WorkshopPLUS Windows PowerShell Desired State Configuration

Appendix 1: Advanced Configurations (Lab 8)

Student Lab Manual

Version 1.0

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.

Contents

Lab 8: Advanced Configurations

Introduction

Real world configuration scripts will grow to include many resources across many servers. In this lab you will explore advanced methods of configuration scripting to manage configurations at scale. In doing so you will introduce multiple layers of abstraction to handle the complexity.

Objectives

After completing this lab, you will be able to write advanced configurations which use the following techniques:

- Parameters
- Scripting
- Composite resources
- Configuration data
- Credential encryption

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.

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

X minutes

For more information

Reusing Existing Configuration Scripts in PowerShell Desired State Configuration
http://blogs.msdn.com/b/powershell/archive/2014/02/25/reusing-existing-configuration-scripts-in-powershell-desired-state-configuration.aspx

Helper Function to Create a PowerShell DSC Composite Resource
http://blogs.technet.com/b/ashleymcglone/archive/2015/02/25/helper-function-to-create-a-powershell-dsc-composite-resource.aspx

Separating "What" from "Where" in PowerShell DSC
http://blogs.msdn.com/b/powershell/archive/2014/01/09/continuous-deployment-using-dsc-with-minimal-change.aspx

Scenario

You have been tasked with writing a configuration to build a **jump server** or **tools server**. After writing the configuration you notice redundant configuration patterns which you would like to simplify. You will experiment with various techniques to automate the configuration script that generates the MOF files.

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 8.1: Scripting a Configuration

Objectives

In this exercise, you will:

- Add parameters and variables to a configuration
- Use a looping technique to simplify redundant resources

Scenario

After writing the configuration script you want to replace hard-coded values with parameters to make the script more flexible. Also, you want to support a variable number of resources where possible.

After each modification to the configuration script you will compare the MOF files for any notable differences.

Prerequisites

Open and run the following setup script to install the latest version of external resources required by this lab: **C:\PShell\Labs\Lab08\Lab08_01_00_Setup.ps1**

> **NOTE:** If prompted, install NuGet as required by the setup script.

Task 8.1.1: Parameters

1. Begin by opening the following lab script in the PowerShell ISE: **C:\PShell\Labs\Lab08\Lab08_01_01a_Original_Configuration.ps1**
2. Read through the configuration script and answer the questions below:

   Question A: What is the configuration trying to accomplish?

   _____

   _____

   Question B: Which parts of the configuration could be more flexible if they were not hard-coded?

   _____

   _____

3. Run the configuration script by pressing **F5**.
4. The script will generate a MOF file and open it in the PowerShell ISE. Briefly read through the MOF file to get an idea of what it is doing. Do not be concerned with details of the file yet. Leave this MOF file open in the ISE for reference later.

5. Now open the following lab script:
   C:\PShell\Labs\Lab08\Lab08_01_01b_Parameters.ps1
6. Edit the script to use a parameter for the **Node** name. On line 3 insert a parameter block with **$ComputerName** as a string array datatype with a default value of the local computer name.

```
Param(
    [string[]]$ComputerName = $Env:COMPUTERNAME
)
```

7. On line 8 change the node name from **ms2** to the new parameter variable:

```
Node $ComputerName
```

8. On line 59 adjust the invocation of the configuration to include the new parameter and the node name:

```
JumpServer_2 -ComputerName ms2
```

9. Save and run the configuration script by pressing **F5**.

> **NOTE:** If you encounter any errors, compare your script edits to the solution file: C:\PShell\Labs\Lab08\Lab08_01_01b_Parameters_Solution.ps1

10. The script will generate a MOF file and open it in the PowerShell ISE. Briefly read through the MOF file and compare it to the previous MOF file still open in the PowerShell ISE.

    Question C: Ignoring the **GenerationDate**, **ConfigurationName**, and **SourceInfo** lines of the file, what significant differences do you notice between the previous MOF file and the new one?

    _____

    _____

11. Leave both MOF files open in the ISE for reference later.

Task 8.1.2: ForEach Loop

1. Now you must edit the multiple **WindowsFeature** resources to be dynamic via a parameter and a loop in the configuration script. Open the following file:
   C:\PShell\Labs\Lab08\Lab08_01_02_Looping.ps1
2. At the end of line 4 add a comma. Then on line 5 insert a new string array parameter called **$Feature**.

```
Param(
    [string[]]$ComputerName = $Env:COMPUTERNAME,
    [string[]]$Feature
)
```

3. On lines 11-18 create a **ForEach** loop to process each feature name supplied in the parameter variable:

```
        ForEach ($WindowsFeature in $Feature) {
            WindowsFeature $WindowsFeature
            {
                Name = $WindowsFeature
                IncludeAllSubFeature = $true
                Ensure = 'Present'
            }
        }
```

> **NOTE:** You may copy and paste parts of this code from a previous script to save typing time.

4. On line 40 modify the invocation of the configuration to include the new **Feature** parameter and the list of features to be installed:

```
JumpServer_3 -ComputerName ms2 -Feature 'Web-Mgmt-Console',
'Web-Scripting-Tools','RSAT-AD-Tools','RSAT-DNS-Server'
```

> **NOTE:** The text above is one single line in the PowerShell ISE, but it is wrapped in the lab document.

5. Save and run the configuration script by pressing **F5**.

> **NOTE:** If you encounter any errors, compare your script edits to the solution file: C:\PShell\Labs\Lab08\Lab08_01_02_Looping_Solution.ps1

6. The script will generate a MOF file and open it in the PowerShell ISE. Briefly read through the MOF file and compare it to the previous MOF files still open in the PowerShell ISE.

Question D: Ignoring the **GenerationDate**, **ConfigurationName**, and **SourceInfo** lines of the file, what significant differences do you notice between the previous MOF file and the new one?

Question E: What advantages does the looping technique offer over the previous version of the configuration script?

7. Close the PowerShell ISE.

Exercise 8.2: Composite Resources

Objectives

In this exercise, you will:

- Create a composite resource
- Use a composite resource in a DSC script

Scenario

From time to time you will encounter repeated combinations of resources that are used together. These patterns can be grouped into a **composite resource**, which then gets referenced in your script similar to a custom resource.

In this exercise you will create and leverage a composite resource for enabling Remote Desktop Protocol (RDP) on a target node.

Task 8.2.1: Create the Composite Structure

1. A **composite resource** has a similar file and directory structure to a **custom resource**. In this exercise you will create the structure below:

```
C:\PROGRAM FILES\WINDOWSPOWERSHELL\MODULES\contosoComposites
|   contosoComposites.psd1
|
\---DSCResources
    \---compositeEnableRDP
            compositeEnableRDP.psd1
            compositeEnableRDP.schema.psm1
```

2. Creating directories and files is easy to script in PowerShell, but we will perform these steps manually for the learning experience. Begin by changing into the PowerShell modules directory.

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

3. Composite resources are stored as sub-modules under a parent PowerShell module. Create a module directory called **contosoComposites**. Change location into the directory.

```
md contosoComposites

cd .\contosoComposites
```

4. In PowerShell a module must have a PSM1 file, a PSD1 file, or one of a few other file types. Create a module manifest (PSD1) file for the empty root parent module.

```
New-ModuleManifest -Path .\contosoComposites.psd1 -ModuleVersion '1.0'
```

5. Just like custom resources, composite resources reside under a subdirectory called **DSCResources**. Create this directory and change into its location.

```
md DSCResources
```

```
cd .\DSCResources
```

6. Now create a module subdirectory to hold the composite resource module and manifest. Name it **compositeEnableRDP**. Change into the new directory.

```
md compositeEnableRDP

cd .\compositeEnableRDP
```

7. For now, we will create an empty module file and a manifest that points to it.

```
New-Item -ItemType File -Name compositeEnableRDP.schema.psm1

New-ModuleManifest -Path .\compositeEnableRDP.psd1 -RootModule
compositeEnableRDP.schema.psm1
```

> **NOTE:** The text above is one single line in the PowerShell ISE, but it is wrapped in the lab document.

8. Next, validate that the file and directory structure is complete and correct. Use the commands below to view the structure:

```
cd \

tree "C:\Program Files\WindowsPowerShell\Modules\contosoComposites" /f /a
```

9. Compare the output from step 8 to the template in step 1 above. Make any corrections that are necessary.

10. Now that you have manually created a composite resource structure, you may optionally view this blog post for one example of a helper script:

    Helper Function to Create a PowerShell DSC Composite Resource
    http://blogs.technet.com/b/ashleymcglone/archive/2015/02/25/helper-function-to-create-a-powershell-dsc-composite-resource.aspx

Task 8.2.2: Create the Composite Resource

1. In the previous task you created the overall structure for a composite resource. Now you must add the actual configuration script to the **schema.psm1** module file. This file will look like any other PowerShell DSC configuration script, except it will not have a **Node** section. It can contain parameters, import DSC resource modules, use scripting techniques as demonstrated earlier, and more. First, in the PowerShell ISE open the empty file you created:

```
psEdit “C:\Program Files\WindowsPowerShell\Modules\contosoComposites
\DSCResources\compositeEnableRDP\compositeEnableRDP.schema.psm1”
```

> **NOTE:** The text above is one single line in the PowerShell ISE, but it is wrapped in the lab document.

2. Next, open the following lab file:
   C:\PShell\Labs\Lab08\Lab08_02_02_Composite.ps1
3. This is the configuration used earlier in the lab. You will remove everything unrelated to the RDP resources. First, remove the **WindowsFeature** resource section and the **ForEach** loop scripting around it.
4. Next, remove the **Node** line and its **opening and closing curly braces**. Be sure to remove the correct closing curly brace at the bottom of the script.
5. Remove the entire **Param** block at the top of the script. This includes the parentheses and everything inside them.
6. The composite only needs to import DSC resource modules for the resources it contains. Remove **PSDesiredStateConfiguration** from the imported modules list.
7. Now you must rename the configuration to match the name of the new composite resource. Edit the first line as follows:

```
Configuration compositeEnableRDP
```

8. Compare your results to the code below. Correct any differences.

```
Configuration compositeEnableRDP
{
    Import-DscResource -ModuleName xRemoteDesktopAdmin, xNetworking

    xRemoteDesktopAdmin RemoteDesktopSettings
    {
        Ensure = 'Present'
        UserAuthentication = 'Secure'
    }

    xFirewall AllowRDP
    {
        Name = 'DSC - Remote Desktop Admin Connections'
        Group = "Remote Desktop"
        Ensure = 'Present'
        Enabled = $true
        Action = 'Allow'
        Profile = 'Domain'
    }
}
```

9. Now copy and paste all of this code into the empty **compositeEnableRDP.schema.psm1** script file open in the other tab.
10. Save the **schema.psm1** file.
11. Finally, check that your composite resource is visible to PowerShell. **Get-DscResource** returns all types of resources, including your new composite resource. Open a fresh PowerShell console window and run the following command:

```
Get-DscResource
```

Question A: Can you find the new composite resource in the output?

Question B: Write down the following properties:

i. ImplementedAs _____

ii. Name _____

iii. ModuleName _____

iv. Version _____

Task 8.2.3: Use the Composite Resource in a Configuration

1. The last step is to reference the new composite resource in the previous configuration. Open the following configuration file in the PowerShell ISE: **C:\PShell\Labs\Lab08\Lab08_02_03_Configuration.ps1**

2. Open the **compositeEnableRDP.schema.psm1** file.

```
psEdit "C:\Program Files\WindowsPowerShell\Modules\contosoComposites
\DSCResources\compositeEnableRDP\compositeEnableRDP.schema.psm1"
```

> **NOTE:** The text above is one single line in the PowerShell ISE, but it is wrapped in the lab document.

3. Examine the **Import-DscResource** dynamic keyword line of each script.

   Question C: Which two resource modules are now duplicated in the configuration script and the composite resource?

4. Since the composite resource is importing **xNetworking** and **xRemoteDesktopAdmin**, these modules can be removed from the original configuration. In their place you must import the new composite resource module created in the previous task. Notice that you import the parent module name, not the composite resource name under the parent module. In the **configuration script** adjust the **Import-DscResource** line as follows:

```
Import-DscResource -ModuleName PSDesiredStateConfiguration, contosoComposites
```

> **NOTE:** The only reason to keep these other resource modules would be if another resource in the configuration came from these modules. This is not the case for our jump server configuration.

5. Next, you must remove the **xFirewall** and **xRemoteDesktop** resources from the configuration and replace them with the composite resource. This reduces the lines in the main configuration script significantly. Remove these two resources from the configuration.
6. Now add a new resource to the configuration. Use **TAB** complete and Intellisense. Before the closing brace of the **Node** block and after the **ForEach**…**WindowsFeature** resource type the following on a new line:

```
compositeEn
```

7. Press **TAB**. Autocompletion should finish the resource name as **compositeEnableRDP**.
8. Next, you must name the resource and insert the opening and closing braces. We did not add any parameters to the resource, so we can leave this composite resource without any property values. Your code should like similar to this:

```
        compositeEnableRDP RDP
        {
        }
```

9. Compare your configuration script to this file and correct any differences: **C:\PShell\Labs\Lab08\Lab08_02_03_Configuration_Solution.ps1**
10. Run the configuration script. This will generate a MOF and open it. Review the MOF. Leave the MOF file open.
11. Now open the original configuration: **C:\PShell\Labs\Lab08\Lab08_01_01a_Original_Configuration.ps1**
12. Compare the new configuration script (using looping and a composite resource) to the original script.

    Question D: How many lines are there in the original configuration from the first line to the closing brace?

    Question E: How many lines are there in the new configuration from the first line to the closing brace?

    ---

13. Run the original configuration script file by pressing **F5**. This will generate a MOF file and open it.
14. Compare the original MOF to the new MOF generated in step 10 above. Study the **ResourceID** and **SourceInfo** properties of the last two resources in the MOF files.

    Question F: What differences do you observe?

Question G: After comparing these two MOF files will they achieve the same end configuration state? Why or why not?

NOTE: Composites resources per se do not have to be published for push or pull, because they contain existing resources. However, the external resources referenced by a composite configuration will have to be published for nodes to use.

Exercise 8.3: Configuration Data

Objectives

In this exercise, you will:

- Use the automatic parameter **ConfigurationData** with a configuration
- Employ various techniques for generating **ConfigurationData**
- Use the automatic variables **$AllNodes**, **$Node**, and **$ConfigurationData** in a configuration

Scenario

In the previous exercises you discovered how to write flexible configurations using scripting, parameters, and composite resources. In this exercise you will further abstract DSC scripts using the **ConfigurationData** automatic parameter of configurations.

Using configuration data offers significant scalability to DSC configuration scripts. This technique allows the configuration script to remain static and checked into source control. Meanwhile, the configuration data that flows through the script can be easily updated or even generated dynamically.

In this exercise you will apply these principles to the jump server configuration from the previous two exercises.

Prerequisites

If you still have the ISE open from the previous exercise, then close it. Close any other lab files that may be open. In a new ISE open and run the following script to create a clean version of the composite resource from the previous exercise:
C:\PShell\Labs\Lab08\Lab08_03_00_Composite_Prerequisite.ps1

Task 8.3.1: Basic Configuration Data

1. In this task you will replace the previous configuration parameters with the **ConfigurationData** design. This involves moving the parameters into a new data structure, updating the parameter references in the configuration, and adjusting the parameters when invoking the configuration. Begin by opening the parameterized version of the configuration file:
   C:\PShell\Labs\Lab08\Lab08_03_01a_Parameter_Configuration.ps1
2. Click on the empty line 27 in the script.
3. Use PowerShell ISE snippets (**CTRL+J**) to insert a **DSC ConfigurationData** template.
4. At the beginning of line 27 insert a variable name assignment in front of the hash table:

```
$ServerData = @{
```

5. At the bottom of the configuration data delete the green comment line about saving the data to a file.

6. Delete the **Node2** and **Node3** sections of the data structure:

```
$ServerData = @{
    AllNodes = @(
        @{
            NodeName = "Node1"
            Role = "WebServer"
            },
        @{
            NodeName = "Node2"
            Role = "SQLServer"
            },
        @{
            NodeName = "Node3"
            Role = "WebServer"
            }
    )
}
```

7. Remove the comma at the end of the **Node1** hash table. Compare your results so far to the code sample below:

```
$ServerData = @{
    AllNodes = @(
        @{
            NodeName = "Node1"
            Role = "WebServer"
        }
    )
}
```

8. At this point we have a clean **ConfigurationData** structure to populate. Change the first line of the data from **NodeName = "Node1"** to this:

```
NodeName = 'ms2'
```

9. Replace the line **Role = "WebServer"** by copying and pasting the **Feature** parameter and values from the lines farther down in the script. Adjust it to look as follows. Remove the dash in front of **Feature**, and insert and equals (=) after **Feature**.

```
Feature = 'Web-Mgmt-Console','Web-Scripting-Tools','RSAT-AD-Tools',
'RSAT-DNS-Server'
```

> **NOTE:** The text above is one single line in the PowerShell ISE, but it is wrapped in the lab document.

10. Since we are passing all values into the configuration using a different method, we can completely delete the **Param** block at the top of the configuration.

```
Param(
    [string[]]$ComputerName = $Env:COMPUTERNAME,
    [string[]]$Feature
```

```
}
```

11. Now you have to change the former parameter values in the configuration to use the automatic variables **$AllNodes** and **$Node**. Change the line **Node $ComputerName** to the following:

```
Node $AllNodes.NodeName
```

12. Change the **ForEach** line to reference the features list coming from the node data:

```
ForEach ($WindowsFeature in $Node.Feature) {
```

13. The final edit is to replace the parameters after the configuration invocation with the automatic parameter **ConfigurationData**. At the bottom of the script remove the parameters after **JumpServer_6**.

14. Click at the end of the line **JumpServer_6**. Type a **SPACE** and a dash (-). Use Intellisense (**CTRL+SPACE** if it does not show automatically) to add the **ConfigurationData** automatic parameter.

15. Provide the variable **$ServerData** for the **ConfigurationData** parameter. Your completed line should look like this:

```
JumpServer_6 -ConfigurationData $ServerData
```

16. The script transformation is complete. Open the solution script to check your work. Make any required adjustments to match your script to the solution: **C:\PShell\Labs\Lab08\Lab08_03_01b_Solution.ps1**

17. Run the script file and review the MOF file result.

    Question A: Think back to the MOF files generated in the previous lab exercises. Is this MOF file significantly different? Why or why not?

18. To understand the power of the **ConfigurationData** structure, observe how easy it is to add nodes. Copy and paste the **ms2** node section multiple times. Adjust the **NodeName** and **Features** list for each one. Do not forget to insert a comma between each node hash table. Use the example below for reference:

```
$ServerData = @{
    AllNodes = @(
        @{
            NodeName = 'ms2'
            Feature = 'Web-Mgmt-Console','Web-Scripting-Tools','RSAT-AD-
Tools','RSAT-DNS-Server'
        },
```

```
        @{
            NodeName = 'ms3'
            Feature = 'Web-Mgmt-Console','Web-Scripting-Tools'
        },
        @{
            NodeName = 'ms4'
            Feature = 'RSAT-AD-Tools','RSAT-DNS-Server'
        }
    )
}
```

NOTE: The **Feature** text for node **ms2** above is one single line in the PowerShell ISE, but it is wrapped in the lab document.

19. Press **F5** to run the entire configuration script. If you edited it correctly, you should have three MOF files open in the PowerShell ISE. If the script generated an error, compare it against the solution file here:
    C:\PShell\Labs\Lab08\Lab08_03_01c_Solution.ps1
20. Briefly review the three MOF files generated.

Question B: Think back to the first configuration script in this lab where the node name and features were all hard-coded. How is this technique better?

Task 8.3.2: Configuration Data File

1. The **ConfigurationData** automatic parameter can accept either a hash table or a **PSD1** file path (which contains the same hash table). Now that the configuration is converted to use this data structure, we will experiment with putting the same data into a file instead. Open the lab file:
   C:\PShell\Labs\Lab08\Lab08_03_02_PSD1.ps1
2. In the PowerShell ISE select lines 23-38. Cut these to the clipboard.
3. Click the new script icon on the PowerShell ISE toolbar. Paste the lines into this blank script file.
4. Remove the variable assignment on the first line, leaving only the opening of the hash table: **@{**
5. Save the file with the following path and name:
   C:\PShell\Labs\Lab08\Lab08_03_02_JumpServer.psd1
6. Return to the configuration script opened in step 1.

7. Edit the configuration invocation at the bottom to use the file path instead of the hash table variable name:

```
JumpServer_7 -ConfigurationData .\Lab08_03_02_JumpServer.psd1
```

8. Run the script.

9. If everything worked correctly, you should see three familiar MOF files opened in the PowerShell ISE. If you received an error, compare your files to the solution files:
   C:\PShell\Labs\Lab08\Lab08_03_02_JumpServer_Solution.psd1
   C:\PShell\Labs\Lab08\Lab08_03_02_PSD1_Solution.ps1

   Question C: What is the advantage of storing the configuration data hash table in a separate file from the configuration script?

HINT: Now that the configuration specifics are separated from the configuration script, imagine the possibilities. You could have multiple unique PSD1 files to cover each environment (dev, test, prod) and then pass them into the exact same configuration. All of the environment specifics are isolated from the static configuration script.

Task 8.3.3: Dynamic Configuration Data

1. In the last two tasks you worked with the **ConfigurationData** hash table. This hash table contains the required key **AllNodes**. This key is an array of **NodeName** hash tables. Unrelated to PowerShell Desired State Configuration, working with hash tables and arrays is a core PowerShell feature. In this task you will explore the possibilities of creating this data structure entirely in memory without manually writing the hash table data structure in code. This technique is truly the pinnacle of PowerShell DSC scalable configurations. First, you will need a data source from which to query node information. Run the following script:
   C:\PShell\Labs\Lab08\Lab08_03_03a_DataSource.ps1

NOTE: This script may take a few minutes to run. If you see red error text or missing certificate data in the output, then reboot the **DC** server using this command:

```
Restart-Computer -ComputerName dc -Wait -For WinRM -Force
```

Then run the script again.

2. This script collected certificate public keys from each target node and stored the data in a CSV file. View the public keys collected by the script:

```
dir C:\PublicKeys
```

3. The public keys are used to encrypt configuration credentials, as demonstrated earlier in **Lab 4**. The CSV file can represent any data source where you desire to store node-specific data (SQL, XML, CSV, etc.). Open the following lab file: **C:\PShell\Labs\Lab08\Lab08_03_03b_Dynamic_Configuration_Data.ps1**

4. Select and run line 1 by pressing **F8**.

```
$nodes = Import-Csv C:\PublicKeys\index.csv
```

5. View the contents of the **$nodes** variable. Type this into the blue command pane of the ISE:

```
$nodes
```

6. This data serves two purposes:

   a. Feed a configuration script for encrypting credentials on the build server

   b. Feed an LCM script for configuring the decryption on the target node

   Now run line 3. This creates an empty **ConfigurationData** structure.

```
$ConfigData = @{ AllNodes = @() }
```

7. Using standard PowerShell scripting, you will loop through the CSV node data to build the **AllNodes** array of node hash tables. This scripting appends an array element for each node in the form of a hash table. Run lines 5-11.

```
ForEach ($node in $nodes) {
    $ConfigData.AllNodes += @{
        NodeName        = $node.Node
        CertificateFile = $node.Path
        Thumbprint      = $node.Thumbprint
    }
}
```

8. Now view the contents of the **AllNodes** array:

```
$ConfigData.AllNodes | %{"----";$_}
```

9. The **CertificateFile** hash table key is used for credential **encryption**. It points to the **.CER** public key file from the node. The **Thumbprint** key is not used on the build server that generates the MOF files. The thumbprint is only used on the target node to determine which certificate will be used for **decryption**. PowerShell DSC uses these hash table keys automatically. No additional scripting is required for encryption or decryption of credential passwords. You will supply each of these pieces of data in the next script. Open the lab file: **C:\PShell\Labs\Lab08\Lab08_03_03c_Dynamic_Encryption.ps1**

10. This script generates a **MOF** and **META.MOF** for each node in the CSV file. Review the script and run it. When prompted, enter the password **PowerShell5!**.

11. The script opens all of the **MOF** and **META.MOF** files generated. Compare the **META.MOF** files.

    Question D: Which property is unique in each META.MOF file? (Ignore **TargetNode** in the comments in the MOF file heading.) Why is this unique?

12. Compare the **MOF** files that were generated for each node.

    Question E: Which property is unique in each **MOF** file? (Ignore **TargetNode** in the comments in the MOF file heading.) Why is this unique?

    Question F: Is the actual password the same or different in these MOF files? Why?

13. This script illustrates the scalability of PowerShell DSC. Imagine pointing to a different data source or query for each environment (dev, test, prod). The configuration, the LCM meta configuration, and the ConfigurationData remain static files. The data sent through the script will determine the unique output per node.

    Question G: How could you apply this technique in your own configuration environments?