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:

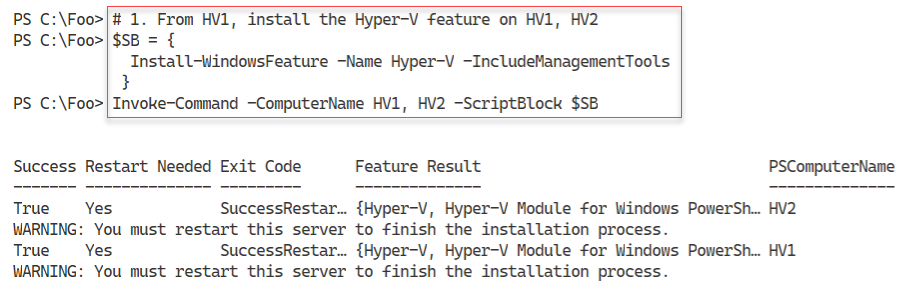


Figure 12.1: Installing the Hyper-V Feature on HV1 and HV2

**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, and in step 6, you specify the host should support NUMA spanning. Then, in step 7, you set enhanced session mode to improve VM connections. Finally, in step 8, you set the two Hyper-V hosts to make 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:

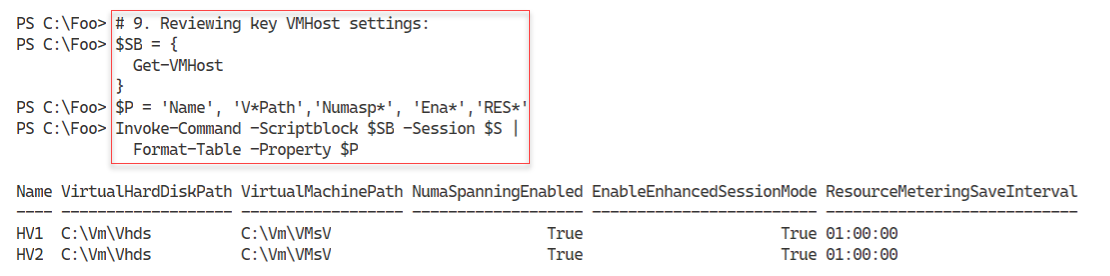


Figure 12.2: Viewing Hyper-V settings

**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 your system is capable of supporting Hyper-V, 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 that you have enabled virtualization in the VM host’s BIOS before 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 installing the Hyper-V features. You then reboot the host in a controlled way in a later step. 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 Hyper-V server 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 VM's DVD drive contents. Once you have created your VM, you need to install the VM's operating system. You have numerous options regarding how you deploy Windows (or Linux) in a Hyper-V VM.

The Windows Assessment and Deployment Kit, a free product from Microsoft, contains various 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 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 you complete the OS installation, 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 hostname which you later change to Tiger. In building the VM, you assign the Windows Server 2022 DVD to the VM's DVD drive. This step ensures that, when you start the VM, Windows commences the GUI setup process. The result should be a fully installed OS inside the VM. The details of performing the actual installation are outside the scope of this recipe.

Two minor issues using the GUI to install Windows Server 2022 are that Windows randomly generates the machine name, 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 deploying Windows Server 2022 in a more automated fashion using a SETUP.XML file that specifies the installation's details. 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 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 complete retail edition – and for this chapter, you could use an image of Windows Server 2022 or even Windows Server 2019. You can also download an ISO of the latest preview copy of Windows Server and use ISO when installing the VM. See https://www.microsoft.com/en-us/software-download/windowsinsiderpreviewserver for preview build downloads, and https://techcommunity.microsoft.com/t5/windows-server-insiders/bd-p/WindowsServerInsiders for more information about Windows Insiders builds.

## 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 VMs and VM hard drives' locations 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:

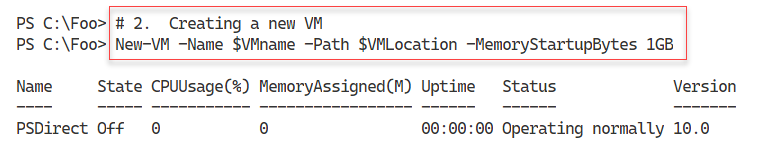


Figure 12.3: Creating a new Hyper-V VM

**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;

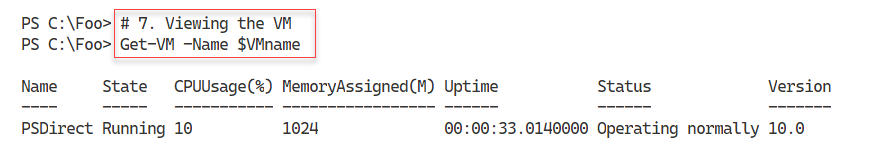


Figure 12.4: Viewing the new Hyper-V VM

**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 it to match the ISO file's name. For released versions of Windows server, you can find ISO images for evaluation versions here https://www.microsoft.com/en-us/evalcenter/evaluate-windows-server. If you like living in the fast lane, consider joining the Windows Insiders project and using the latest preview build – see https://www.microsoft.com/en-us/software-download/windowsinsiderpreviewserver. Note that Windows Server Insiders builds are free. However, you do need to sign up first.

In step 2, you create a VM. Although that step should succeed, you could not run this VM yet, 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 runs and completes Windows Server's installation. To complete installing Windows Server, you need to use the GUI. For this chapter's purposes, accept all the installation defaults and ensure you set the password to 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:

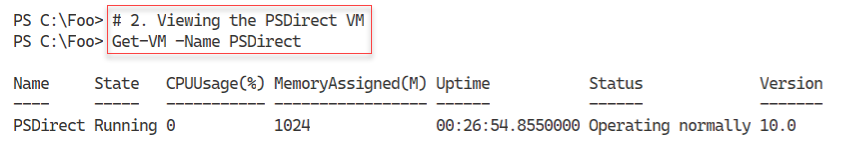


Figure 12.5: Viewing the PSDirect Hyper-V VM

**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:

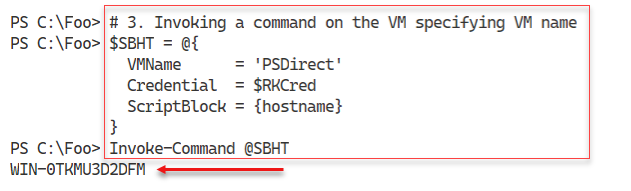


Figure 12.6: Checking hostname of PSDirect VM

**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:

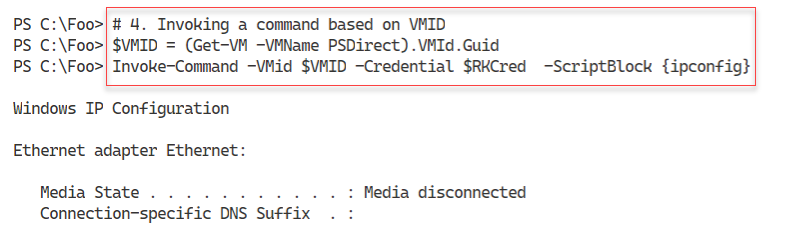


Figure 12.7: Checking hostname of PSDirect VM by VM guid

**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:

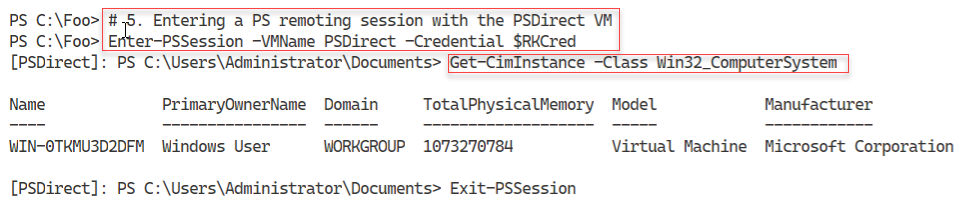


Figure 12.8: Checking hostname of PSDirect VM by VM guid

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

Now that you have created a working VM in HV1, you run the remainder of the VM setup steps 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:

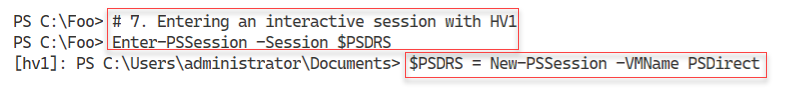


Figure 12.9: Using the remoting session on HV`

**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:

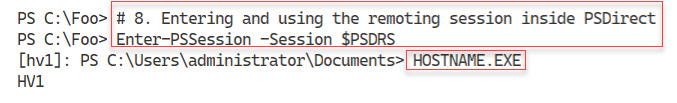


Figure 12.10: Running HOSTNAME.EXT in session to HV1

**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:

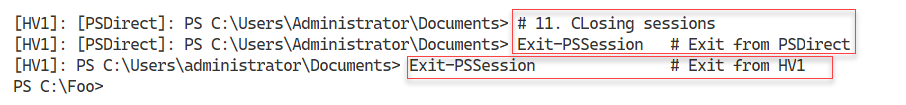


Figure 12.12: Exiting from remoting sessions

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

## There's more...

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

With step 8 and step 9, you first create 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).

# Using Hyper-V VM Groups

Hyper-Vs VM groups allow you to group VMs for 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.

With VN groups, you could 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 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. In production, you might just consider using simple arrays of VM names (for example, $ACCTSQVMS to hold the names of the accounting VMs)

## 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 with the output of this step looking like this:

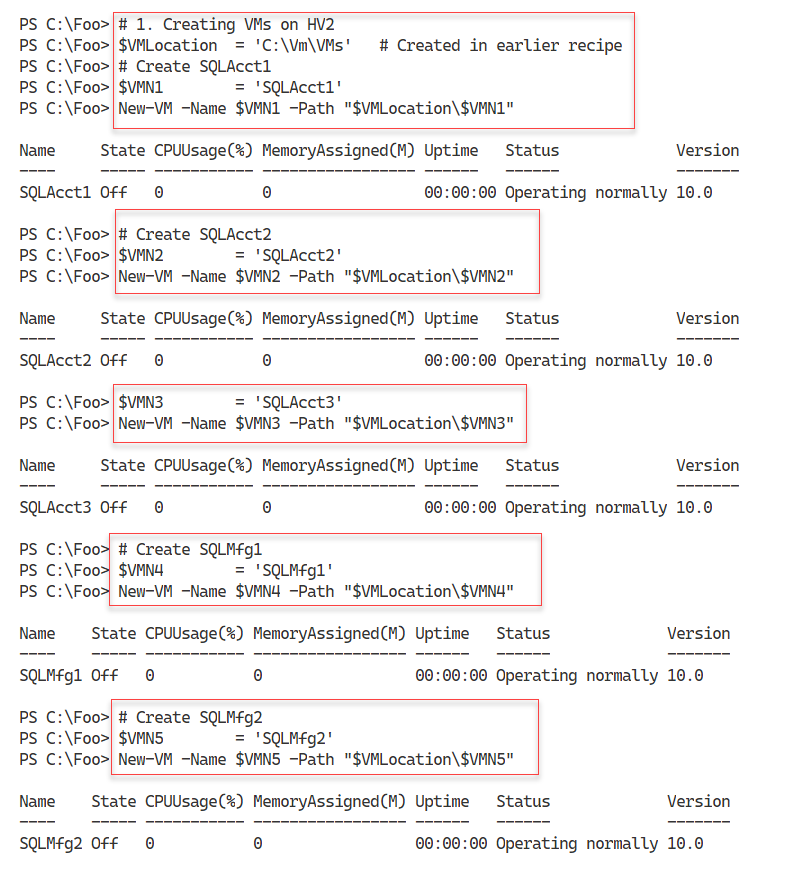


Figure 12.13: Creating VMs for this recipe

**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:

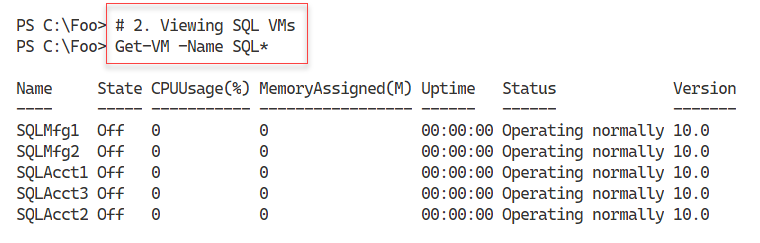


Figure 12.14: Viewing SQL VMs

**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:

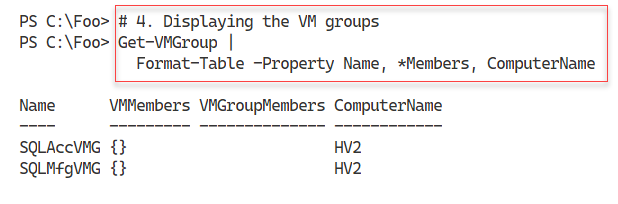


Figure 12.15: Viewing VM Groups on HV2

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

To simplify VM collection groups' creation, 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:

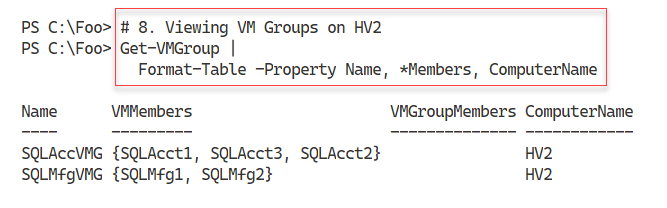


Figure 12.16: Viewing VM Groups on HV2

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

In step 9, you 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:

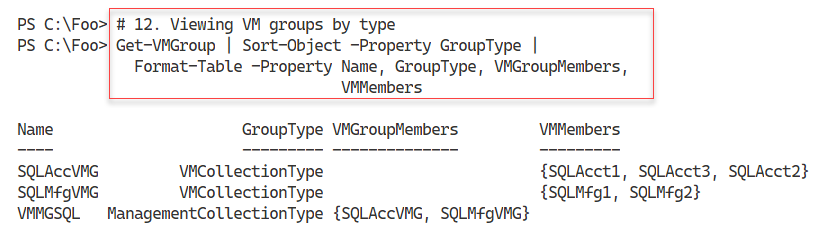


Figure 12.17: Viewing fully populated VM Groups on HV2

**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 processors, 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:

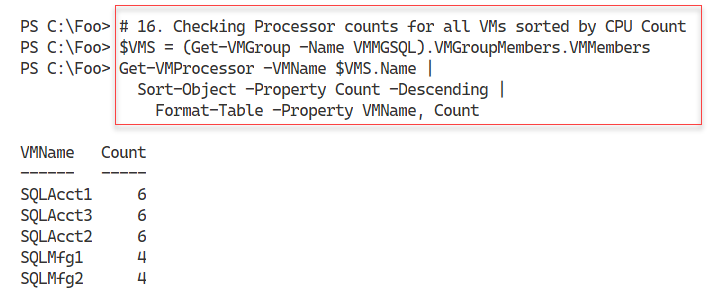


Figure 12.18: Viewing virtual processors assigned to each SQL VM

**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 use at most 4 members of each 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.

Like most physical servers, you can not change all of these virtual components while your virtual server is up and 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 Configure VM networking in the Configuring Hyper-V networking recipe later in this chapter.

## 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 SCSI controller 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 SCSI controller

$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:

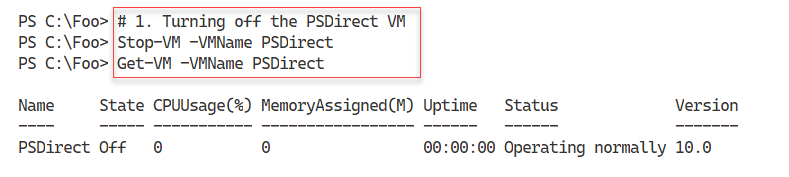


Figure 12.19: Stopping the PSDirect VM

**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:

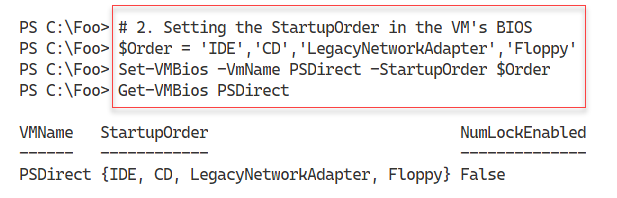


Figure 12.20: Updating and viewing the VM’s BIOS

**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:

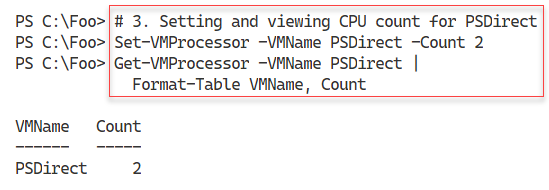


Figure 12.21: Updating and viewing the VM’s BIOS

**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:

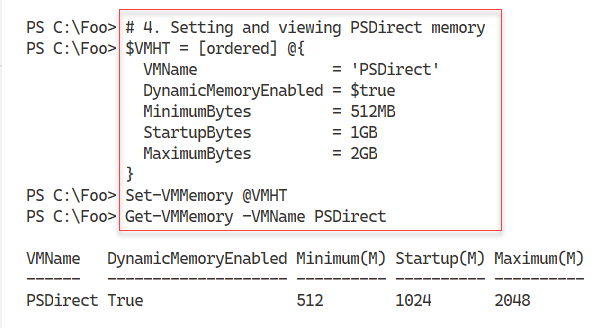


Figure 12.22: Changing and viewing VM memory

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

In step 5, you add a SCSI controller to the PSDirect VM, and when you then view the SCSI controllers inside the PSDirect VM, you should see output like this:

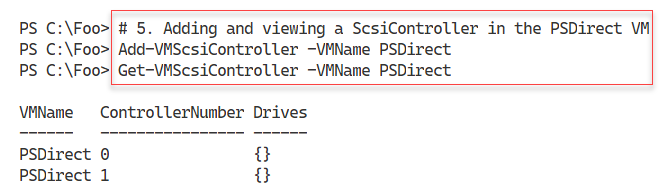


Figure 12.23: Adding and viewing SCSI virtual disk controllers

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

Now that you have adjusted the virtual hardware for the PSDirect VM, in step 6, you restart the VM and 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:

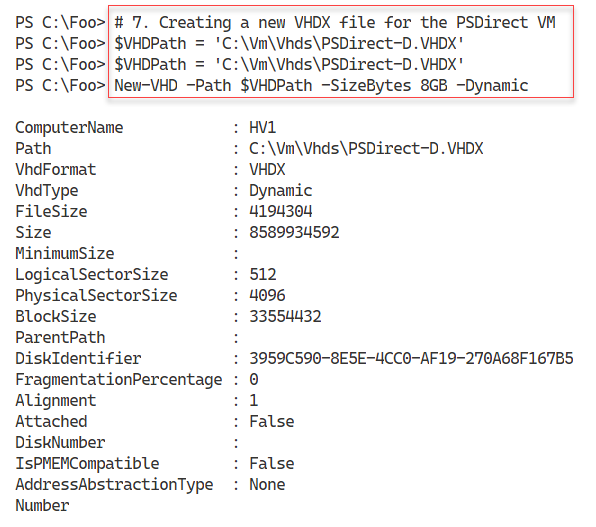


Figure 12.24: Creating a new virtual disk drive

**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:

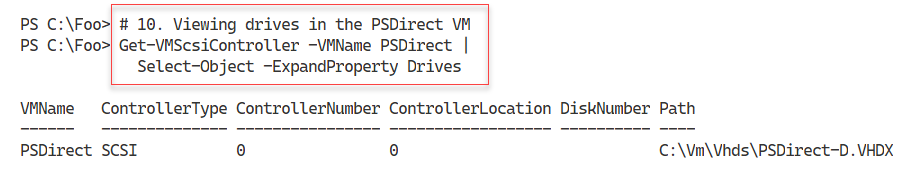


Figure 12.25: Viewing the SCSI disk drives in the PSDirect VM

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

## There's more...

With Hyper-V, you can only update some VM hardware configuration settings when you have turned off the VM. In step 1, you switch the VM to enable you to make virtual BIOS settings. Once you have made these BIOS (and possibly other) changes, you can restart 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 this recipe on HV1, which uses the PSDirect VM you created in the Creating a virtual machine recipe. This recipe also makes use of a DHCP server running on DC1. You set the DHCP service up in the Installing and authorizing a DHCP Server recipe. Then you configured the DHCP server in the Configure DHCP scopes recipe.

This chapter uses the PSDirect VM you created earlier. When you built this machine using the normal setup routine, Windows assigns a random machine 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 PSDirect VM’s networking card to enable MAC address spoofing. This step enables the VM to see the network and 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. You should also enable MAC spoofing on HV2.

## 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:

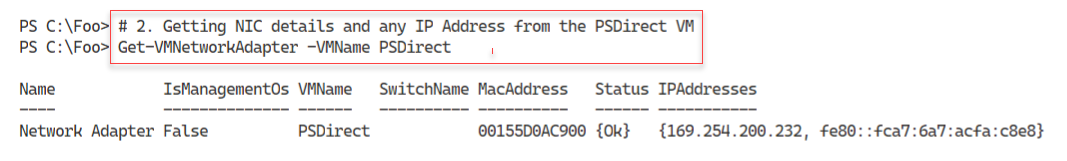


Figure 12.26: Viewing the NIC details for the PSDirect VM

**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:

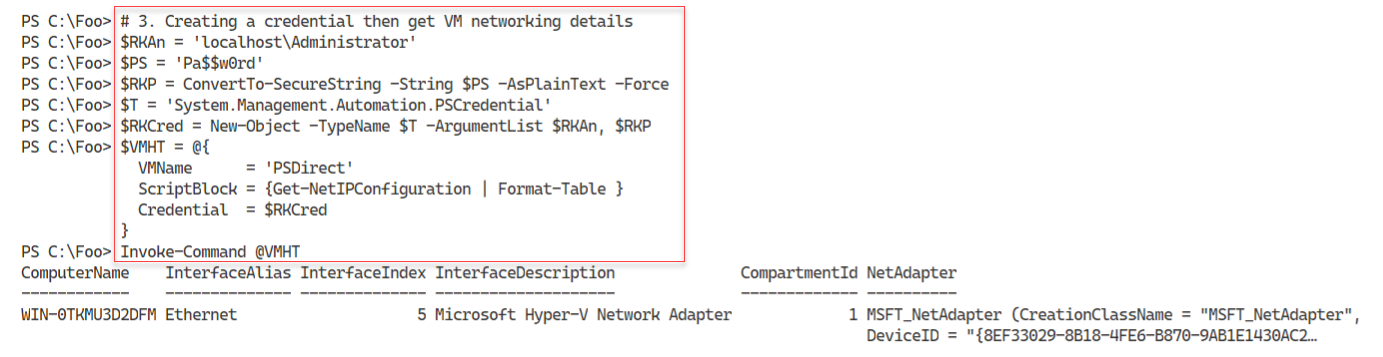


Figure 12.27: Viewing the IP configuration inside the PSDirect VM

**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:

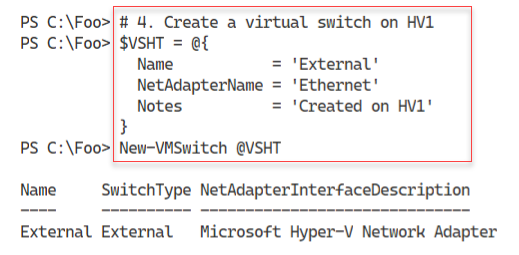


Figure 12.28: Creating a new virtual switch inside HV1

**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:

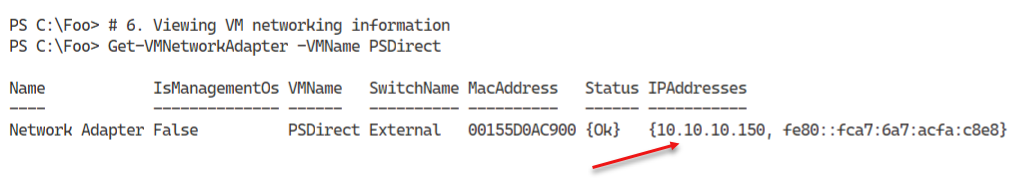


Figure 12.29: Viewing PSDirect NIC settings

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

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

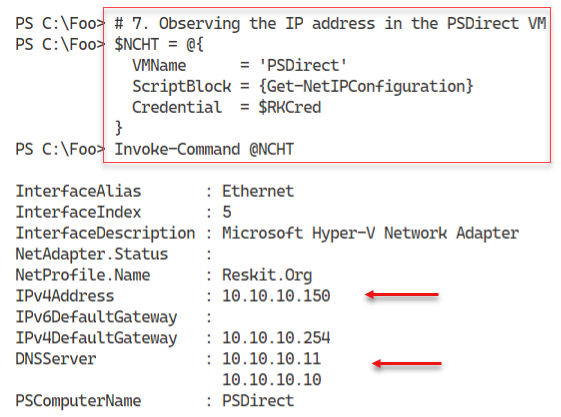


Figure 12.30: Observing PSDirect Network settings

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

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

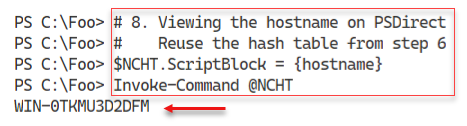


Figure 12.31: Viewing PSDirect Host Name

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

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

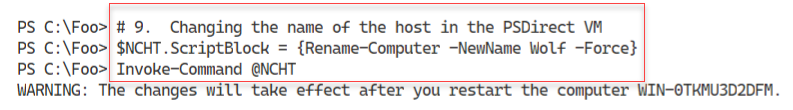


Figure 12.32: Changing the VM hostname

**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:

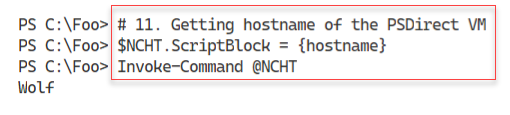


Figure 12.33: Viewing the updated hostname in the PSDirect VM

**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 many uses. First, nested MVs hosted in one VM are provided hardware isolation from nested VMs run in other VMs. In this case, nested virtualization provides a further level of security for virtual machines.

Nested virtualization is also helpful for testing and education/training. You could give students a single VM (on a large blade server, for example). Nested virtualization enables them to create additional VMs as part of the course lab work. And most IT pros just find it cool! You can, for example, run all the recipes in this chapter using nested virtualization.

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. Changing the BIOS to do this must be done after you turn off the VM. 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. These details are 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...

1. Stopping the PSDirect VM

Stop-VM -VMName PSDirect

1. Setting the VM's processor to support virtualization

$VMHT = @{

  VMName                         = ‘PSDirect’

  ExposeVirtualizationExtensions = $true

}

Set-VMProcessor @VMHT

Get-VMProcessor -VMName PSDirect |

  Format-Table -Property Name, Count,

                         ExposeVirtualizationExtensions

1. Starting the PSDirect VM

Start-VM -VMName PSDirect

Wait-VM  -VMName PSDirect -For Heartbeat

Get-VM   -VMName PSDirect

1. Creating credentials for PSDirect

$User = 'Wolf\Administrator'

$PHT = @{

  String      = 'Pa$$w0rd'

  AsPlainText = $true

  Force       = $true

}

$PSS  = ConvertTo-SecureString @PHT

$Type = 'System.Management.Automation.PSCredential'

$CredRK = New-Object -TypeName $Type -ArgumentList $User, $PSS

1. Creating a script block for remote execution

$SB = {

  Install-WindowsFeature -Name Hyper-V -IncludeManagementTools

}

1. Creating a remoting session to PSDirect

$Session = New-PSSession -VMName PSDirect -Credential $CredRK

1. Installing Hyper-V inside PSDirect

$IHT = @{

  Session     = $Session

  ScriptBlock = $SB

}

Invoke-Command @IHT

1. Restarting the VM to finish adding Hyper-V to PSDirect

Stop-VM  -VMName PSDirect

Start-VM -VMName PSDirect

Wait-VM  -VMName PSDirect -For IPAddress

Get-VM   -VMName PSDirect

1. Creating a nested VM inside the PSDirect VM

$SB2 = {

        $VMname = 'NestedVM'

        New-VM -Name $VMname -MemoryStartupBytes 1GB | Out-Null

        Get-VM

}

$IHT2 = @{

  VMName = 'PSDirect'

  ScriptBlock = $SB2

}

Invoke-Command @IHT2 -Credential $CredRK

## How it works...

In step 1, you ensure the PSDirect VM has stopped, creating no console output. In step 2, you adjust the VM’s virtual processor to support virtualization. Then you review the processor settings, which looks like this:

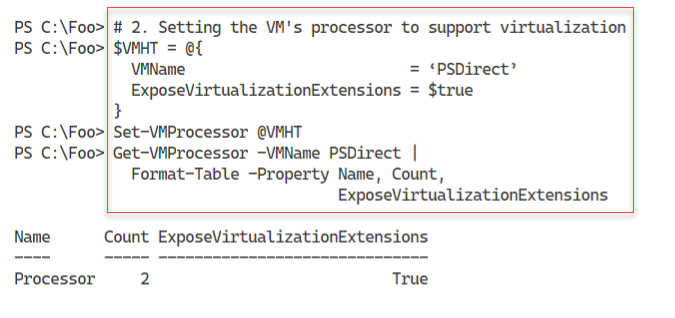


Figure 12.34: Viewing the virtual processor inside PSDirect

**Insert image B42024\_12\_34.png**

In step 3, you start the PSDirect VM then view it using Get-VM, which looks like this:

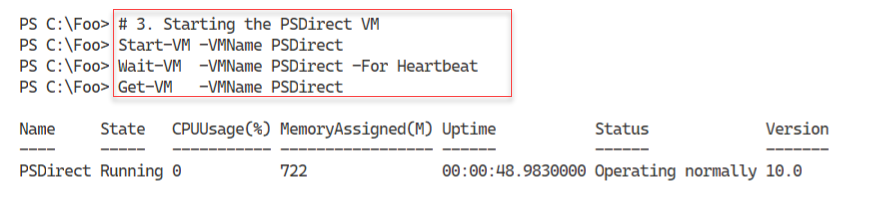


Figure 12.35: Starting and viewing PSDirect

**Insert image B42024\_12\_35.png**

In step 4, you create a PowerShell credential object for the PSDirect VM. Then in step 5, you create a remote session to the PSDirect VM. These two steps produce no output to the console.

In step 7, you use the remoting session to install the Hyper-V feature inside the PSDirect VM, producing output like this:

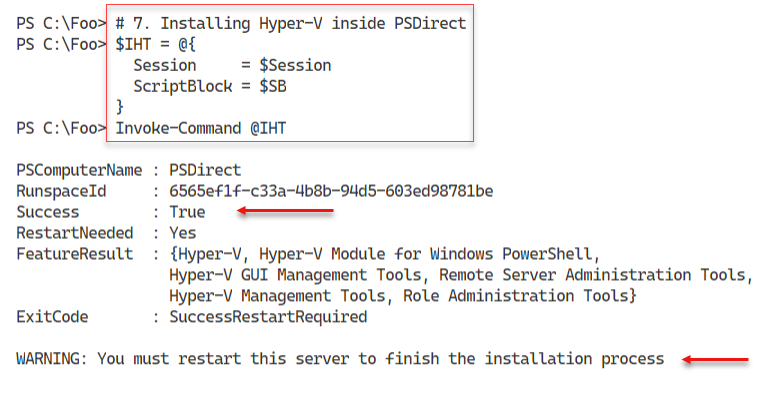


Figure 12.36: Installing Hyper-V inside the PSDirect VM

**Insert image B42024\_12\_36.png**

As you can see from the output, after installing the Hyper-V feature inside PSDirect, you need to reboot the VM to complete the installation process. In step 8, you restart PSDirect, wait for it to start, and view the VM details with output like this:

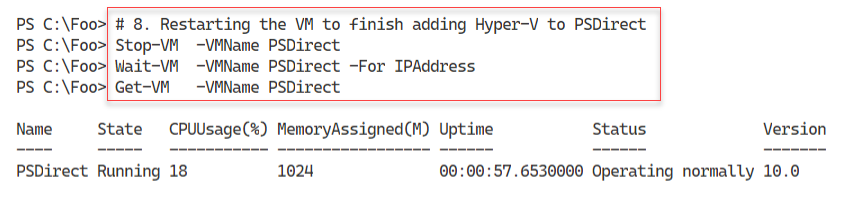


Figure 12.37: Restarting the PSDirect VM

**Insert image B42024\_12\_37.png**

In the final step in this recipe, step 9, you create a new nested VM, NestedVM, inside the PSDirect VM, which produces console output like this:

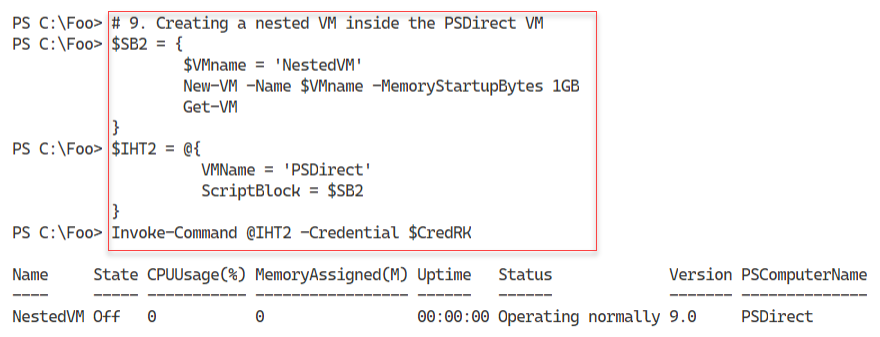


Figure 12.38: Creating a nested VM inside PSDirect

**Insert image B42024\_12\_38.png**

## There's more...

In step 7, you install the Hyper-V feature in the nested VM (NestedVM) inside the PSDirect VM. The installation of Hyper-V inside the PSDirect VM is successful, as you can see in this step's output. Had the PSDirect VM not supported virtualization, the Install-WindowsFeature cmdlet would have thrown an error. Likewise, had you not installed Hyper-V in PSDirect, step 9 would have failed. If you choose, you can adjust the steps from the Creating a Hyper-V VM recipe to install and configure an OS in the NestedVM.

# Managing VM State

Hyper-V provides you with the ability to start, stop, and pause a Hyper-V VM. You can also save and restore a VM. You use the Hyper-V cmdlets to manage your VMs either locally (that is, on the Hyper-V host in an RDP or PowerShell remoting session) or use RSAT tools to manage the state of VMs on remote Hyper-V hosts.

You can start and stop VMs either directly or via the task scheduler. You might want to start up a few VMs every working morning and stop them up each evening. If you have provisioned your Hyper-V host with spinning disks, starting multiple VMs at once stresses the storage subsystem, especially if you are using any form of RAID on the disk drives you use to hold your virtual disks. Depending on your hardware, you can some times hear the IO Blender effect starting up a small VM farm. And even with solid-state disks, starting several VMs at once puts a considerable load on the Windows storage system. In such cases, you might pause a VM and let others start faster.

Saving a VM can be helpful to avoid a long startup process. If you have created multiple VMs to test out the recipes in this book, you might save VMs and restart them as you move through the chapters.

## Getting Ready

You run this recipe on the HV1 host after you have created the PSDirect VM. You created that VM in the Creating a virtual machine recipe.

## How to do it...

1. Getting the VM's state to check if it is off

Stop-VM -Name PSDirect -WarningAction SilentlyContinue

Get-VM -Name PSDirect

1. Starting the VM

Start-VM -VMName PSDirect

Wait-VM -VMName PSDirect -For IPAddress

Get-VM -VMName PSDirect

1. Suspending and viewing the PSDirect VM

Suspend-VM -VMName PSDirect

Get-VM -VMName PSDirect

1. Resuming the PSDirect VM

Resume-VM -VMName PSDirect

Get-VM -VMName PSDirect

1. Saving the VM

Save-VM -VMName PSDirect

Get-VM -VMName PSDirect

1. Resuming the saved VM and viewing the status

Start-VM -VMName PSDirect

Get-VM -VMName PSDirect

1. Restarting the PSDirect VM

Restart-VM -VMName PSDirect -Force

Get-VM     -VMName PSDirect

1. Waiting for the PSDirect VM to get an IP address

Wait-VM    -VMName PSDirect -For IPaddress

Get-VM     -VMName PSDirect

1. Performing a hard power off on the PSDirct VM

Stop-VM -VMName PSDirect -TurnOff

Get-VM  -VMname PSDirect

## How it works...

In step 1, you stop the PSDirect VM and check its status. The output looks like this;

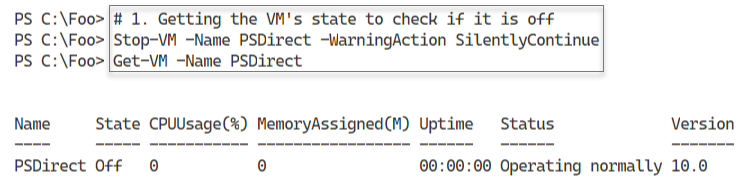


Figure 12.39: Stopping the PSDirect VM

**Insert image B42024\_12\_39.png**

In step 2, you use Start-VM to start the PSDirect VM, then use Wait-VM to wait for the VM to start up and get an IP address. The output from this step looks like this:

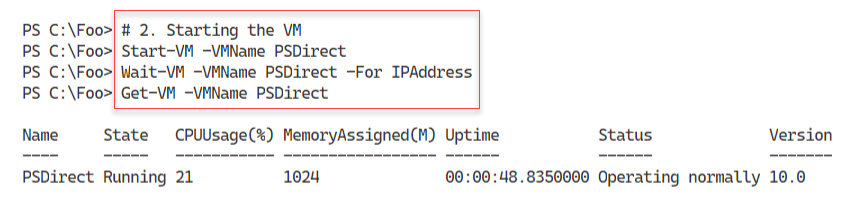


Figure 12.40: Starting the PSDirect VM

**Insert image B42024\_12\_40.png**

In step 3, you use Suspend-VM to pause the PSDirect VM and observe its status, with the following output:

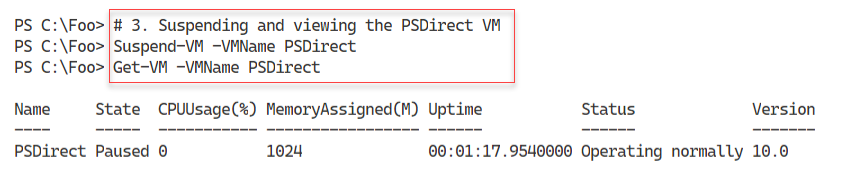


Figure 12.41: Pausing the PSDirect VM

**Insert image B42024\_12\_41.png**

Having paused the PSDirect VM, in step 4, you use the Resume-VM cmdlet to un-pause the VM, with the following output:

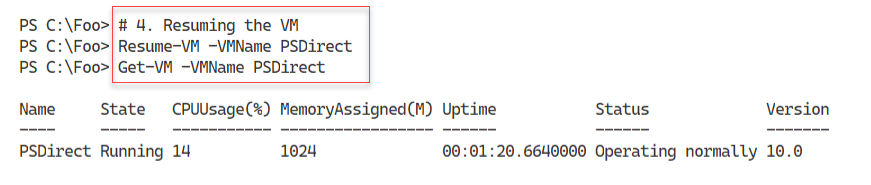


Figure 12.42: Resuming the PSDirect VM

**Insert image B42024\_12\_42.png**

In step 5, you save the PSDirect VM, with output like this:

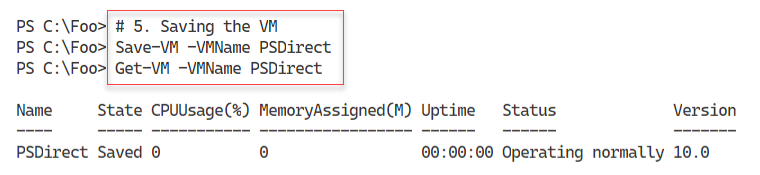


Figure 12.43: Saving the PSDirect VM

**Insert image B42024\_12\_43.png**

You can resume a saved VM by using the Start-VM command, as you see in step 6, which creates output like this:

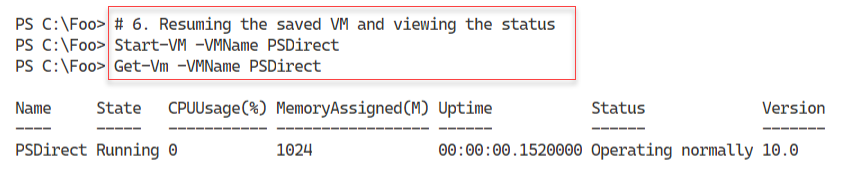


Figure 12.44: Resuming the PSDirect VM (after you saved it)

**Insert image B42024\_12\_44.png**

Step 7 demonstrates using the Restart-VM cmdlet to restart the PSDirect VM. The output from this step looks like this:

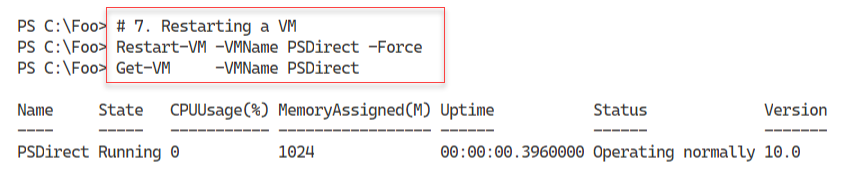


Figure 12.45: Restarting the PSDirect VM

**Insert image B42024\_12\_45.png**

In the previous step, you restarted the VM, which can take some time, depending on your VM host and VM size. In step 8, you wait for the VM to have restarted, then use Get-VM to observe the status of the PSDirect VM, like this:

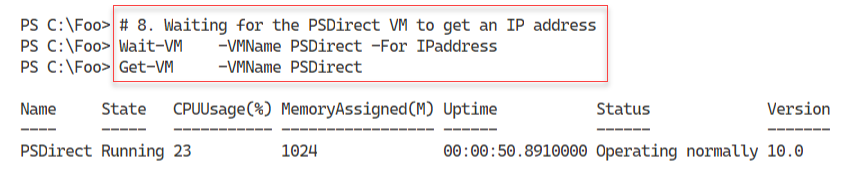


Figure 12.46: Waiting for the PSDirect VM to restart

**Insert image B42024\_12\_46.png**

In the final step in this recipe, step 9, you perform a hard power-off of the PSDirect VM, with output like this:

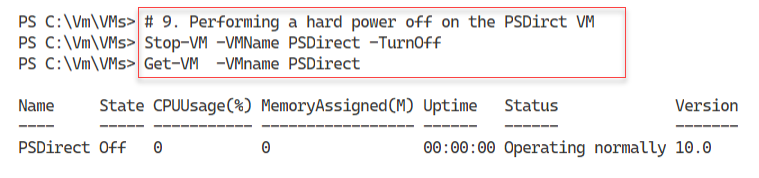


Figure 12.47: Turning off the PSDirect VM

**Insert image B42024\_12\_47.png**

## There's more...

Managing the state of VMs is straightforward with the cmdlets in the Hyper-V module.

# Managing VM and storage movement

Hyper-V enables you to move a VM to a new VM host and move a VM’s storage 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. Storage migration 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 recipe Creating a virtual machine 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. You perform the migration while the VM is running. This VM movement is known as live migration – you are migrating a live VM that stays up; and running during the migration.

## Getting Ready

In this recipe, you use the HV1 and HV2 systems (Windows 2022 Server with Hype-V loaded) as setup in the Installing and configuring Hyper-V recipe and the PSDirect VM created in the recipe Creating a virtual machine.

In the first part of this recipe, you first move the storage for the PSDirect VM. You created this VM in the Creating a virtual machine recipe.

You run all the first part of this recipe on HV1 to move the PSDirect VM from HV1 to HV2. Once you have moved the VM to HV2, you run the final part of this recipe on HV2 to move the VM back to HV1. If you attempted to run the last part of this recipe remotely, for example, for a client machine or from HV1 (suitably wrapped up in a script block you run with Invoke-Command), you would see a Kerberos error due to the Kerberos double hop problem. To overcome that issue, you could have deployed Windows Credential Security System Provider (CredSSP).

## How to do it...

1. Viewing the PSDirect VM on HV1 and verifying that it is turned on and running

Get-VM -Name PSDirect -Computer HV1

1. Getting the VM configuration location

(Get-VM -Name PSDirect).ConfigurationLocation

1. Getting virtual hard drive locations

Get-VMHardDiskDrive -VMName PSDirect |

  Format-Table -Property VMName, ControllerType, Path

1. Moving the VM's to the C\PSDirectNew folder

$MHT = @{

  Name                   = 'PSDirect'

  DestinationStoragePath = 'C:\PSDirectNew'

}

Move-VMStorage @MHT

1. Viewing the configuration details after moving the VM's storage

(Get-VM -Name PSDirect).ConfigurationLocation

Get-VMHardDiskDrive -VMName PSDirect |

  Format-Table -Property VMName, ControllerType, Path

1. Getting the VM details for VMs from HV2

Get-VM -ComputerName HV2

1. Creating External virtual switch on HV2

$SB = {

  $NSHT = @{

    Name           = 'External'

    NetAdapterName = 'Ethernet'

    ALLOWmAnagementOS = $true

  }

  New-VMSwitch @NSHT

}

Invoke-Command -ScriptBlock $SB -ComputerName HV2

1. Enabling VM migration from both HV1 and HV2

Enable-VMMigration -ComputerName HV1, HV2

1. Configuring VM Migration on both hosts

$SVHT = @{

  UseAnyNetworkForMigration                 = $true

  ComputerName                              = 'HV1', 'HV2'

  VirtualMachineMigrationAuthenticationType =  'Kerberos'

  VirtualMachineMigrationPerformanceOption  = 'Compression'

}

Set-VMHost @SVHT

1. Moving the PSDirect VM to HV2

$Start = Get-Date

$VMHT = @{

  Name                   = 'PSDirect'

  ComputerName           = 'HV1'

  DestinationHost        = 'HV2'

  IncludeStorage         =  $true

  DestinationStoragePath = 'C:\PSDirect' # on HV2

}

Move-VM @VMHT

$Finish = Get-Date

($Finish - $Start)

1. Displaying the time taken to migrate

$OS = "Migration took: [{0:n2}] minutes"

($OS -f ($($Finish-$Start).TotalMinutes))

1. Checking the VMs on HV1

Get-VM -ComputerName HV1

1. Checking the VMs on HV2

Get-VM -ComputerName HV2

1. Looking at the details of the PSDirect VM on HV2

((Get-VM -Name PSDirect -Computer HV2).ConfigurationLocation)

Get-VMHardDiskDrive -VMName PSDirect -Computer HV2  |

  Format-Table -Property VMName, Path

Run the remainder of this recipe on HV2 directly.

1. Moving the PSDirect VM back to HV1

$Start2 = Get-Date

$VMHT2 = @{

    Name                  = 'PSDirect'

    ComputerName           = 'HV2'

    DestinationHost        = 'HV1'

    IncludeStorage         =  $true

    DestinationStoragePath = 'C:\vm\vhds\PSDirect' # on HV1

}

Move-VM @VMHT2

$Finish2 = Get-Date

1. Displaying the time taken to migrate back to HV1

$OS = "Migration back to HV1 took: [{0:n2}] minutes"

($OS -f ($($Finish-$Start).TotalMinutes))

## How it works...

In step 1, you check the status of the PSDirect VM to ensure it is running. The output should look like this:

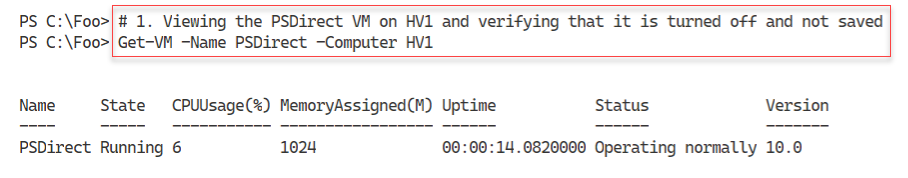


Figure 12.48: Checking the status of the PSDirect VM

**Insert image B42024\_12\_48.png**

In step 2, you determine the location Hyper-V is using to store VM configuration for the PSDirect VM, with console output like this:

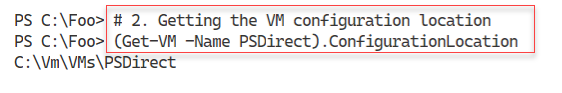


Figure 12.49: Checking the status of the PSDirect VM

**Insert image B42024\_12\_49.png**

In step 3, you view the locations Hyper-V is using to store the virtual hard disks for the PSDirect VM, with output like this:

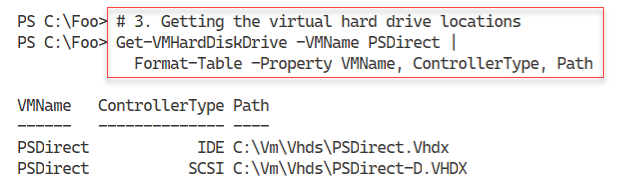


Figure 12.50: Viewing PSDirect’s virtual hard drives

**Insert image B42024\_12\_50.png**

In step 4, you move the storage of the running PSDirect VM. This step generates no output. After moving the VM’s storage, in step 5, you re-view the VM configuration location and the details of the hard drives for the PSDirect VM. The output of this step looks like this:

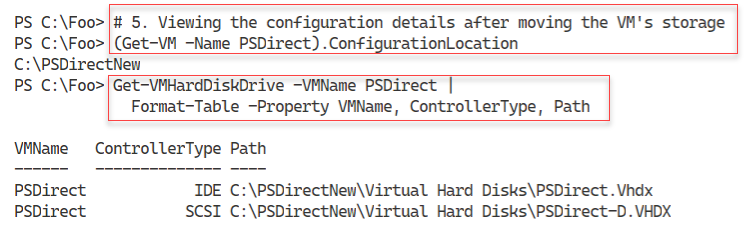


Figure 12.50: Viewing PSDirect’s virtual hard drives

**Insert image B42024\_12\_50.png**

In step 6, you check to see which VMs are available on HV2, with output like this:

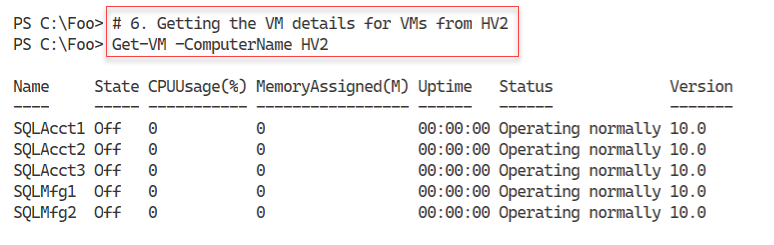


Figure 12.51: Viewing VMs on HV2

**Insert image B42024\_12\_51.png**

In step 7, you create a new External switch on HV2 to enable networking on HV2. The output you see is as follows:

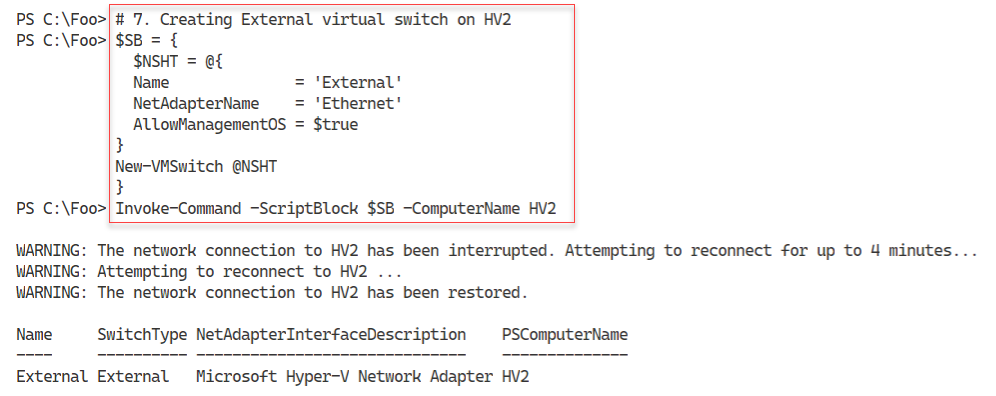


Figure 12.52: Creating the External virtual switch on HV2

**Insert image B42024\_12\_52.png**

In step 8, you enable VM migration on both HV1 and HV2. In step 9, you configure Hyper-V VM migration on both servers. Next, in step 10, you perform a live migration of the PSDirect VM to HV2. These three steps produce no output.

In step 11, you display how long it took Hyper-V to migrate the PSDirect VM, with output like this:

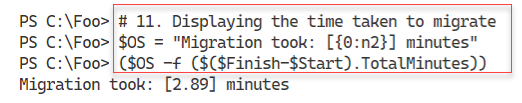


Figure 12.53: Displaying the VM migration time

**Insert image B42024\_12\_53.png**

In step 12, you use the Get-VM cmdlet to view the VMs on HV1. Since you migrated the PSDirect VM to HV2, there are no VMs on HV2; thus, there is no console output from this step. You then, in step 13, view the VMs on HV2, which produces output like this:

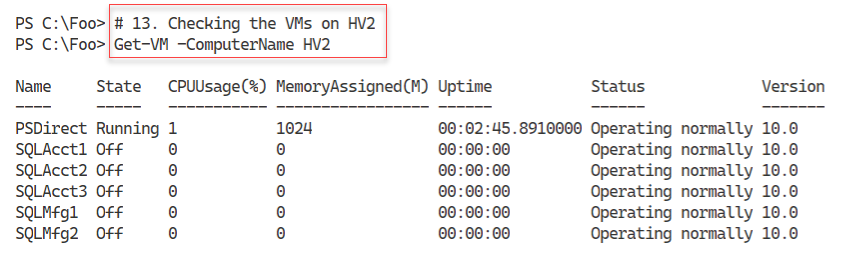


Figure 12.54: Displaying VMs on HV2

**Insert image B42024\_12\_54.png**

In step 14, you examine the details of where Hyper-V has stored the VM configuration and virtual hard drives for the PSDirect VM. The output of this step looks like this:

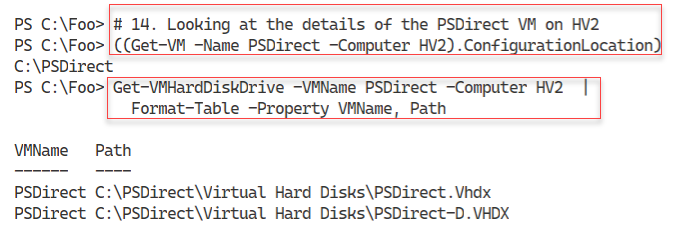


Figure 12.55: Viewing PSDirect VM details on HV2

**Insert image B42024\_12\_55.png**

In step 15 (which you run from HV2), you migrate the PSDirect VM back to HV1, producing no console output. In the final step in this recipe, step 16, you view how long it took Hyper-V to migrate the VM back to HV1, with output like this:

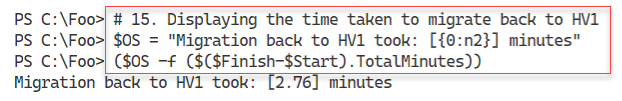


Figure 12.56: Viewing PSDirect migration time

**Insert image B42024\_12\_56.png**

## There's more...

In step 1, you check the status of the PSDirect VM to ensure it is running. If the output you see shows the VM is not running, you should use the Start-VM cmdlet to start it before proceeding with this recipe.

In step 6, you used Get-VM to view the VMs defined on HV2. The output shows the VMs created in the earlier recipe “Using VM Groups”.

In step 13, you view the VMs on HV2, which now shows the PSDirect VM. Note that the VM continues to be running since you performed a “live migration”. Consider opening an RDP or VM Connect session to the PSDirect VM and view the VM as you carry out the migration. When the migration is complete, you may need to re-login to your VM session.

# Managing VM replication

VM replication is a disaster recovery feature within Hyper-V. Hyper-V creates a VM replica on a remote Hyper-V Server and then keeps that replica up to date as the original VM changes. The VM on the remote host is not active during the normal operation of the original VM. You can make the replica active should the VM's host, for some reason, fail.

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

Once you have a VM 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 the replication process losing those changes.

In this recipe, you create and use a replica of the PSDirect VM, which you have used in previous recipes in this chapter.

## Getting Ready

This recipe creates a Hyper-V VM replica of the PSDirect VM running in HV1 to HV2. You created this VM in the recipe Creating a virtual machine. You also use this VM in the Configuring VM and storage movementrecipe.

## How to do it...

1. Configuring HV1 and HV2 to be trusted for delegation in AD on DC1

$SB = {

  Set-ADComputer -Identity HV1 -TrustedForDelegation $True

  Set-ADComputer -Identity HV2 -TrustedForDelegation $True

}

Invoke-Command -ComputerName DC1 -ScriptBlock $SB

1. Rebooting the HV1 and HV2 hosts

Restart-Computer -ComputerName HV2 -Force

Restart-Computer -ComputerName HV1 -Force

1. Configuring Hyper-V replication on HV1 and HV1

$VMRHT = @{

  ReplicationEnabled              = $true

  AllowedAuthenticationType       = 'Kerberos'

  KerberosAuthenticationPort      = 42000

  DefaultStorageLocation          = 'C:\Replicas'

  ReplicationAllowedFromAnyServer = $true

  ComputerName                    = 'HV1.Reskit.Org',

                                    'HV2.Reskit.Org'

}

Set-VMReplicationServer @VMRHT

1. Enabling PSDirect on HV1 to be a replica source

$VMRHT = @{

  VMName             = 'PSDirect'

  Computer           = 'HV1'

  ReplicaServerName  = 'HV2'

  ReplicaServerPort  = 42000

  AuthenticationType = 'Kerberos'

  CompressionEnabled = $true

  RecoveryHistory    = 5

}

Enable-VMReplication  @VMRHT

1. Viewing the replication status of HV1

Get-VMReplicationServer -ComputerName HV1

1. Checking PSDirect on Hyper-V Hosts

Get-VM -ComputerName HV1 -VMName PSDirect

Get-VM -ComputerName HV2 -VMName PSDirect

1. Starting the initial replication

Start-VMInitialReplication -VMName PSDirect -ComputerName HV1

1. Examining the initial replication state on HV1 just after you start the initial replication

Measure-VMReplication -ComputerName HV1

1. Examining the replication status on HV1 after replication completes

Measure-VMReplication -ComputerName HV1

1. Testing PSDirect failover to HV2

$SB = {

  $VM = Start-VMFailover -AsTest -VMName PSDirect -Confirm:$false

  Start-VM $VM

}

Invoke-Command -ComputerName HV2 -ScriptBlock $SB

1. Viewing the status of PSDirect VMs on HV2

Get-VM -ComputerName HV2 -VMName PSDirect\*

1. Stopping the failover test

$SB = {

  Stop-VMFailover -VMName PSDirect

}

Invoke-Command -ComputerName HV2 -ScriptBlock $SB

1. Viewing the status of VMs on HV1 and HV2 after failover stopped

Get-VM -ComputerName HV1

Get-VM -ComputerName HV2

1. Stopping PSDirect on HV1 before performing a planned failover

Stop-VM PSDirect -ComputerName HV1

1. Starting VM failover from HV1 to HV2

Start-VMFailover -VMName PSDirect -ComputerName HV2 -Confirm:$false

1. Completing the failover

$CHT = @{

  VMName       = 'PSDirect'

  ComputerName = 'HV2'

  Confirm      = $false

}

Complete-VMFailover @CHT

1. Starting the replicated VM on HV2

Start-VM -VMname PSDirect -ComputerName HV2

Set-VMReplication -VMname PSDirect -reverse -ComputerName HV2

1. Checking the PSDirect VM on HV1 and HV2 after the planned failover

Get-VM -ComputerName HV1 -Name PSDirect

Get-VM -ComputerName HV2 -Name PSDirect

1. Removing the VM replication on HV2

Remove-VMReplication -VMName PSDirect -ComputerName HV2

1. Removing the PSDirect VM replica on HV1

Remove-VM -Name PSDirect -ComputerName HV1 -Confirm:$false -Force

Run the remainder of this recipe on HV2.

1. Moving the PSDirect VM back to HV1

$VMHT2 = @{

    Name                  = 'PSDirect'

    ComputerName           = 'HV2'

    DestinationHost        = 'HV1'

    IncludeStorage         =  $true

    DestinationStoragePath = 'C:\Vm\Vhds\PSDirect' # on HV1

}

Move-VM @VMHT2

## How it works...

In step 1, you configure HV1 and HV2 to be trusted for delegation as required to support VM replication, and then in step 2, you reboot both hosts. These two steps produce no console output.

After rebooting both hosts, in step 3, you configure both Hyper-V hosts to support VM replication. And in step 4, you set up a Hyper-V replication partnership between HV1 and HV2. These two steps create no output.

In step 5, you review the replication status of the HV1 Hyper-V server, which creates output like this:

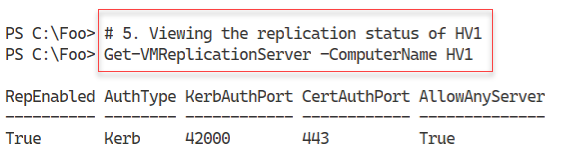


Figure 12.57: Viewing replication status of HV1

**Insert image B42024\_12\_57.png**

Next, in step 6, you check the replication status of the PSDirect VM, which you do on both Hyper-V servers. The output of this step looks like this:

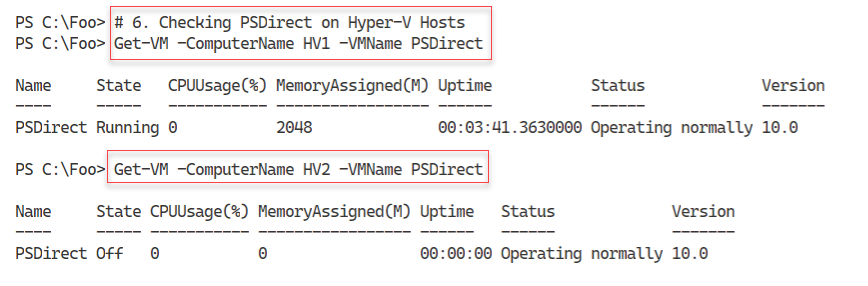


Figure 12.58: Checking PSDirect VM status on HV1 and HV2

**Insert image B42024\_12\_58.png**

In step 7, you start the VM replication process. This step involves Hyper-V creating a duplicate VM on HV2, essentially a full copy of the PSDirect VM, producing no output.

If you run step 8 immediately after completing step 7, you can observe the initial replication process. You should see output like this

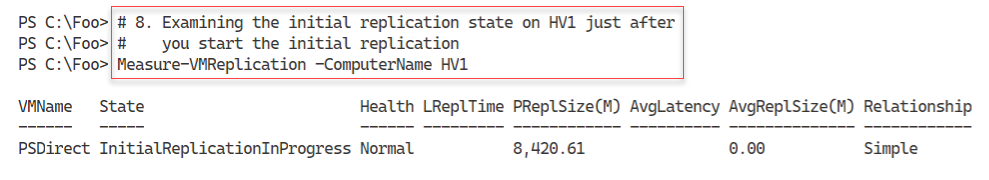


Figure 12.59: Viewing replication process

**Insert image B42024\_12\_59.png**

The initial replication process should take a few minutes. However, the replication speed also depends on the Hyper-V host hardware components, VM memory and virtual processors, and the underlying network and disk subsystem speeds. The disk sizes of any virtual disks are also a significant factor in the initial replication speed. After Hyper-V completes the initial replication of PSDirect, you can view the replication status, as shown in step 9. The output of this step looks like this:

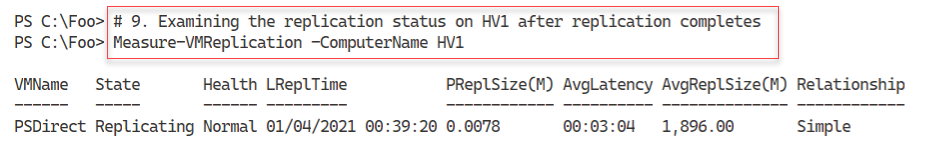


Figure 12.60: Viewing replication process after initial replication is complete

**Insert image B42024\_12\_60.png**

Now that you have the PSDirect replication invoked and Hyper-V has completed an initial replication, you can test the failover capability. In step 10, you run a test failover of the PSDirect VM from HV1 to HV2. This step generates no console output.

In step 11, you view the status of the PSDirect VMs on HV2, with output like this:

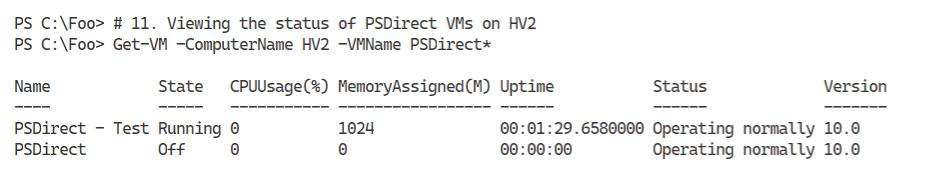


Figure 12.61: Viewing PSDirect VMs on HV2

**Insert image B42024\_12\_61.png**

In step 12, you stop the failover test, which generates no output. In step 13, you view the status of VMs on HV1 and HV2 after failover stopped, which produces output like this:

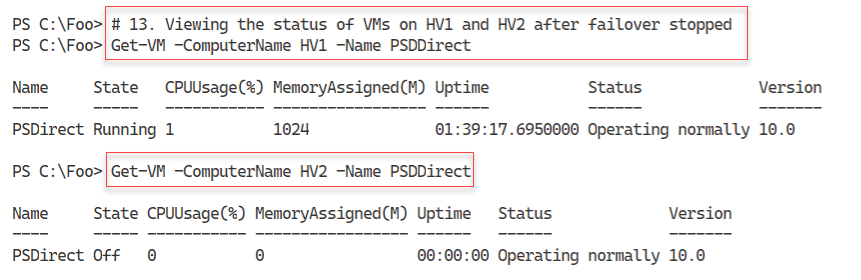


Figure 12.62: Viewing status of PSDirect VMs on HV2

**Insert image B42024\_12\_62.png**

In step 14, you stop the PSDirect VM before carrying out a planned failover. Next, in step 15, you perform the planned failover of PSDirect from HV1 to HV2. Once the planned failover has completed (that is, once the cmdlet completes), in step 16, you complete the failover process. In step 17, you start the PSDirect VM on HV2. These three steps produce no console output.

In step 18, you check the status of the PSDirect VM on both Hyper-V hosts. This step generates the following output:

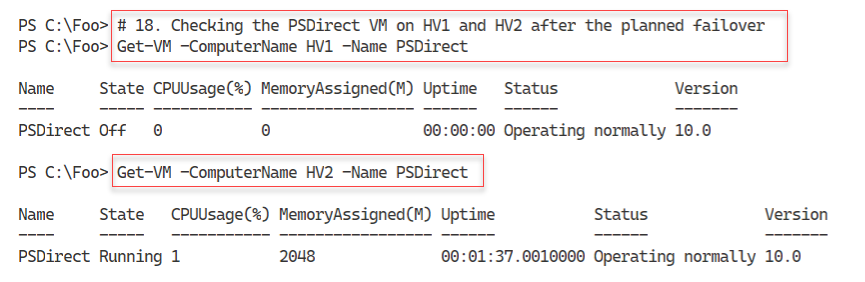


Figure 12.63: Checking status of PSDirect VMs on HV1 and HV2

**Insert image B42024\_12\_63.png**

In step 19, you remove the replication of the PSDirect VM from HV2. This step leaves the PSdirect VM running on HV2, although no longer replicating to another server. In step 20, you move the PSDirect VM back to HV1 to complete this recipe. These last two steps generate no console output.,

## There's more...

In step 11, you view the two PSDirect VMs on HV2. The first is the VM that you are replicating from HV1. During this step, PSDirect is up and running on HV1 and replicating any changes to HV2. You see a separate test VM showing that the replica VM is up and running on HV2. It is always good to run a test failover after setting up replication to ensure the replication and failure over work as you wish.

In step 18, you view the PSDirect VM status on both your Hyper-V hosts. As you can see in the output from this step, the PSDirect VM is up and running on HV2. On HV1, you still have an (older) copy of the VM. With the VM failed over (and running) on HV2, you could now either reverse the replication (that is, begin replication back to HV1 from HV2 for this VM). Or, as you do here, you can remove the VM from HV1, which allows you to move this VM back to HV1 as you do in step 20, which completes this recipe.

# Managing VM Checkpoints

With Hyper-V in Server 2022, 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 change of terminology was then consistent with System Center and avoided confusion with respect to the Volume Shadow Copy Service (VSS) snapshots used by many 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. Hyper-V 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 excellent for a variety of scenarios. It can be helpful in troubleshooting. Get the VM to the point where some bug is triggered and take a checkpoint. Then you can try a fix—if it doesn't work, you can just roll the VM back to the checkpoint and try some other fix. Checkpoints are also helpful for training. You could create a VM in which you perform all the lab exercise steps and create a checkpoint after each successful lab. That way, the student can make a mistake in a lab exercise and skip forward to a later checkpoint to carry on.

Using checkpoints in production is a different matter. In general, you should avoid using checkpoints on your production systems for several reasons. If your servers use any type of replication or transaction-based applications, the impact of rolling back a VM to an earlier time can be lead to issues. Since checkpoints rely on differencing disks that feature constantly growing physical disk files, using checkpoints can also result in poor performance.

In this recipe, you create a snapshot of the PSDirect VM, and then you create a file inside the VM. You take a further checkpoint and create a second file. Then you revert 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 taking the snapshot. Then you remove all the snapshots. After each checkpoint operation, you observe the VHDX and AVHD files that Hyper-V uses in the PSDirect VM.

## Getting Ready

You run this recipe on HV1. This recipe makes use of the PSDirect VM that you created and used earlier in this chapter. Depending on which other recipes you have run from this chapter, the virtual disks may be in different folders, but the recipe copes with the disk files in any folder known to Hyper-V.

## How to do it...

1. Creating credentials for PSDirect

$RKAn = 'Wolf\Administrator'

$PS   = 'Pa$$w0rd'

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

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

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

1. Examining the C:\ in the PSDirect VM before we start

$SB = { Get-ChildItem -Path C:\ | Format-Table}

$ICHT = @{

  VMName      = 'PSDirect'

  ScriptBlock = $SB

  Credential  = $RKCred

}

Invoke-Command @ICHT

1. Creating a snapshot of PSDirect on HV1

$CPHT = @{

  VMName       = 'PSDirect'

  ComputerName = 'HV1'

  SnapshotName = 'Snapshot1'

}

Checkpoint-VM @CPHT

1. Examining the files created to support the checkpoints

$Parent = Split-Path -Parent (Get-VM -Name PSdirect |

            Select-Object -ExpandProperty HardDrives).Path |

              Select-Object -First 1

Get-ChildItem -Path $Parent

1. Creating some content in a file on PSDirect and displaying it

$SB = {

  $FileName1 = 'C:\File\_After\_Checkpoint\_1'

  Get-Date | Out-File -FilePath $FileName1

  Get-Content -Path $FileName1

}

$ICHT = @{

  VMName      = 'PSDirect'

  ScriptBlock = $SB

  Credential  = $RKCred

}

Invoke-Command @ICHT

1. Taking a second checkpoint

$SNHT = @{

  VMName        = 'PSDirect'

  ComputerName  = 'HV1'

  SnapshotName  = 'Snapshot2'

}

Checkpoint-VM @SNHT

1. Viewing the VM checkpoint details for PSDirect

Get-VMSnapshot -VMName PSDirect

1. Looking at the files supporting the two checkpoints

Get-ChildItem -Path $Parent

1. Creating and displaying another file in PSDirect (i.e. after you have taken Snapshot2)

$SB = {

  $FileName2 = 'C:\File\_After\_Checkpoint\_2'

  Get-Date | Out-File -FilePath $FileName2

  Get-ChildItem -Path C:\ -File | Format-Table

}

$ICHT = @{

  VMName    = 'PSDirect'

  ScriptBlock = $SB

  Credential  = $RKCred

}

Invoke-Command @ICHT

1. Restoring the PSDirect VM back to the checkpoint named Snapshot1

$Snap1 = Get-VMSnapshot -VMName PSDirect -Name Snapshot1

Restore-VMSnapshot -VMSnapshot $Snap1 -Confirm:$false

Start-VM -Name PSDirect

Wait-VM -For IPAddress -Name PSDirect

1. Seeing what files we have now on PSDirect

$SB = {

  Get-ChildItem -Path C:\ | Format-Table

}

$ICHT = @{

  VMName    = 'PSDirect'

  ScriptBlock = $SB

  Credential  = $RKCred

}

Invoke-Command @ICHT

1. Rolling forward to Snapshot2

$Snap2 = Get-VMSnapshot -VMName PSdirect -Name Snapshot2

Restore-VMSnapshot -VMSnapshot $Snap2 -Confirm:$false

Start-VM -Name PSDirect

Wait-VM -For IPAddress -Name PSDirect

1. Observing the files you now supporting PSDirect

$SB = {

    Get-ChildItem -Path C:\ | Format-Table

}

$ICHT = @{

  VMName      = 'PSDirect'

  ScriptBlock = $SB

  Credential  = $RKCred

}

Invoke-Command @ICHT

1. Restoring to Snapshot1 again

$Snap1 = Get-VMSnapshot -VMName PSDirect -Name Snapshot1

Restore-VMSnapshot -VMSnapshot $Snap1 -Confirm:$false

Start-VM -Name PSDirect

Wait-VM -For IPAddress -Name PSDirect

1. Checking snapshots and VM data files again

Get-VMSnapshot -VMName PSDirect

Get-ChildItem -Path $Parent | Format-Table

1. Removing all the snapshots from HV1

Get-VMSnapshot -VMName PSDirect |

  Remove-VMSnapshot

1. Checking VM data files again

Get-ChildItem -Path $Parent

## How it works...

In step 1, you create a Windows credential object to use with the PSDirect VM. This step creates no output.

In step 2, you examine the C:\ drive inside the PSDirect VM, generating output like this:

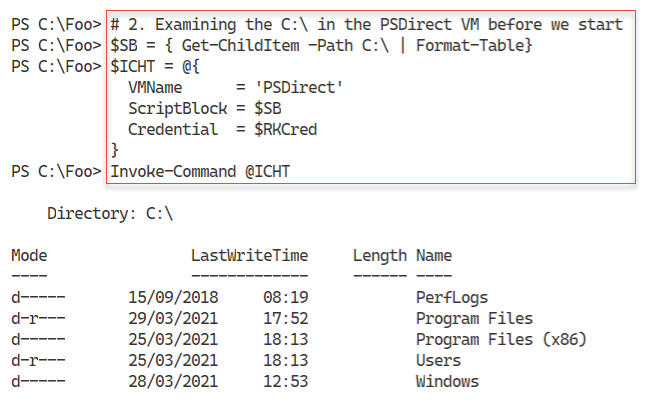


Figure 12.64: Examing C:\ on PSDirect VM

**Insert image B42024\_12\_64.png**

In step 3, you create a VM snapshot of the PSDirect VM, which generates no console output. After you create the snapshot, in step 4, you examine the files which Hyper-V uses to support the VM snapshot. The output of this step looks like this:

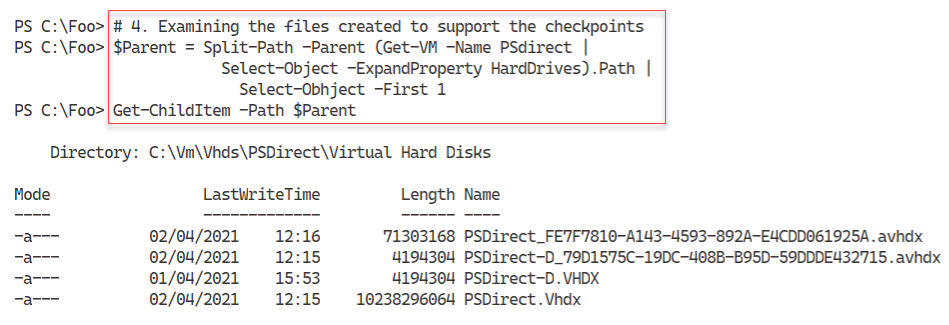


Figure 12.65: Examing files supporting a snapshot

**Insert image B42024\_12\_65.png**

In step 5, you create a new file on the C:\ in the PSDirect VM. Then you view the contents of this file. The output looks like this:

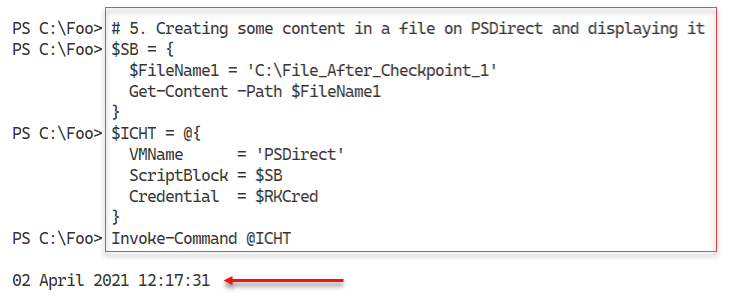


Figure 12.66: Creating a file in the PSDirect VM

**Insert image B42024\_12\_66.png**

After creating a file inside the VM, in step 6, you create a second snapshot that generates no output. In step 7, you use the Get-VMSnapshot cmdlet to view the snapshots for the PSDirect VM currently on HV1, which looks like this:

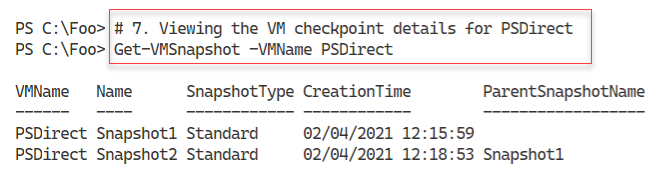


Figure 12.67: Viewing snapshots of PSDirect on HV1

**Insert image B42024\_12\_67.png**

In step 8, you examine the files which Hyper-V uses to store your virtual disk images and snapshot files for the PSDirect VM, like this:

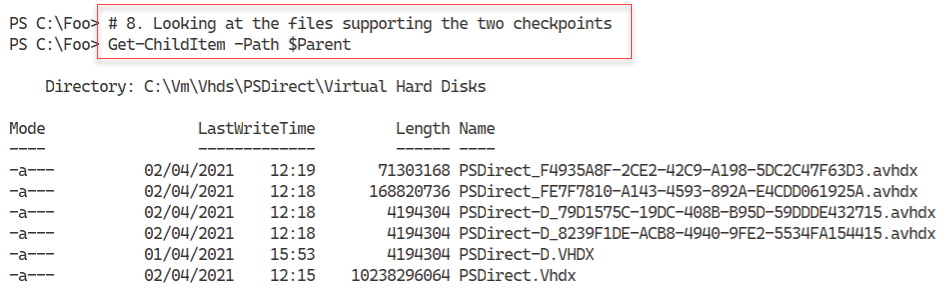


Figure 12.68: Viewing snapshots of PSDirect on HV1

**Insert image B42024\_12\_68.png**

In step 9, you create a new file in the PSDirect VM, add contents to the file, and then view those files creates so far, like this:

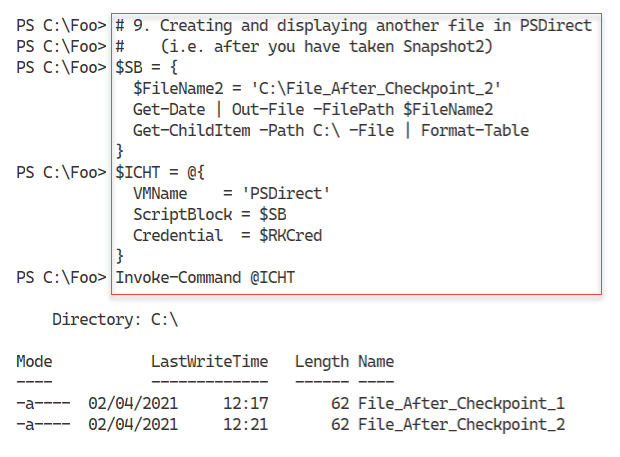


Figure 12.69: Creating a second file in PSDirect

**Insert image B42024\_12\_69.png**

In step 10, you revert the PSDirect VM back to the first snapshot, which creates no output to the console. In step 11, you see what files you now have on the C:\ drive, like this:

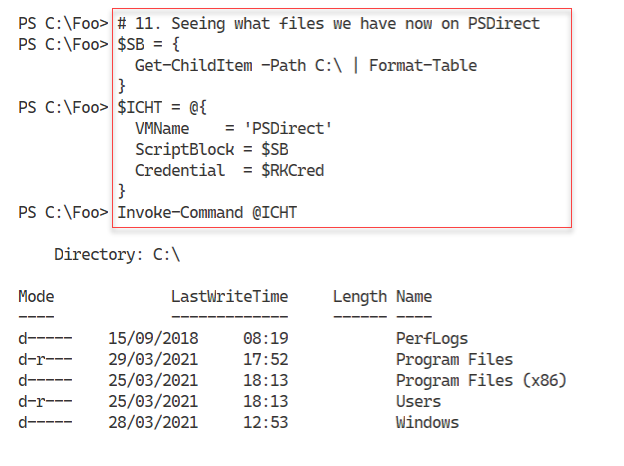


Figure 12.70: Checking files in PSDirect

**Insert image B42024\_12\_70.png**

In step 12, you restore the PSDirect VM to the second snapshot, which creates not output. In step 13, you examine the files create in PSDirect, which looks like this:

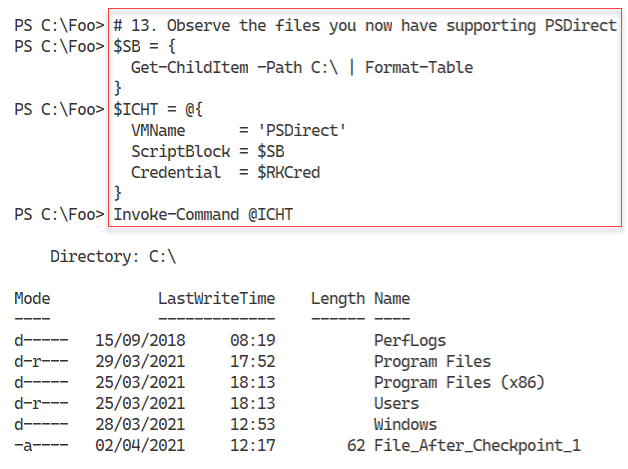


Figure 12.71: Checking files in PSDirect after restoring Checkpoint2

**Insert image B42024\_12\_71.png**

In step 14, you restore PSDirect to the first snapshot, which creates no output. In step 15, you check the snapshots and the virtual disk files again, with output like this:

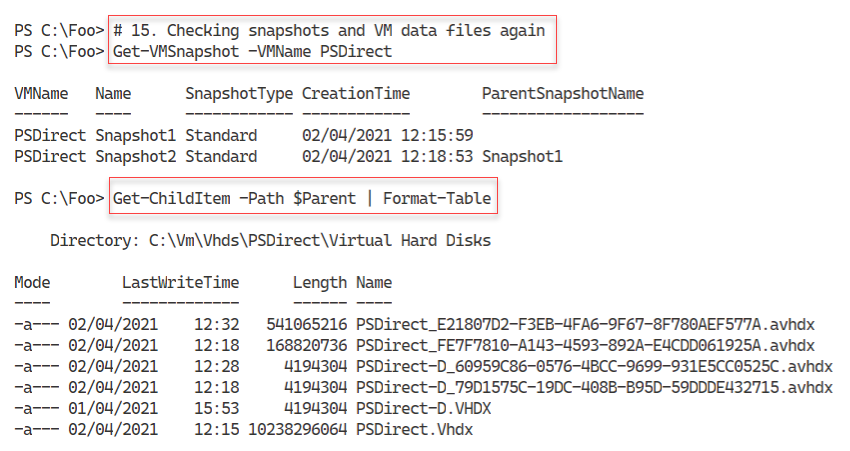


Figure 12.72: Checking snapshots on HV1

**Insert image B42024\_12\_72.png**

In step 16, you remove all the checkpoints for PSDirect on HV1, which generates no output. In the final step in this recipe, you check the VM data files on HV1, which produces the following output:

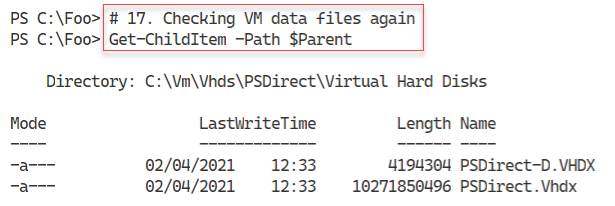


Figure 12.73: Viewing disk files after removing the snapshots

**Insert image B42024\_12\_73.png**

## There's more...

In step 8, you examine the files that Hyper-V uses to support the PSDirect VM snapshots. The two VHDX files held the starting state when you took the first snapshot. Then, there are two more sets of AVHDX files. Each file represents any work you have done after taking a snapshot and before taking another.

In step 9, you view the two files you created in the PSDirect VM. As you can see, there are the two files you created in earlier steps in the recipe.

After reverting o the first snapshot, in step 11, you see neither of the two files you created to exist on the C:\ drive. You saw no files in PSDirect after restoiring the snapshot because you created those files after creating the first snapshot. This situation is the expected result of restoring a snapshot.