12

Managing Hyper-V

In this chapter, we cover the following recipes:

* Installing Hyper-V inside Windows Server
* Creating a Hyper-V VM
* Using PowerShell Direct
* Using Hyper-V VM Groups
* Configuring VM hardware
* Configuring VM networking
* Implementing Nested Virtualization
* Managing VM state
* Managing storage movement
* Configure VM replication
* Managing VM checkpoints

# Introduction

Hyper-V is Microsoft's virtual machine (VM) hypervisor. Both Windows Server 2019 and Windows 10 include Hyper-V as an option you can install in all versions of Windows Server 2019 and the Enterprise, Professional, and Education editions of Windows 10.

Microsoft first released Hyper-V with Server 2008. Microsoft has improved it significantly with each successive version of Windows Server. Improvements include additional features, support of the latest hardware, and improvements in scalability.

Hyper-V supports nested virtualization, the ability to run Hyper-V inside a Hyper-V VM. Nested virtualization has some good use cases, such as in training—give each student a VM on a large blade in which are the VMs needed for the course labs. Nested virtualization also provides an additional layer of security that might be useful in multi-tenant scenarios.

Microsoft also ships a free version of Hyper-V, the Microsoft Hyper-V Server. The Hyper-V Server runs virtual machines with no GUI. You configure and manage remotely using recipes like the ones in this chapter.

This chapter focuses solely on Hyper-V inside Windows Server 2022, although you can manage a Hyper-V Server using the tools used in this chapter's recipes. References to your Hyper-V servers refer to your Windows 2022 servers that have the Hyper-V feature added. Hyper-V's management tools enable you to configure and manage both the Hyper-V service and the virtual machines running on your Hyper-V servers.

This chapter starts with installing and configuring the Hyper-V feature. After installing Hyper-V, you go on to create a VM. PSDirect, which requires you to download an ISO image of Windows Server from the Internet.

After you create the PSDirect VM, you use the VM. You use PowerShell Direct to use a remoting session into a VM without using a network connection. You also configure a VM’s hardware and networking capability. You then use the PowerShell cmdlets to manage the state of a VM.

Hyper-V allows you to move a VM and/or a VM’s storage between Hyper-V hosts. For disaster recovery, you can replicate a running VM (and use that replica should the primary VM fail.

You can take snapshots, or checkpoints, of a VM to save a VM's state and restore your VM to that point as you can see in the Managing VM Checkpoints recipe.

# Installing Hyper-V inside Windows Server

In this recipe, you add and configure the Hyper-V feature on HV1, a domain-joined Windows Server 2022 host, on which you have installed PowerShell 7 and VS Code. Logon to HV1 as Reskit\Administrator.

## Getting Ready

This recipe uses SRV2, a recently added workgroup host. By default, this host is a DHCP client.

## How to do it...

1. Installing the Hyper-V feature on HV1, HV2

$SB = {

  Install-WindowsFeature -Name Hyper-V -IncludeManagementTools

}

Invoke-Command -ComputerName HV1, HV2 -ScriptBlock $Sb

1. Rebooting the servers to complete the installation

Restart-Computer -ComputerName HV2 -Force

Restart-Computer -ComputerName HV1 -Force

1. Creating a PSSession with both HV Servers (after reboot)

$S = New-PSSession HV1, HV2

1. Creating and setting the location for VMs and VHDs on HV1 and HV2

$SB = {

    New-Item -Path C:\Vm -ItemType Directory -Force |

        Out-Null

    New-Item -Path C:\Vm\Vhds -ItemType Directory -Force |

        Out-Null

    New-Item -Path C:\Vm\VMs -ItemType Directory -force |

        Out-Null

Invoke-Command -ScriptBlock $SB -Session $S | Out-Null

1. Setting the default paths for Hyper-V VM disk/config information

$SB = {

  $VMs  = 'C:\Vm\Vhds'

  $VHDs = 'C:\Vm\VMsV'

  Set-VMHost -ComputerName Localhost -VirtualHardDiskPath $VMs

  Set-VMHost -ComputerName Localhost -VirtualMachinePath $VHDs

}

Invoke-Command -ScriptBlock $SB -Session $S

1. Setting NUMA spanning

$SB = {

  Set-VMHost -NumaSpanningEnabled $true

}

Invoke-Command -ScriptBlock $SB -Session $S

1. Setting EnhancedSessionMode

$SB = {

 Set-VMHost -EnableEnhancedSessionMode $true

}

Invoke-Command -ScriptBlock $SB -Session $S

1. Setting host resource metering on HV1, HV2

$SB = {

 $RMInterval = New-TimeSpan -Hours 0 -Minutes 15

 Set-VMHost -ResourceMeteringSaveInterval $RMInterval

}

Invoke-Command -ScriptBlock $SB -Session $S

1. Reviewing key VM host settings

$SB = {

  Get-VMHost

}

$P = 'Name', 'V\*Path','Numasp\*', 'Ena\*','RES\*'

Invoke-Command -Scriptblock $SB -Session $S |

  Format-Table -Property $P

## How it works...

In step 1, you install the Hyper-V feature to both HV1 and HV2. The output from this step looks like this:

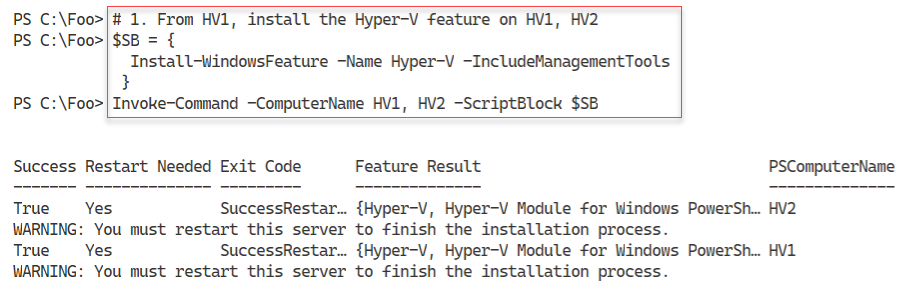


Figure 12.1: Installing the Hyper-V Feature on HV1 and HV2

**Insert image B42024\_12\_01.png**

To complete the installation of Hyper-V, in step 2 you reboot both HV1 and HV2. This step creates no output but does reboot both hosts.

After HV1 and HV2 have rebooted, you login back into the host using Reskit\Administrator. In step 3, you create two new PowerShell remoting sessions to HV1 and HV2. In step 4, you use the remoting sessions and create new folders on HV1 and HV2 to hold Hyper-V VMs and Hyper-V virtual disks. With step 5, you configure Hyper-V on both hosts to use these new locations to store VMs and virtual drives. In step 6 you specify the host should support NUMA spanning. In step 7, you set enhanced session mode to improve VM connections. In step 8, you set the two Hyper-V hosts to makes use of resource metering. These six steps produce no console output.

In step 9, you review the key Hyper-V host settings with output like this:

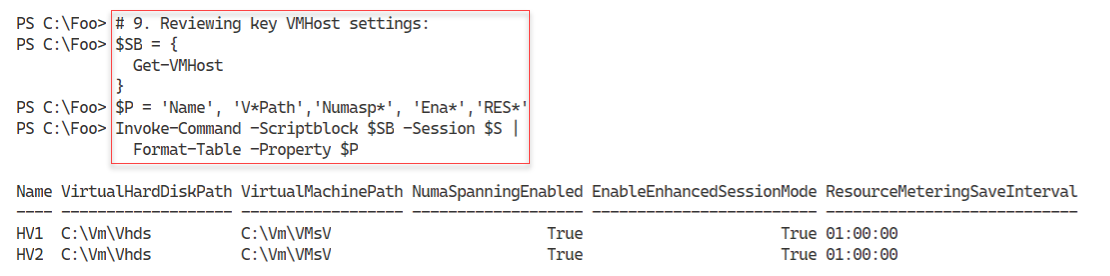


Figure 12.2: Viewing Hyper-V settings

**Insert image B42024\_12\_02.png**

## There's more...

In step 1, you install the Hyper-V feature on two servers. You can only do this successfully if the host you are using supports the necessary virtualization capabilities and you have enabled them in your system's BIOS. To ensure if your system is capable, see this link: http://mikefrobbins.com/2012/09/06/use-powershell-to-check-for-processorcpu-second-level-address-translation-slat-support/. Additionally, ensure you double-check the BIOS to ensure virtualization is enabled prior to running this step.

In step 2, you restart both servers. You could have allowed Install-WindowsFeature (used in step 1) to restart the servers automatically by using the -Restart switch. In automation terms, this could have meant that the system started rebooting before the remote script had completed, which could cause Invoke-Command to error out. The recipe avoids this by not rebooting after the installation of the Hyper-V features, then rebooting in a controlled way. Once the restart has completed, your script can carry on managing the servers.

In step 5through step 8, you set up one aspect of the VM hosts in each step. You could have combined  
these steps and just called Set-VMHost once with all of the properties specified.

## See also

You can find more information on some of the Hyper-V features used in this recipe (details of  
which are outside the scope of this book), as follows:

|  |  |
| --- | --- |
| Features | Links for more information |
| Connecting to a VM, including enhanced session mode | https://docs.microsoft.com/en-us/windows-server/ virtualization/hyper-v/learn-more/use-local resources-on-hyper-v-virtual-machine-with-vmconnect |
| Understanding the hard disk options | https://www.altaro.com/hyper-v/understanding working-vhdx-files/ |
| Hyper-V and NUMA | https://blogs.technet.microsoft.com/ pracheta/2014/01/22/numa-understand-it-its usefulness-with-windows-server-2012/ |
| Configuring Hyper-V Resource Metering | https://redmondmag.com/articles/2013/08/15/hyper-v resource-metering.aspx |

# Creating a Hyper-V VM

Creating a Hyper-V virtual machine is relatively straightforward and consists of a few simple steps.

First, you need to create the VM itself inside Hyper-V. Then, you create the VM's virtual hard drive, and add it to the VM. You may also wish to adjust the number of processors and memory for the VM and set the contents of the VM's DVD drive. Once you have created your VM, you need to install the VM's operating system. You have a number of options in terms of how you deploy Windows (or Linux) in a Hyper-V VM.

The Windows Assessment and Deployment Kit, a free product from Microsoft, contains a variety of tools to assist in the automation of deploying Windows. These include Deployment Image Servicing and Management (DISM), Windows Imaging and Configuration Designer (Windows ICD), Windows System Image Manager (Windows SIM), User State Migration Tool (USMT), and a lot more. For more information on the tools and deploying Windows, see https://docs.microsoft.com/en-us/windows/deployment/windowsdeployment-scenarios-and-tools.

Another way to install the OS into a VM is to just create the VM (either with PowerShell or the Hyper-V Manager) and attach the operating system's ISO image into the VM's DVD drive. After starting the VM, you do a manual installation and once the OS is installed, you can use the recipes in this book to configure the server to your needs.

In this recipe, you create a VM, PSDirect. Initially, this has a Windows-assigned host name which you later change to Tiger. In building the VM, you assign the Windows Server 2022 DVD to the VM's DVD drive. This ensures that, when you start the VM, Windows commences the GUI setup process, ending up with a fully installed OS inside the VM. The details of performing the actual installation are outside the scope of this recipe.

Two small issues using the GUI to install Windows Server 2022 are that the machine name is randomly generated by Windows and the VM is set up as a workgroup computer and not joined to the domain. You can easily script both renaming the server and joining the domain. The scripts used to generate the VM farm used in this book are examples of how to deploy Windows Server 2019 in a more automated fashion using a SETUP.XML file that specifies the details of the installation. The scripts that create the VMs used are available online at GitHub. See https://github.com/doctordns/ReskitBuildScripts for the scripts and documentation on them.

## Getting Ready

You run this this recipe on the VM host HV1 that you created in the Installing and configuring  
Hyper-Vrecipe. You also need the Windows Server ISO image. For testing purposes, this  
could be an evaluation version, or a full retail edition – and for this chapter you could use an image of Windows Server 2022 or Windows Servere 2019.. You can download a

## How to do it...

1. Setting up the VM name and paths for this recipe

$VMname      = 'PSDirect'

$VMLocation  = 'C:\Vm\VMs'

$VHDlocation = 'C:\Vm\Vhds'

$VhdPath     = "$VHDlocation\PSDirect.Vhdx"

$ISOPath     = 'C:\builds\en\_windows\_server\_2019\_x64.iso'

If ( -not (Test-Path -Path $ISOPath -PathType Leaf)) {

  Throw "Windows Server ISO DOES NOT EXIST"

}

1. Creating a new VM

New-VM -Name $VMname -Path $VMLocation -MemoryStartupBytes 1GB

1. Creating a virtual disk file for the VM

New-VHD -Path $VhdPath -SizeBytes 128GB -Dynamic | Out-Null

1. Adding the virtual hard drive to the VM

Add-VMHardDiskDrive -VMName $VMname -Path $VhdPath

1. Setting ISO image in the VM's DVD drive

$IHT = @{

  VMName           = $VMName

  ControllerNumber = 1

  Path             = $ISOPath

}

Set-VMDvdDrive @IHT

1. Starting the VM

Start-VM -VMname $VMname

1. Viewing the VM

Get-VM -Name $VMname

## How it works...

In step 1, you specify the locations for VMs and VM hard drives and assign them to variables. You also check to ensure that the Windows Server ISO is in the correct place. Assuming the ISO exists and you have it named correctly then this step produces no output. If the step fails to find the file, it aborts.

In step 2, you create a new VM using the New-VM cmdlet, with output that looks like this:

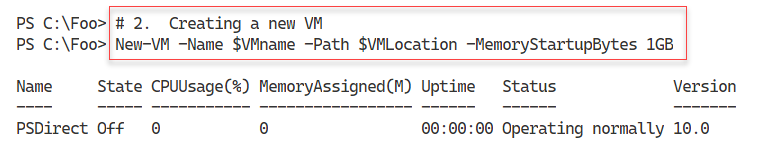


Figure 12.3: Creating a new Hyper-V VM

**Insert image B42024\_12\_03.png**

In step 3, you create a virtual disk file for the VM and you add this VHDX to the PSDirect VM in step 4. In step 5, you add the ISO image to the PSDirect VM and then in step 6, you start the VM. These 4 steps create no output.

In the final step in this recipe, you use Get-VM to view the VM details, producing output like this;

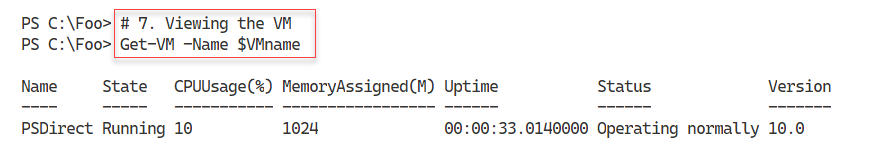


Figure 12.4: Viewing the new Hyper-V VM

**Insert image B42024\_12\_04.png**

## There's more...

In step 1, you specify the name of the ISO image for Windows Server. Depending on which image you are using, the file name of the ISO you download may be different. Either adjust the file name to match this step, or change this step to match the name of the ISO file. For released versions of Windows server, you can find ISO imagers for evaluation versions here { }. If you like living in the fast lane, consider joining the Windows Insiders project and using the latest preview build – see {}.

In step 2, you create a VM. Although that step should succeed, the VM you create would not yet be usable as you need to define, at least, one virtual hard drive to use with the VM. You carry out these configuration steps in subsequent steps.

Once you start the VM, in step 6, the VM will undertake the installation of Windows Server. You need to use the GUI to complete the installation. For the purposes of this chapter, accept all the defaults and ensure you create a new user with the password Pa$$w0rd. As with all the passwords used in this book, feel free to use whatever passwords you wish – just ensure you don’t forget the passwords(s) later.

# Using PowerShell Direct

PowerShell direct is a Hyper-V feature that enables you to open PowerShell remoting sessions on a VM without using a network connection. This feature enables you to create a remoting session inside the VM to fix issues, such as networking misconfiguration. An administrator might be commissioning a new host and configures its host IP address nearly correctly, meaning a network connectivity Catch-22 situation. Without a working network connection to the VM, you can’t fix the issue – but until you fix the issue, you cannot make a connection to the VM. With PS direct, you can create a remoting session in the VM, as you see in this recipe.

## Getting Ready

This recipe uses HV1, a Windows Server Datacenter host on which you have installed the Hyper-V feature. You should have also created a VM of Windows Server called PSDirect. This recipe demonstrates PowerShell direct, so it doesn’t matter which version of the operating system you install so long as it is a supported version of Windows (and you complete the installation of the OS inside the PSDirect VM.

You run the final steps of this recipe on DC1, showing how to connect to a VM remotely from the Hyper-V host.

## How to do it...

1. Creating a credential object for Reskit\Administrator

$Admin = 'Administrator'

$PS    = 'Pa$$w0rd'

$RKP   = ConvertTo-SecureString -String $PS -AsPlainText -Force

$Cred  = [System.Management.Automation.PSCredential]::New(

          $Admin, $RKP)

1. Viewing the PSDirect VM

Get-VM -Name PSDirect

1. Invoking a command on the VM specifying VM name

$SBHT = @{

  VMName      = 'PSDirect'

  Credential  = $Cred

  ScriptBlock = {hostname}

}

Invoke-Command @SBHT

1. Invoking a command based on VMID

$VMID = (Get-VM -VMName PSDirect).VMId.Guid

Invoke-Command -VMid $VMID -Credential $RKCred  -ScriptBlock {ipconfig}

1. Entering a PS remoting session with the PSDirect VM

Enter-PSSession -VMName PSDirect -Credential $Cred

Get-CimInstance -Class Win32\_ComputerSystem

Exit-PSSession

You now run the rest of this recipe from DC1.

1. Creating a remoting session to HV1 (Hyper-V Host)

$RS = New-PSSession -ComputerName HV1

1. Entering an interactive session with HV1

Enter-PSSession $RS

$PSDRS = New-PSSession -VMName PSDirect

1. Entering and using the remoting session inside PSDirecet

Enter-PSSession -Session $PSDRS

hostname

1. Creating credential for PSDirect inside HV1

$Admin = 'Administrator'

$PS    = 'Pa$$w0rd'

$RKP   = ConvertTo-SecureString -String $PS -AsPlainText -Force

$Cred  = [System.Management.Automation.PSCredential]::New(

          $Admin, $RKP)

1. Using PS Direct session to the VM

Enter-PSSession -VMName PSDirect -Credential $Cred

1. Closing sessions

Exit-PSSession # Exit session on PSDirect

Exit-PSSession # Exit session on HV1

## How it works...

In step 1, you create a PowerShell credential object. You use this object later in this recipe to enable connection to the PSDirect VM. This step creates no output.

In step 2, you use the Get-VM cmdlet to view the PSDirect VM, with output like this:

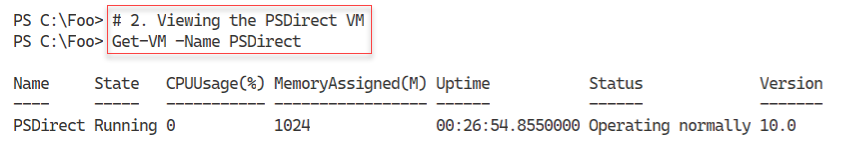


Figure 12.5: Viewing the PSDirect Hyper-V VM

**Insert image B42024\_12\_05.png**

In step 3, you use Invoke-Command to run the hostname command in the PSDirect VM, with output like this:

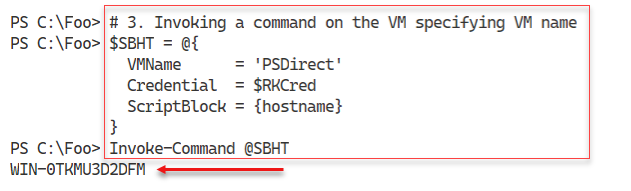


Figure 12.6: Checking host name of PSDirect VM

**Insert image B42024\_12\_06.png**

In step 4, you invoke a command inside the PSDirect VM, but using the VM’s GUID ID, with output like this:

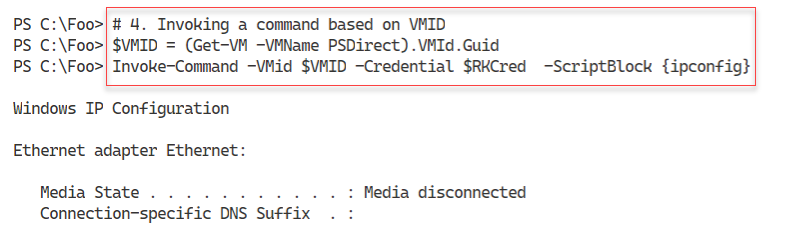


Figure 12.7: Checking host name of PSDirect VM by VM guif

**Insert image B42024\_12\_07.png**

In step 5, you use the Enter-PSSession command to enter an interactive session with the PSDirect VM. Inside the remoting session, you run Get-CimInstance to return details of the computer system of the VM. The output should look like this:

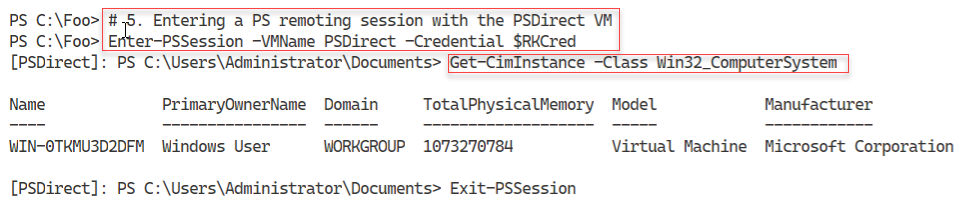


Figure 12.8: Checking host name of PSDirect VM by VM guif

**Insert image B42024\_12\_08.png**

Now that you have created a working VM in HV1, you run the remainder of the VM remotely, on DC1. In step 6, you create a PowerShell remoting session with the Hyper-V Host (HV1) which creates no console output.

In step 7, you enter a remoting session with HV1. Inside the remoting session, you create a PS Direct based remoting session with the PSDirect VM, with output like this:

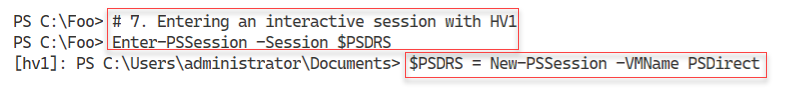


Figure 12.9: Using the remoting session on HV`

**Insert image B42024\_12\_09.png**

In step 8, within the remoting session to HV1, you enter the remoting session to PSDirect and run the hostname command. The output of this step looks like this:

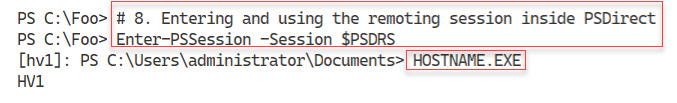


Figure 12.10: Running HOSTNAME.EXT in session to HV1

**Insert image B42024\_12\_10.png**

In step 9, you create credentials for the PSDirect VM inside the HV1 VM, producing no output.

In step 10, you enter an interactive session on PSDirect from the remoting session on HV1. Inside that session, you run the HOSTNAME.EXE command, which produces output like this:

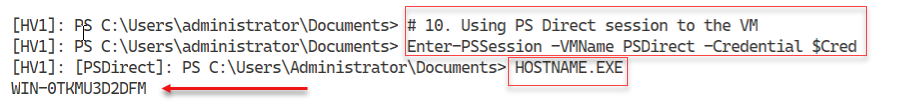


Figure 12.11: Running HOSTNAME.EXT in session to PSDirect from HV1

**Insert image B42024\_12\_11.png**

In the final step in this recipe, step 11, you exit from the interactive sessions on PSDirect and HV1 VMs. The output from this step looks like this:

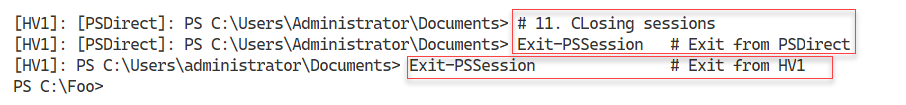


Figure 12.12: Exiting from remoting sessions

**Insert image B42024\_12\_12.png**

## There's more...

In step 2, you use PowerShell Direct to enter a session inside a VM and run a command. Assuming you installed Windows in the PSDirect VM using a supported version of Windows, the setup process generatres a random machine name, in this case WIN-0TKMU3D2DFM. The machine name of your VM is liketly to be different.

With step 8 and step 9, you fiurst ctreate a remoting session from DC1 to HV1. Inside that session you create a further, PowerShell Direct session with the VM, PSDirect on which you run a command (HOSTNAME.EXE). This is a second hop, but is not a demonstration or a real workaround to the Kerberos double hop issue.

# Using Hyper-V VM Groups

Hyper-Vs VM groups allow you to group VMs for the purposes of automation. There are two types of VM groups you can create: a VMCollectionType and a ManagementCollectionType. A VMCollectionType VM group contains VMs, while the ManagementCollectionType VM group contains VMCollectionType VM groups.

The might enable you to have two VMCollectionType VM groups, SQLAccVMG (that contains the VMs SQLAcct1, SQLAcct2, and SQLAcct3) and a group, SQLAccVMG, that contains the VMs SQLMfg1 and SQLMfg2. You could then create a ManagementCollectionType VM group, VM-All, containing the two VMCollectionType VM groups.

The VMGroup feature is one feels incomplete. For example, there is not -VMGroup parameters on any of the Hyper-V cmdlets enabling you to configure a VM group. And having two types of groups seems over the top. All in all, this feature could be improved.

## Getting Ready

You run this recipe on the HV2, a Windows Server 2022 server with the Hyper-V feature added., You configured this host in the Installing and configuring Hyper-Vrecipe.

## How to do it...

1. Creating VMs on HV2

$VMLocation  = 'C:\Vm\VMs'   # Created in earlier recipe

# Create SQLAcct1

$VMN1        = 'SQLAcct1'

New-VM -Name $VMN1 -Path "$VMLocation\$VMN1"

# Create SQLAcct2

$VMN2        = 'SQLAcct2'

New-VM -Name $VMN2 -Path "$VMLocation\$VMN2"

 # Create SQLAcct3

$VMN3        = 'SQLAcct3'

New-VM -Name $VMN3 -Path "$VMLocation\$VMN3"

# Create SQLMfg1

$VMN4        = 'SQLMfg1'

New-VM -Name $VMN4 -Path "$VMLocation\$VMN4"

# Create SQLMfg2

$VMN5        = 'SQLMfg2'

New-VM -Name $VMN5 -Path "$VMLocation\$VMN5"

1. Viewing SQL VMs

Get-VM -Name SQL\*

1. Creating Hyper-V VM groups

$VHGHT1 = @{

  Name      = 'SQLAccVMG'

  GroupType = 'VMCollectionType'

}

$VMGroupACC = New-VMGroup @VHGHT1

$VHGHT2 = @{

  Name      = 'SQLMfgVMG'

  GroupType = 'VMCollectionType'

}

$VMGroupMFG = New-VMGroup @VHGHT2

1. Displaying the VM groups on HV2

Get-VMGroup |

  Format-Table -Property Name, \*Members, ComputerName

1. Creating arrays of group member VM Names

$ACCVMs = 'SQLAcct1', 'SQLAcct2','SQLAcct3'

$MFGVms = 'SQLMfg1', 'SQLMfg2'

1. Adding members to the Accounting SQL VMgroup

Foreach ($Server in $ACCVMs) {

    $VM = Get-VM -Name $Server

    Add-VMGroupMember -Name SQLAccVMG -VM $VM

}

1. Adding members to the Manufacturing SQL VM Group

Foreach ($Server in $MfgVMs) {

    $VM = Get-VM -Name $Server

    Add-VMGroupMember -Name  SQLMfgVMG -VM $VM

}

1. Viewing VM Groups on HV2

Get-VMGroup |

 Format-Table -Property Name, \*Members, ComputerName

1. Creating a management collection VMGroup

$VMGHT = @{

  Name      = 'VMMGSQL'

  GroupType = 'ManagementCollectionType'

}

$VMMGSQL = New-VMGroup  @VMGHT

1. Adding the two VMCollectionType groups to the VMManagement group

Add-VMGroupMember -Name VMMGSQL -VMGroupMember $VMGroupACC,

                                               $VMGroupMFG

1. Setting Format Enumeration Limit to 99

$FormatEnumerationLimit = 99

1. Viewing VM groups by type

Get-VMGroup | Sort-Object -Property GroupType |

  Format-Table -Property Name, GroupType, VMGroupMembers,

                         VMMembers

1. Stopping all the SQL VMs

Foreach ($VM in ((Get-VMGroup VMMGSQL).VMGroupMembers.vmmembers)) {

  Stop-VM -Name $vm.name -WarningAction SilentlyContinue

}

1. Setting CPU count in all SQL VMs to 4

Foreach ($VM in ((Get-VMGroup VMMGSQL).VMGroupMembers.VMMembers)) {

  Set-VMProcessor -VMName $VM.name -Count 4

}

1. Setting Accounting SQL VMs to have 6 processors

Foreach ($VM in ((Get-VMGroup SQLAccVMG).VMMembers)) {

  Set-VMProcessor -VMName $VM.name -Count 6

}

1. Checking Processor counts for all VMs sorted by CPU Count

$VMS = (Get-VMGroup -Name VMMGSQL).VMGroupMembers.VMMembers

Get-VMProcessor -VMName $VMS.Name |

  Sort-Object -Property Count -Descending |

    Format-Table -Property VMName, Count

1. Removing VMs from VM Groups

$VMs = (Get-VMGroup -Name SQLAccVMG).VMMEMBERS

Foreach ($VM in $VMS)  {

  $X = Get-VM -vmname $VM.name

  Remove-VMGroupMember -Name SQLAccVMG -VM $x

  }

$VMs = (Get-VMGroup -Name SQLMFGVMG).VMMEMBERS

Foreach ($VM in $VMS)  {

  $X = Get-VM -vmname $VM.name

  Remove-VMGroupMember -Name SQLmfgvMG -VM $x

}

1. Removing VM Groups from VMManagementGroups

$VMGS = (Get-VMGroup -Name VMMGSQL).VMMembers

Foreach ($VMG in $VMGS)  {

  $X = Get-VMGroup -vmname $VMG.name

  Remove-VMGroupMember -Name VMMGSQL -VMGroupName $x

}

1. Removing all the VMGroups

Remove-VMGroup -Name SQLACCVMG -Force

Remove-VMGroup -Name SQLMFGVMG -Force

Remove-VMGroup -Name VMMGSQL   -Force

## How it works...

In step 1, you create several VMs, using the New-VM cmdlet, the out put of this step looks like this:

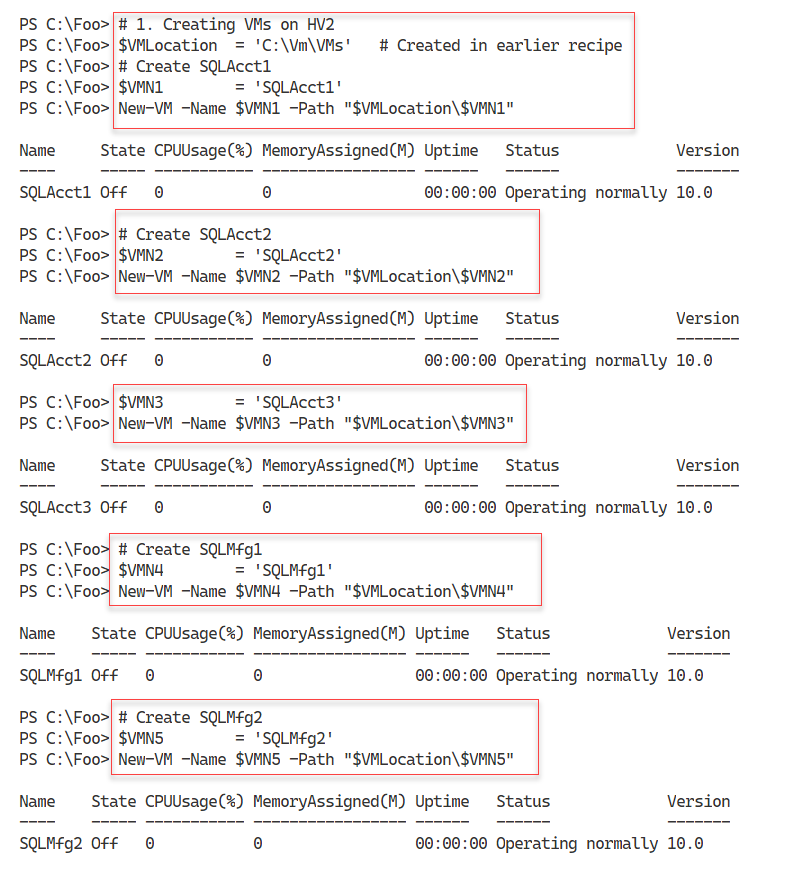


Figure 12.13: Creating VMs for this recipe

**Insert image B42024\_12\_13.png**

In step 2, you use the Get-VM cmdlet to look at the six VMs you just created, with output like this:

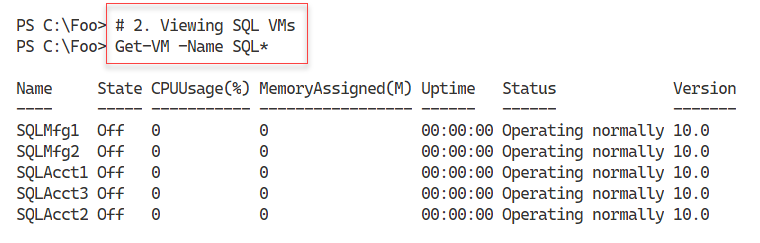


Figure 12.14: Viewing SQL VMs

**Insert image B42024\_12\_14.png**

In step 3, you create several Hyper-V VM collection type VM groups, creating no output. In step 4, you examine the existing VM groups on HV2, with output like this:

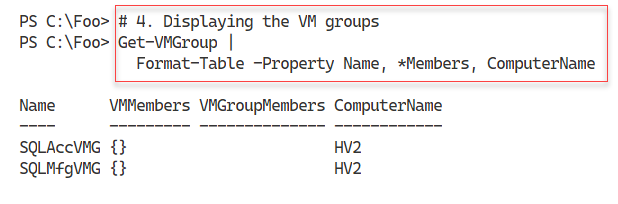


Figure 12.15: Viewing VM Groups on HV2

**Insert image B42024\_12\_15.png**

To simplify the creation of VM collection groups, in step 5, you create arrays of the VM names. In step 6, you add VMs to the SQLAccVMG VM group, while in step 7, you add VMs to the SQLMfgVMG VM group. These three steps produce no console output.

Then, in step 8, you view the VM groups again, with output like this:

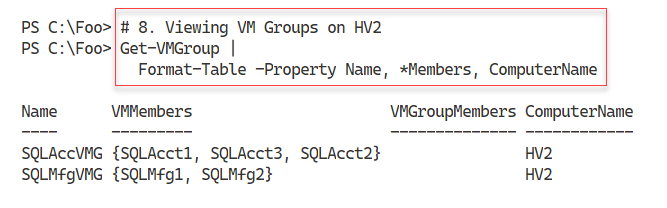


Figure 12.16: Viewing VM Groups on HV2

**Insert image B42024\_12\_16.png**

In step 9, youy create a VM Management collection group and in step 10, you populate the VM Management collection. To simplify the output, in step 11, you set the $FormatEnumerationLimit to 99. These two steps create no output.

In step 12, you view all the fully populated VM groups, sorted by VM group type, with output like this:

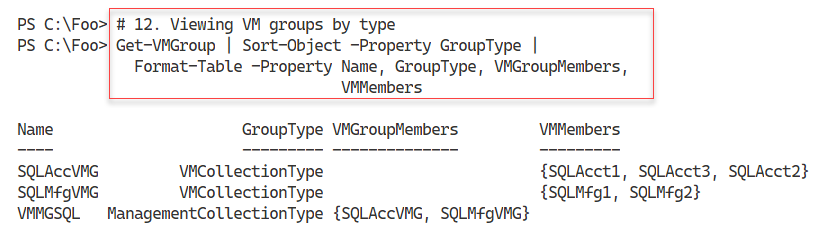


Figure 12.17: Viewing fully populated VM Groups on HV2

**Insert image B42024\_12\_17.png**

In step 13, you use the VM groups you have created to stop, explicitly, all the SQL VMs. In step 14, you set the VMs in the VMMGSQL VM management collection group to have four virtual processoers, and in step 15, you set the VMs in the SQLAccVMG VM collection group. These three steps produce no output to the console.

In step 16, you review the virtual processors assigned to each SQL VM, like this:

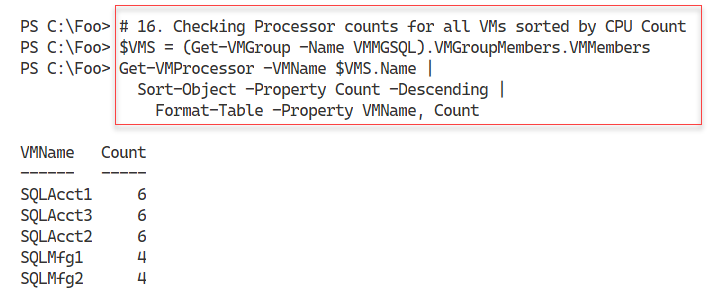


Figure 12.18: Viewing virtual processors assigned to each SQL VM

**Insert image B42024\_12\_18.png**

In step 17, you remove all the VM from the VM collection groups and in step 18, you remove all the members from the VM management VM groups. Finally, in step 19, you remove all the VM groups. These three steps produce no output.

## There's more...

In step 11, you set a value to the $FormatEnumerationLimit automatic variable. This variable controls how many repeating groups formatted by the Format-\* cmdlets. By default, PowerShell initializes this variable with a value of 4. Using this default, in step 11, you would se at most 4 members ot eacvh of the VM groups with PowerShell displaying “…” to indicate you have more than 4 members. With this step, PowerShell can display up to 99 VM members for each VM group.

# Configuring VM Hardware

Configuring hardware in your virtual machine is very much like configuring a physical computer, just without the need for a screwdriver. With a physical computer, you can adjust the CPUs and BIOS settings and adjust physical RAM, network interfaces, disk interfaces, disk devices, DVD drives (with/without a loaded DVD), and more. You can find each of these components within a Hyper-V VM. As you can see in the recipe, the PowerShell cmdlets make it simple to configure the virtual hardware available to any Hyper-V VM.

In this recipe, you adjust the VM's BIOS, CPU count, and memory and then add a SCSI controller. With the controller in place, you create a new virtual disk and assign it to the SCSI controller. Then, you view the results.

Just like in most physical servers, not all of these components can be changed while the server is running. You run this recipe from HV1 and turn the PSDirect VM off before configuring the virtual hardware.

This recipe does not cover the VM's virtual NIC. By default, Virtual Machines (such as you created in the Creating a virtual machine recipe) contain a single virtual NIC. You can add additional NICs to any VM should you wish to. You view how to configure VM networking networking in the Configuring Hyper-V networking recipe.

## Getting Ready

This recipe uses HV1, a Windows Server Datacenter host on which you have installed the Hyper-V feature. You should have also created a VM of Windows Server called PSDirect.

## How to do it...

1. Turning off the PSDirect VM

Stop-VM -VMName PSDirect

Get-VM -VMName PSDirect

1. Setting the StartupOrder in the VM's BIOS

$Order = 'IDE','CD','LegacyNetworkAdapter','Floppy'

Set-VMBios -VmName PSDirect -StartupOrder $Order

Get-VMBios PSDirect

1. Setting and viewing CPU count for PSDirect

Set-VMProcessor -VMName PSDirect -Count 2

Get-VMProcessor -VMName PSDirect |

  Format-Table VMName, Count

1. Setting and viewing PSDirect memory

$VMHT = [ordered] @{

  VMName               = 'PSDirect'

  DynamicMemoryEnabled = $true

  MinimumBytes         = 512MB

  StartupBytes         = 1GB

  MaximumBytes         = 2GB

}

Set-VMMemory @VMHT

Get-VMMemory -VMName PSDirect

1. Adding and viewing a ScsiController in the PSDirect VM

Add-VMScsiController -VMName PSDirect

Get-VMScsiController -VMName PSDirect

1. Starting the PSDirect VM

Start-VM -VMName PSDirect

Wait-VM -VMName PSDirect -For IPAddress

1. Creating a new VHDX file for the PSDirect VM

$VHDPath = 'C:\Vm\Vhds\PSDirect-D.VHDX'

New-VHD -Path $VHDPath -SizeBytes 8GB -Dynamic

1. Getting Controller number of the newly added SCSI controller

$VM    = Get-VM -VMName PSDirect

$SCSIC = Get-VMScsiController -VM $VM|

           Select-Object -Last 1

1. Adding the VHD to the ScsiController

$VHDHT = @{

    VMName            = 'PSDirect'

    ControllerType    = $SCSIC.ControllerNumber

    ControllerNumber  =  0

    ControllerLocation = 0

    Path               = $VHDPath

}

Add-VMHardDiskDrive @VHDHT

1. Viewing virtual drives in the PSDirect VM

Get-VMScsiController -VMName PSDirect |

  Select-Object -ExpandProperty Drives

## How it works...

In step 1, you turn off the PSDirect VM and check the VM’s status, with output like this:

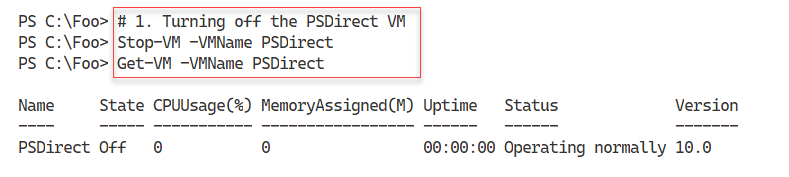


Figure 12.19: Ensuring PSDirect VM is turned off

**Insert image B42024\_12\_19.png**

In step 2, you set the startup order for the VM, then view the VM’s virtual BIOS. The output of this step is:

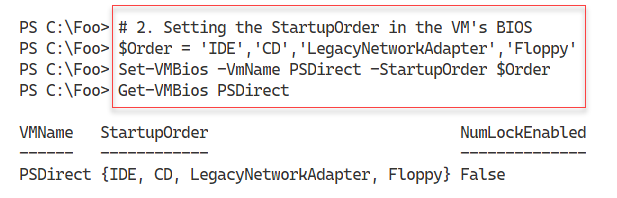


Figure 12.20: Updating and viewing the VM’s BIOS

**Insert image B42024\_12\_20.png**

In step 3, you adjust then view the number of virtual processors for the PSDirect VM, with output like this:

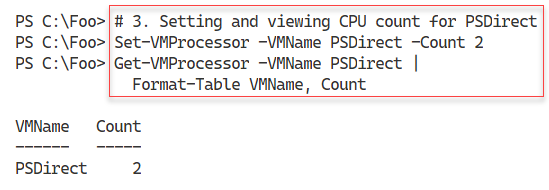


Figure 12.21: Updating and viewing the VM’s BIOS

**Insert image B42024\_12\_21.png**

In step 4, you set the PS Direct’s virtual memory then view the memory settings, with the following output:

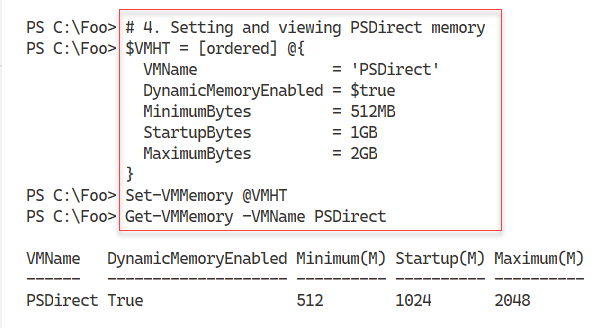


Figure 12.22: Changing and viewing VM memory

**Insert image B42024\_12\_22.png**

In step 5, you add a SCSI controller to the PSDirect VM then you view the SCSI controllers in the VM with output like this:

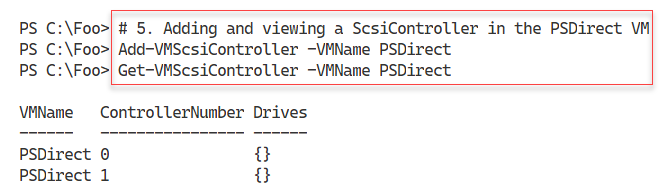


Figure 12.23: Adding and viewing SCSI virtual disk controllers

**Insert image B42024\_12\_23.png**

Now that you have asjusted the virtual hardware for the PSDirect VM, in step 6 you restart the VM aned wait for the VM to come up. This step generates no output.

In step 7, you create a new virtual disk file for the PSDirect VM, with output like this:

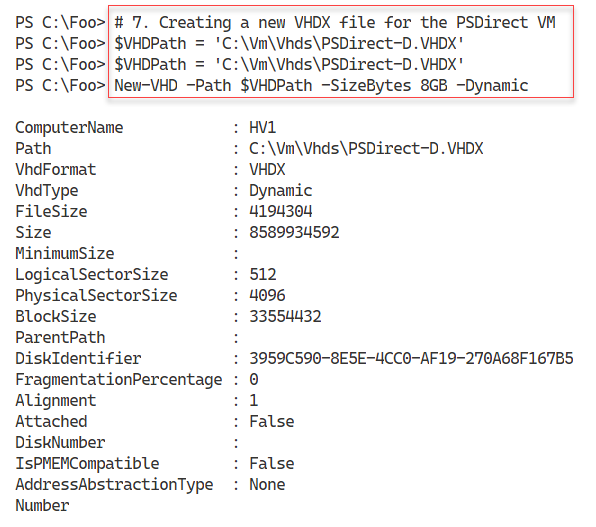


Figure 12.24: Creating a new virtual disk drive

**Insert image B42024\_12\_24.png**

In step 8, you get the SCSI controllers in the PSDirect VM and the controller you added in step 5. Then, in step 9, you add the newly created virtual disk drive to the newly added SCSI controller. These two steps create no console output.

In step 10, you get the disk drives attached to SCSI controllers in the PSDirect VM, with output like this:

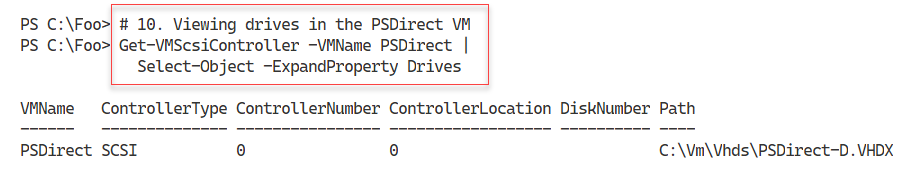


Figure 12.25: Viewing the SCSI disk drives in the PSDirect VM

**Insert image B42024\_12\_25.png**

## There's more...

With Hyper-V, you can only update some of the VM hardware configuration settings when the VM is switched off. In step 1, you switch the VM off so you can make virtual BIOS settings then turn on the VM.

Some settings, such as SCSI disk drives, can be added and removed from a running VM. In step 9, you add a newly created virtual disk drive to the running PSDirect VM.

# Configuring VM Networking

In the Creating a virtual machine recipe, you created a VM, PSDirect. This virtual machine has, by default, a single network card that Hyper-V sets to acquire IP address details from DHCP. In this recipe, you assign the NIC to a switch and configure IP address details for the virtual network adapter.

## Getting Ready

You run the recipe on HV1 and assumed you have created the PSDirect VM, as per the Creating a virtual machinerecipe. This recipe also makes use of a DHCP server running on DC1. You set this DHCP server up in the Installing and authorizing a DHCP Serverrecipe, and configured the DHCP server in the Configure DHCP scopesrecipe, in the Managing Windows Network Services chapter.

This chapter uses the PSDirect VM you created earlier. When you built this machnine using the normal setup routine, Windowsd assigns a random machnine name, which you saw in a previous recipe (Using PowerShell Direct|). In this recipe, you also change the name of the host inside to Wolf.

In this recipe, you set the the PSDirect VM’s networking card to enable MAC address spoofing. This ehables the VM to see the network and to get an IP address from the DHCP server on DC1. If you are running HV1 as a VM, you must also enable MAC spoofing on the NIC(s) in this VM on HV1’s VM host.

## How to do it...

1. Setting the PSDirect VM's NIC

Get-VM PSDirect |

  Set-VMNetworkAdapter -MacAddressSpoofing On

1. Getting NIC details and any IP Address from the PSDirect VM

Get-VMNetworkAdapter -VMName PSDirect

1. Creating a credential then get VM networking details

$RKAn = 'localhost\Administrator'

$PS = 'Pa$$w0rd'

$RKP = ConvertTo-SecureString -String $PS -AsPlainText -Force

$T = 'System.Management.Automation.PSCredential'

$RKCred = New-Object -TypeName $T -ArgumentList $RKAn, $RKP

$VMHT = @{

    VMName      = 'PSDirect'

    ScriptBlock = {Get-NetIPConfiguration | Format-Table }

    Credential  = $RKCred

}

Invoke-Command @VMHT | Format-List

1. Creating a virtual switch on HV1

$VSHT = @{

    Name           = 'External'

    NetAdapterName = 'Ethernet'

    Notes          = 'Created on HV1'

}

New-VMSwitch @VSHT

1. Connecting the PSDirect VM's NIC to the External switch

Connect-VMNetworkAdapter -VMName PSDirect -SwitchName External

1. Viewing VM networking information

Get-VMNetworkAdapter -VMName PSDirect

1. Observing the IP address in the PSDirect VM

$NCHT = @{

    VMName      = 'PSDirect'

    ScriptBlock = {Get-NetIPConfiguration}

    Credential  = $RKCred

}

Invoke-Command @NCHT

1. Viewing the hostname on PSDirect

$NCHT.ScriptBlock = {hostname}

Invoke-Command @NCHT

1. Changing the name of the host in the PSDirect VM

$NCHT.ScriptBlock = {Rename-Computer -NewName Wolf -Force}

Invoke-Command @NCHT

1. Rebooting and wait for the restarted PSDirect VM

Restart-VM -VMName PSDirect -Wait -For IPAddress -Force

1. Getting hostname of the PSDirect VM

$NCHT.ScriptBlock = {HOSTNAME}

Invoke-Command @NCHT

## How it works...

In step 1, you set the PSDirect VM’s NIC to enable MAC address spoofing. There is no output from this step.

In step 2, you get the NIC details for the NIC assigned to the PSDirect VM, with output like this:

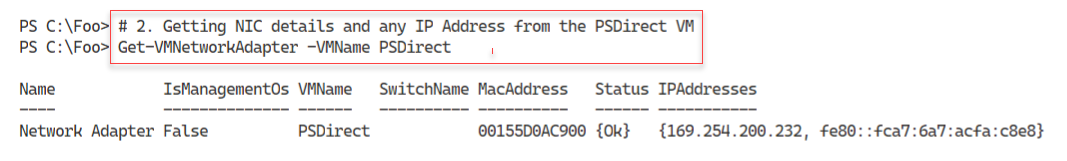


Figure 12.26: Viewing the NIC details for the PSDirect VM

**Insert image B42024\_12\_26.png**

In step 3, you create a credential object for the PSDirect VM. Then you use that credential to run the Get-NetIPConfiguration information from inside the VM, with output like this:

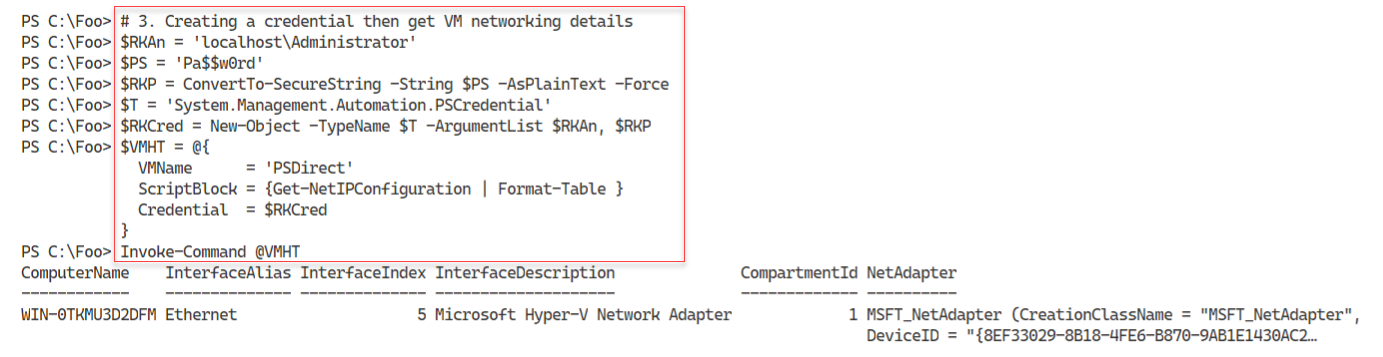


Figure 12.27: Viewing the IP configuration inside the PSDirect VM

**Insert image B42024\_12\_27.png**

In step 4, you create a new Hyper-V VM switch inside the HV1 VM host, which creates the following output:

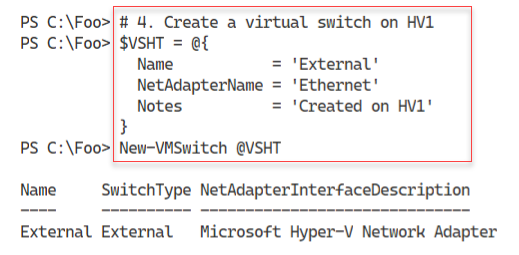


Figure 12.28: Creating a new virtual switch inside HV1

**Insert image B42024\_12\_28.png**

In step 5, you connect the NIC in the PSDirect VM to the VM switch, creating no output. In step 6, you can re-view the networking information for the PSDirect VM, with output like this:

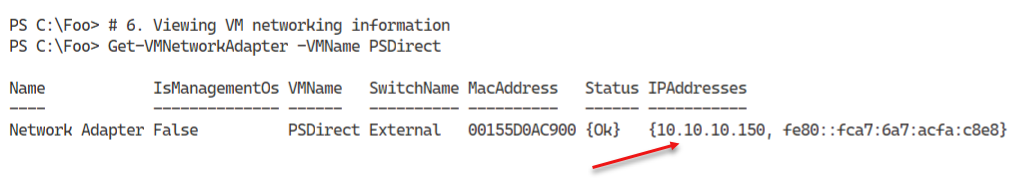


Figure 12.29: Viewing PSDirect NIC settings

**Insert image B42024\_12\_29.png**

In step 7, you execute a script block inside ther PSDirect VM to return the VM’s network details, which produces output that looks like this:

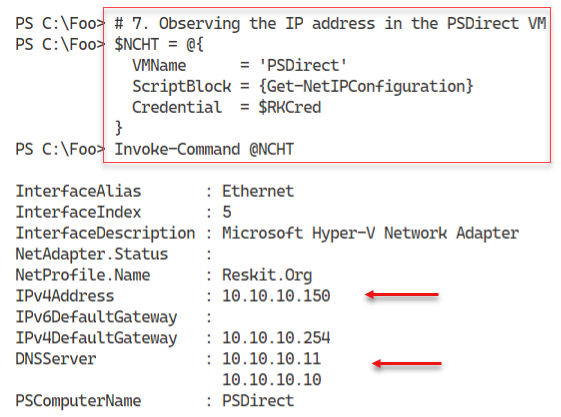


Figure 12.30: Observing PSDirect Network settings

**Insert image B42024\_12\_30.png**

In step 8, you view the existing host name of the PSDirect VM, with output that looks like this:

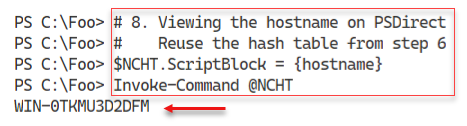


Figure 12.31: Viewing PSDirect Host Name

**Insert image B42024\_12\_31.png**

In step 9, you use the Rename-Computer cmdlet, running inside the PSDirect VM. This steop changes the host name of tgher VM to Wolf, and produces the following output:

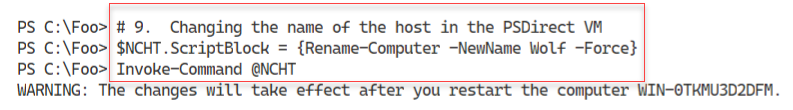


Figure 12.32: Changing the VM host name

**Insert image B42024\_12\_32.png**

In step 10, you reboot the PSDirect VM, generating no console output. After the VM has restarted, in step 11, you run the HOSTNAME command inside the VM, with output like this:

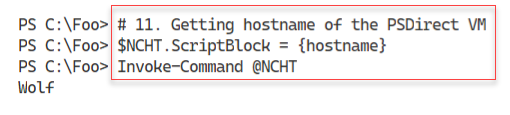


Figure 12.33: Viewing the updated hostname in the PSDirect VM

**Insert image B42024\_12\_33.png**

## There's more...

In step 1, you set the PSDirect VM’s NIC to enable MAC address spoofing. You can read more about MAC Address spoofing with Hyper-V at <https://charbelnemnom.com/how-to-set-dynamic-mac-address-on-a-hyper-v-vm-with-powershell/>.

In step 5, you connect the PSDirect VM’s NIC to the virtual switch you created on HV1. Since the PSDirect VM’s NIC is (by default) set to get IP addresses via DHCP, as soon as you connect the virtual NIC to the switch, the VM should acquire IP address configuration from the DC1 host which is running DHCP. You see the DHCP acquired IP configuration details in step 6.

# Implementing nested virtualization

Nested virtualization is a feature that enables a Hyper-V VM to host VMs, which also have virtualization enabled. You could, for example, take a Hyper-V host (say, HV1) and on that host run a VM (say PSDirect). With nested virtualization, you could enable your PSDirect VM to host VMs and then create a nested VM inside it called Nested1.

Nested VMs have a number of uses. First, nested MVs hosted in one VM are provided hardware isolation from nested VMs run in other VMs. This provides a further level of security for virtual machines. It's also useful for testing and education/training. In a training course, you could give students a single VM and enable then to create additional VMs as part of the course. And most IT pros just find it cool! You could, for example, run all the recipes in this chapter using Nested VMs.

Enabling nested Hyper-V is very simple. First, you must update the virtual CPU in the VM  
you want to support nesting. Therefore, in this recipe, you adjust the virtual CPU in the PSDirect  
VM to expose the virtualization extensions. This has to be done while the VM is turned off.  
After you restart the VM, you install the Hyper-V feature and create the Nested1 nested  
VM. This recipe does not show the details of configuring the Nested1 VM, which are left as  
an exercise for the reader

## Getting Ready

This recipe uses the HV1 Hyper-V host, with an existing Hyper-V VM, PSDirect, available. The recipe assumes you have setup PSDirect as discussed in the Creating a virtual machinerecipe earlier in this chapter

## How to do it...

## How it works...

## There's more...

# Managing VM State

Managing the VM state involves stopping and starting or pausing and resuming a VM. You  
can also save and restore a VM.

## Getting Ready

This recipe uses SRV2, a recently added workgroup host. By default, this host is a DHCP client.

## How to do it...

## How it works...

## There's more...

# Managing VM and storage movement

Hyper-V enables you to both move a VM and to move the storage for a VM to a new location. Moving a VM and moving a VM's storage are two important features you can use to manage your Hyper-V hosts.

With live migration, you can move a Hyper-V VM to a different VM host with no downtime. This works best when the VM is held on shared storage (via a fibre channel SAN, iSCSI, or SMB). You can also move a VM's storage (any VHD/VHDX associated with the VM) to a different location. You can also combine these and move a VM supported by local storage to another Hyper-V host (moving both the VM and the underlying storage).

In this recipe, you first move the storage for the PSDirect VM. You created this VM in the Creating a virtual machine recipe and stored the VM configuration and the VM's VHD on the H: drive. To move the storage, you create a new SMB share and then move the VM's storage to the new SMB share.

In the second part of this recipe, you do a live migration of the PSDirect VM from HV1 to HV2 while the VM is running. This is possible since the VM is using shared storage (that is the SMB share you create).

## Getting Ready

This recipe uses SRV2, a recently added workgroup host. By default, this host is a DHCP client.

## How to do it...

## How it works...

## There's more...

# Managing VM replication

VM replication is a disaster recovery feature withing Hype-V. It creates a replica of a VM on a remote Hyper-V Server and then keeps the replica up to date. The VM on the remote host is not active, but can be made active should the VM's host for some reason fail.

With Hyper-V replication, the source VM host bundles up any changes in a running VM's VHD file(s) and sends them to the replica server on a regular basis. The replica server then applies those changes to the dormant replica.

Once you have a replica established, you can test the replica to ensure it can start should you need that. Also, you can failover to the replica—bringing the replicated VM up based on the most recently replicated data. If the source VM host becomes inoperable before it can replicate changes on the source VM, there is a risk of those changes being lost.

In this recipe, you create and use a replica of a VM, PSDirect , that you have running on your HV1 server. The recipe sets up the replica on the HV2 server

## Getting Ready

This recipe uses SRV2, a recently added workgroup host. By default, this host is a DHCP client.

## How to do it...

## How it works...

## There's more...

# Managing VM Checkpoints

With Hyper-V in Server 2016, a checkpoint captures the state of a VM into a restore point. Hyper-V then enables you to roll back a VM to a checkpoint. Windows Server 2008's version of Hyper-V provided this feature. With Server 2008, these restore points were called snapshots.

With Server 2012, Microsoft changed the name to checkpoint. This made the terminology consistent with System Center, and avoided confusion with respect to the Volume Shadow Copy Service (VSS) snapshots used by backup systems. Whilst the Hyper-V team did change the terminology, some of the cmdlet names remain unchanged. To restore a VM to a checkpoint, you use the Restore-VMSnapShot cmdlet.

When you create a checkpoint, Hyper-V temporarily pauses the VM. It then creates a new differencing disk (AVHD). Hyper-V then resumes the VM which writes all data to the differencing disk. You can create a variety of checkpoints for a VM.

Checkpoints are great for a variety of scenarios. It can be useful for troubleshooting. Get the VM to the point where some bug is triggered, take a checkpoint. Then try a fix—if it doesn't work, you can just roll back to the checkpoint and try some other fix. Checkpoints are also useful for training. You could create a VM for a course, and create a checkpoint after each successful lab. That way, the student can make a mistake in a lab, and skip forward to a later checkpoint and carry on.

Using checkpoints in production is a different matter. In general, you should avoid using checkpoints on your production systems for a number of reasons. If your servers use any sort of replication or transaction based applications, the impact of resetting the clock to an earlier time can be bad. Since checkpoints rely on differencing disks that feature constantly growing physical disk files, the use of checkpoints can result in poor performance.

Checkpoints have their place—but should not be used as a backup strategy. In this recipe, you create a snapshot of the PSDirect vm, then you create a file inside the VM. You take a further checkpoint and create a second file. Then you revert back to the first snapshot, observing that there are no files created. Then you roll forward to the second snapshot to see that the first file is there but not the second (because you created the second file after the snapshot was taken. Then you remove all the snapshots. After each key checkpoint operation, you observe the VHDX and AVHD files which support the PSDirect VM.

## Getting Ready

This recipe uses SRV2, a recently added workgroup host. By default, this host is a DHCP client.

## How to do it...

## How it works...

## There's more...