WorkshopPLUS Windows PowerShell Desired State Configuration

Module 7: Troubleshooting

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 7: Troubleshooting

Introduction

PowerShell Desired State Configuration (DSC) has many moving parts. When an error occurs, you need to understand the capabilities for troubleshooting. In this lab you will investigate configuration errors using LCM properties, event logs, and the **xDscDiagnostics** module. Then you will diagnose configuration issues with remote resource debugging.

Objectives

After completing this lab, you will be able to:

- Recognize the various LCM status and detail conditions
- Research errors in the events logs using **Get-WinEvent** and the **xDscDiagnostics** module
- Analyze resource errors with remote resource debugging

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

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

75 minutes

For more information

See the WMF 5.0 Release Notes, section *DSC Resource Script Debugging*.
C:\PShell\WMF 5.0 August 2015 Production Preview\Windows Management Framework 5.0 Production Preview Release Notes.docx

```
Get-Help about_Debuggers
```

Using Event Logs to Diagnose Errors in Desired State Configuration
http://blogs.msdn.com/b/powershell/archive/2014/01/03/using-event-logs-to-diagnose-errors-in-desired-state-configuration.aspx

DSC Diagnostics Module– Analyze DSC Logs instantly now!
http://blogs.msdn.com/b/powershell/archive/2014/02/11/dsc-diagnostics-module-analyze-dsc-logs-instantly-now.aspx

Troubleshooting DSC

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

Scenario

Consider a configuration being applied via a pull scenario. There is no blue verbose scrolling output. The process is essentially a *black box* with no interactive visibility. As the new PowerShell DSC expert on your team a peer has asked for your help identifying the root cause for a configuration failure. You want to train your peer in DSC troubleshooting by demonstrating various techniques they can use when issues occur. You will explore these techniques in the following exercises.

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 7.1: LCM Status

Objectives

In this exercise, you will:

- Observe changes in the **LCMState** and **LCMStateDetail**
- Report on configuration status using **Get-DscConfigurationStatus**

Scenario

You need to view the current configuration status of a node as the configuration is being applied. Then you want to report on configuration errors of this node for the last week.

Task 7.1.1: LCM State and Details

1. Begin by opening the lab script "C:\PShell\Labs\Lab07\Lab_07_01_01a_LCM_State.ps1".
2. The PowerShell ISE can run multiple PowerShell hosts in separate memory spaces. These hosts have their own set of tabs where scripts share memory and variables. To create a second host in the ISE choose the menu options **File** and **New PowerShell Tab (CTRL+T)**.
3. Click the new script icon or press **CTRL+N**. Notice that there are now two rows of tabs in the ISE. The upper ones have square corners, and the lower ones have rounded corners. Each PowerShell tab with square corners represents a different host process for PowerShell. This allows you to run two or more scripts at the same time in different hosts. Click on each of the PowerShell tabs. **PowerShell1** should have the lab script you opened in step 1.
4. Click on **PowerShell2**. In this tab open the script "C:\PShell\Labs\Lab07\Lab_07_01_01b_Check_LCM_State.ps1".
5. Click on the **PowerShell1** tab. Select and run the code in the first script region titled **SETUP**. This will reset the local LCM settings.
6. Select and run the command **Get-DscLocalConfigurationManager** in the next script region titled **IDLE**. In the output locate the properties **LCMState** and **LCMStateDetail**.

   Question A: Are there any configurations running at the moment?

   _____

7. Scroll down to the next script region titled **BUSY**. Examine the code. This configuration uses the **Script** resource three times, each simply waiting 10 seconds. The **Script** resource is capable of much more, but all we need is a long-running configuration for this task.

8. Select and run the code in the **BUSY** script region.
9. Immediately click on the **PowerShell2** tab. Select the following line and press **F8**.

```
Get-DscLocalConfigurationManager | Select-Object LCMState, LCMStateDetail
```

10. Continue to press **F8** until the **LCMState** changes back to **Idle**.
11. Click on the **PowerShell1** tab.
12. Select and run the code in the next script region titled **EXISTING**.
13. Immediately click on the **PowerShell2** tab. Select the following line and press **F8**.

```
Get-DscLocalConfigurationManager | Select-Object LCMState, LCMStateDetail
```

14. Continue to press **F8** until the **LCMState** changes back to **Idle**.

Question B: The **LCMState** was **Busy** for the last two examples, but what was different about the **LCMStateDetail**? Scroll up through the output to find your answer.

15. Click on the **PowerShell1** tab.
16. Select and run the code in the next script region titled **PENDING**.

Question C: What is the value of the LCM property **ConfigurationModeFrequencyMins**?

Question D: How is **ConfigurationModeFrequencyMins** related to a **Pending** configuration?

17. Select and run the code in the next script region titled **STOP**.
18. Immediately click on the **PowerShell2** tab. Select the following line and press **F8**.

```
Stop-DscConfiguration -Verbose
```

19. Read the message. Then immediately select the following line and press **F8**.

```
Stop-DscConfiguration –Verbose -Force
```

20. Read the message.

Question E: What is the difference when using the **Force** switch with **Stop-DscConfiguration**?

21. Click on the **PowerShell1** tab.

Question F: What is different about the configuration output now?

22. Imagine the possibilities of collecting the LCM state detail across multiple servers. Again select and run the code in the last script region titled **STOP**.

23. Immediately click on the **PowerShell2** tab. Select the following lines and press **F8**.

```
Get-DscLocalConfigurationManager -CimSession pull,ms1,ms2 |
    Format-Table PSComputerName, LCMState, LCMStateDetail
```

Question G: Based on what you have observed with the **LCMState** and **LCMStateDetail** properties how can you see using these techniques in your own DSC troubleshooting scenarios?

24. Close the PowerShell ISE.

Task 7.1.2: Configuration Status

1. Open the lab file "C:\PShell\Labs\Lab07\Lab_07_01_02_Status_Solution.ps1". Use the commands in the script file to complete the steps below.

2. Run the following lines. Study the output of each command.

```
Get-DscConfigurationStatus

Get-DscConfigurationStatus | Format-List *

Test-DscConfiguration -Detailed
```

3. Now scale these commands across multiple servers and view the output in a grid. Keep both grids open. Run the following lines:

```
Get-DscConfigurationStatus -CimSession pull,ms1,ms2 |
 Select-Object * | Out-GridView

Test-DscConfiguration -Detailed -CimSession pull,ms1,ms2 |
 Select-Object * | Out-GridView
```

4. Compare the output of the two commands in the grids.

   Question H: Which property names are the same?

   Question I: Which important property is in the output of **Test-DscConfiguration** but not in the output of **Get-DscConfigurationStatus**?

   Question J: Which one of these two cmdlets includes **JobId** in the output?

5. Check the properties of the LCM:

```
Get-DscLocalConfigurationManager
```

   Question K: What is the value of **StatusRetentionTimeInDays**?

6. View the file system location for the data in the **Get-DscConfigurationStatus** cmdlet. Then open one of the files and study the contents:

```
dir C:\Windows\System32\Configuration\ConfigurationStatus |
 Sort-Object LastWriteTime

Get-Content (dir C:\Windows\System32\Configuration\ConfigurationStatus |
 Sort-Object LastWriteTime -Descending)[0].FullName
```

7. View the status history MOF:

```
Get-Content C:\Windows\System32\Configuration\DSCStatusHistory.mof
```

   This file acts as an index of the files listed in the previous step.

8. Now view this same history through the cmdlet:

```
Get-DscConfigurationStatus -All | Out-GridView
```

9. Now filter the output for configurations that experienced issues. Study the output.

```
Get-DscConfigurationStatus -All | Where-Object {$_.Status -ne 'Success'} |
 Select-Object * | Out-GridView
```

10. Look at the **Error** and **Status** columns. Try to identify the stopped configuration(s) from the previous lab task.

11. Finally, find all of the unsuccessful configurations from three lab servers:

```
Get-DscConfigurationStatus -All -CimSession pull,ms1,ms2 |
 Where-Object {$_.Status -ne 'Success'} | Select-Object * | Out-GridView
```

Question L: How could you use this report in your environment?

Exercise 7.2: Event Logs

Objectives

In this exercise, you will:

- Query DSC events using Get-WinEvent
- Install and evaluate the xDscDiagnostics module

Scenario

A key benefit of using PowerShell Desired State Configuration (DSC) is the automatic logging. You must use this logging to identify and explain a configuration error for a remote server.

Task 7.2.1: Get-WinEvent

1. Open the lab file "C:\PShell\Labs\Lab07\Lab_07_02_01_GetWinEvent_Solution.ps1". Use the commands in the script file to complete the steps below.
2. In order to query event logs remotely you must either open the Windows Firewall rules for **Remote Event Log Management** or execute all queries inside a PowerShell remoting session. Both are valid solutions. Opening the firewall is easier, while using remoting is more secure. In our lab we will open the firewall rules.

```
Invoke-Command -ComputerName pull,ms1,ms2 -ScriptBlock {
    Set-NetFirewallRule -DisplayGroup 'Remote Event Log Management' -Enabled True
}
```

3. List the available DSC event logs:

```
Get-WinEvent -ListLog *dsc*,*desired* -Force | Out-GridView
```

Question A: Look at the record count column in the output grid. How would you explain the only two logs with data in them?

4. View the most recent 50 events from the **Operational** log:

```
Get-WinEvent -LogName Microsoft-Windows-DSC/Operational -MaxEvents 50 |
 Out-GridView
```

Question B: Can you identify anything in the **Message** property of these log entries that resembles the blue verbose output from a configuration event?

Task 7.2.2: xDscDiagnostics Module

1. Open the lab file "C:\PShell\Labs\Lab07\Lab_07_02_02_xDscDiagnostics_Solution.ps1". Use the commands in the script file to complete the steps below.
2. The **Operational** event log information above is helpful, but it does not recreate the full verbose logging that you hope to find for troubleshooting. To see that level of detail you must enable the **Analytic** and **Debug** logs. One way is using **WEVTUTIL.EXE**, but we prefer to use the **xDscDiagnostics** PowerShell module.

> **NOTE:** For more information on **xDscDiagnostics** see the blog post linked at the beginning of this lab document under **For More Information**.

3. Install the **xDscDiagnostics** module from the PowerShell Gallery:

```
Install-Module xDscDiagnostics -Force
```

> **NOTE:** If you are prompted to install **NuGet**, answer **Yes**.

4. Explore the module cmdlets and their syntax:

```
Get-Command -Module xDscDiagnostics

Get-Command Get-xDscOperation –Syntax

Get-Command Trace-xDscOperation –Syntax

Get-Command Update-xDscEventLogStatus -Syntax
```

5. Enable the **Analytic** and **Debug** logs on **MS1**.

```
'Analytic','Debug' | ForEach-Object {
    Update-xDscEventLogStatus -Channel $_ -Status Enabled -ComputerName ms1
}
```

> **NOTE:** If you receive the error message below, it means that these logs have already been enabled. Safely ignore the error.
>
> *The channel fails to activate.*
> *Failed to save configuration or activate log Microsoft-Windows-Dsc/Debug. The requested operation cannot be performed over an enabled direct channel. The channel must first be disabled before performing the requested operation.*

6. These two logs are now enabled, but each one is empty. First we will create a configuration error to troubleshoot. Then we will use the **xDscDiagnostics** cmdlets to identify the problem. Run the lab configuration error script:

```
& C:\PShell\Labs\Lab07\Lab_07_02_02_Configuration_Error.ps1
```

7. Wait for the script to display the message ***** PROCEED WITH THE NEXT LAB STEP *****, and then continue.
8. The **xDscOperation** cmdlets collect and correlate events across all three DSC configuration logs: Operational, Analytic, Debug. This gives a complete picture of each configuration event. List the last 10 configuration events for **MS1**:

```
Get-xDscOperation -ComputerName ms1 -Newest 10
```

9. Locate the line with **Failure** in the **Result** column. Copy and paste the **JobId** value of the failure into the next command:

```
Trace-xDscOperation -ComputerName ms1 -JobId 'paste-here' | Out-Gridview
```

> **NOTE:** Ignore any error messages that may appear in the output pane.

10. Scroll through the grid to view the **Message** column. Note the **EventType** and **TimeCreated** columns.
11. Filter the grid by typing into the *Filter* box at the top. Filter for each of these values and study the types of messages for each one:
    a. Operational
    b. Analytic
    c. Debug
    d. Verbose
    e. Error
12. Remove the filter value and scan the entire **Message** column stream.

Question C: What was this configuration trying to do?

Question D: Approximately how long did it take from start to finish?

Question E: Why do you think the configuration failed?

13. Run these two commands. Compare the output, specifically the **JobId** and time stamp columns.

```
Get-DscConfigurationStatus -CimSession ms1 -All |
 Select-Object Status, StartDate, Type, Mode, JobId -First 10 |
 Format-Table -AutoSize

Get-xDscOperation -ComputerName ms1 -Newest 10
```

14. In the **Get-DscConfigurationStatus** output identify the row with the **Type** of **Initial** and the **Status** of **Failure**. Locate the same **JobId** from this row in the output of the **Get-xDscOperation** output.

> **NOTE:** If you cannot find the related **JobId**, then repeat the commands and look for 20 instead of 10 items in each command.

15. Finally, get detailed status of this failed event:

```
Get-DscConfigurationStatus -CimSession ms1 -All |
 Where-Object {$_.Type -eq 'Initial' -and $_.Status -eq 'Failure'} |
 Select-Object * -First 1
```

16. Review the output.

Question F: What is the value of the property **ResourcesNotInDesiredState**?

_____

_____

17. Now you see that the **xDscDiagnostics** module cmdlets can correlate data from the **Get-DscConfigurationStatus** cmdlet. Together these cmdlets help you reconstruct the evidence surrounding a configuration error.

18. Reset the LCM and configuration state on **MS1** by deleting the MOF files:

```
Invoke-Command -ComputerName ms1 -ScriptBlock {
    Remove-Item C:\Windows\System32\Configuration\*.mof -Force
}
```

Exercise 7.3: Remote Resource Debugging

Objectives

In this exercise, you will:

- Configure the LCM for debug mode
- Remotely debug a configuration resource

Scenario

A configuration continues to fail on a remote node. You have reviewed the log, but you need to go deeper into a specific resource that is failing. You must do a live debug session over PowerShell remoting to investigate the issue.

Task 7.3.1: DscDebug Cmdlets

1. In PowerShell v5 there are new cmdlets to enable and disable DSC debugging status on the LCM. List these cmdlets:

```
Get-Command -Noun DscDebug
```

2. Get the syntax for **Enable-DscDebug**.

```
Get-Command Enable-DscDebug -Syntax
```

> **NOTE:** At the time of this writing the cmdlet is still under development. The **BreakAll** parameter is required even though it may not be indicated in the syntax output.

3. Now enable DSC debugging on **MS1**. We will create a CIM session to reuse for multiple commands against **MS1**.

```
$cim = New-CimSession -ComputerName ms1

Enable-DscDebug -BreakAll -CimSession $cim
```

4. Check the LCM configuration on **MS1**.

```
Get-DscLocalConfigurationManager -CimSession $cim
```

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

5. Open the WMF 5.0 release notes in the **C:\PShell** folder (C:\PShell\WMF 5.0 August 2015 Production Preview\Windows Management Framework 5.0 Production Preview Release Notes.docx). Find and read the section *DSC Resource Script Debugging*.

Question B: What does **ResourceScriptBreakAll** do?

6. Close the PowerShell ISE.

Task 7.3.2: Remote Debugging

1. Begin in a new PowerShell ISE by opening the lab script "C:\PShell\Labs\Lab07\ Lab_07_03_02_Remote_Debugging.ps1".

> **TIP:** In Windows Explorer right click the file and choose **Edit**.

2. Set the working path to the **Lab07** folder.

```
Set-Location C:\PShell\Labs\Lab07
```

3. Create a second host in the ISE. Choose the menu options **File** and **New PowerShell Tab (CTRL+T)**. Click the new script icon.
4. Click on the first square tab **PowerShell1**.
5. Only DSC resources written in PowerShell script can be remotely debugged. List the built-in PowerShell DSC resources and sort by the column **ImplementedAs**.

```
Get-DscResource -Module PsDesiredStateConfiguration  | Sort-Object ImplementedAs
```

Question C: What are the two resources implemented as **Binary**? These cannot be remotely debugged.

6. Run the provided script lines to create and apply the configuration **RemoteDebugSample**:

```
Configuration RemoteDebugSample
{
    Import-DscResource -ModuleName PsDesiredStateConfiguration

    Node ms1
    {
        Environment EnvVar
        {
            Name   = 'PS_DSC_Workshop'
            Value  = 'A113'
            Ensure = 'Present'
        }
```

```
        Service Bits
        {
            Name        = 'Bits'
            StartupType = 'Automatic'
            State       = 'Running'
        }
    }
}

RemoteDebugSample

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

7. Scroll through the output and read the two orange warning messages.
8. Find the last three lines of the warning text. Copy and paste them into the **PowerShell2** tab. In our lab we will not need the **Credential** parameter. Remove the last part of the first line you pasted (-Credential <credentials>).

```
Enter-PSSession -ComputerName MS1
Enter-PSHostProcess -Id 1188 -AppDomainName DscPsPluginWkr_AppDomain
Debug-Runspace -Id 5
```

> **NOTE:** The number values in the command you paste will likely be different than the sample provided above.

9. Use the **F8** key to run each one of these lines individually. Each line may take a few seconds to run.
10. Notice all of the prefixes on the remote command prompt created by these commands:

```
[MS1]: [DBG]: [Process:1188]: [Runspace5]: PS C:\Windows\system32>>
```

You are now in a remote debug session on MS1 for the process and runspace listed.

Question D: Look above in the PowerShell ISE script pane. What is the title on the current script tab?

_____

_____

Question E: Look in the code and find the yellow highlighting. It may be small on a single curly brace. What is the name of the function where execution is paused?

_____

_____

11. You are actually looking at the **Environment** DSC resource code on the remote server **MS1**. This code was brought across the remoting session into your local

PowerShell ISE for debugging. Debugging techniques are covered in the Microsoft Premier PowerShell Part 2 workshop. You can read more about them in PowerShell help:

```
Get-Help about_Debuggers
```

> **NOTE:** If you choose to reference this help now you will need to do so from another PowerShell console. Both PowerShell tabs in the ISE are in use.

12. Click on the PowerShell ISE **Debug** menu. Click on **Run/Continue F5**.

    Question F: Look again in the code and find the yellow highlighting. It may be small on a single curly brace. What is the name of the function where execution is paused?

    _____

13. Now click over to the **PowerShell1** tab in the PowerShell ISE. Notice that the resource code is paused again. Read the blue verbose output between the two warning messages. You completed the **Test** function by running it, and now you are paused on the **Set** function of the **Environment** resource.
14. The debug session is still running on **PowerShell2** tab. There is no need to copy and paste the warning code again. Click over to the **PowerShell2** tab.
15. Click on the **Debug** menu. Then choose **Step Into F11**.
16. Choose the same menu command again.
17. Now press **F11** to continue one line at a time through the remote function. Notice that some commands may take longer than others as you step through the script. You can observe this by watching for the red **Stop** icon on the task bar at the top of the ISE.
18. As you step through using **F11**, move the mouse to hover over a variable. Notice that its value pops up in a shaded box.
19. Continue to press **F11** until you get to the next function, **Test-TargetResource** in the **Service** resource.
20. Now click in the command pane at the bottom. Type a question mark:

```
?
```

    Scroll up through the output. This is the help for the command line debugger experience.

21. Type ‘c’ to continue and press **ENTER**.

```
c
```

22. Type the letter ‘k’ and press ENTER to see the call stack:

```
k
```

23. Study the **Command**, **Arguments**, and **Location** properties that are returned.

    Question G: Which function in what resource are you currently debugging?

24. Type ‘c’ to continue and press **ENTER**.

```
c
```

> **NOTE:** The debug session should now be completed. If you do not get back to a prompt, press **CTRL+C** to end the debug session.

25. Now click on the **PowerShell1** tab. Notice that the configuration has completed successfully.
26. Now click on the **PowerShell2** tab. Notice that the remoting session is still opened. You may see an error that the runspace pool is broken. This is expected. Type ‘exit’ to leave the remote session.

```
exit
```

    Question H: What happened to the tabs in the PowerShell ISE after you typed ‘exit’?

27. Return to the **PowerShell1** tab. Apply the configuration again.

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

28. Wait for the warning message. Copy and paste the bottom three debug lines again to **PowerShell2** tab. Remove the credential parameter. Run the lines one at a time to launch another debug session.

```
Enter-PSSession -ComputerName MS1
Enter-PSHostProcess -Id 1236 -AppDomainName DscPsPluginWkr_AppDomain
Debug-Runspace -Id 5
```

> **NOTE:** The number values in the command you paste will likely be different than the sample provided above.

29. When you receive the remote **[DBG]** prompt, press **F5** to continue.
30. Click back to the **PowerShell1** tab. Notice that the configuration is still paused.

31. Sometimes you may want to end a debug session early instead of stepping through every resource to get to the end. Run the following commands from the **PowerShell2** tab:

```
detach

exit
```

Question I: What happened on the **PowerShell1** tab?

32. Press **CTRL+T** to open a third new PowerShell tab. Run the following command from the **PowerShell3** tab:

```
Get-DscConfiguration -CimSession ms1
```

33. Repeat the process of copying the warning debug message into a new PowerShell tab as you did in the above steps.

Question J: At what function is the debugger paused this time?

NOTE: Normally at this point you would set break points and step through the code for troubleshooting.

34. Run the following command from the debugger session:

```
detach

exit
```

35. Finally, disable debug mode on **MS1**.

```
Disable-DscDebug -CimSession ms1
```

36. Close the PowerShell ISE.