WorkshopPLUS Windows PowerShell Desired State Configuration

Module 5: Resources

Student Lab Manual

Version 1.2

Contents

Lab 5: Resources

Introduction

In PowerShell Desired State Configuration resources are the code that implement the configuration. Many are provided out-of-the-box, and others can be acquired online. In this lab you will work with built-in and external resources. Then you will deploy them using both the push and pull methods.

Objectives

After completing this lab, you will be able to:

- Integrate configuration resources that are both built-in and open source
- Stage resources for node use via push
- Stage resources for node use via pull

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

These scripts will reboot the MS1, MS2, and PULL servers once complete. You will need to reconnect to the PULL virtual machine.

If you have not completed the **Module 4** lab, then you will need to configure the HTTPS pull server by running the following two scripts in the order listed:

- C:\PShell\Labs\Lab00\Lab_00_02_01_Prep_HTTPS_Pull_Server.ps1
- C:\PShell\Labs\Lab00\Lab_00_02_02_Build_HTTPS_Pull_Server.ps1

The **C:\PShell** folder on server **PULL** must be shared out with **Everyone** having **Read** access. This should have been completed in the **Module 1** lab. Run the following line to be sure. You will see an error if the share already exists.

```
New-SmbShare -ReadAccess Everyone -Path C:\PShell -Name PShell
```

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

Estimated time to complete this lab

90 minutes

For more information

DSC Resources

https://msdn.microsoft.com/en-us/powershell/dsc/resources

Setting up a DSC web pull server

https://msdn.microsoft.com/en-us/powershell/dsc/pullserver#placing-configurations-and-resources

PowerShell Gallery

http://www.powershellgallery.com

Scenario

In the PowerShell Gallery you found a resource that meets your needs. You want to install it and then deploy it to nodes with configurations.

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 5.1: Exploring Resources

Objectives

In this exercise, you will:

- Discover built-in resources and their source code
- Download and use resources from the PowerShell Gallery
- Write a configuration using these resources

Scenario

You need to configure a file server with a new share. You want to create the folder, populate the files, and then share it out for users. You will also need to open the firewall for SMB traffic.

Task 5.1.1: Built-In

1. Get a list of all available resources installed on the **PULL** server.

```
Get-DscResource
```

Question A: You may see both the built-in resources and possibly others installed from previous labs. Which one resource module name does not begin with an “x”?

____

> **HINT:** Scroll up to see the **ModuleName** column in the output.

2. List just the DSC resources in the built-in module **PSDesiredStateConfiguration**.

```
Get-DscResource -Module PSDesiredStateConfiguration
```

3. Reference the help for the cmdlet **Get-DscResource**.
4. Display the syntax for the **File** and **Registry** resources. You will use this command later for writing configurations with these resources.
5. Explore the properties of the built-in module **PSDesiredStateConfiguration**.

```
Get-Module -Name PSDesiredStateConfiguration

Get-Module -Name PSDesiredStateConfiguration | fl *
```

Question B: Which property of the module holds the module install path?

____

6. List the contents of the module directory.

```
dir (Get-Module -Name PSDesiredStateConfiguration).ModuleBase
```

7. List the contents of the DSCResources subdirectory.

```
dir (Join-Path (
Get-Module -Name PSDesiredStateConfiguration).ModuleBase 'DSCResources')
```

NOTE: This is one command line, but it is wrapped in the lab document.

8. Now explore the source code behind the **Registry** resource.

```
Get-DscResource Registry | fl *

dir (Get-DscResource Registry).ParentPath

psedit (Get-DscResource Registry).Path
```

WARNING: **DO NOT** edit any code in the **MSFT_RegistryResource.psm1** file.

9. Close the Registry resource file.

10. The point here is that you can find and review the source code for most built-in DSC resources by exploring the module and resource directories. Those paths are found on the respective objects.

Task 5.1.2: External

1. Change into the lab directory:

```
cd C:\PShell\Labs\Lab05
```

2. Many PowerShell DSC resources begin as open source projects on GitHub. Briefly browse the site:

```
Start-Process https://github.com/PowerShell

Start-Process https://github.com/PowerShell/xNetworking
```

3. Resources managed by Microsoft get released from GitHub to the PowerShell Gallery. Briefly browse the site:

```
Start-Process https://powershellgallery.com
```

4. You can browse, download and install modules and resources from the PowerShell Gallery via the PowerShell command line. NuGet is the service that makes this possible, and you will be prompted to install it the first time you connect. If prompted, accept the NuGet install. Run the following commands:

```
Find-Module

Find-Module -Includes DscResource
```

```
Find-DscResource

Find-DscResource | Out-GridView

Find-DscResource -ModuleName xActiveDirectory -AllVersions
```

5. Notice the difference in the output of **Find-Module** and **Find-DscResource**. In order to download a **resource**, you must reference the **module** name that contains the **resource**. Before you install a module or resource, you should first save a local copy to review. When prompted, answer **Yes** to install from an untrusted repository.

```
Save-Module -Name xNetworking -Path .\
```

6. Now explore a downloaded resource:

```
dir .\xNetworking -Recurse

psEdit .\xNetworking\*\DSCResources\MSFT_xFirewall\MSFT_xFirewall.psm1
```

NOTE: The version subfolder will likely be different when you run this psEdit command. Adjust the path if necessary.

7. Once you are comfortable that the code is safe for use in your environment you can install it from the PowerShell Gallery. Answer **Yes** when prompted regarding the untrusted repository.

```
Install-Module xNetworking
```

8. You can even use **Out-GridView** for a convenient selection graphical interface. Sort the output and choose **xSmbShare** to install. The **-Force** parameter ignores the untrusted repository prompt.

```
Find-DscResource |
  Out-GridView -Title "Choose the modules you wish to install..." -PassThru |
  Install-Module -Force
```

9. Now you can find these resource modules installed in the standard PowerShell module path:

```
dir "C:\Program Files\WindowsPowerShell\Modules"
```

10. Use **Get-DscResource** to list the newly-installed resources. Your list should include the following rows:

```
ImplementedAs   Name                         ModuleName      Version
-------------   ----                         ----------      -------
PowerShell      xDefaultGatewayAddress       xNetworking     3.1.0.0
PowerShell      xDnsConnectionSuffix         xNetworking     3.1.0.0
PowerShell      xDNSServerAddress            xNetworking     3.1.0.0
PowerShell      xFirewall                    xNetworking     3.1.0.0
PowerShell      xIPAddress                   xNetworking     3.1.0.0
PowerShell      xNetConnectionProfile        xNetworking     3.1.0.0
PowerShell      xSmbShare                    xSmbShare       2.0.0.0
```

NOTE: Version numbers will differ from this sample output.

11. Finally, view the syntax of the two new resources you installed:

```
Get-DscResource xFirewall -Syntax
Get-DscResource xSmbShare -Syntax
```

Task 5.1.3: Using Resources in a Configuration

1. Now that we have the resources required to build a file share we are ready to write the configuration. This configuration will use both built-in and external resources. Open a new script tab in the **PowerShell ISE**.

2. Begin by creating a new **Configuration** block with the name **FileServer**:

```
Configuration FileServer
{
}
```

3. Inside the configuration block, create a **Node** block that targets **MS1**.

```
Configuration FileServer
{
    Node ms1
    {
    }
}
```

4. Above the **Node** block we must import the resource modules we will reference in the configuration.

```
Configuration FileServer
{
    Import-DscResource -ModuleName xNetworking
    Import-DscResource -ModuleName xSmbShare
    Import-DscResource -ModuleName PSDesiredStateConfiguration

    Node ms1
    {
    }
}
```

5. First, our configuration must open the firewall for SMB traffic, both inbound and outbound. Insert a blank line inside the curly braces of the **Node** block.

6. In the blue command pane at the bottom of the ISE display the syntax for the **xFirewall** resource.

```
Get-DscResource xFirewall -Syntax
```

7. Now copy and paste this syntax inside the **Node** block of your configuration script.

```
   Node ms1
    {

xFirewall [String] #ResourceName
{
    Name = [string]
    [Action = [string]{ Allow | Block | NotConfigured }]
    [Authentication = [string]{ NoEncap | NotRequired | Required }]
    [DependsOn = [string[]]]
    [Description = [string]]
    [Direction = [string]{ Inbound | Outbound }]
    [DisplayName = [string]]
    . . .
    . . .
    . . .
}
    }
```

NOTE: Property names may differ from this sample output. This sample output has been truncated.

8. Edit your **xFirewall** resource to use the name, properties and values listed below:

```
        xFirewall SMB-In
        {
            Name          = 'SMB-In'
            DisplayName   = 'SMB-In'
            Group         = 'File and Printer Sharing'
            Direction     = 'Inbound'
            Action        = 'Allow'
            Enabled       = $true
            LocalPort     = 445
            Protocol      = 'TCP'
            Profile       = @('Domain','Private')
            Ensure        = 'Present'
        }
```

NOTE: Because external resources are constantly updated in the community, it is very possible that the property names and values have changed since the writing of this lab. If necessary, use **Get-DscResource resourcename -Syntax** to find and modify new property names to achieve the same goals set out in the sample code above. You may also research the resource documentation on GitHub: http://github.com/powershell.

9. Copy and paste your completed **xFirewall SMB-In** resource as a template for the **SMB-Out** resource. Then edit the name and values as shown below:

```
        xFirewall SMB-Out
        {
            Name          = 'SMB-Out'
            DisplayName   = 'SMB-Out'
            Group         = 'File and Printer Sharing'
            Direction     = 'Outbound'
            Action        = 'Allow'
            Enabled       = $true
            RemotePort    = 445
            Protocol      = 'TCP'
            Profile       = @('Domain','Private')
            Ensure        = 'Present'
        }
```

10. Next we will use Intellisense to write the configuration for the **File** resource. Insert two blank lines after the **xFirewall SMB-Out** resource.

11. Now insert a File resource and name as below:

```
        File DirectorySource
        {
        }
```

12. Click inside the curly braces of the **File** resource and insert a blank line. Use Intellisense and **TAB** completion to define resource properties.
13. Press **CTRL+SPACE** to invoke Intellisense.
14. Use the arrow keys to select the **Ensure** property and press **ENTER**.
15. After the equals sign type a single quote and then press **TAB** twice to complete the line as:

```
Ensure = 'Present'
```

16. Use Intellisense and **TAB** completion as much as possible to fill in the remaining resource properties as follows:

```
        File DirectorySource
        {
            Ensure          = 'Present'
            Type            = 'Directory'
            DestinationPath = 'C:\FileShare\'
            SourcePath      = '\\pull\pshell\labs'
            Recurse         = $true
            MatchSource     = $true
            Checksum        = 'ModifiedDate'
            DependsOn       = @('[xFirewall]SMB-In','[xFirewall]SMB-Out')
        }
```

NOTE: You will need to do some code formatting to keep the script clean and readable. Use **TABs** and **SPACEs** as appropriate to create the layout illustrated above. Also, be sure that the closing curly braces "}" for Configuration and Node are positioned correctly at the end of the script.

17. Use the same process to add the final **xSmbShare** resource with the following properties:

```
        xSmbShare CreateShare
        {
            Name       = 'SourceShare'
            Path       = 'C:\FileShare'
            ReadAccess = 'Everyone'
            Ensure     = 'Present'
            DependsOn  = '[File]DirectorySource'
        }
```

Question A: Read through the configuration script you have created. Study the **DependsOn** property of each resource. Which resource will run last?

> **NOTE:** Because external resources are constantly updated in the community, it is very possible that the property names and values have changed since the writing of this lab. If necessary, use **Get-DscResource resourcename -Syntax** to find and modify new property names to achieve the same goals set out in the sample code above. You may also research the resource documentation on GitHub: http://github.com/powershell.

18. Finally, add one line at the bottom of the script to call the **Configuration**. This should be a new line below the closing curly brace of the entire configuration.

```
FileServer
```

19. Save and run your configuration script. This should generate a **MOF** file for **MS1** in a subdirectory named **FileServer**.
20. The configuration is now complete, using both built-in and external resources. In the following exercises we will publish the external resources and configuration in both push and pull modes. Compare your script to the solution in lab file **Lab_05_01_03_Configuration_Solution.ps1**. Correct any difference you find in your script.

Exercise 5.2: Staging Resource for Push

Objectives

In this exercise, you will:

- Push a configuration containing external resources to a remote node
- Test a correction to configuration drift

Scenario

You have downloaded some custom resource modules from the internet and installed them onto your configuration build server. You have also written a configuration that uses these resources. Now you need to push both the configuration and the new resources to a target node.

Task 5.2.1: Configure the LCM for Push

1. Before a target node can receive a pushed configuration, the Local Configuration Manager (**LCM**) must be set to **Push** mode. First, query the LCM configuration of **MS1**.

```
Get-DscLocalConfigurationManager -CimSession ms1
```

Question A: What is the value of the **RefreshMode** property?

____

2. Open up the lab script **Lab_05_02_01_LCM_Push.ps1**. Read the script and run it.
3. Repeat the command from step 1.

```
Get-DscLocalConfigurationManager -CimSession ms1
```

Question B: What is the value of the **RefreshMode** property?

____

Task 5.2.2: Push the Resources

1. Attempt to push the configuration created in Exercise 5.1.

```
cd C:\PShell\Labs\Lab05

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

Question C: Did it succeed? Why or why not?

____

2. Pushing PowerShell DSC resource models to a target node is nothing more than a traditional file copy operation. If this is done using UNC paths or mapped drives, then an SMB firewall inbound rule will need to be enabled on the target node. However, using the new PowerShell v5 session parameters of **Copy-Item** we can copy files over WinRM/WSMan without opening any additional ports. First create two variables to hold the source and destination file paths:

```
$Source = 'C:\Program Files\WindowsPowerShell\Modules\[xc]*'

$Dest   = 'C:\Program Files\WindowsPowerShell\Modules\'
```

3. Notice that the source path only includes modules that begin with 'x' or 'c'. This keeps us from disturbing any other modules that may be installed in the path, since most DSC modules begin with one of those characters. Next notice that the destination path is a local path. This is because the Copy-Item session parameters use local paths in the context of PowerShell remoting on the remote node.

4. Open a new PowerShell remoting session to **MS1**.

```
$pss = New-PSSession ms1
```

5. The **Copy-Item** cmdlet has two new parameters in PowerShell v5: **FromSession** and **ToSession**. These are used in combination with the usual **Path** (source) and **Destination** parameters. In this case we need to copy a local path to a destination path inside a remoting session. We want this to be recursive. Finally, we will add the **Force** parameter in case something is already in the path, and we will add the **Verbose** parameter to see a list of every file as it is copied. Run this line:

```
Copy-Item -Path $Source -ToSession $pss -Destination $Dest -Recurse -Force
-Verbose
```

NOTE: This is one line, but it is wrapped in the lab document.

NOTE: You may see an **Access Denied** error message for unrelated files. Safely ignore the error for now.

6. Now verify the remote files are present with **Invoke-Command**. Notice the **$using:** variable prefix; this makes a local variable available in a remoting session. Observe that there may be additional modules from previous labs copied as well.

```
Invoke-Command -Session $pss -ScriptBlock {dir $using:Dest}
```

7. Repeat the same command and make it recursive:

```
Invoke-Command -Session $pss -ScriptBlock {dir $using:Dest -Recurse}
```

8. Finally, remove the remoting session:

```
Remove-PSSession $pss
```

Task 5.2.3: Push the Configuration

1. Now that the configuration **MOF** has been generated and the module files have been copied to the target node, we are ready to push the configuration. Make sure you are in the proper path and try this line again:

```
cd C:\PShell\Labs\Lab05

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

Question D: Did this command succeed? Why or why not?

____

____

____

2. The failed configuration attempt from the previous lab task is still pending. You must use the **-Force** switch to apply the configuration:

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

3. Verify the configuration with DSC cmdlets:

```
Test-DscConfiguration -CimSession ms1

Get-DscConfiguration -CimSession ms1
```

Task 5.2.4: Configuration Drift

1. Unauthorized changes can introduce **configuration drift**, when settings no longer match the desired state. View the contents of the file share created in the previous lab task:

```
Invoke-Command -ComputerName ms1 -ScriptBlock {dir C:\FileShare}
```

2. Recursively delete the **Lab03** directory in the path:

```
Invoke-Command -ComputerName ms1 -ScriptBlock {del C:\FileShare\Lab03 -Force
-Recurse}
```

NOTE: This is one line, but it is wrapped in the lab document.

3. View the directory again to see the deletion:

```
Invoke-Command -ComputerName ms1 -ScriptBlock {dir C:\FileShare}
```

Question E: What are two ways PowerShell DSC can repair this change from the desired state?

4. Study the parameters available on the cmdlet **Start-DscConfiguration**.

```
Get-Command Start-DscConfiguration -Syntax
```

Question F: Instead of pushing the entire configuration again, which parameter would you use to reapply the current configuration?

5. Repair the configuration on **MS1** using the following command:

```
Start-DscConfiguration -CimSession ms1 -UseExisting -Wait -Verbose -Force
```

6. Inspect the contents of the remote directory:

```
Invoke-Command -ComputerName ms1 -ScriptBlock {dir C:\FileShare}
```

Question G: Has the **Lab03** directory been replaced? If yes, where did the files come from?

Exercise 5.3: Staging Resource for Pull

Objectives

In this exercise, you will:

- Stage modules on the pull server
- Stage a configuration on the pull server
- Configure a node to pull a configuration and the required modules

Scenario

You have downloaded some custom resource modules from the internet and installed them onto your configuration build server. You have also written a configuration that uses these resources. Now you need to publish both the configuration and the new resources for a target node to pull.

Task 5.3.1: Stage the Modules

1. In this exercise we will **pull** the **FileServer** configuration to **MS2**, instead of using **push** as we did for **MS1**. This configuration uses the **xNetworking** and **xSmbShare** resource modules. Begin by setting the current directory:

```
cd C:\PShell\Labs\Lab05
```

2. View the modules that are installed locally on the pull server. Note that these are for local use and have not been published for pull.

```
dir "$Env:PROGRAMFILES\WindowsPowerShell\Modules"
```

3. Now inspect the DSC pull service folder to see if any modules have been published there yet. Use a variable for the path to save typing in future commands.

```
$PullModulePath = "$Env:PROGRAMFILES\WindowsPowerShell\DscService\Modules"

dir $PullModulePath
```

Question A: Are there any modules here? Why or why not?

4. To publish a module for pull you must do the following:
   a. Compress the module directory into a ZIP file
   b. Rename the ZIP file to include the module version
   c. Put the ZIP file into the DSC pull service module directory
   d. Create a checksum for the file

First we will do this using manual script steps. Put the module into a variable, and view its version and install path:

```
$m = Get-Module xNetworking -ListAvailable

$m | Select-Object Name, Version, ModuleBase
```

Question B: Why do you need the **ListAvailable** parameter for **Get-Module**?

____

____

____

5. Next you must construct the destination path for the compressed module ZIP file:

```
$PullModuleFile = Join-Path -Path $PullModulePath -ChildPath
"$($m.Name)_$($m.Version).zip"
```

NOTE: This is one line, but it is wrapped in the lab document.

6. Use one of the new ZIP cmdlets in PowerShell v5 to compress the module folder and locate the file in the DSC pull service module path:

```
Compress-Archive -Path "$($m.ModuleBase)\*" -DestinationPath $PullModuleFile
-Update -Verbose
```

NOTE: This is one line, but it is wrapped in the lab document.

7. Create the checksum file. It is strongly recommended to always use the **Force** switch with **New-DscChecksum**, because it will not overwrite any existing checksum file by default.

```
New-DscChecksum $PullModuleFile –Force
```

8. Finally, view the pull service module path to see the published module.

```
dir $PullModulePath
```

9. It would be time consuming to manually publish every resource module in this manual fashion. The obvious solution is to create a custom function for publishing modules. Open up the lab script file **Lab_05_03_01_Publish_Modules_Function.ps1**.

10. At the bottom of the script file add a line to call the function and publish the **xSmbShare** module. Save and run the updated script file.

```
Publish-DscResourcePull -Module xSmbShare
```

11. View the pull service module path to see the published modules:

```
dir "$Env:PROGRAMFILES\WindowsPowerShell\DscService\Modules"
```

Task 5.3.2: Stage the Configuration

1. Previously in Module 3 we discussed the steps to publish a configuration for pull:
   a. Generate the **MOF**
   b. Rename it with the Configuration **GUID** of the target node
   c. Move it to the pull service **Configuration** directory
   d. Generate the **checksum**

   Begin by opening the lab file **Lab_05_03_02a_Configuration.ps1**.
2. This is the same configuration we used earlier for server MS1. This time we will apply it to MS2. Edit line 6 and change the **Node** to **MS2**.

```
Node ms2
```

3. Save and run the script.
4. Open up the lab script **Lab_05_03_02b_Staging.ps1**.
5. This script renames and publishes the MOF file with the ConfigurationID (GUID). Read the script and run it.
6. Verify that the output displays a ***GUID*.mof** and a ***GUID*.mof.checksum** file with the current date and time.

Task 5.3.3: Configure the LCM

1. Open the lab script **Lab_05_03_03_LCM_Pull.ps1**.
2. Read the script.

   Question C: Which essential setting needs a value in this LCM script?

   ____

3. Set the **RefreshMode** value to **Pull**.

```
RefreshMode = 'Pull'
```

4. Save and run the script.

   Question D: Did this configure the LCM on MS2? If not, what additional step is required?

   ____

   ____

5. Configure the LCM on MS2 by running the following command:

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

Task 5.3.4: Pull the Configuration and Modules

1. MS2 is now ready to pull a configuration. We can wait for the interval, or we can trigger it manually. However, we want to guarantee that the configuration pulls and installs the required modules. View the **Modules** directory on **MS2**:

```
$ModulesPath = "$Env:PROGRAMFILES\WindowsPowerShell\Modules\[xc]*"

Invoke-Command -ComputerName ms2 -ScriptBlock {dir $using:ModulesPath}
```

2. You may find modules there from previous labs, but you should not find **xNetworking** or **xSmbShare**. If you find these, it is possible that they were pulled dynamically in the previous minutes since the last task.

3. Force **MS2** to pull its configuration and modules:

```
Update-DscConfiguration -CimSession ms2 -Wait -Verbose
```

4. Once the configuration completes, check the modules folder again:

```
Invoke-Command -ComputerName ms2 -ScriptBlock {dir $using:ModulesPath}
```

Question E: Are the **xNetworking** and **xSmbShare** modules present now? Why or why not?