A layman's guide to PowerShell 2.0 remoting

Ravikanth Chaganti

Learn the basics of PowerShell 2.0 remoting, methods of remoting and how to use remoting to manage systems in a datacenter.

A layman's guide to PowerShell 2.0 remoting

Ravikanth Chaganti

Jan Egil Ring

Acknowledgments

Thanks to everyone who read my blog posts on PS remoting and provided feedback. Your encouragement and support helped me write quite a bit about remoting and now this e-book. Thanks to Jan Egil Ring (http://blog.powershell.no/) for contributing to appendix B in this updated version of the ebook.

Contents

PART1 ..... **5**

CHAPTER 1: INTRODUCTION TO REMOTING **5**

TRADITIONAL REMOTING IN POWERSHELL 5
OVERVIEW OF POWERSHELL 2.0 REMOTING 6
PowerShell 2.0 remoting requirements *6*
OVERVIEW OF REMOTING CMDLETS 7
Enable-PSRemoting *7*
Disable-PSRemoting *7*
Invoke-Command *7*
New-PSSession *8*
Enter-PSSession *8*
Exit-PSSession *8*
Get-PSSession *8*
Remove-PSSession *8*
Import-PSSession *8*
Export-PSSession *8*
Register-PSSessionConfiguration *9*
Unregister-PSSessionConfiguration *9*
Disable-PSSessionConfiguration *9*
Enable-PSSessionConfiguration *9*
Get-PSSessionConfiguration *9*
Set-PSSessionConfiguration *9*
Test-WSMan *9*
Enable-WSManCredSSP *9*
Disable-WSManCredSSP *10*

CHAPTER 2: ENABLE/DISABLE POWERSHELL REMOTING **11**

TEST POWERSHELL REMOTING 12
REMOTING IN WORKGROUP ENVIRONMENTS 13
On Windows XP *13*
Modify WSMan trusted hosts setting *13*
REMOTING IN MIXED DOMAIN ENVIRONMENT 14
DISABLE REMOTING 14
ENABLE REMOTING FOR ONLY A SPECIFIC NETWORK ADAPTER 14
REMOTING IN AN ENTERPRISE 15
SUMMARY 15

CHAPTER 3: EXECUTE REMOTE COMMANDS **16**

RUN SCRIPT BLOCKS ON LOCAL OR REMOTE COMPUTER 16
RUN SCRIPT FILES ON REMOTE COMPUTERS 16
PASSING VARIABLES TO REMOTE SESSION 17
USING PERSISTENT SESSIONS WITH INVOKE-COMMAND 17
RUNNING REMOTE COMMAND AS A BACKGROUND JOB 17

Specifying credentials required for remoting....18
Summary....19

CHAPTER 4: INTERACTIVE REMOTING SESSIONS....20

Starting an interactive remoting session....20
Exiting an interactive session....21
Using persistent sessions with interactive remoting....21
Starting interactive remoting with an existing session....21
Method 1: Using session Id....22
Method 2: Using session instance Id....22
Method 3: Using session name....22
Method 3: Using –session parameter....22
Summary....22

CHAPTER 5: IMPLICIT REMOTING IN POWERSHELL....23

Why implicit remoting?....23
Creating an implicit remoting session....23
Avoiding name conflicts while importing a remote session....24
Importing modules and snap-ins to local session....24
Limitations of implicit remoting....25
Summary....25

CHAPTER 6: SAVING REMOTE SESSIONS TO DISK....26

Export remote session to a module on disk....26
Importing a module saved on disk....26
Limitations of Export-PSSession....27
Summary....27

PART 2....28

CHAPTER 7: UNDERSTANDING SESSION CONFIGURATIONS....28

What is a PS Session configuration?....28
Cmdlets available to manage session configurations....28
Creating a new session configuration....29
List available session configurations....29
From the local computer....29
From a remote computer....29
Custom permissions and PS Session configurations....30
Invoking a custom session configuration....30
Disable a session configuration....31
Delete a session configuration....31
Summary....31

CHAPTER 8: USING CUSTOM SESSION CONFIGURATIONS....32

CHAPTER 9: INTERPRETING, FORMATTING AND DISPLAYING REMOTE OUTPUT....35

How remote output comes over to local computer?....36
Formatting remote output....37

CHAPTER 10: USING CREDSSP FOR MULTI-HOP AUTHENTICATION 39

DELEGATING CREDENTIALS 40
SUMMARY 42

APPENDIX A: FREQUENTLY ASKED QUESTIONS 43

APPENDIX B: ENABLE POWERSHELL REMOTING USING GROUP POLICY 44

Part1

Chapter 1: Introduction to remoting

Traditional remoting in PowerShell

A few cmdlets in PowerShell support accessing information on a remote system. These cmdlets have a –ComputerName parameter. For example the following cmdlets support the computername parameter and hence can be used to access information from a remote computer.

- Get-WmiObject
- Invoke-WmiMethod
- Limit-EventLog
- Set-Service
- Set-WmiInstance
- Show-EventLog
- Stop-Computer
- Clear-EventLog
- Get-Counter
- New-EventLog
- Register-WmiEvent
- Remove-EventLog
- Remove-WmiObject
- Restart-Computer
- Get-EventLog
- Get-HotFix
- Get-Process
- Get-Service
- Get-WinEvent

The remoting capability of these cmdlets is independent of PowerShell. It is up to the cmdlet author to implement the remote access using methods such as remote procedure call (RPC), etc. This method of remoting can be called traditional remoting or classic remoting.

One obvious disadvantage is that not all PowerShell cmdlets implement this type of remoting. So, for example, if you want to execute Get-PSDrive or Get-ChildItem remotely on a different computer, it is not possible. This is where the new PowerShell 2.0 remoting feature plays an important role. So, throughout this guide, whenever we refer to remoting, we refer to the new remoting technology but not traditional or classic remoting methods.

Overview of PowerShell 2.0 remoting

One of the most exciting and important features of PowerShell 2.0 is the remoting capability. PowerShell remoting enables management of computers from a remote location. Remoting is built on top of Windows remote management (WinRM)[1]. WinRM is Microsoft's implementation of WS-Management[2] protocol.

This feature enables what is known as Universal Code Execution Model[3] in Windows PowerShell 2.0. UCEM means that whatever runs locally should run anywhere. PowerShell remoting also lets you import remote commands in to a local session — a feature known as **implicit remoting** and also enables you to save or export these imported commands to local disk as a module for later use. There are bunch of other features such as interactive sessions, etc. We will look in to all these features -- one thing at a time.

PowerShell remoting allows for multiple ways of connecting. These ways include interactive (1:1), fan-out (1: many), and fan-in (many: 1 by using the IIS hosting model, for example, Quest Software's MobileShell[4]). This guide will walk though each of these ways and explain how to configure your system for these scenarios.

PowerShell 2.0 remoting requirements

To enable PowerShell remoting, all computers participating in remote management should have the following software

1. Windows PowerShell 2.0
2. NET framework 2.0 SP1 or later
3. Windows Remote Management (WinRM) 2.0

All of the above are installed by default on Windows 7 and Windows Server 2008 R2. However, earlier versions of Windows will require you to download the updates from Microsoft website and install them yourself.

PowerShell 2.0 and WinRM 2.0 are included as a part of Windows Management Framework download and are available for Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008. WinRM 2.0 and PowerShell 2.0 can be installed on the following supported operating systems

1. Windows Server 2008 with Service Pack 1
2. Windows Server 2008 with Service Pack 2
3. Windows Server 2003 with Service Pack 2
4. Windows Vista with Service Pack 2
5. Windows Vista with Service Pack 1
6. Windows XP with Service Pack 3

[1] WinRM: http://msdn.microsoft.com/en-us/library/aa384426(VS.85).aspx
[2] WS-Management: http://msdn.microsoft.com/en-us/library/aa384470(VS.85).aspx
[3] UCEM as explained by Jeffery Snover: Universal Code Execution Model
[4] MobileShell

7. Windows Embedded POSReady 2009
8. Windows Embedded for Point of Service 1.1

PowerShell 2.0 remoting is supported only on the operating systems listed above.

To be able run scripts and commands on remote computers, the user performing remote script execution must be

- a member of the administrators group on the remote machine **OR**
- should be able to provide administrator credentials at the time of remote execution **OR**
- should have access the PS session configuration on the remote system

For a complete discussion on PS Session configurations refer to chapter << >>.

Also, on client OS versions of Windows such as Windows Vista and Windows 7, network location must be set either to Home or Work. WS-Management may not function properly if the network location for any of the network adapters is set to **public**.

Overview of remoting cmdlets

This section provides a quick overview of some of the important cmdlets that are used in PowerShell remoting. This list will also include cmdlets that are not directly used within remoting but help configure various aspects of remoting. The knowledge of some of these cmdlets such as WSMan cmdlets is not mandatory for basic usage of PowerShell remoting. Subsequent chapters will discuss these cmdlets in detail.

Enable-PSRemoting

The Enable-PSRemoting cmdlet configures the computer to receive Windows PowerShell remote commands that are sent by using the WS-Management technology. This cmdlet will be the first one to run if you want to use PowerShell 2.0 remoting features and needs to be run just once. This cmdlet internally calls Set-WSManQuickConfig to configure WinRM service, enable firewall exceptions for WS Management and finally enables all registered PowerShell configurations.

Note: You need to enable PowerShell remoting only if you want the computer receive commands from a remote machine. To only send commands to a remote machine, you don't need to enable PowerShell remoting.

Disable-PSRemoting

The Disable-PSRemoting cmdlet disables all PowerShell session configurations on the local computer to prevent the computer from receiving any remote commands. You will have to manually stop the WinRM service if you don't want the service to be running after you disable PowerShell remoting.

Invoke-Command

The Invoke-Command cmdlet runs commands on a local or remote computer and returns all output from the commands, including errors. With a single Invoke-Command command, you can run commands

on multiple computers. This cmdlet — in its default form — opens a session for running a command against a remote computer and closes it once the execution is complete. This method may be slow and can be worked around by specifying pre-defined session information.

New-PSSession

Invoke-Command cmdlet supports specifying an existing session to enhance the speed of overall command execution. By specifying an existing session, we eliminate the need for creating/destroying the sessions on the fly. New-PSSession cmdlet can be used to create a persistent connection to a remote computer. By creating a persistent session, we will be able to share data, such as a function or the value of a variable between different commands executing within the PSSession.

Enter-PSSession

Enter-PSSession cmdlet starts an interactive session with a single remote computer. During the session, the commands that you type run on the remote computer, just as though you were typing directly on the remote computer. You can have only one interactive session at a time. You can specify the PSSession you created using New-PSSession as a parameter to this cmdlet.

Exit-PSSession

Exit-PSSession exits an interactive PS Session created using Enter-PSSession cmdlet.

Get-PSSession

The Get-PSSession cmdlet gets the Windows PowerShell sessions (PSSessions) that were created in the current session. This cmdlet gets all the PSSessions returns all the PSSessions in to a variable when no parameters are specified. You can then use the session information with other cmdlets such as Invoke-Command, Enter-PSSession, Remove-PSSession, etc.

Remove-PSSession

The Remove-PSSession cmdlet closes PS session(s). It stops any commands that are running in the PSSessions, ends the PSSession, and releases the resources that the PSSession was using. If the PSSession is connected to a remote computer, Remove-PSSession also closes the connection between the local and remote computers.

Import-PSSession

Import-PSSession cmdlet uses the implicit remoting feature of PowerShell 2.0. Implicit remoting enables you to import commands from a local/remote computer in to an existing PS session and run those commands as if they were local to the session.

Export-PSSession

The Export-PSSession cmdlet gets cmdlets, functions, aliases, and other command types from another PSSession on a local or remote computer and saves them to local disk as a Windows PowerShell module. We can now use the Import-Module cmdlet to add the commands from the saved module to a PS Session.

Register-PSSessionConfiguration

Any PS session created using Invoke-Command or New-PSSession or any other PowerShell remoting cmdlet for that matter uses the default PS Session configuration as specified in the $PSSessionConfigurationName variable. PS Session configuration determines which commands are available in the session, and it can include settings that protect the computer, such as those that limit the amount of data that the session can receive remotely in a single object or command. So, you can use the Register-PSSessionConfiguration cmdlet creates and registers a new session configuration on the local computer.

Unregister-PSSessionConfiguration

The Unregister-PSSessionConfiguration cmdlet deletes registered session configurations from the computer. It is possible to delete the default PSSession configurations (Microsoft.PowerShell or Microsoft.PowerShell32) using this cmdlet. In such a case, you can use Enable-PSRemoting cmdlet to re-create and register the default PS Session configurations.

Disable-PSSessionConfiguration

Disable-PSSessionConfiguration disables a registered PS Session configuration. Remember, this only disables the configuration but not un-register or delete the information from local computer. These disabled session configurations cannot be used to establish a remoting session.

Enable-PSSessionConfiguration

The Enable-PSSessionConfiguration cmdlet re-enables registered session configurations that have been disabled by using the Disable-PSSessionConfiguration cmdlet.

Get-PSSessionConfiguration

The Get-PSSessionConfiguration cmdlet gets the session configurations that have been registered on the local computer.

Set-PSSessionConfiguration

The Set-PSSessionConfiguration cmdlet changes the properties of the registered session configurations on the local computer.

Test-WSMan

PowerShell remoting requires WinRM service to be running on the remote machines. You can use Test-WSMan cmdlet to quickly check if you can establish a remoting session with other computers. If WinRM is not enabled on remote machine, you can safely assume that PowerShell remoting is not enabled. However, you cannot assume that PowerShell remoting is enabled just by verifying that WinRM service is running. Remember, this cmdlet checks only for WinRM service and remoting requires many other components to function.

Enable-WSManCredSSP

PowerShell remoting supports CredSSP authentication and the same can be enabled by using Enable-WSManCredSSP cmdlet. The Enable-WSManCredSSP cmdlet enables CredSSP authentication on a client or on a server computer. When CredSSP authentication is used, the user's credentials are passed to a

remote computer to be authenticated. This type of authentication is designed for commands that create a remote session from within another remote session. For example, you use this type of authentication if you want to run a background job on a remote computer.

Disable-WSManCredSSP

The Disable-WSManCredSPP cmdlet disables CredSSP authentication on a client or on a server computer.

There are other WSMan cmdlets introduced in PowerShell 2.0 such as Connect-WSMan, Disconnect-WSMan, Get-WSManInstance, New-WSManInstance, New-WSManSessionOption, Remove-WSManInstance and Set-WSManInstance. These cmdlets are not really meant for PowerShell remoting but we will discuss them as required.

Chapter 2: Enable/Disable PowerShell remoting

Remoting in PowerShell 2.0 can be enabled by just running the following cmdlet in an elevated PowerShell prompt

Enable-PSRemoting

Yes. That is it. You will be asked to respond to a couple of questions — based on OS architecture – as you see in the screenshot here.

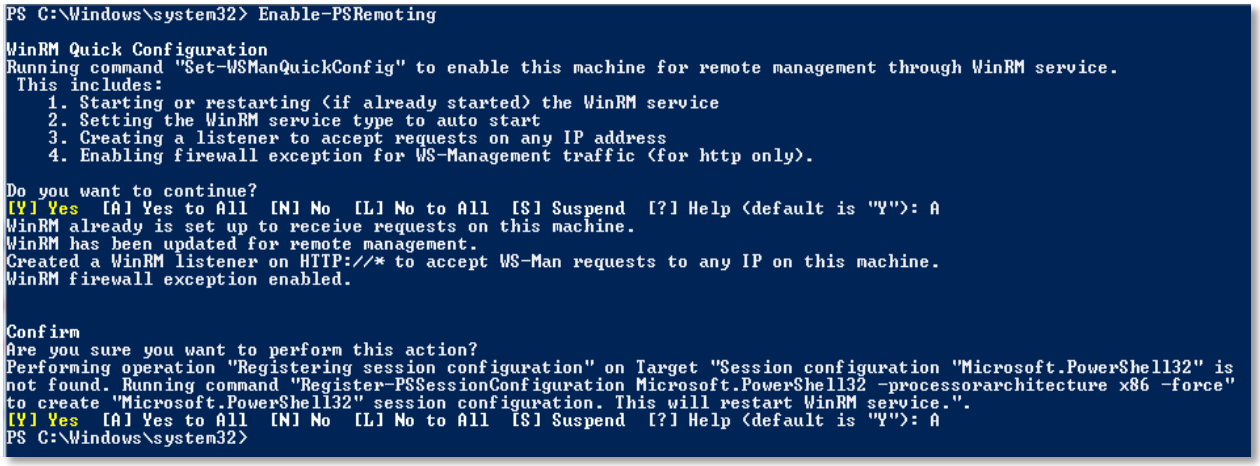

Figure 1 Enable Remoting

The following things happen when you run this cmdlet.

1. WinRM service gets enabled and startup type is set to auto start.
2. WinRM listener gets created to accept remoting requests on any IP addresses assigned to local computer
3. Windows firewall exceptions for WinRM service will be created. This is essentially the reason why network location cannot be set to public if you want to enable PS remoting. Windows firewall exceptions cannot be enabled if the network location is set to public.
4. Enables all registered PS session configurations. We will talk about this in detail later.

By default, WinRM only enables http transport for accepting remoting requests. You can manually enable https transport using either winrm command or New-WSManIntance cmdlet. For now, let us not overwhelm with so much information. We will look at this in part 2 of this guide.

Note

By default, PowerShell remoting uses port number 5985 (for http) and 5986 (for https). This can be changed by modifying wsman:\Localhost\listener\listener*\port to a different value using Set Item cmdlet. However, beware that this will change port number for every WinRM listener on the system.

You should always use the more comprehensive Enable-PSRemoting cmdlet. You can use -force parameter along with this cmdlet to silently enable remoting.

Trivia

PowerShell remoting cannot be enabled remotely ☺

Test PowerShell remoting

You can use the Enter-PSSession cmdlet to test if remoting is enabled on the local machine or not.

```
Enter-PSSession -ComputerName localhost
```

If remoting is enabled and functional, you will see the prompt changing to something like this

Figure 2 Enter-PSSession on localhost

Note

A PowerShell session (PS Session) is an environment to run the remote commands and scripts. PowerShell 2.0 provides various cmdlets to manage these sessions. To see a list of all PSSession cmdlets, use ***Get-Command –noun PSSession***.

There is also a ***New-PSSessionOption*** cmdlet to change default behavior of a PS session. New-PSSession and Enter-PSSession cmdlets have a parameter, -sessionOption, to specify custom session options. You can use this to specify options such as

IdleTimeOut
Determines how long the PSSession stays open if the remote computer does not receive any communication from the local computer, including the heartbeat signal. When the interval expires, the PSSession closes.

OpenTimeOut
Determines how long the client computer waits for the session connection to be established. When the interval expires, the command to establish the connection fails.

OperationTimeOut
Determines the maximum time that any operation in the PSSession can run. When the interval expires, the operation fails.

SkipCACheck
Specifies that when connecting over HTTPS, the client does not validate that the server certificate is signed by a trusted certification authority (CA).

SkipCNCheck
Specifies that the certificate common name (CN) of the server does not need to match the hostname of

the server. This option is used only in remote operations that use the HTTPS protocol.

SkipRevocationCheck
Does not validate the revocation status of the server certificate.

Remoting in workgroup environments

You will not be able to connect to a computer in workgroup just by running Enable-PSRemoting cmdlet. This is essentially because the security levels on a workgroup joined computer are more stringent than on a domain joined computer. So, on workgroup joined computers, you need to enable a few more things before you can create remoting sessions.

On Windows XP

You need to make sure the local security policy to enable classic mode authentication for network logons. This can be done by opening "Local Security Policy" from Control Panel -> Administrative Tools. Over there, navigate to **Local Policies -> Security Options** and double click on **"Network Access: Sharing and Security Model for local accounts"** and set it to classic.

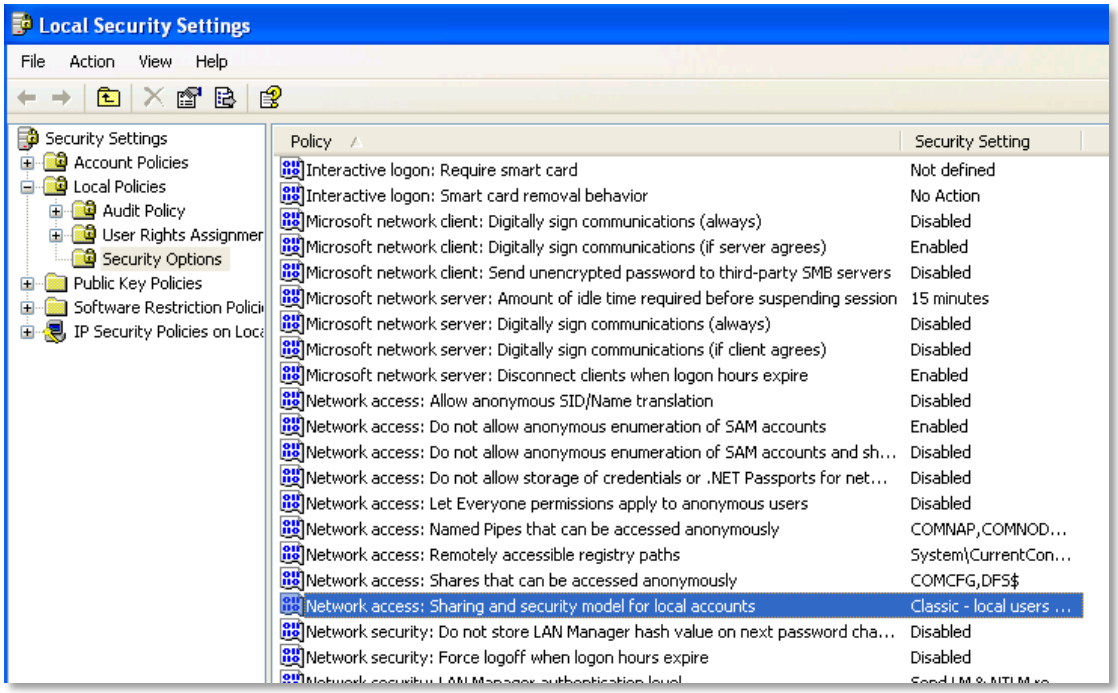

Figure 3 Windows XP security policy change

Modify WSMan trusted hosts setting

On all the workgroup joined computers – including Windows XP, Windows Vista and later – you need to add the IP addresses of all remoting clients to the list of trusted hosts. To do this,

```
Set-item wsman:localhost\client\trustedhosts -value *
```

Using * as the value will add all computers as trusted hosts. If you want to add only a specific set of computers,

```
Set-item wsman:localhost\client\trustedhosts -value Computer1,Computer2
```

If you want to add all computers in a specific domain,

```
Set-item wsman:localhost\client\trustedhosts -value *.testdomain.com
```

If you want to add an IP address of a remote computer to the trusted hosts list,

```
Set-item wsman:localhost\client\trustedhosts -value 10.10.10.1
```

Once the above changes are made, you can use Enable-PSRemoting cmdlet to enable remoting on these workgroup joined computers.

Remoting in mixed domain environment

By default, a user from a different domain cannot connect to a computer in another domain even when the user is a part of local administrators group. This is because remote connections from other domains run with only standard user privileges.

To work around this, you can change the LocalAccountTokenFilterPolicy registry entry (set it to 1) to allow members of other domain to remote in to the local computer.

```
new-itemproperty -name LocalAccountTokenFilterPolicy -path `
HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System -propertyType DWord -value 1
```

Disable remoting

You can use Disable-PSRemoting to disable remoting on the local computer. Disable-PSRemoting will only disable the session configurations. This will *not* remove all the changes done by Enable-PSRemoting. This includes leaving the WinRM service in enabled state and leaving all the listeners created to enable PS remoting. You will have to manually undo these changes if they are not required by any other component or service on the local computer.

If no other service or components on the local computer need WinRM service, you can disable WinRM service by running

```
Set-Service winrm -StartupType Manual
Stop-Service winrm
```

To remove all WinRM listeners listening on default PS remoting port (5985)

```
Get-ChildItem WSMan:\localhost\Listener –Recurse | Foreach-Object { $_.PSPath } | Where-Object { (Get-Item
"$_\Port").Value -eq 5985 } | Remove-Item
```

Enable remoting for only a specific network adapter

When you enable remoting on a computer using Enable-PSRemoting cmdlet, an http listener will be created to listen for remoting requests on all IP addresses on the local computer. This may not be a great security practice in an enterprise.

For example, you have an Internet facing server with two network connections. One – obviously – is the Internet connection and a second one connecting to your internal network. You don't need remoting be enabled on the network adapter connected Internet. But, since you used Enable-PSRemoting cmdlet, remoting will be enabled and there is a WinRM listener on the Internet facing network too. **So, how do we disable remoting on the Internet facing adapter?**

Enable-PSRemoting is a comprehensive cmdlet that does lot of things for you in one shot. This is also the recommended way to enable remoting. So, if we need to disable remoting on a particular IP address, all you need to do is remove the WinRM listener create by Enable-PSRemoting cmdlet and re-create your own listener for a specified IP address.

We use Remove-WSManInstance and New-WSManInstance cmdlets to do this. You can also use winrm command-line to achieve this. It is just a preference.

To remove the http listener created by Enable-Remoting,

```
Remove-WSManInstance winrm/config/Listener -SelectorSet @{Address="*";Transport="http"}
```

This will remove the listener.

Now, to re-create the http listener on a specified IP address

```
New-WSManInstance winrm/config/Listener -SelectorSet @{Address="IP:192.168.100.2";Transport="http"}
```

Once this listener is created successfully, you need to restart the WinRM service using Restart-Service cmdlet. From this point onwards, system will listen only on 192.168.100.2 IP address for any remoting requests.

You can follow the same approach for HTTPS transport too. However, you will have to specify the CertificateThumbPrint.

Remoting in an enterprise

To enable remoting on multiple computers in an enterprise or domain environment, you can use group policy[5]. For more information on this, refer to Appendix B: Enable PowerShell remoting using group policy

Summary

In this chapter, we looked at how to enable remoting for basic usage. We also looked at how to configure computers in a workgroup and mixed domain environment to participate in PowerShell remoting. Beware that disabling remoting won't undo changes done by Enable-PSRemoting. If remoting is not required on the local computer, you should manually undo all the changes. This is a good security practice. There are few more things you should be aware of while configuring a computer for remoting. We will look at each of these in detail in part 2.

[5] http://technet.microsoft.com/en-us/library/dd347642.aspx

Chapter 3: Execute remote commands

Within remoting, there are a couple of ways to run commands or scripts on a remote machine. This includes Invoke-Command cmdlet and interactive remoting sessions. These two methods deserve a detailed discussion for each and hence we will see Invoke-Command method in this chapter and discuss interactive remoting in the next chapter.

Once you have enabled remoting on all your computers, you can use Invoke-Command cmdlet to run commands and scripts on local computer or on remote computer(s). There are many possible variations of this cmdlet.

Run script blocks on local or remote computer

You can invoke a command on local or remote computer(s) using the below method

```
Invoke-Command -ComputerName SP2010-WFE -ScriptBlock {Get-Process}
```

The ScriptBlock parameter can be used to specify a list of commands you want to run on the remote computer. ComputerName parameter is not required for running commands on the local machine. If you want to run the same command on multiple remote computers, you can supply the computer names as a comma separated list to ComputerName parameter or use a text file as shown in the example here

```
Invoke-Command -ComputerName SP2010-WFE, SP2010-DB -ScriptBlock {Get-Process}
```

OR

```
Invoke-Command -ComputerName (get-content c:\scripts\servers.txt) -ScriptBlock {Get-Process}
```

This method is also called fan-out or 1: many remoting. You run the same command on multiple computers in just a single command.

All commands and variables in the ScriptBlock are evaluated on the remote computer. So, if you do something like -ScriptBlock {Get-Process -Name $procName}, PowerShell expects the remote computer session to have $procName defined. You can however pass variables on the local computer to a remote session when using Invoke-Command. This brings us to the next point in our discussion.

Run script files on remote computers

There are a couple of ways of doing this.

First method is to use the –FilePath parameter.

```
Invoke-Command -ComputerName SP2010-WFE -FilePath C:\scripts\Test.PS1
```

Note that the script you provide as a value to -FilePath must exist on the local machine or at a place accessible to the local machine. So, what if you want to run a script that exists only on the remote server?