|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | | |
|  |  |  |  | |
| PowerShell CI/CD Tools |  | PowerShell |  |  |
|  |  |  |  |
|  |  |  | **Hands-on lab** |
|  |  |  |  | |
| PowerShell Desired State Configuration can be a core component of the Continuous Integration / Continuous Deployment pipeline. Come build and run an end-to-end pipeline using PowerShell features and tooling. You will leave this ILL with hands on experience in building and running a PowerShell based pipeline. |  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  | | |

This document supports a preliminary release of a software product that may be changed substantially prior to final commercial release. This document is provided for informational purposes only and Microsoft makes no warranties, either express or implied, in this document. Information in this document, including URL and other Internet Web site references, is subject to change without notice. The entire risk of the use or the results from the use of this document remains with the user. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Copyright 2016 © Microsoft Corporation. All rights reserved.

Microsoft ***<list other MS trademarks used alphabetically>*** are trademarks of the Microsoft group of companies.

***<This is where mention of specific, contractually obligated to, third party trademarks should be listed.>***

All other trademarks are property of their respective owners.

# Introduction

### Estimated time to complete this lab

90 minutes

### Objectives

After completing this lab, you will be able to:

* Learn how to implement a Continuous Integration pipeline with PowerShell
* Utilize the latest PowerShell tooling

### Prerequisites

Before working on this lab, you must have:

* an understanding of Continuous Integration and Continuous Deployment concepts
* an understanding of Git source control concepts
* experience with PowerShell scripting

### Overview of the lab

PowerShell can be a core component of the Continuous Integration / Continuous Deployment pipeline. Come build and run an end-to-end pipeline using PowerShell features and tooling. You will leave this ILL with hands on experience in building and running a PowerShell based pipeline.

### Scenario

In this lab you will configure a Continuous Integration (CI) pipeline using Visual Studio Team Foundation Server and the MyGet online services. Once the pipeline is configured, you will use it to fully build, test, and deploy a PowerShell module to a private PowerShellGet feed. Although this Lab will not cover it, the same process can be extended to take the changes through a test and production for managing infrastructure environments as well.

### Reference material

* [The Release Pipeline Model](http://releasepipeline.blob.core.windows.net/pdf/The%20Release%20Pipeline%20Model.pdf): This document provides a great framework and defines all of the concepts use in this lab.
* [Team Foundation Server](https://www.visualstudio.com/products/visual-studio-team-services-vs?utm_source=Google&utm_medium=CPC&utm_term=Branded&utm_campaign=VSTS%20EvergreenWT.srch=1&WT.mc_id=SEM_xeWGA7kH): Used as source code repository as well as build system for release pipeline in this lab.



# Exercise 1: Setup VSTS Build Environment

In this first exercise, you will setup a VSTS account and configure the build definitions.

## Task 0 – Setup a VSTS account

In this task, you will setup a free account in Visual Studio Team Services.

\*\*Note\*\* - This task requires a (free) Microsoft account. (https://account.microsoft.com/)

* Task instructions

1. In a browser, open https://www.visualstudio.com
2. Click Free Visual Studio
3. In the Visual Studio Team Services box, Click Free Account
4. Sign in using your Microsoft Account
5. Enter a unique name for your project and select the radio button next to Git (this is the default), then click Continue.
6. Click Generate Git Credentials, enter an alias and password, then click Save Git Credentials.

The next task will continue in the same browser session, so do not close the window.

## Task 1 – Clone code from GitHub in to your new project

In this task, you will import an example PowerShell module stored in a public GitHub repository.

* Task instructions

1. Continuing in the same browser session, expand the text “or import a repository” and click Import
2. In the Source Type window, keep the default choice, Git. In the Clone URL window, type https://github.com/PowerShell/Demo\_CI/

The browser window should automatically redirect to the Code tab

1. Click on the  button to select a different branch, select All branches and then module-example

The code changes to the *ExampleModule* file content automatically

The next task will continue in the same browser session, so do not close the window.

## Task 2 – Create and configure Build Definition

In this task, you will configure VSTS with a build definition.

* Task instructions

1. Minimize the PowerShell Console to expose the desktop
2. Click the Team Foundation Server shortcut in the center of the desktop.

* NOTE: This brings up the Team Foundation Server console that is hosted on the TFSSrv1 VM.
* The first time this is loaded, it might take 45-60 seconds to load the page.

1. Click the Demo\_CI project in the list of recent projects and teams to bring up the project overview page.
2. Click the Build tab located just under the blue ribbon at the top of the page.
3. Click the Add button .
4. Select the Empty build definition type at the bottom of the dialog and click Next.
5. Leave the defaults on the Create new build definition page selected and click Create.
6. Click the Save button  on the newly created build definition and type “DNS Infra” into the Name field. Click the OK button.
7. Now create our first build step by clicking Add build step…  on the Build tab.
8. Click Utility from the left pane to show a list of utility tasks.
9. Scroll down and click Add next to the PowerShell task .

* You may not have noticed, but this did add a build step to the screen behind the dialog so do not click add again or multiple steps will be added.

1. Click Test from the left pane to show list of Test tasks.
2. Click Add next to the Publish Test Results task .

* This step copies Unit test results generated by Pester into VSTS so that the results are integrated.

1. Click Utility from the left pane to show a list of utility tasks again.
2. Click Add next to the Copy Files task .

* This step copies all artifacts that will be needed for deployment to a staging directory.

1. Click Add next to the Publish Build Artifiacts .

* This step publishes all artifacts from the staging directory so that they are available for releases.

1. Click Close button on the Add Tasks dialog.
2. Select PowerShell step and type initiate.ps1 into Script filename and -fileName build into the Arguments field.
3. Select Publish Test Results step and change the following properties:

* Select NUnit for Test Result Format.
* Type “InfraDNS/Tests/Results/\*.xml” into Test Result Files field
* Enter “Unit” for Test Run Title.
* Check the box for Always run under Control Options.

1. Select the Copy Files step and and change the following properties:

* Type the following into the Contents field each item followed by carriage return:
  + initiate.ps1
  + \*\*\deploy.ps1
  + \*\*\Acceptance\\*\*
  + \*\*\Integration\\*\*
* Type $(Build.ArtifactStagingDirectory)\ into the Target folder field.

1. Select the Publish Build Artifacts step and change the following properties:

* Type $(Build.ArtifactStagingDirectory)\ into Path to Publish.
* Type Deploy into Artifact Name.
* Select Server for Artifact Type.

1. Click Save  to save the build definition.
2. Click OK on the Save dialog.

* IMPORTANT: Do not close the Edge browser. You will need it in the following exercises and tasks.

# Exercise 2: Review project code

In this exercise, you will review the code that was downloaded to understand how it is structured and what the different files do.

## Task 1 – Open Project

In this Task, you will use Visual Studio Code to open the folder that contains all of the code.

* Task instructions

1. Open Visual Studio Code by clicking  on the task bar.
2. On the Menu, click File and select Open Folder….
3. In the select Folder dialog browse to “c:\Git\Demo\_CI” and click Select Folder.

* IMPORTANT: Do not close the Visual Studio Code. You will need it in the following exercises and tasks.

## Task 2 – DSC Configuration

In this task, you will locate and review the DSC configuration that will be used to configure DNS on TestAgent1. This configuration uses configuration data generated during the build process.

* Task instructions

1. In the VS Code file Explorer, expand InfraDNS and then expand Configs.
2. Select DNSServer.ps1. This displays the DSC configuration code that will be used to configure the DNSserver.
3. Two things to notice about this configuration:

* Node name(s) are not static and come from configuration data:

Node $AllNodes.Where{$\_.Role -eq 'DNSServer'}.NodeName

{. . . }

This is important when doing Continuous Integration because node information will likely change between environments and this allows you to easily make changes without changing this code.

* Foreach loop around xDnsRecord resources:

foreach ($ARec in $Node.ARecords.keys) {

xDnsRecord $ARec

{ . . . }

This code allows you to pass in any number of DNS records that need to be configured for this DNS Zone. Each will be individually validated to ensure it is present. This means that you can easily add more records without changing this code.

## Task 3 – [PSake](http://psake.readthedocs.io/en/latest/) Build Script

In this task, you will locate and review the PSake build script used to run all of the steps in the build and test process.

* Task instructions

1. If not already there, in the VS Code file Explorer, expand InfraDNS.
2. Select Build.ps1. This displays the PSake build script. It has tasks that generate DSC MOF documents and run unit tests.
3. Select anywhere inside the right pane and press Ctrl+K Ctrl+0 to fold (collapse) all regions. This will make the code a bit more readable for now.
4. Few things to notice about this configuration:

* **Tasks**: Contain PowerShell code and logic for executing a specific task. The default task is required and is run when no task is provided.
* **Depends**: Define task dependencies. All dependent tasks must be run successfully before a task that depends on them will be run.
* **Properties**: Are variables that can be shared across all tasks.
* **Failures**: A terminating error by any task will cause PSake to stop processing any dependent task. Since Pester does not throw terminating errors for failed tests, logic is added to each test task to throw a terminating error if any tests fail.

1. Select Deploy.ps1. This displays the PSake deployment script. It has tasks for deploying and testing (Integration and Acceptance) the DNS configuration.

## Task 4 – Pester and PoshSpec Tests

In this task, you will locate and review the test scripts used in each test phase (Unit, Integration and Acceptance). You will also learn about the types of tests that should be performed in each test phase.

* Task instructions

1. In the VS Code file Explorer, expand InfraDNS and then expand Tests.

Under this folder there is a folder for each type of test: Unit, Integration and Acceptance. There is also a folder where test results output will be stored.

**Unit Tests**: Tests configurations themselves to ensure the configuration is going to do what is expected when run. [Pester](https://github.com/pester/Pester/wiki) syntax is used to describe the tests.

**Integration Tests**: Tests configuration of system to ensure that when integrated with other components, the system is configured as expected. These tests run on the target node after it has been configured with DSC. These tests use a mix of Pester and [PoshSpec](https://github.com/Ticketmaster/poshspec/wiki) syntax.

**Acceptance Tests**: Tests the system to ensure it behaves as expected. For example, it tests to ensure a web page returns the right information when queried. These tests do not run after the integration tests and run remotely from the target node in order to test real world scenarios. These tests use a mix of Pester and [PoshSpec](https://github.com/Ticketmaster/poshspec/wiki) syntax.

1. Now expand Unit.
2. Select DNSServer.tests.ps1.

In this file you will see numerous Pester Tests. You should be able to read through the examples to understand the types of tests that will be performed when it is run.

1. Expand Integration and Acceptance and select DNSServer.tests.ps1 to familiarize yourself with those tests as well.
2. NUnit XML output from pester tests will be used to integrate test result into the TFS pipeline. This same output can be used to integrate test results with other build systems.

## Task 5 – DSC Configuration Data

In this task, you will locate and review the script used to generate the configuration data. This data allows environment specific information to be passed into the DSC Configuration so that the structural configuration, DNSServer.ps1 in this case, does not need to change when moving between environments such as between development, test and production environments. Generating this configuration data from a script like this is recommended as it allows you to more easily and dynamically change the environmental data at compile time.

* Task instructions

1. If not already there, in the VS Code file Explorer, expand InfraDNS.
2. Select DevEnv.ps1.

In this file you will see test environment specific data for the DNS server configuration. This script is used to generate the configuration data file that is passed into the structural configuration from DNSServer.ps1.

## Task 6 – Build Runner Script

In this task, you will locate and review the script that was used in the first step of the build definition in exercise 1 above. This script simply invokes the PSake script and handles any errors returned.

* Task instructions

1. In the VS Code file Explorer, select initiate.ps1.

This file is a simple script used by TFS to kick off the PSake build and deploy scripts.

# Exercise 3: Run Build

In this exercise, you will manually run the build pipeline that you defined in the previous exercises and review the results of the run.

## Task 1 – Check-in Code to TFS

In this task, you will connect your local repo with the TFS server and push the code that you checked into your local Git repo up to the TFS server.

* Task instructions

1. If not already open, launch PowerShell console by clicking  on the taskbar.
2. To configure a local Git repo with code from Exercise 1, type the following in the console, and then press ENTER after each line:

* Cd \git\Demo\_CI
* Git init
* Git add .
* Git config --global user.email [demo@contoso.com](mailto:demo@contoso.com)
* Git config --global user.name “Mr. Demo”
* Git commit -m “Initial Configuration and tests.”
* IMPORTANT: Do not close the PowerShell console. You will need it in the following exercises and tasks.

1. Click the Code tab located just under the blue ribbon at the top of the page.
2. Scroll down to the section with command line instructions to Push an existing repository and copy each of the commands and paste them into the PowerShell console as follows, and then press ENTER after each line:

* Git remote add origin http://tfssrv1:8080/tfs/Pipeline/\_git/Demo\_CI
* Git push -u origin --all

1. Enter the following credentials into the authentication dialog that pops up and click Ok:

Username: Administrator

Password: pass@word1

1. Now go back to Microsoft Edge and click the refresh button . You will now see the code that was uploaded to TFS.

## Task 2 – Run Build

In this task, you will manually kick off the build.

* Task instructions

1. In Microsoft Edge select the Build tab located just under the blue ribbon at the top of the page.
2. Select the DNS Infra build definition that you created in Exercise 1.
3. Click the Queue build… button to manually start a build.
4. No need to make any changes to the defaults in the dialog that comes up so just click OK.

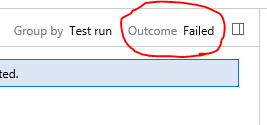
You will start to see the build progressing. It should complete successfully in a minute or so.

## Task 3 – Review results

In this task, you will review the results of the manual build that you kicked off in the previous task.

* Task instructions

1. In the console window that was scrolling by during the build you will see details about all of the PSake tasks that were run as part of the build. Each task is formatted with ‘--------------’ on either side of the task title so that you can find them easily. This formatting is defined in the PSake build script (build.ps1) using the **FormatTaskName** property.
2. Select Build 1 in the left-hand pane to show the summary for the build. In addition to build summary details, this shows a summary of the pester tests as well. This is a result of the Outputfile and OutputFormat parameters of the Pester cmdlet and the Publish Test Results steps that we added to the TFS build after the PowerShell step.
3. Click the Detailed report > at the bottom of the summary page (Might require scrolling down on the page). This brings up test run details.
4. By default, only failing test cases are shown here but you can see details about all tests run by selecting Failed next to Outcome and changing it to All.



# Exercise 4: Continuous Integration

In this exercise, you will enable Continuous Integration in TFS, make changes to the code, watch the build automatically happen, and review the results of the build.

## Task 1 – Enable Continuous Integration

In this task, you will enable Continuous Integration for the build definition defined in Exercise 1 above. This will cause any changes pushed to the master branch of this repo to cause a build to be automatically kicked off.

* Task instructions

1. In Microsoft Edge, click the Build tab located just under the blue ribbon at the top of the page.
2. Select DNS Infra from left pane.
3. Click Edit at the top of the right hand pane.
4. Click the Triggers tab.
5. Check the box next to Continuous integration. This will cause all check-ins to kick of this build automatically.
6. Leave the rest of the defaults and click Save  button.
7. Click OK in the SAVE dialog.

## Task 2 – Change Static Script Analysis

In this task, you will remove the following warning generated by PSScriptAnalyzer in the ScriptAnalysis build step:

------------- ScriptAnalysis --------------

Starting static analysis…

RuleName : PSMissingModuleManifestField

Severity : Warning

…

This warning indicates that a ModuleManifest file is missing from the TestEnv.psd1 file that is generated as part of the build. This file is a DSC configuration data file and not a module manifest so this rule is not applicable and can be safely ignored. In this step, you will tell PS Script Analyzer to exclude that rule when running its analysis.

* Exclude rule from PS Script Analyzer

1. If not already there, in Visual Studio Code, expand InfraDNS
2. Select build.ps1.
3. Select anywhere inside the right pane and press Ctrl+K Ctrl+0 to fold (collapse) all regions.
4. Expand Task ScriptAnalysis
5. Add the -ExcludeRule parameter to the Invoke-ScriptAnalyzer command as follows:

* Invoke-ScriptAnalyzer -Path $ConfigPath -ExcludeRule 'PSMissingModuleManifestField'

1. Press Ctrl + S to save the change.

* Push changes to TFS

1. In PowerShell console, type the following and then press ENTER after each line:

* Cd \git\Demo\_CI
* Git add .
* Git commit -m ”Excluding Script Analyzer rule causing warning”
* Git push
* Verify CI process

1. In Microsoft Edge, click the Build tab located just under the blue ribbon at the top of the page.
2. Select the DNS Infra build definition that you created in Exercise 1.
3. Click Queued to see the build that was automatically kicked off when the change you just made was pushed to TFS.

* If you take too long, the build may not be in Queued any longer. If this happens, check in Completed.

1. When the build completes, it will disappear from the Queued list and appear in the Completed list.
2. Click Completed to view the list of completed builds. You will see a new build and it will have a new icon  indicating that it was a result of continuous integration.
3. Double click this new build in the list to view the build details.
4. Click the PowerShell: initiate.ps1 step. If you scroll down, you will see that the warning message no longer is displayed in the ScriptAnalysis task.

## Task 3 – Change Build Script

In this task, you will update the PSake build script so that it runs all of the tasks. In its current state the build script is running only up to the UnitTest task. This is because the Default task is configured to depend on the UnitTest task so it will start processing all of the tasks that depend on that task and stop when it completes the unit tests. To cause all tasks to run you will change the Default task to depend on the CompileConfigs task instead.

* Change PSake Default task dependency

1. If not already there, in Visual Studio Code, expand InfraDNS
2. Select build.ps1.
3. Select anywhere inside the right pane and press Ctrl+K Ctrl+0 to fold (collapse) all regions.
4. Update the default task by replacing UnitTests with CompileConfigs as follows:

* Task Default -depends CompileConfigs

1. Press Ctrl + S to save the change.

* Push changes to TFS

1. In PowerShell console type the following, and then press ENTER after each line:

* Cd \git\Demo\_CI
* Git add .
* Git commit -m ”Run all build tasks”
* Git push
* Verify CI process

1. In Microsoft Edge, click the Build tab located just under the blue ribbon at the top of the page.
2. Select the DNS Infra build definition that you created in Exercise 1.
3. Click Queued to see the build that was automatically kicked off when the change you just made was pushed to TFS.
4. Double click the queued build to follow the process of the build.
5. When the build completes, Click Build 3 in the left hand pane to view the build summary.

* Verify Artifacts

1. Click Artifacts at the top of the build summary.
2. Click Explore next to the Deploy Artifact. This will bring up a dialog where you can see the artifacts that were generated by the build process and will be available for the release process.
3. Expand the Deploy artifact and you should see the following:

* InfraDNS – This contains the Deploy PSake script, the Integration tests, and the Acceptance tests.
* Modules – This contains the DSC resource modules required by the configuration.
* MOF – This contains the DSC configuration MOFs.
* WebApp – This is not used in this lab, but represents another configuration.
* Initiate.ps1 – This is the build runner script described in exercise 2.

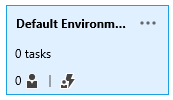
1. Click the Close button to close the Artifact Explorer.

# Exercise 5: Release and Continuous Deployment

## Task 1 - Create and Configure Release Definition

In this task, you will create a release definition for the development environment and enable it for continuous deployment. Doing this will result in every code check-in being deployed to the development environment.

* Create Release Definition

1. In Microsoft Edge, select the Release tab located just under the blue ribbon at the top of the page.
2. Click the Add button  in the left hand pane.
3. Select the Empty type at the bottom of the dialog
4. Click OK.
5. Enter DNS Infra in the definition field at the top of the right pane.
6. Click the Save button  on the newly created release definition.
7. Rename the default environment  to Development by clicking on the text and typing the new name.
8. Add a PowerShell and two Publish Test Results tasks the same way you did for the build steps.
9. Click Close button on the Add Tasks dialog.

* Configure the Release tasks

1. Select PowerShell step and type “$(Build.DefinitionName)\Deploy\initiate.ps1” into Script filename and “-fileName Deploy” into the Arguments field.
2. Select the first Publish Test Results task and change the following properties:

* Select NUnit for Test Result Format.
* Type “$(Build.DefinitionName)\Deploy\InfraDNS\Tests\Results\Integration\*.xml” into Test Result Files field
* Enter “Integration” for Test Run Title.
* Check the box for Always run under Control Options.

1. Select the second Publish Test Results task and change the following properties:

* Select NUnit for Test Result Format.
* Type “$(Build.DefinitionName)\Deploy\InfraDNS\Tests\Results\Acceptance\*.xml” into Test Result Files field
* Enter “Acceptance” for Test Run Title.
* Check the box for Always run under Control Options.

1. Link the release to the build definition that you created in exercise 3 by clicking Link to a build definition link located above the release tasks that you just created and selecting DNS Infra from the Source dropdown.
2. Click the LINK button to close the Link to Artifact Source dialog.

* Enable continuous deployment

1. Click the Triggers tab at the top of the right hand pane.
2. Select the Continuous Deployment radio button.
3. Select DNS Infra from the trigger on artifact source dropdown.
4. At the bottom of the page click  next to the Development environment. This brings up a dialog to configure Deployment Conditions for the environment.
5. Select the After release creation radio button. This will kick off a deployment automatically when a new release is triggered.
6. Click OK to save and close the Deployment Conditions dialog.
7. Click the Save button  at the top of the right hand pane.

## Task 2 – Create Manual Release

In this task, you will manually kick off a development release.

* Create Release

1. In Microsoft Edge, select the Release tab located just under the blue ribbon at the top of the page.
2. Select the DNS Infra release definition that you created in Task 1.
3. Click the  button and select Create Release to manually start a release.
4. Select the latest (largest number) artifact version in the dialog that comes up. Leave all of the other options with the default values.
5. Click Create.
6. A new release will be created (Should be Release - 1). You will see the release in the list of releases.
7. Double click the release that was just created to view its summary. It will show as .
8. After about a minute, you will see the Deployment status change to Succeeded.  You have successfully deployed the DNS server!

* Review Tests

1. In the Release summary view, you now have test results for both the Integration and Acceptance tests on the bottom right of the summary page.
2. Click Detailed Report > to see details for the Integration and Acceptance tests. This test report is the same as the test report for the unit tests. Notice all tests have passed .

* Manually Verify Release

1. In Microsoft Edge, add a new tab.
2. Browse to [www.contoso.com](http://www.contoso.com) to verify that the DNS records were set correctly and the website is accessible.

* Even though the configuration processed successfully and all tests passed this will fail.

## Task 3 – Change Configuration and Tests

In this task, you will update the configuration and tests to resolve the issue that was discovered in the manual validation in the previous task. To do this, you will add a cname record for www and change the tests to ensure that they are validating the DNS name instead of NetBIOS name of the web server.

* Update configuration data

1. If not already there, in VS Code file explorer, expand InfraDNS
2. Select DevEnv.ps1.
3. Add the ;’www’ = ‘TestAgent2’ to the CNameRecords property:

* CNameRecords = @{‘DNS’ = ‘TestAgent1.contoso.com’; ‘www’ = ‘TestAgent2.contoso.com’;};

1. Press Ctrl + S to save the change.

* Add acceptance tests

1. In Visual Studio Code, expand InfraDNS, Tests and Acceptance.
2. Select DNSServer.tests.ps1.
3. Add a test to the Context ‘DNS Addresses’ block as follows:

* It "Should resolve WWW to TestAgent2" {
* (Resolve-DnsName -Name www.contoso.com -Type CName -DnsOnly -NoHostsFile).NameHost | Should be 'TestAgent2'
* }

1. Replace TestAgent2 in the ‘Web server ports’ & ‘Website Content’ Context blocks with www as follows:

* $PortTest = Test-NetConnection -ComputerName www.contoso.com -Port 80
* $WebRequest = Invoke-WebRequest -Uri http://www.contoso.com -UseBasicParsing

1. Press Ctrl + S to save the change.

* Push changes to TFS

1. In PowerShell console, type the following and then press ENTER after each line:

* Cd \git\Demo\_CI
* Git add .
* Git commit -m ”New acceptance test”
* Git push
* Verify CI/CD process

1. In Microsoft Edge, click the Build tab located just under the blue ribbon at the top of the page.
2. Select the DNS Infra build definition that you created in Exercise 1.
3. Click Builds at the top of the right pane to view the list of builds.
4. Double click the latest build either in Queued or Completed to view the summary of the build.
5. Once the build shows Completed, click the Release tab located just under the blue ribbon at the top of the page.
6. Double click the latest release to view the summary of the build.
7. Wait for the Release to show a deployment status of Succeeded.

* Manually verify

1. In Microsoft Edge, add a new tab.
2. Browse to www.contoso.com to verify that the DNS records were set correctly and the web site is accessible.

**Congratulations**. You have successfully configured and run a Continuous Integration / Continuous Deployment pipeline for a development environment. Now take these concepts forward and build your own CI pipelines.