WorkshopPLUS Windows PowerShell Desired State Configuration

Module 2: Push

Student Lab Manual

Version 1.2

Conditions and Terms of Use

Microsoft Confidential - For Internal Use Only

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Contents

Lab 2: Push

Introduction

In this exercise we will write a DSC configuration, run the configuration, review the generated MOF file, and then push that MOF file to a node to apply the configuration.

Using PowerShell v5 ISE and the built-in DSC snippets, the creation of a configuration document can be fairly simple. Most settings can be discovered with Intellisense, which assists in the authoring process. Most errors in syntax within a configuration will be underlined in red.

> **NOTE:** Sometimes red underlining may indicate modules that have not been installed or loaded. The red underline will go away once the modules are installed and the ISE relaunched.

Objectives

After completing this lab, you will be able to:

- Write a DSC configuration
- Configure the LCM
- Push a DSC configuration to the localhost and remote nodes
- Describe how the MOF file is used within the push mode of DSC
- Determine the status on an applied DSC configuration
- Identify resource dependencies
- Troubleshoot a DSC configuration

Prerequisites

If you have not completed the Module 1 lab, then you will need to install WMF 5.0 by running the following two scripts in the order listed:

- C:\PShell\Labs\Lab00\Lab_00_01_01_Stage_Resources.ps1
- C:\PShell\Labs\Lab00\Lab_00_01_02_Install_WMF_5.ps1

Start all VMs provided for the workshop labs.

Logon to the **PULL** server as **Administrator** with the password **PowerShell5!**

Estimated time to complete this lab

90 minutes

For more information

Get Started with Windows PowerShell Desired State Configuration

https://msdn.microsoft.com/en-us/PowerShell/DSC/overview

Scenario

You work in the Windows server team, and you are tasked with creating a build script for new administrative jump box servers that are being deployed to manage the Windows file servers.

You need to configure the servers' LCM settings so they automatically reboot when applying DSC configurations and then you also need to push the configurations that will install the required RSAT tools for managing other file servers.

NOTE: Use the Windows PowerShell Integrated Scripting Environment (ISE) for typing and editing the PowerShell scripts in this course. The automatic Intellisense will be very helpful in the learning process. If you need to reactivate Intellisense, use the **CTRL SPACE** keyboard shortcut.

Each lab has its own files. For example, the lab exercises for module 2 have a folder at **C:\PShell\Labs\Lab02** on the **PULL** virtual machine. Use these provided files to avoid typing long sections of code.

We recommend that you connect to the virtual machines for these labs through Remote Desktop rather than connecting through the Hyper-V console. When you connect with Remote Desktop, you can copy and paste from one virtual machine to another.

Exercise 2.1: Authoring DSC Configurations

Objectives

In this exercise, you will:

- Use PowerShell ISE Snippets to author a DSC configuration
- Execute the DSC configuration to create the MOF file
- Identity the components of a DSC configuration document
- Describe some of the key attributes within a MOF file
- Create a meta configuration to configure the node LCM

Scenario

One of the best ways to learn DSC is to start with writing a DSC configuration. Creating configuration documents is an essential skill to master. With Intellisense and syntax highlighting in the Integrated Scripting Environment (ISE), this task is greatly simplified.

Before applying a configuration, you should review the target node LCM settings to ensure they match the requirements within your organization. This includes settings such as the frequency that configurations should reapply and whether a node should automatically reboot.

Finally, it is helpful to understand the MOF file format, because these documents are used to transport configurations to nodes. You do not need to write these by hand. This open specification file format applies to both Windows and non-Windows platforms, which is why it was chosen for DSC.

Task 2.1.1: Author a configuration

1. Open the PowerShell Integrated Scripting Environment (ISE) and begin a new script using one of these techniques:
   a. Press **CTRL+N**
   b. From the menu choose **File** > **New**
   c. Click the NEW icon on the taskbar.
2. It is recommended that you type all of the commands below each on a new line in this new script. Run the following command:

```
Set-Location C:\PShell\Labs\Lab02
```

> **NOTE:** You can highlight a selection of code and run it with the **F8** key OR execute only the current line by pressing the **F8** key (even without highlighting anything).

3. Run the following command to view which RSAT features are installed:

```
Get-WindowsFeature -Name RSAT*
```

Question A: Are the **File Services Tools** installed?

4. View the DSC-related snippets. On a new line in the ISE script pane press **CTRL+J**.
5. From the list of snippets, arrow down (or type "DSC") and select the **DSC Configuration (Simple)** snippet. Press **Enter** to paste it into the ISE script.

> **NOTE:** You can also start snippets from the **Edit** > Start **Snippets** menu option.

6. Now we will customize this template configuration for our own purpose. Change the configuration name from **Name** to **JumpServer**.
7. Update the node from **("Node1","Node2","Node3")** to **localhost**.
8. Update the WindowsFeature name **FriendlyName** to **FileServices**.
9. Inside the WindowsFeature resource, Update the Name = **"Feature Name"** to **"RSAT-File-Services"**.
10. Delete the **File** resource. Use care to remove the proper curly braces for this resource.
11. Delete the comment lines (any line beginning with #).
12. The configuration should look like the script below:

```
configuration JumpServer
{
    node localhost
    {
        WindowsFeature FileServices
        {
           Ensure = "Present"
           Name = "RSAT-File-Services"
        }
    }
}
```

13. Select the configuration lines as shown above and run them by pressing the **F8** key.

Question B: Is there any output? Why or why not?

14. Run the following command in the console pane of the ISE:

```
Get-Command -CommandType Configuration
```

15. You can see the configuration **JumpServer** has been loaded into memory and is available to be executed. Run the following command:

```
Get-Command -Name Ju*
```

Question C: What was the output?

NOTE: A **Configuration** shares similar properties to a Function. It is only loaded into the current session after you run it. Each time you make a change to the configuration you are required to run it again to reload it in memory.

16. In the console pane start typing the name of the configuration and use **TAB** complete for the remainder of the name. You may have to press **TAB** multiple times.

```
Ju<tab>

JumpServer
```

17. Now press **ENTER** to run the configuration **JumpServer**

Question D: What is the output?

NOTE: The snippet we used to create the DSC configuration was originally created in PowerShell v4 and has not been updated for PowerShell v5. The warning message was added in v5 when we do not explicitly import the DSC resources that we reference from within the configuration. You can safely ignore this warning for now.

18. Read the warning and follow the instructions to avoid the warning message from being shown when the configuration runs again. In the script pane above, before the **node** block, add the following line:

```
Import-DscResource -ModuleName PSDesiredStateConfiguration
```

19. Now highlight and execute the configuration to reload it into memory. The configuration should look the same as below.

```
configuration JumpServer
{
    Import-DscResource -ModuleName PSDesiredStateConfiguration
```

```
    node localhost
    {
        WindowsFeature FileServices
        {
           Ensure = "Present"
           Name = "RSAT-File-Services"
        }
    }
}
```

> **NOTE:** You can discover a configuration like a standard cmdlet or function with **TAB** complete.
>
> The first time you tab to discover a configuration you might discover the directory (e.g. **.**JumpServer). Press **TAB** again to discover the command (without the **.**).
>
> 1) Ju**<tab>**
>
> 2) **.**JumpServer**<tab>**
>
> 3) JumpServer

20. In the console pane rerun the configuration **JumpServer**.

Question E: Did the warning appear this time?

_____

Question F: What is the name of the **directory** and **file** that was output?

_____

_____

21. If you wish, compare your code with the final solution.

```
psEdit .\Lab_02_01_01_PUSH_Scenarios_Solution.ps1
```

Task 2.1.2: Review the MOF file format

1. Open the MOF file that was created in the previous task:

```
psEdit .\JumpServer\localhost.mof
```

Question G: Notice the logging comments at the beginning of the MOF file. Compare these to the properties at the bottom of the MOF file. Fill in the blanks with the value of each property:

i. TargetNode _____

ii. GeneratedBy _____

iii. GenerationDate _____

iv. GenerationHost _____

Question H: Why are these details important?

Question I: Compare the **MSFT_RoleResource** section of the MOF file to your configuration script. Do the property values of **Name** and **Ensure** match in both places?

Task 2.1.3: Author a meta configuration

1. Verify that you are in the **Lab02** directory.

```
Set-Location -Path C:\PShell\Labs\Lab02
```

2. Open up the lab script:

```
psEdit .\Lab_02_01_03_PUSH_Author_Meta_Local.ps1
```

3. Select and execute the following code in **#region 1** to query the current LCM settings from one of the nodes.

```
Get-DscLocalConfigurationManager
```

Question A: What is the current value for the following?

i. RebootNodeIfNeeded _____

ii. ConfigurationMode _____

NOTE: The LCM configuration is not default on the servers after Lab 1, which sets **RebootNodeIfNeeded** to **$true**. The default value for this property is **$false**.

4. Select and execute the following code in **#region 2** to create an LCM meta configuration for localhost:

```
[DscLocalConfigurationManager()]
Configuration LCMPushDisableReboot
{
    Node localhost
    {
        Settings
        {
            ActionAfterReboot              = 'ContinueConfiguration'
            ConfigurationMode              = 'ApplyAndAutoCorrect'
            RebootNodeIfNeeded             = $False
            RefreshMode                    = 'Push'
```

```
        }
    }
}

Set-Location -Path C:\PShell\Labs\Lab02

LCMPushDisableReboot
```

Question B: What is the output from running the configuration?

NOTE: In this exercise we created two **MOF** files. The **localhost.mof** file holds our configuration and the **localhost.meta.mof** file holds and LCM (meta) configuration. In the following exercise we will use these files to configure the localhost.

Exercise 2.2: Push DSC configurations

Objectives

In this exercise, you will:

- Deploy the meta configuration to configure the local node LCM
- Deploy a configuration to configure the local node

Scenario

During this exercise we will push the meta configuration that we previously created to the localhost. This will ensure the node does not allow configuration drift (**ApplyAndAutoCorrect**) and prevents reboots (**RebootNodeIfNeeded**).

Once the LCM is configured, the next step is to deliver the configuration to the target node. There are two ways to deliver the document, pull and push modes. During this exercise we will push the configuration to the localhost.

Task 2.2.1: Push a meta configuration

1. Change to the **Lab02** directory.

```
Set-Location -Path C:\PShell\Labs\Lab02
```

2. Open up the lab script:

```
psEdit .\Lab_02_02_01_PUSH_Meta_Local.ps1
```

3. Select and execute the following code in **#region 1** to review the current LCM settings:

```
Get-DscLocalConfigurationManager
```

4. Select and execute the following code in **#region 2** to view the previously-compiled meta configuration document:

```
Get-ChildItem -Path .\LCMPushDisableReboot
```

Question A: What is the name of the meta configuration file that we previously compiled?

5. Select and execute the following code in **#region 3** to push the LCM meta configuration for localhost:

```
Set-DSCLocalConfigurationManager -Path .\LCMPushDisableReboot -Verbose
```

6. Select and execute the following code in **#region 4** to view the current LCM settings for the localhost:

```
Get-DscLocalConfigurationManager
```

Question B: What is the new value for the following?

i. RebootNodeIfNeeded _____

ii. ConfigurationMode _____

Task 2.2.2: Push a configuration

1. Now we can apply the configuration (**localhost.mof**) to the **PULL** node.
2. First, review the **PSDesiredStateConfiguration** cmdlets. Run the following:

```
Get-Command -Module PSDesiredStateConfiguration
```

3. Verify that you see the cmdlet **Start-DscConfiguration** in the list.
4. Review the help for the **Start-DscConfiguration** cmdlet:

```
Help Start-DscConfiguration –Full
```

Question C: Read the **Description** portion of the help output. How can you run the cmdlet interactively?

5. From the help output copy and paste **Example 2** into a new ISE script. Update the path as follows:

```
-Path .\JumpServer
```

6. Execute the complete command:

```
Start-DscConfiguration -Path .\JumpServer -Wait -Verbose
```

> **NOTE:** This command applies all of the **MOF** files within the directory by matching the file name to a node name.

Question D: What was the output from this?

Question E: Why do we see the blue text?

Question F: How long did it take to run?

Question G: What node is this executing on?

7. Run the following command again:

```
Get-WindowsFeature -Name RSAT*
```

Question H: How is the output different from when we ran this command previously?

8. Notice that within the configuration **JumpServer** we used the **WindowsFeature** resource. View the syntax of **WindowsFeature** by executing the following command:

```
Get-DscResource -Name WindowsFeature -Syntax
```

Question I: What are the required parameters?

HINT: Anything without the square brackets [ ].

9. When looking at the syntax output from the command above, notice the extra properties that we did not use in the **JumpServer** configuration. Now add two new properties for this resource. Be sure to use the Intellisense in the ISE to assist. Continue to edit the exact same **JumpServer** configuration that you have already been writing in Task 2.1.1. In the **WindowsFeature** resource create a newline after:

```
Name = "RSAT-File-Services"
```

10. Press **CTRL+SPACE** to start the Intellisense.

NOTE: You can also use the **Edit** menu, **Start Intellisense**.

```
        WindowsFeature FileServices
        {
           Ensure = "Present"
           Name = "RSAT-File-Services"
           CTRL+SPACE
        }
```

11. Use **CTRL+SPACE** Intellisense to add each of the following:

```
LogPath              = 'c:\windowsfeatureinstall.log'
IncludeAllSubFeature = $true
```

12. Adjust the text alignment to look as follows:

```
        WindowsFeature FileServices
```

```
        {
            Ensure               = "Present"
            Name                 = "RSAT-File-Services"
            LogPath              = 'c:\windowsfeatureinstall.log'
            IncludeAllSubFeature = $true
        }
```

13. Highlight and run the configuration **JumpServer** with the updates.

> **NOTE:** This reloads the configuration but does not apply it.

14. Call the configuration again to regenerate the MOF file:

```
    JumpServer
```

15. Apply the new configuration again, however this time do not use the **Wait** or **Verbose** parameters.

```
Start-DscConfiguration -Path .\JumpServer
```

Question J: What is the output?

> **NOTE:** Revisit the help output for **Start-DscConfiguration** if you are unsure.

16. Repeat the following command until you see the job **State** of **Completed**:

```
Get-Job
```

17. Run the following command to find out if the log file was created:

```
Test-Path -Path C:\windowsfeatureinstall.log
```

18. Review the contents of the log file:

```
Get-Content -Path C:\windowsfeatureinstall.log
```

19. Remove the log file:

```
Remove-Item -Path C:\windowsfeatureinstall.log
```

20. Run the following command to view the output of the completed job:

```
Get-Job | Receive-Job -Keep -Verbose
```

Question K: How long did the configuration take to complete?

---

21. Now remove the job(s).

```
Get-Job | Remove-Job
```

22. Apply the new configuration one more time. You can see this run in real time again by using the **Verbose** and **Wait** switches.

```
Start-DscConfiguration -Path .\JumpServer -Wait -Verbose
```

Question L: How long did it take to run?

Question M: Why did it run so fast?

Question N: Which functions from the WindowsFeature resource were executed (Test, Set, Get)?

23. Now execute the following command:

```
Test-DscConfiguration
```

Question O: What is the output? What does this mean?

24. Now execute the following command:

```
Test-DscConfiguration -Detailed
```

Question P: What is the difference when you add the **-Detailed** switch?

Question Q: How can this be useful?

25. Confirm that the WindowsFeature **RSAT-File-Services** and all subfeatures are installed:

```
Get-WindowsFeature -Name RSAT*

Get-DscConfiguration
```

26. Check the LCM status, by executing the following:

```
Get-DscConfigurationStatus
```

27. Review the LCM status history, by executing the following:

```
Get-DscConfigurationStatus -All
```

28. If you wish, compare your code with the final solution.

```
psEdit .\Lab_02_02_02_PUSH_Scenarios_Solution.ps1
```

Task 2.2.3: Restore the previous configuration

1. The Intellisense for writing DSC configurations was enhanced with WMF 5.0. This dramatically assists with authoring documents by discovery of properties and property values. Modify the configuration **JumpServer**. Under **WindowsFeature FileServices** next to **Ensure = "Present"** delete **"Present"**. Then press **TAB** until the setting changes to **'Absent'**.

```
Ensure = TAB
Ensure = 'Absent'
```

2. We use **Ensure = 'Absent'** to remove a setting which we had been previously applied. Unlike settings applied with Group Policy, a DSC setting will not be undone if the DSC configuration is removed. The explicit **Absent** value is required to remove a setting. Also, remove the line **IncludeAllSubFeature = $true**, since we don't use this when we are removing features. The resource should appear as follows:

```
        WindowsFeature FileServices
        {
           Ensure = 'Absent'
           Name = "RSAT-File-Services"
           LogPath = 'c:\windowsfeatureinstall.log'
        }
```

3. Highlight and run the configuration **JumpServer** with the update. This reloads the configuration but does not apply it.

4. Run the configuration to create the MOF file:

```
JumpServer
```

5. Push the MOF file:

```
Start-DscConfiguration -Path .\JumpServer -Wait –Verbose
```

Question A: How long did that take to execute?

Question B: Why does it take longer than the last time you applied it?

6. Run the following command:

```
Get-WindowsFeature -Name RSAT*
```

Question C: What is the **Install State** of the **File Services Tools** feature and sub-features?

NOTE: There is a small chance that the removal of these features may request a reboot.

7. Review the **MOF** files under the **Configuration** directory:

```
Get-ChildItem -Path C:\Windows\System32\Configuration -Filter *.mof
```

8. Notice the **Current.mof** and the **Previous.mof**. View these two MOF files and examine the configurations.

```
psEdit C:\Windows\System32\Configuration\Current.mof
psEdit C:\Windows\System32\Configuration\Previous.mof
```

Question D: Can you read the MOF files or do they look encrypted?

9. Close the **MOF** files.

10. Run the following:

```
 Get-Command -Module PSDesiredStateConfiguration
```

Question E: Which cmdlet allows us to return to a previous configuration?

11. Execute a roll back using the following command:

```
Restore-DscConfiguration –Verbose
```

12. Having the ability to easily roll back a configuration is a very important feature of DSC. However, keep in mind that settings will not necessarily be reversed. It simply ensures the previous configuration is reapplied. Run the following command:

```
Get-WindowsFeature -Name RSAT*
```

Question F: Are the RSAT features as expected? What changed?

13. Remove the WindowsFeature install log:

```
Remove-Item -Path c:\windowsfeatureinstall.log
```

14. If you wish, compare your code with the final solution.

```
psEdit .\Lab_02_02_03_PUSH_Scenarios_Solution.ps1
```

Exercise 2.3: Exploring DSC Configurations

Objectives

In this exercise, you will:

- Define resource sequence using the **DependsOn** property
- Recognize and resolve configuration errors

Scenario

Writing and applying a simple DSC configuration with only a few resources should be relatively simple. However, as the configurations grow in size and complexity, we need the skills to ensure they are written and applied successfully. Over time configurations will change and errors may be created.

It is important to recognize that the order that the resources are written within the configuration does not relate to the order that the configurations will be applied. We rely on the use of the **DependsOn** property to define prerequisites of the order that resoruces are applied. An individual resource will only apply after all prerequisites have successfully applied.

Task 2.3.1: The DependsOn resource property

1. Add a **File** resource to the **JumpServer** configuration after the **WindowsFeature** resource. Use Intellisense to complete the settings below. **CTRL+SPACE** will invoke Intellisense.

```
File LogDir
{
    DestinationPath = 'H:\Logs'
    Type            = 'Directory'
    Ensure          = 'Present'
}
```

> **NOTE:** Some early versions of WMF 5.0 have issues with Intellisense in the ISE hanging. You may notice grey RUN icons on the toolbar when this happens. In this case close the ISE and complete the script in a new ISE without using TAB completion or Intellisense.

2. Within the **WindowsFeature** resource, add a new property value for **DependsOn**. Use Intellisense and the **TAB** key to assist when writing new values.

```
DependsOn = '[File]LogDir'
```

3. The result of these configuration changes should resemble the script below:

```
WindowsFeature FileServices
 {
    Ensure               = 'Absent'
    Name                 = "RSAT-File-Services"
    LogPath              = 'c:\windowsfeatureinstall.log'
```

```
           DependsOn            = '[File]LogDir'
       }

       File LogDir
       {
           DestinationPath = 'H:\Logs'
           Type            = 'Directory'
           Ensure          = 'Present'
       }
```

4. By using the **DependsOn** property we ensure that **FileServices** resource will not execute until the **LogDir** resource has successfully processed. Select and run the configuration **JumpServer** with the updates. This reloads the configuration but does not apply it.

5. Run the configuration to create the MOF file:

```
JumpServer
```

6. Push the MOF file:

```
Start-DscConfiguration -Path .\JumpServer -Wait -Verbose
```

Question A: What was the outcome from applying the configuration?

Question B: How long did it take to execute?

NOTE: This configuration should fail with the message **The system cannot find the path specified.** This is because the H:\ drive that was used in the value for the DestinationPath does not exist.

7. Run the following commands:

```
Get-DscConfiguration

Test-DscConfiguration -Detailed
```

Question C: What is the output?

Question D: What is the warning?

Question E: Why did we get the results shown?

Question F: Does the order of the resources inside of the configuration matter in relation to the order in which the resources are executed?

NOTE: It is important to understand that the resource **[WindowsFeature]FileServices** was not executed because of the setting: **DependsOn = '[File]LogDir'**. Only after the resource **[File]LogDir** applies successfully will the resource **[WindowsFeature]FileServices** apply.

8. If you wish, compare your code with the final solution.

```
psEdit .\Lab_02_03_01_PUSH_Scenarios_Solution.ps1
```

Task 2.3.2: Pending, Current, and Previous MOF files

1. Update the **JumpServer** configuration so that the **DestinationPath** is set to **C:\Logs**:

```
DestinationPath = 'C:\Logs'
```

2. Update the **JumpServer** configuration so that the **LogPath** is set to **C:\Logs\windowsfeatureinstall.log**:

```
LogPath = 'c:\Logs\windowsfeatureinstall.log'
```

3. Highlight and run the configuration **JumpServer** with the updates. This reloads the configuration but does not apply it.

4. Run the configuration to create the MOF file:

```
JumpServer
```

5. Push the MOF file:

```
Start-DscConfiguration -Path .\JumpServer -Wait -Verbose
```

Question G: What is the outcome from applying the configuration? Read the error message.

6. Check for the pending configuration MOF file:

```
Get-ChildItem -Path C:\Windows\System32\Configuration -Filter *.mof
```

Question H: What is the name of the **MOF** file?

Question I: What are the names of the other two **MOF** files we saw earlier in this lab?

7. Check the status of the LCM:

```
Get-DscLocalConfigurationManager
```

Question J: What is the **LCMState**?

8. Add the **Force** switch and push the configuration again:

```
Start-DscConfiguration -Path .\JumpServer -Wait -Verbose -Force
```

Question K: What was the outcome?

9. Confirm that the directory is created, that the Windows features are removed, and that we are in the desired state:

```
Get-ChildItem -Path C:\Logs

Get-WindowsFeature -Name RSAT*

Test-DscConfiguration -Detailed
```

NOTE: Now the resource **[File]MyDir** is listed under **ResourcesInDesiredState**, because the resource **[WindowsFeature]FileServices** has been executed

10. Run the following command:

```
Get-Command -Module PSDesiredStateConfiguration
```

Question L: Which command can delete the current and previous **MOF** files?

11. Run the following commands:

```
Remove-DscConfigurationDocument -Stage Current -Verbose
Remove-DscConfigurationDocument -Stage Previous –Verbose
```

12. Check the **MOF** files:

```
Get-ChildItem -Path C:\Windows\System32\Configuration -Filter *.mof
```

13. Now check the LCM status:

```
Get-DscLocalConfigurationManager
```

Question M: What is the LCM State?

14. Finally, clean up the items we created during this lab:

```
Remove-Item -Path C:\Logs -Recurse
```

15. If you wish, compare your code with the final solution:

```
psEdit .\Lab_02_03_02_PUSH_Scenarios_Solution.ps1
```

Exercise 2.4: Push configurations to remote nodes

Objectives

In this exercise, you will:

- Modify LCM settings on remote nodes
- Push configurations to remote nodes

Scenario

We have previously written both the LCM meta configuration and standard node configurations. Then we applied them to the localhost. In this exercise we will follow the same process. However, we will push these to remote nodes.

Task 2.4.1: Push a meta configuration to remote nodes

1. Change to the **Lab02** directory.

```
cd C:\PShell\Labs\Lab02
```

2. Open up the lab script **Lab_02_04_01_PUSH_LCM.ps1**.
3. Select and execute the following code in **#region 1** to query the current LCM settings from the node **MS1**.

```
Get-DscLocalConfigurationManager -CimSession MS1
```

Question A: What is the current value for the following?

i. RebootNodeIfNeeded _____

ii. ConfigurationMode _____

NOTE: The LCM configuration is not default on the servers after Lab 1, which sets **RebootNodeIfNeeded** to **$true**. The default value for this property is **$false**.

4. Select and execute the following code in **#region 2** to create an LCM meta configuration:

```
[DscLocalConfigurationManager()]
Configuration LCMPushDisableReboot
{
    Node ('MS1','MS2')
    {
        Settings
        {
            ActionAfterReboot  = 'ContinueConfiguration'
            ConfigurationMode  = 'ApplyAndAutoCorrect'
            RebootNodeIfNeeded = $False
            RefreshMode        = 'Push'
        }
    }
}
```

```
Set-Location -Path C:\PShell\Labs\Lab02
LCMPushDisableReboot
```

Question B: What is the output from running the configuration?

5. Select and execute the following code in **#region 3** to push the meta configurations to the two nodes:

```
Set-DSCLocalConfigurationManager -Path .\LCMPushDisableReboot -Verbose
```

6. Confirm the changes have applied by executing the following code in **#region 4**:

```
$CS = New-CimSession -ComputerName MS1,MS2
Get-DscLocalConfigurationManager -CimSession $CS
```

Task 2.4.2: Push a configuration to remote nodes

1. Change to the **Lab02** directory.

```
cd C:\PShell\Labs\Lab02
```

2. Open up the lab script **Lab_02_04_02_PUSH_Remote_Nodes.ps1**.
3. Select and execute the following code in **#region 1** to define the **BaselineConfig** configuration and compile the MOF files.

```
Configuration BaselineConfig
{
    Import-DscResource -ModuleName PSDesiredStateConfiguration

    Node ('MS1','MS2')
    {
        WindowsFeature RemoveTelnetServer
        {
            Name = 'Telnet-Server'
            Ensure = 'Absent'
        }
    }
}

Set-Location -Path C:\PShell\Labs\Lab02
BaselineConfig
```

Question C: How many MOF files were created and why?

4. Select and execute the following code in **#region 2** to apply the configuration.

```
Start-DscConfiguration -Path .\BaselineConfig -Verbose -Wait
```

Question D: How many nodes applied the configuration and why?