating a release.



Create release

Version 1.0.0.0

Packages

Step	Package	Latest	Specific
NugetDeploy	Package	● 1.0.0	○

Release notes

Release

Octopus Deploy 2.6.5.1010

Figure 2-18. Creating a release

Note In the current scenario, we are creating a release to deploy the NuGet package to multiple Azure websites.

A release is created with the specified version. The Deploy page opens. Here, we can select the desired environment to which we want to deploy the created release. We can also click the Change button to change the environment.

7. Click the Deploy Now button to deploy the created release, as shown in Figure 2-19.

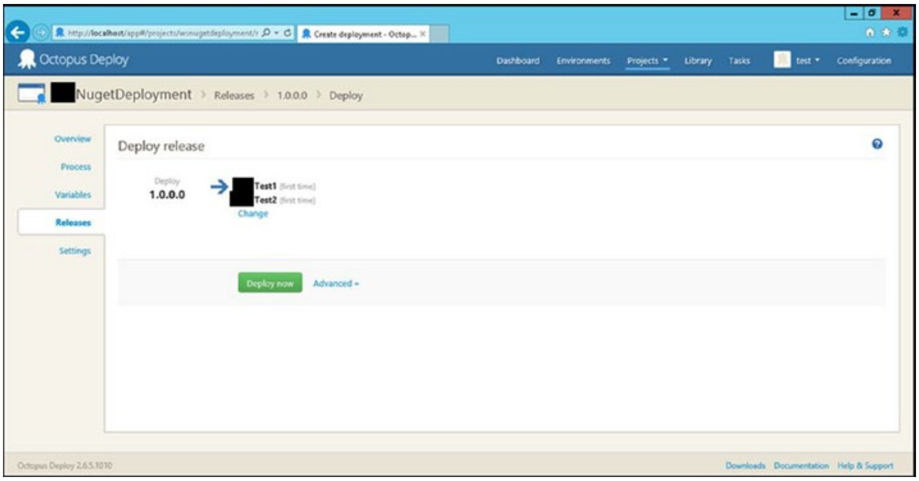
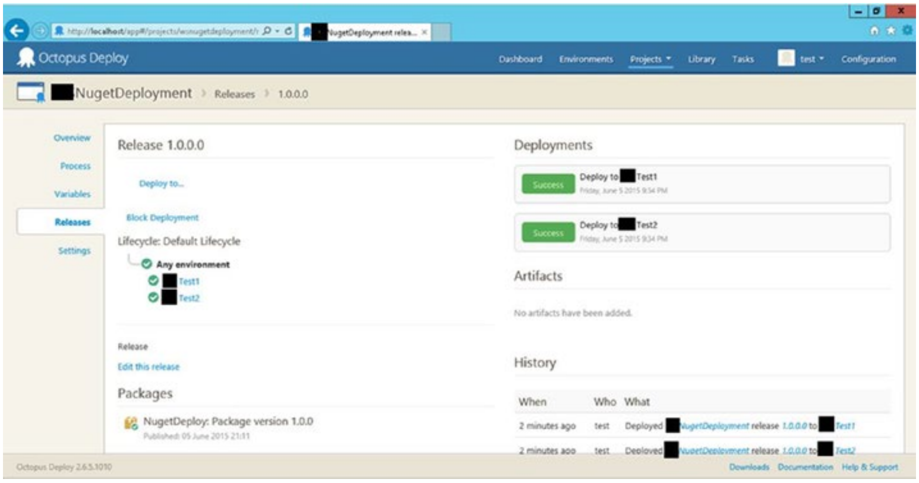


Figure 2-19. Deploying a release

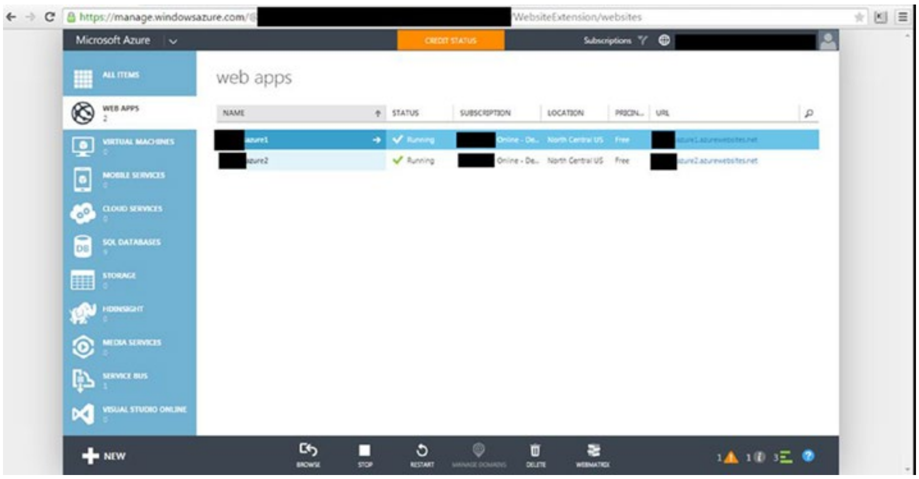
The release is deployed successfully to both Azure websites, as shown in Figure 2-20.



When	Who	What
2 minutes ago	test	Deployed [redacted]NugetDeployment release 1.0.0.0 to [redacted]Test1
2 minutes ago	test	Deployed [redacted]NugetDeployment release 1.0.0.0 to [redacted]Test2

Figure 2-20. Deployment result

We can now navigate to the Azure portal where we see that two Azure websites have been created for multiple deployments of the NuGet package, as shown in Figure 2-21.



web apps

NAME	STATUS	SUBSCRIPTION	LOCATION	PRICIN...	URL
[redacted]azure1	✓ Running	[redacted]Online - De...	North Central US	Free	[redacted]azure1.azurewebsites.net
[redacted]azure2	✓ Running	[redacted]Online - De...	North Central US	Free	[redacted]azure2.azurewebsites.net

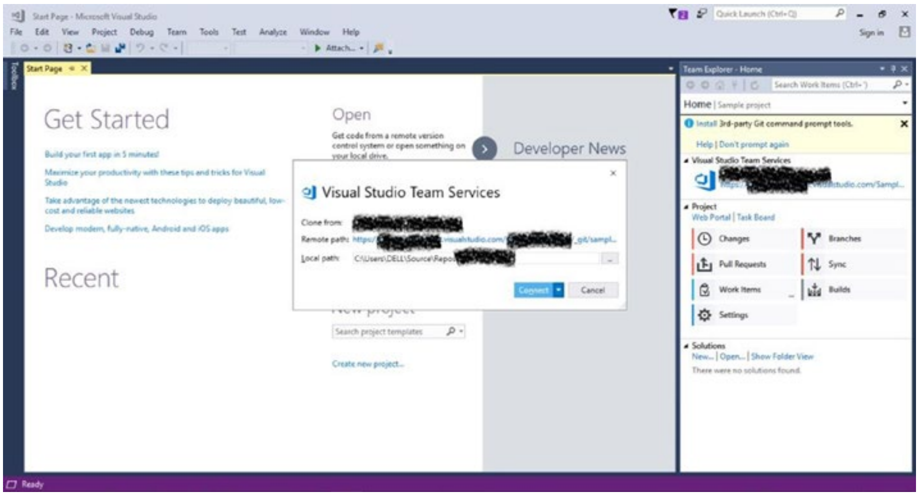
Figure 2-21. Displaying the created websites on Azure

Summary

In this chapter, we discussed the CI tool called TeamCity and the release management software or CD tool called Octopus Deploy. TeamCity builds the source code using MSBuild. Initially, we configured TeamCity by creating a new project and providing the SVN path to fetch the latest code onto the build agent. We then configured the source code and set parameters for the PowerShell script file. The target path settings were modified to create a NuGet package. This package was copied from the build agent to a location where Octopus Deploy could pick it up.

In Octopus Deploy, we created a project and two environments to test multiple deployment scenarios. Then, we uploaded the package. We also created two steps NugetDeploy and Web Deploy-Publish Website (MSDeploy). The former was created to deploy the uploaded NuGet

4. Click the Connect button to connect Visual Studio with Visual Studio Team Services, as shown in Figure 3-10.


Start Page - Microsoft Visual Studio
File Edit View Project Debug Team Tools Test Analyze Window Help
Start Page
Get Started
Build your first app in 5 minutes!
Recent
Open
Developer News
Team Explorer - Home
Home | Sample project
Visual Studio Team Services
Project
Web Portal | Task Board
Changes
Branches
Pull Requests
Sync
Work Items
Builds
Settings
Solutions
New... | Open... | Show Folder View
There were no solutions found.
Connect
Cancel
Ready

Visual Studio Team Services

Clone from: [redacted]

Remote path: https://[redacted]visualstudio.com/[redacted]_git/sampl...

Local path: C:\Users\DELL\Source\Repo[redacted]

Figure 3-10. Connecting Visual Studio to Visual Studio Team Services

After we click Connect, the cloning and connection processes are complete.

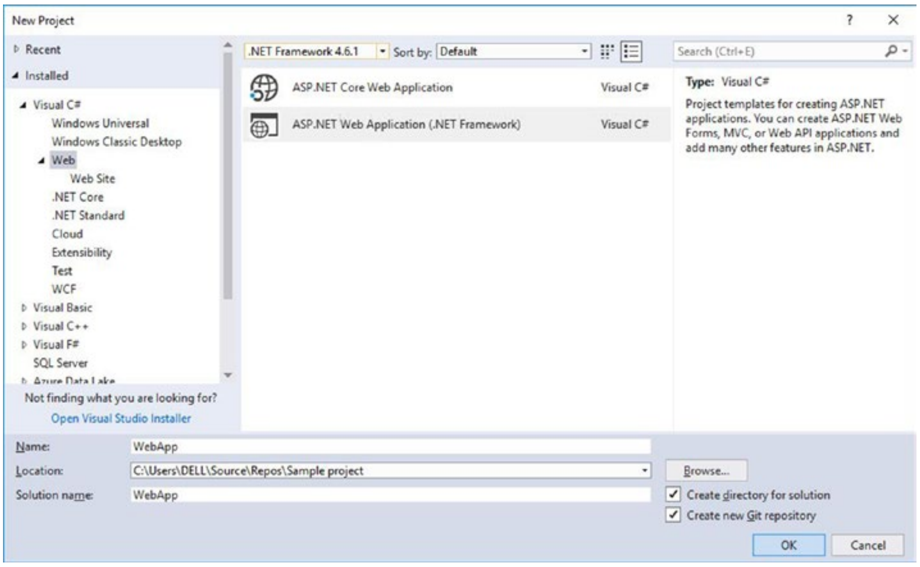
Adding a New Solution

Here, we need to add a new solution, which can be done by performing the following steps:

1. Click the New link under the Solutions section in the Team Explorer panel. The New Project window appears.

2. Select the desired option from the left pane. In this case, we selected Web. e related templates appear in the middle pane based on the selection.

3. Select the desired template in the middle pane. In this case, we selected ASP.NET Web Application (.NET Framework).
4. Enter the desired name for the selected template in the Name text box. In this case, we entered WebApp.
5. Specify the desired location for the template in the Location text box.
6. Select the Create Directory for Solution checkbox.
7. Select the Create New Git Repository checkbox.
8. Click the OK button, as shown in Figure 3-11.



Type: Visual C#

Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Name: WebApp

Location: C:\Users\DELL\Source\Repos\Sample project

Solution name: WebApp

✓ Create directory for solution

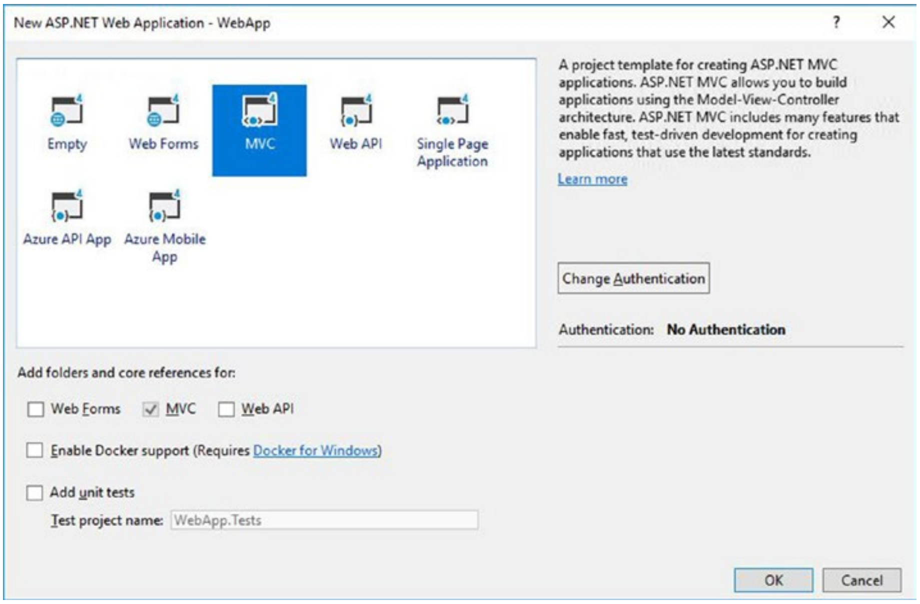
✓ Create new Git repository

Figure 3-11. Creating a new project

The New ASP.NET Web Application WebApp window appears.

9. Select the MVC option to create the MVC application.

10. Click the OK button, as shown in Figure 3-12.



New ASP.NET Web Application - WebApp

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Authentication: **No Authentication**

Add folders and core references for:

☐ Web Forms ☑ MVC ☐ Web API

☐ Enable Docker support (Requires Docker for Windows)

☐ Add unit tests

Test project name: WebApp.Tests

Figure 3-12. The New ASP.NET Web Application WebApp window

The Microsoft Visual Studio progress bar appears displaying the status of the project. Once complete, the project is created and added to the Solutions section.

Committing Changes

Once the required changes are made, we can commit them. Perform the following steps to commit the changes:

1. Click the Changes button under the Project section in the Team Explorer panel, as shown in Figure 3-13.

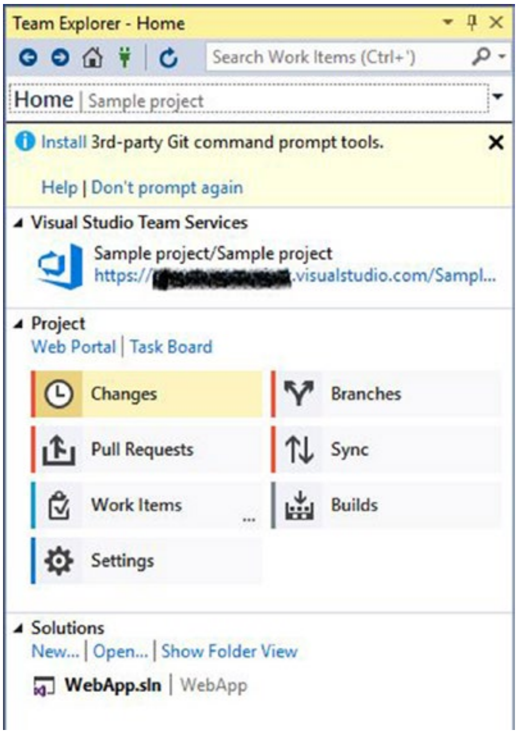
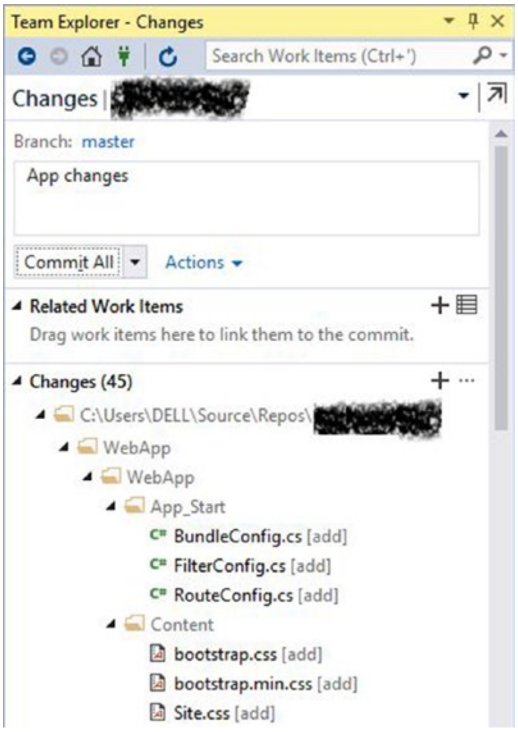

Team Explorer - Home
Search Work Items (Ctrl+')
Home | Sample project
Install 3rd-party Git command prompt tools.
Help | Don't prompt again
Visual Studio Team Services
Sample project/Sample project
https://[redacted].visualstudio.com/Sampl...
Project
Web Portal | Task Board
Changes
Branches
Pull Requests
Sync
Work Items
Builds
Settings
Solutions
New... | Open... | Show Folder View
WebApp.sln | WebApp

Figure 3-13. Steps to commit changes

The changes made to the project are displayed in the Changes section.

2. Enter the desired commit message in the Enter a Commit Message text box.

3. Click the Commit All button, as shown in Figure 3-14.



App changes

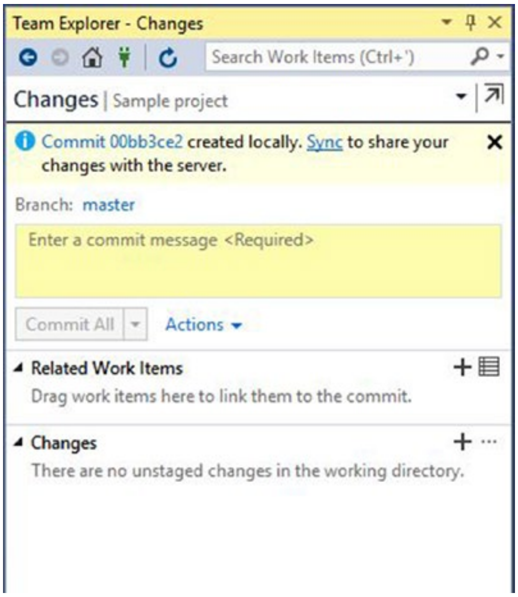
Drag work items here to link them to the commit.

C:\Users\DELL\Source\Repos\ [redacted]
WebApp
WebApp
App_Start
BundleConfig.cs [add]
FilterConfig.cs [add]
RouteConfig.cs [add]
Content
bootstrap.css [add]
bootstrap.min.css [add]
Site.css [add]

Figure 3-14. Steps to commit changes

A commit is created locally.

4. Click the Sync link to share the changes with the server, as shown in Figure 3-15.



Commit 00bb3ce2 created locally. Sync to share your changes with the server.

Branch: master

Enter a commit message <Required>

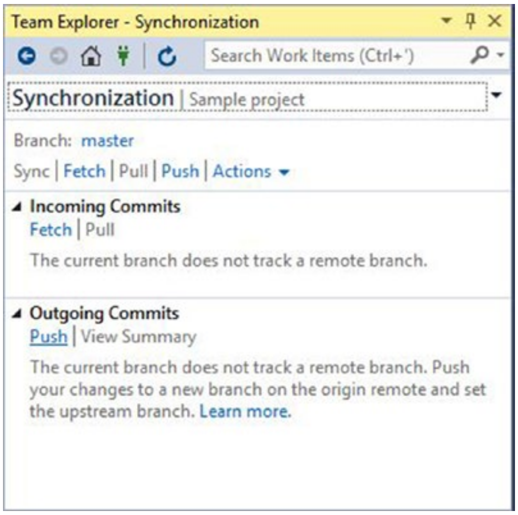
Drag work items here to link them to the commit.

There are no unstaged changes in the working directory.

Figure 3-15. Sharing the changes with the server

The Synchronization page appears in the Team Explorer panel.

5. Click the Push link under the Outgoing Commits section, as shown in Figure 3-16.


Team Explorer - Synchronization
Search Work Items (Ctrl+')
Synchronization | Sample project

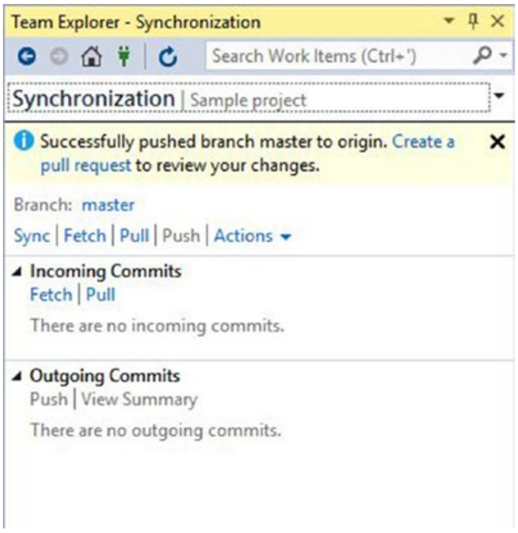
Branch: master
Sync | Fetch | Pull | Push | Actions

Incoming Commits
Fetch | Pull
The current branch does not track a remote branch.

Outgoing Commits
Push | View Summary
The current branch does not track a remote branch. Push your changes to a new branch on the origin remote and set the upstream branch. Learn more.

Figure 3-16. The Push link enables synchronization

The synchronization is successful, as shown in Figure 3-17.


Team Explorer - Synchronization
Search Work Items (Ctrl+')
Synchronization | Sample project

Successfully pushed branch master to origin. Create a pull request to review your changes.

Branch: master
Sync | Fetch | Pull | Push | Actions

Incoming Commits
Fetch | Pull
There are no incoming commits.

Outgoing Commits
Push | View Summary
There are no outgoing commits.

Figure 3-17. Successful synchronization

At this point, the code is added to the server.

Next, verify the repository in VSTS. For this, navigate to the Code section of the project created earlier. A folder with the same name as that of the project created in Visual Studio appears, as shown in Figure 3-18.

Figure 3-18. The repository

Creating a Build

Once the source control repository is available, we can set up (or create) a build. Perform the following steps to create a build:

1. Hover the mouse over the Build and Release tab.

   A list of options appears.

2. Click the Builds option, as shown in Figure 3-19.