11

Managing Hyper-V

In this chapter, we cover the following recipes:

* Installing and configuring Hyper-V
* Creating a virtual machine
* Using PowerShell Direct
* Working with VM groups
* Configuring VM hardware
* Configuring Hyper-V networking
* Implementing nested Hyper-V
* Managing VM state
* Configuring VM and storage movement
* Configuring 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. The Hyper-V feature is included in all versions of Windows Server 2019, as well as in the Enterprise, Professional, and Education editions of Windows 10.

Hyper-V was first released with Server 2008 and has been improved with each successive version of Windows Server. Improvements include additional features, support of the latest hardware, and scalability.

Hyper-V supports nested Hyper-V, the ability to run Hyper-V inside a Hyper-V VM. Nested Hyper-V has some great 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 Hyper-V 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 2019, although you can manage Hyper-V Server using the tools used in this chapter's recipes. References to your Hyper-V servers refer to your Windows 2019 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. Later in the chapter, you create and manage virtual machines, and use PSDirect.

# Installing and configuring Hyper-V

With Windows Server 2019, to add Hyper-V to your server, you install the Hyper-V feature. This recipe installs Hyper-V on two servers: HV1 and HV2, which are domain-joined Windows 2019 servers with no added features.

## Getting ready

In this recipe, you do the set up remotely from a client machine, CL1, using the Hyper-V cmdlets and PowerShell's remoting capabilities. CL1 is a domain-joined Windows 10 system with the RSAT tools installed. You previously set up the CL1 client in the Installing RSAT tools on Windows 10 and Windows Server 2019 recipe.

## How to do it...

1. From CL1, install the Hyper-V feature on HV1 and HV2:

$Sb = {

Install-WindowsFeature -Name Hyper-V -IncludeManagementTools

}

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

1. Reboot both servers to complete the installation:

Restart-Computer -ComputerName HV1, HV2 -Force

1. Create a PSSession with both HV servers (after the restart has completed):

$S = New-PSSession HV1, HV2

1. Create and set the location for VMs and VHDs on HV1 and HV2, then view the results:

$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

Get-ChildItem -Path C:\Vm }

Invoke-Command -ScriptBlock $Sb -Session $S

1. Set default paths for Hyper-V VM hard disks and VM configuration information:

$SB = {

$VMs = 'C:\Vm\Vhds'

$VHDs = 'C:\Vm\VMs\Managing Hyper-V'

Set-VMHost -ComputerName Localhost -VirtualHardDiskPath $VMs

Set-VMHost -ComputerName Localhost -VirtualMachinePath $VHDs

}

Invoke-Command -ScriptBlock $SB -Session $S

1. Set up NUMA spanning:

$SB = {

Set-VMHost -NumaSpanningEnabled $true

}

Invoke-Command -ScriptBlock $SB -Session $S

1. Set up EnhancedSessionMode:

$SB = {

Set-VMHost -EnableEnhancedSessionMode $true

}

Invoke-Command -ScriptBlock $SB -Session $S

1. Set up 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. Review key VMHost settings:

$SB = {

Get-VMHost

}

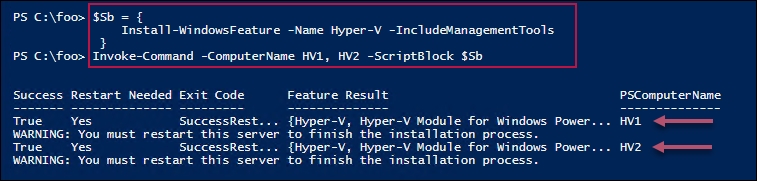
$P = 'Name', 'V\*Path','Numasp\*', 'Ena\*','RES\*'

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

Format-Table -Property $P

## How it works...

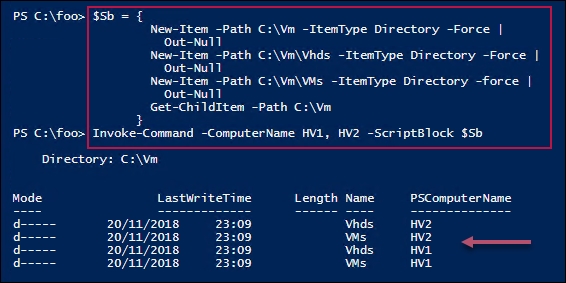
In step 1, you use the Install-WindowsFeature cmdlet to install Hyper-V on HV1 and HV2 remotely, which looks like this:



In step 2, you reboot HV1 and HV2, which completes the process of installing Hyper-V on both servers. There is no output from this step.

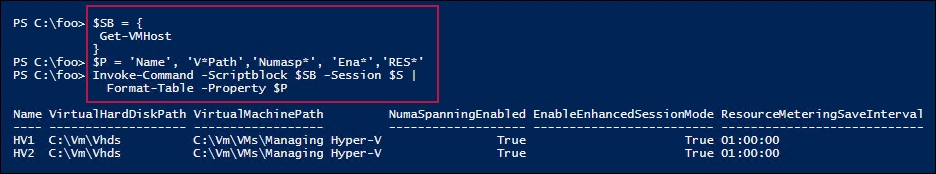
After the reboot, in step 3, you create a remoting session on both HV1 and HV2. You use this in the following steps, but there is no output.

In step 4, you create new folders to hold virtual machines and virtual hard drives on both servers. The output of this step looks like this:



In step 5, you configure Hyper-V to save VMs and VHDs in the newly created folder on both servers. In step 6, you configure Hyper-V to support NUMA. In step 7, you configure EnhancedSessionMode. In step 8, you configure host resource metering. These four steps produce no output.

In step 9, you look to see how HV1 and HV2 are configured, which looks like this:



## 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-processor-cpu-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 5 through step 8, you set up one aspect of the VM hosts. 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/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 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/windows/deployment/windows-deployment-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, which has a hostname of Tiger. In building the VM, you assign the Windows Server 2019 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 2019 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-V recipe. You also need the Windows Server 2019 ISO image. For testing purposes, this could be an evaluation version, or a full retail edition.

## How to do it...

1. Set 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\windows\_server\_2019\_x64\_dvd.iso'

1. Create a new VM:

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

1. Create a virtual disk file for the VM:

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

1. Add the virtual hard drive to the VM:

Add-VMHardDiskDrive -VMName $VMName -Path $VhdPath

1. Set the ISO image in the VM's DVD drive:

$IHT = @{

VMName = $VMName

ControllerNumber = 1

Path = $ISOPath

}

Set-VMDvdDrive @IHT

1. Start the VM:

Start-VM -VMname $VMname

1. View the results:

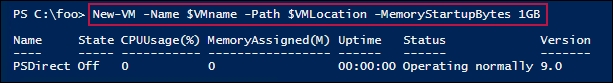
Get-VM -Name $VMname

1. Use the VM connect tool from the Hyper-V console and complete the OS installation of the VM using the GUI setup tool.

## How it works...

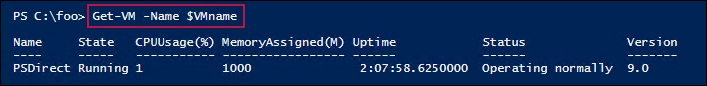
In step 1, you create a number of variables representing values you use in this recipe. This step produces no output.

In step 2, you use the New-VM cmdlet to create a new Hyper-V VM, which produces the following output:

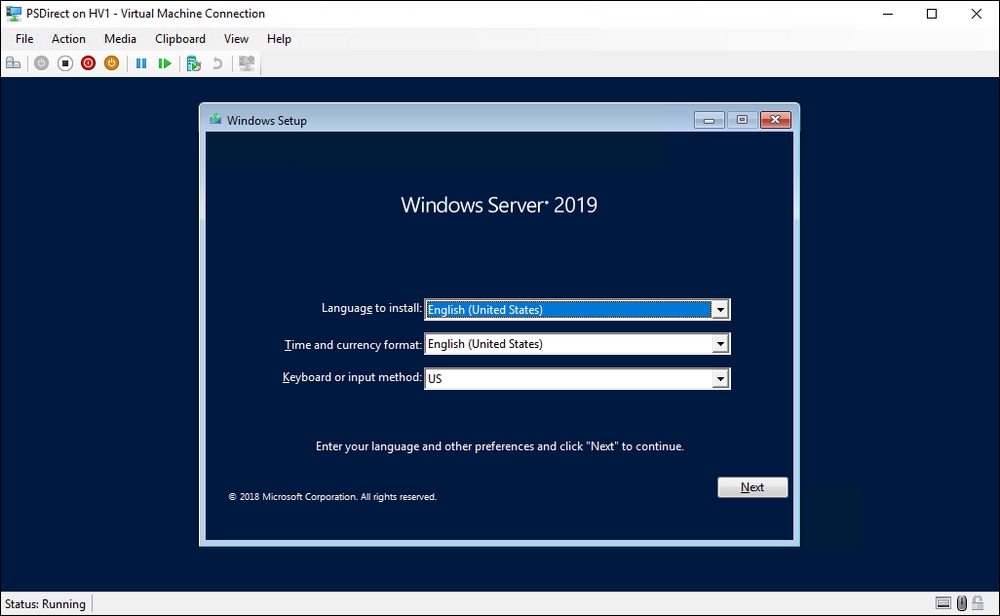


In step 3, you create a new VHDX file, which is to serve as the VM's C: drive. In step 4, you add the newly created VHDX to the PSDirect VM. In step 5, you add the Windows Server 2019 ISO image to the VM. Then, in step 6, you start the VM. These four steps produce no output.

In step 7, you use the Get-VM cmdlet to see the newly created VM, which looks like this:



Using the VMConnect feature of the Hyper-V GUI, in step 8, you can view the VM. Initially, you see something like this:



From that screen, you click Next, and so on, to complete the GUI installation process. To get the VM ready for the next step, you should rename the VM (Tiger) and join the domain.

## There's more...

In step 1, you use the Windows 2019 Server ISO image, saved on HV1 as C:\builds\windows\_server\_2019\_x64\_dvd.iso. Depending on where you obtain the image, the file name may differ.

In step 8, you view the VM, which should have started and have reached the initial dialog in the setup process. From here, you can use the GUI to complete the installation of Windows Server 2019. The details of Windows Server setup are outside the scope of this book and are left as an exercise for the reader.

# Using PowerShell Direct

PowerShell Direct (PSD) is a new feature with Windows Server 2016 (and on Windows 10 Anniversary Update or later). PSD enables you to use PowerShell remoting to access a Hyper-V VM without needing to set up networking and firewall settings inside the VM. With PSD, you use Invoke-Command, specifying either the VM's name or the VM's VMID (the VMID is a GUID used internally by Hyper-V to identify a VM) rather than the VM's hostname. You can also use the VM name or VMID to enter a remote session using Enter-PSSession.

In earlier versions of Hyper-V, you needed a networking connection between your Hyper-V host and the guest OS in order to remote into the guest. This meant setting up and establishing network connectivity. With PSD, you can use the VM's name or ID and remote straight in. This is useful when a VM is misconfigured and its network connectivity is unavailable.

## Getting ready

For this recipe, you need the Hyper-V host that you set up in the Installing and configuring Hyper-V recipe. You also need the VM setup in the Creating a virtual machine recipe. The VM has a VM name of PSDirect, but has a hostname of Tiger.

To ensure security, you need to specify credentials when you call Invoke-Command or Enter-PSSession. You can either specify the -Credential parameter or let either cmdlet prompt for credentials. With Hyper-V and PSDirect, the VM name and the hostname of the OS running inside the VM do not need to be the same.

## Getting ready

This recipe uses the HV1 host you created in the Installing and configuring VM recipe, along with a Windows Server VM running on HV1. This VM has a VM name of PSDirect, a hostname of Tiger, and a local administrator password of Pa$$w0rd. You can use the Creating a VM recipe, suitably adapted, to create the PSDirect VM.

## How to do it...

1. Create a credential object for the local administrator on PSDirect:

$RKAn = '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. Retrieve and display the details of the PSDirect VM:

Get-VM -Name PSDirect

1. Invoke a command on the VM, specifying the VM name:

$SBHT = @{

VMName = 'PSDirect'

Credential = $RKCred

ScriptBlock = {hostname}

}

Invoke-Command @SBHT

1. Invoke a command based on VMId:

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

$ICMHT = @{

VMid = $VMID

Credential = $RKCred

ScriptBlock = {hostname}

}

Invoke-Command @ICMHT

1. Enter a PS remoting session with the PSDirect VM:

Enter-PSSession -VMName PSDirect -Credential $RKCred

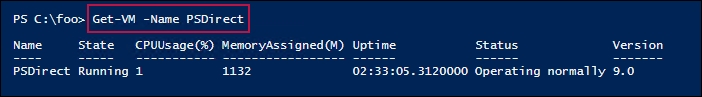
Get-CimInstance -Class Win32\_ComputerSystem

Exit-PSSession

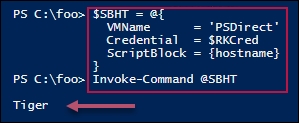
## How it works...

In step 1, you create a credential object for the local administrator of the PSDirect virtual machine. This step creates no output.

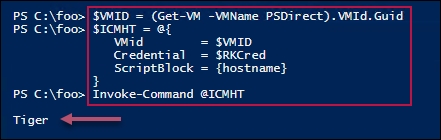
In step 2, you retrieve the details of the PSDirect virtual machine, which looks like this:



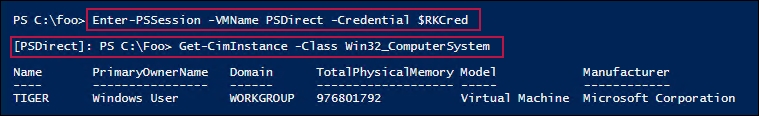
In step 3, you use the PSD feature and invoke a command inside the PSDirect VM using just the virtual machine name, which looks like this:



In step 4, you invoke a command inside the PSDirect virtual machine but using the VM's VMID value, like this:



In step 5, you enter a PS remoting session with the PSDirect VM, which looks like this:



## There's more...

In step 3 and step 4, you make use of PSD by invoking a command using either the VM's name or the VM's VMID value. In step 5, you use Enter-PSSession and enter a PowerShell session in the PSDirect VM, and as you can see, the hostname of that VM is Tiger.

# Working with VM groups

VM groups are a feature of Hyper-V that allow you to group VMs for the purposes of automation. With Hyper-V, there are two types of VM groups: 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.

## Getting ready

You run this recipe on the HV2 Hyper-V server, which you created in the Installing and configuring Hyper-V recipe. This Hyper-V server has a number of VMs defined. For the purposes of this recipe, you can create the necessary VMs using the following:

# Create HV2 VMs for Hyper-V Chapter

$VMLocation = 'C:\Vm\VMs'

# Create VM1

$VMN1 = 'SQLAcct1'

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

# Create VM2

$VMN2 = 'SQLAcct2'

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

# Create VM3

$VMN3 = 'SQLAcct3'

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

# Create VM4

$VMN4 = 'SQLMfg1'

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

# Create VM5

$VMN5 = 'SQLMfg2'

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

## How to do it...

1. Set up Hyper-V VM groups and display them:

$VHGHT1 = @{

Name = 'SQLAccVMG'

GroupType = 'VMCollectionType'

}

$VMGroupACC = New-VMGroup @VHGHT1

$VHGHT2 = @{

Name = 'SQLMfgVMG'

GroupType = 'VMCollectionType'

}

$VMGroupMFG = New-VMGroup @VHGHT2

1. Create arrays of group member VM names:

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

$MFGVms = 'SQLMfg1', 'SQLMfg2'

1. Add members to the accounting SQL VM group:

Foreach ($Server in $ACCVMs) {

$VM = Get-VM -Name $Server

Add-VMGroupMember -Name SQLAccVMG -VM $VM

}

1. Add members to the manufacturing SQL VM group:

Foreach ($Server in $MfgVMs) {

$VM = Get-VM -Name $Server

Add-VMGroupMember -Name SQLMfgVMG -VM $VM

}

1. Create a management collection VM group:

$VMGHT = @{

Name = 'VMMGSQL'

GroupType = 'ManagementCollectionType'

}

$VMMGSQL = New-VMGroup @VMGHT

1. Add the two VM collection type groups to the VM management group:

Add-VMGroupMember -Name VMMGSQL -VMGroupMember $VMGroupACC,

$VMGroupMFG

1. Set FormatEnumerationLimit to a higher value, then view the VM groups:

$FormatEnumerationLimit = 99

Get-VMGroup |

Format-Table -Property Name, GroupType, VMGroupMembers,

VMMembers

1. Stop all the SQL VMs:

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

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

}

1. Set the CPU count in all SQL VMs to 4:

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

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

}

1. Set accounting SQL VMs to have 6 processors:

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

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

}

1. Check 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. Remove VMs from VM groups:

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

Foreach ($VM in $ACCVMS) {

$X = Get-VM -VMName $VM.name

Remove-VMGroupMember -Name SQLAccVMG -VM $x

}

$MFGVMs = (Get-VMGroup -Name SQLMfgVMG).VMMEMBERS

Foreach ($VM in $MFGVMS) {

$X = Get-VM -VMName $VM.Name

Remove-VMGroupMember -Name SQLMfgVMG -VM $x

}

1. Remove all the VM groups from VM management groups:

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

Foreach ($VMG in $VMGS) {

$X = Get-VMGroup -VMName $VMG.Name

Remove-VMGroupMember -Name VMMGSQL -VMGroupName $x

}

1. Remove all the VM groups:

Remove-VMGroup SQLAccVMG -Force

Remove-VMGroup SQLMfgVMG -Force

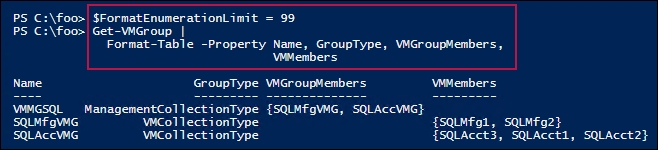
Remove-VMGroup VMMGSQL -Force

## How it works...

In the first part of this recipe, you create and populate two VM collection type VM groups and a VM management type VM group. Then, you use these VM groups in some management activities—in this case, updating the VM configuration for several VMs.

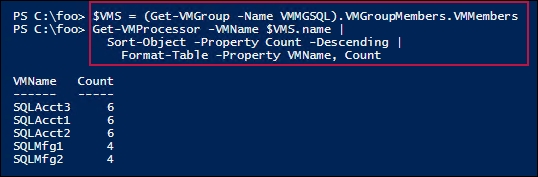
In step 1, you create two VM groups: SQLAccVMG and SQLMfgVMG. In step 2, you create two arrays of VM names. In step 3 and step 4, you add the VMs to the two VM collection type VM groups. In step 5, you create a ManagementCollectionType VM group and in step 6, you populate the VM management collection type group with the two VM collection type VM groups. There is no output from these steps.

In step 7, you view the VM groups and their members, which looks like this:



In step 8, you use the management collection type VM group to stop all the SQL VMs. In step 9, you set all of the SQL VMs to have four virtual processors, and in step 10, you set the VMs in the SQLAccVMG to have four CPUs. These three steps produce no output.

In step 11, you view the processor counts for all the SQL VMs, which looks like this:



In step 12, you remove the VMs from the VM collection type VM groups, and in step 13 you remove the VM collection type VM groups from the VM management groups. Finally, in step 14, you remove all the VM groups from HV2. There is no output from these three steps.

## There's more...

In step 12 and step 13, you explicitly remove members from the three VM groups before, in step 14, you remove the three VM groups. If you attempt to remove the VM groups that still have members, the cmdlet fails.

The VM group is a feature of Hyper-V that feels only partly finished. The distinction between VM collection groups and VM management groups seems unnecessary. It might be better if there were just one group type (VM collection) that collected VMs and VM collections. Additionally, it would be useful to add a -VMGroup parameter to most of the cmdlets as an alternative to VMName, VMID, and so on—that way, the cmdlet would apply to all the VMs in the specified VM group and not just a single VM.

## See also

In this recipe, you updated the CPU count for several VMs. See the Configuring VM hardware recipe for more on configuring the VM's virtual hardware.

# 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. You can also adjust physical RAM, network interfaces, disk interfaces and disk devices, and DVD drives (with/without a loaded DVD), and so on. Each of these physical components is provided within a Hyper-V VM, and 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. You then create a 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. But you can always add additional NICs. Configuring the VM's networking is covered in the Configuring Hyper-V networking recipe.

## Getting ready

This recipe uses the VM you created in the Creating a virtual machine recipe. The VM is called PSDirect.

## How to do it...

1. Turn off the PSDirect VM:

Stop-VM -VMName PSDirect

Get-VM -VMName PSDirect

1. Set the StartupOrder in the VM's BIOS:

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

Set-VMBios -VmName PSDirect -StartupOrder $Order

Get-VMBios PSDirect

1. Set and view CPU count for PSDirect:

Set-VMProcessor -VMName PSDirect -Count 2

Get-VMProcessor -VmName PSDirect |

Format-Table VMName, Count

1. Configure PSDirect memory settings:

$VMHT = [ordered] @{

VMName = 'PSDirect'

DynamicMemoryEnabled = $true

MinimumBytes = 512MB

StartupBytes = 1GB

MaximumBytes = 2GB

}

Set-VMMemory @VMHT

Get-VMMemory -VMName PSDirect

1. Add a SCSI controller to PSDirect and view the controllers available in the VM:

Add-VMScsiController -VMName PSDirect

Get-VMScsiController -VMName PSDirect

1. Restart the VM:

Start-VM -VMName PSDirect

Wait-VM -VMName PSDirect -For IPAddress

1. Create a new VHDX file:

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

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

1. Add the VHD to the SCSI controller:

$VHDHT = @{

VMName = 'PSDirect'

ControllerType = 'SCSI'

ControllerNumber = 0

ControllerLocation = 0

Path = $VHDPath

}

Add-VMHardDiskDrive @VHDHT

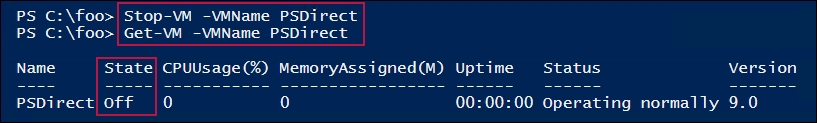
1. Get volumes from PSDirect VM:

Get-VMScsiController -VMName PSDirect |

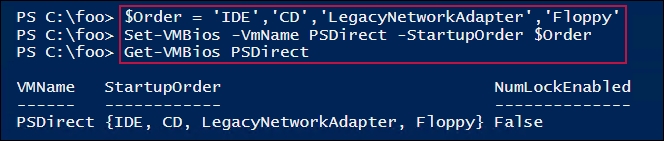
Select-Object -ExpandProperty Drives

## How it works...

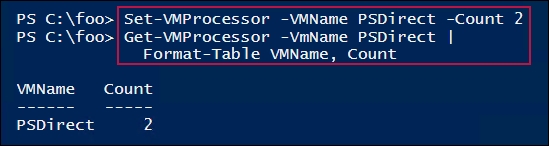
In step 1, you turn off the PSDirect VM, then get the VM's details, as follows:



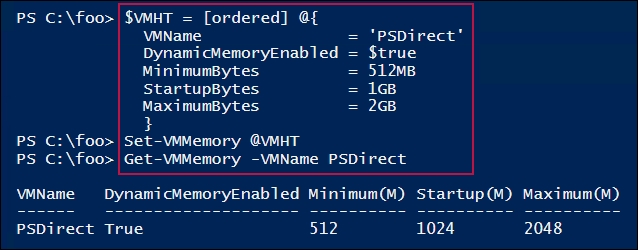
In step 2, you adjust then view the startup order in the VM's virtual BIOS, which looks like this:



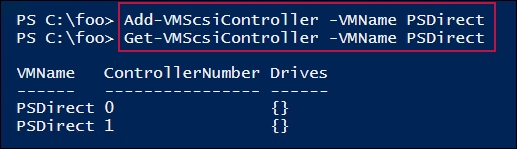
In step 3, you set and then view the CPU count for the PSDirect VM, which looks like this:



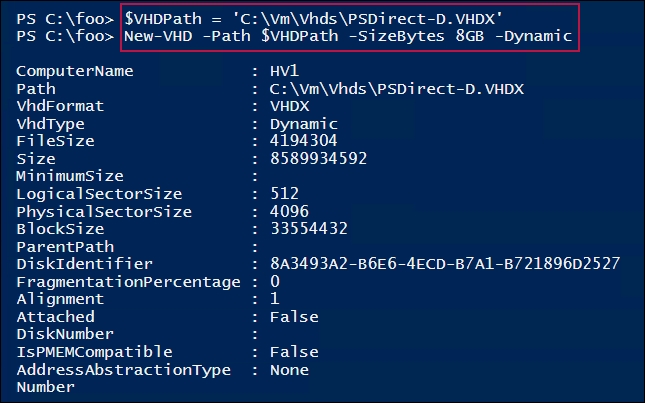
In step 4, you adjust and view the VM's memory settings, which looks like this:



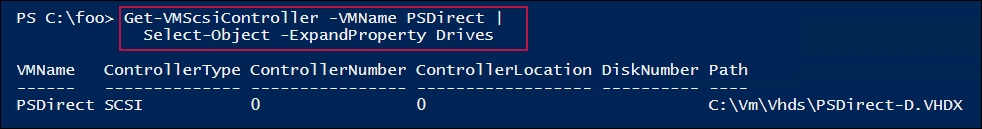
In step 5, you add a new SCSI controller to the PSDirect VM, then view the controllers, which looks like this:



In step 6, you restart the PSDirect VM, which produces no output. Once the VM has restarted, in step 7, you create a new virtual hard drive, that looks like this:



In step 8, you add the VHDX to the VM, which produces no output. In step 9, you view the details of the virtual hard drive in the SCSI controller in the PSDirect VM, which looks like this:



## There's more...

In addition to the hardware components covered in this recipe, you can also manage a VM's COM ports and diskette drives. While you cannot directly connect a VM's COM port to the host's COM port, you can configure a VM to communicate with the physical computer via a named pipe on the host computer. A typical use for this is kernel debugging—probably something most IT pros rarely ever do. For more information on named pipes, see <https://docs.microsoft.com/windows/desktop/ipc/named-pipes>.

You can also use a virtual floppy drive in a VM. There is no cmdlet support to create a virtual floppy drive file (a .vfd file) in the Hyper-V module, nor is there support for mounting a VFD file in Windows. You can create VFD files using Hyper-V Manager and then use Set-VMFloppyDiskDrive to attach the VFD file as a floppy disk drive in the VM.

# Configuring Hyper-V networking

Each VM is essentially another host in your infrastructure. It’s important therefore to set up networking for your Hyper-V hosts. There are three basic types of networking:

* External: This enables a VM to share the host’s physical NIC and participate in host-to-host networking
* VM to VM only: This is a secure network that enables traffic between VMs only
* VM to VM and VM Host: This is a fairly secure network that shares traffic between VMs and the VM host

You start the configuration process by creating a VM switch. If you create either an external or VM to VM and VM host switch, then Windows adds a NIC into the VM host to enable the host to communicate via the VM Switch. Once you have the VM switch(es) set up, you can configure a VM’s NIC to use a specific switch. This recipe shows some of the basic steps involved in getting started with Hyper-V networking.

## Getting ready

You run this recipe on HV1, which you set up in the Installing and configuring the Hyper-V feature recipe. HV1 has no switch defined—you create a virtual switch in this recipe. Additionally, this Hyper-V server has a VM, PSDirect, which you have set up and configured in earlier recipes in this chapter. This VM has a hostname of Tiger. Also, the local administrator has a password of Pa$$w0rd.

## How to do it...

1. Get NIC details and IP address from the PSDirect VM:

Get-VMNetworkAdapter -VMName PSDirect

1. Create 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 }

Credential = $RKCred

}

Invoke-Command @VMHT | Format-List

1. Create a virtual switch on HV1:

$VSHT = @{

Name = 'External'

NetAdapterName = 'Ethernet'

Notes = 'Created on HV1'

}

New-VMSwitch @VSHT

1. Connect PSDirect to the switch:

Connect-VMNetworkAdapter -VMName PSDirect -SwitchName External

1. See the VM's network adapter information:

Get-VMNetworkAdapter -VMName PSDirect

1. With PSDirect now in the network, observe the IP address in the VM:

$NCHT = @{

VMName = 'PSDirect'

ScriptBlock = {Get-NetIPConfiguration}

Credential = $RKCred

}

Invoke-Command @NCHT

1. View the hostname on PSDirect, reusing the hash table from step 6:

$NCHT.ScriptBlock = {hostname}

Invoke-Command @NCHT

1. Change the name of the host in VM1, making use of the hash table from step 6 and step 7:

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

Invoke-Command @NCHT

1. Reboot and wait for the restarted PSDirect:

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

1. Get the hostname of the PSDirect VM:

$NCHT.ScriptBlock = {hostname}

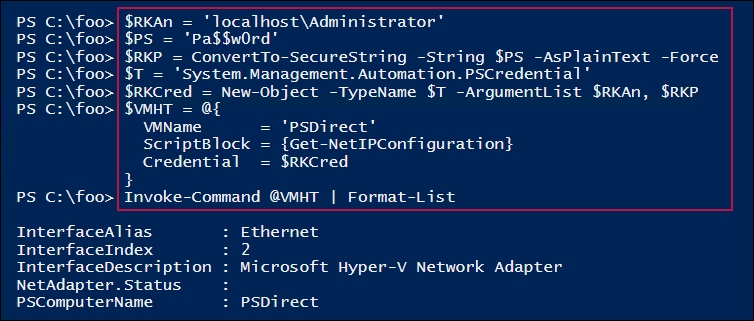
Invoke-Command @NCHT

## How it works...

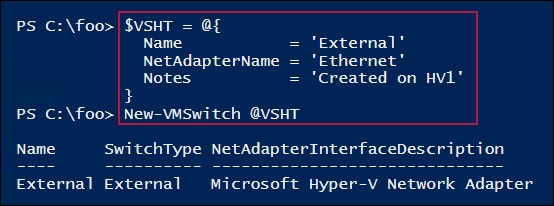
In step 1, you get the NIC details and IP addresses assigned to the PSDirect VM, which looks like this:



In step 2, you create a credential object for the PSDirect virtual machine. Then, you use this to execute a script block on PSDirect, which returns the host's network IP configuration, and looks like this:

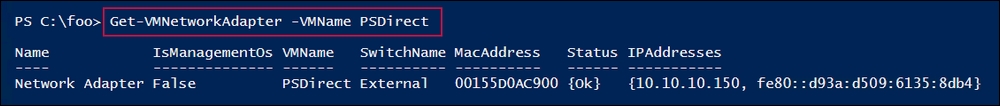


In step 3, you create a new virtual switch on HV1 with the switch name set to External. The output of this step looks like this:

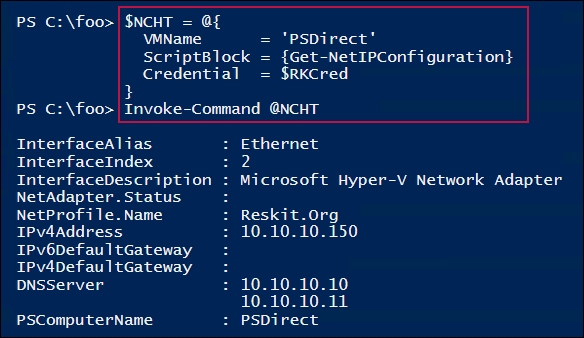


In step 4, you add the NIC in the PSDirect VM to the new External switch. This step produces no output.

With the VM now having network connectivity, in step 5, you look at the VM's networking details, which look like this:



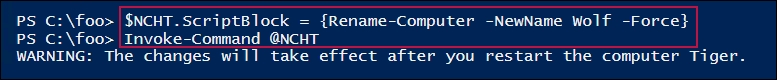
In step 6, you get the host's networking details, showing the IP address and DNS servers (obtained from the DHCP server), like this:



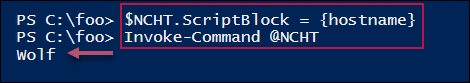
In step 7, you view the hostname of the PSDirect VM, which looks like this:



In step 8, you change the name of the PSDirect VM to Wolf, which looks like this:



In step 9, you restart the PSDirect VM, producing no output. In step 10, you re-check the PSDirect VM's hostname, which looks like this:



## There's more...

In step 1, you use the Get-VMNetworkAdapter for the PSDirect VM. The output from this step shows that the virtual NIC inside the PSDirect VM is not connected to any switch. It also shows that the VM has assigned an IPV4 Automatic Private IP Address (APIPA) address to the NIC (in this case, 169.254.141.180). Since the address chosen by Windows is random, you may see a different address in the 169.254.0.0/16 network. Note that even though Windows has an IP address for the NIC, since you have not connected the VM's NIC to a Hyper-V switch, no networking is possible with the VM until you execute the subsequent steps that resolve that issue. In such cases, using PowerShell Direct, as shown in the Using PowerShell Direct recipe, is useful.

In step 3, you create a new switch. If you already have an External switch created on your Hyper-V host, you can use it in this recipe as opposed to the External switch created in this step.

In step 5 and step 6, you view the PSDirect VM's networking configuration after you created the External VM switch and added the NIC in the PSDirect VM to the switch. The initial IP address, which you saw in step 1, was automatically assigned when the VM was unable to obtain a DHCP address. After connecting the VM to the network, the VM was able to contact the DHCP server that you set up in the Installing and authorizing a DHCP server recipe, and configured in the Configuring DHCP Scopes recipe. This DHCP server is used to get an IP address for the PSDirect VM.

In step 7, you obtain the VM's configured hostname. If you created the VM simply from the default installation via the product DVD, Windows automatically creates a hostname, such as WIN-O5LPHTHBB5U. In the Using PowerShell Direct recipe, you installed Windows Server 2019 to the PSDirect VM and gave the VM a host name of Tiger. In this recipe, regardless of the current hostname, you update the hostname to Wolf.

# Implementing nested Hyper-V

Nested Hyper-V is a cool feature of Windows 2019 and Windows 10 (Anniversary Update and later). Nested Hyper-V enables a Hyper-V VM to host VMs that also have virtualization enabled. You could, for example, take a physical host (say, HV1) and on that host run a VM (PSDirect). With nested Hyper-V, you could install Hyper-V in the PSDirect VM to enable that VM to host further VMs.

Nested VMs have a number of uses. First, nested VMs 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. Nested Hyper-V is also useful for testing and education/training. In a training course, you could give a student one VM (running in a large blade server) and enable the student to create additional VMs as part of the course. You could, for example, run 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 NestedVM VM. This recipe does not show the details of configuring the NestedVM, which are left as an exercise for the reader.

## Getting ready

This recipe uses the PSDirect VM running on the HV1 Windows Server 2019 system with Hyper-V loaded. You created the Hyper-V server in the Installing and configuring Hyper-V recipe. You also make use of the PSDirect VM, which you created in the Creating a virtual machine recipe. You also updated the VM's hostname (to Wolf) in the Configuring Hyper-V networking recipe.

## How to do it...

1. Stop the PSDirect VM:

Stop-VM -VMName PSDirect

1. Change and view 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. Start the PSDirect VM:

Start-VM -VMName PSDirect

Wait-VM -VMName PSDirect -For Heartbeat

Get-VM -VMName PSDirect

1. Create credentials for the PSDirect machine. Note that this assumes the VM has been renamed Wolf:

$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. Create a script block for remote execution:

$SB = {

Install-WindowsFeature -Name Hyper-V -IncludeManagementTools

}

1. Install Hyper-V inside the PSDirect VM:

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

$IHT = @{

Session = $Session

ScriptBlock = $SB

}

Invoke-Command @IHT

1. Restart the VM to finish adding Hyper-V to the PSDirect VM:

Stop-VM -VMName PSDirect

Start-VM -VMName PSDirect

Wait-VM -VMName PSDirect -For IPAddress

1. Create a nested VM inside the PSDirect VM:

$SB2 = {

$VMname = 'NestedVM'

New-VM -Name $VMname -MemoryStartupBytes 1GB

}

$IHT2 = @{

VMName = 'PSDirect'

ScriptBlock = $SB2

}

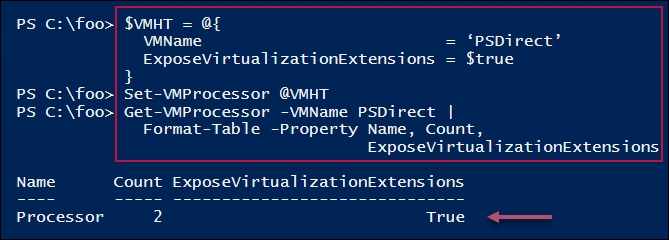
Invoke-Command @IHT2 -Credential $CredRK

## How it works...

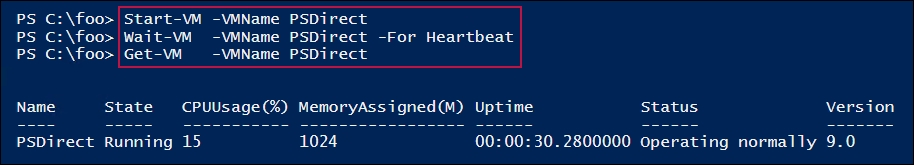
This recipe stops, reconfigures, then restarts the PSDirect VM. Once the VM is running, you create an embedded VM (NestedVM) inside the PSDirect VM.

In step 1, you stop the PSDirect VM on the HV1 server, which produces no output.

In step 2, you update the virtual CPU in the PSDirect VM to expose the virtualization extensions (in the PSDirect VM), which looks like this:



In step 3, you restart the PSDirect VM, which looks like this:



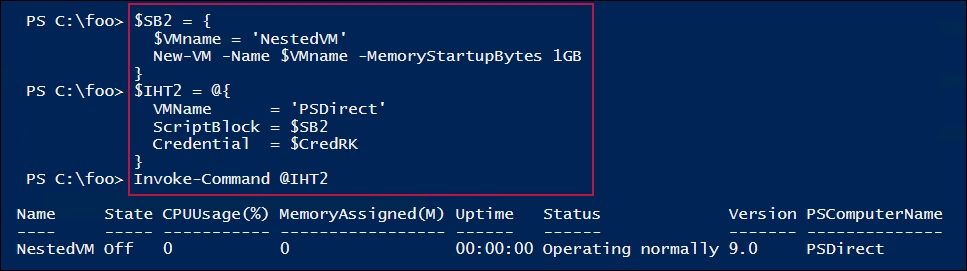
In step 4, you create a credentials object. In step 5, you create a script block for execution on PSDirect. These two steps produce no output.

In step 6, you install the Hyper-V feature inside the PSDirect VM, adding the management tools, which looks like this:



In step 7, after you install the Hyper-V to the PSDirect virtual machine, you reboot the VM, then wait for it to start. There is no output from this step.

In step 8, you create a VM, NestedVM, inside the PSDirect VM, which looks like this:



## There's more...

In step 2, you look at the properties of the virtual CPU(s) in the PSDirect VM. If you have not yet executed the Configuring VM hardware recipe previously, you may see a different CPU count.

In step 5, you stopped then started the PSDirect VM. As an alternative, you could have used the Restart-VM cmdlet.

In step 6, you create a new VM, but you have not loaded an operating system or configured the VM. Naturally, you can use the techniques in this chapter to configure your new VM as needed.

# Managing VM state

Managing the VM state involves stopping and starting, or pausing and resuming, a VM. You can also save and restore, as shown in this recipe.

## Getting ready

This recipe uses the PSDirect VM created in the Creating a virtual machine recipe. This recipe assumes the PSDirect VM is stopped when you start this recipe. If this VM is running, then first stop it using the Stop-VM cmdlet.

## How to do it...

1. Get the VM's state to check if whether is off:

Stop-VM -Name PSDirect -WarningAction SilentlyContinue

Get-VM -Name PSDirect

1. Start the VM, get its status, then wait until the VM has an IP address assigned and the networking stack is working, then examine the VM's state:

Start-VM -VMName PSDirect

Wait-VM -VMName PSDirect -For IPAddress

Get-VM -VMName PSDirect

1. Suspend and view the PSDirect VM:

Suspend-VM -VMName PSDirect

Get-VM -VMName PSDirect

1. Resume the VM:

Resume-VM -VMName PSDirect

Get-VM -VMName PSDirect

1. Save the VM and check its status:

Save-VM -VMName PSDirect

Get-VM -VMName PSDirect

1. Resume the saved VM and view the status:

Start-VM -VMName PSDirect

Get-Vm -VMName PSDirect

1. Restart the VM:

Restart-VM -VMName PSDirect -Force

Get-VM -VMName PSDirect

1. Wait for the VM to get an IP address:

Wait-VM -VMName PSDirect -For IPaddress

Get-VM -VMName PSDirect

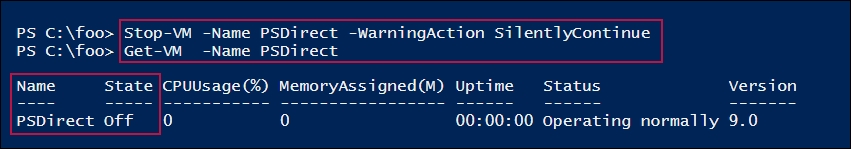
1. Perform a hard power off on the VM:

Stop-VM -VMName PSDirect -TurnOff

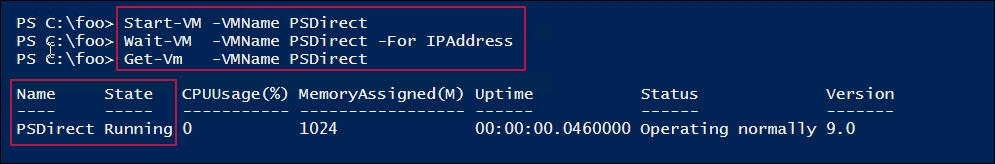
Get-VM -VMname PSDirect

## How it works...

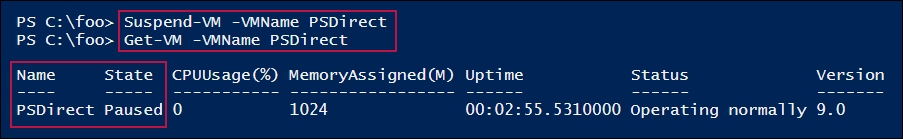
This recipe shows you how to manage VM state. In step 1, you view the properties of the VM, which is not running. As you can see from the screenshot, the PSDirect VM is turned off and is not running (and has an uptime of 00:00:00), as follows:



With step 2, you start the PSDirect VM and retrieve the VM's status, which looks like this:



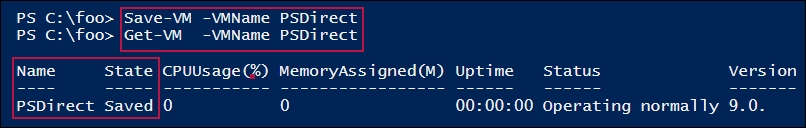
Next, in step 3, you suspend the PSDirect VM, which looks like this:



In step 4, you resume the PSDirect VM, which looks like this:



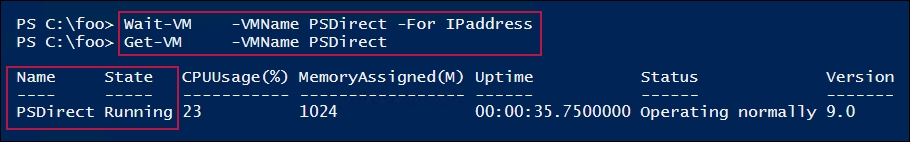
With step 5, you save a VM and view the status, like this:



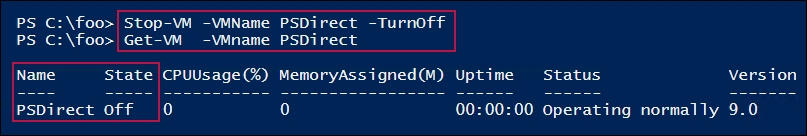
With step 6, you start the saved PSDirect VM and view the VM's details, which looks like this:



With step 7, you forcibly shut down the VM, which looks like this:



In step 8, you wait until the PSDirect VM is up and running to re-view the VM's status, which looks like this:



## There's more...

With step 2, you start the VM and retrieve the VM's status. Next, in step 3, you suspend then resume a VM.

With step 3 and step 4, you suspend then resume the PSDirect VM. While the VM is suspended, the VM is not active and therefore receives and sends no network traffic. The VM's memory is maintained, as is the current state, and the VM can be resumed at any moment.

In step 6 you save the PSDirect VM. When you save a VM, Hyper-V saves the VM's memory to disk and the VM's virtual disks are not used. Saving a VM is similar to pausing it, except that with a saved VM, all the VM's memory is written to disk then released.

In step 7, you perform a forced shutdown of the PSDirect VM. This is equivalent to pulling the power from a running computer then restarting it. When you do this, all VM state is lost, and it is possible to introduce disk corruption due to data being still in memory and not written to disk prior to the power off. While Windows and the most-used Windows filesystems (NTFS and ReFS) are fairly resilient to errors, you should avoid hard shutdown if possible.

# Configuring VM and storage movement

Hyper-V enables you to move VM details and VM 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.

## Getting ready

In this recipe, you are going to move configuration details for the PSDirect VM within the HV1 server. Then, you move the entire VM to another server, HV2, and view the results. The two Hyper-V servers, HV1 and HV2, were set up in the Installing and configuring the Hyper-V recipe. The VM, PSDirect, was created in the Create a virtual machine recipe.

## How to do it...

1. View the PSDirect VM on HV1 and verify that it is turned off:

Get-VM -Name PSDirect -Computer HV1

1. Get the VM configuration location:

(Get-VM -Name PSDirect).ConfigurationLocation

1. Get virtual hard drive locations:

Get-VMHardDiskDrive -VMName PSDirect |

Format-Table -Property VMName, ControllerType, Path

1. Move the VM's storage to the C:\PSDirectNew folder:

$MHT = @{

Name = 'PSDirect'

DestinationStoragePath = 'C:\PSDirectNew'

}

Move-VMStorage @MHT

1. View 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. Get the VM details for the VMs from HV2:

Get-VM -ComputerName HV2

1. Enable VM migration from both HV1 and HV2:

Enable-VMMigration -ComputerName HV1, HV2

1. Configure VM migration on both hosts:

$SVHT = @{

UseAnyNetworkForMigration = $true

ComputerName = 'HV1', 'HV2'

VirtualMachineMigrationAuthenticationType = 'Kerberos'

VirtualMachineMigrationPerformanceOption = 'Compression'

}

Set-VMHost @SVHT

1. Move the 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

1. Display the time taken to migrate:

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

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

1. Check which VMs on are on HV1:

Get-VM -ComputerName HV1

1. Check the VMs on HV2:

Get-VM -ComputerName HV2

1. Look at the details of the moved VM:

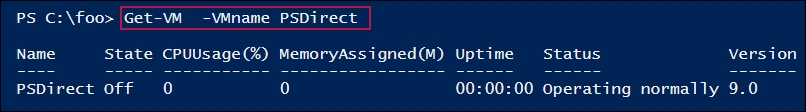
((Get-VM -Name PSDirect -Computer HV2).ConfigurationLocation)

Get-VMHardDiskDrive -VMName PSDirect -Computer HV2 |

Format-Table -Property VMName, Path

## How it works...

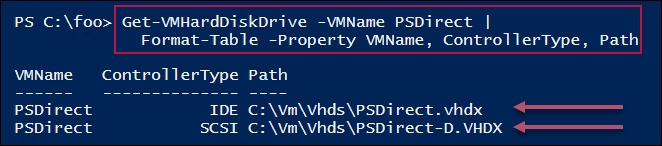
In step 1, you ensure that the PSDirect VM is stopped, which looks like this:



In step 2, you view the VM configuration location for the PSDirect VM, which looks like this:

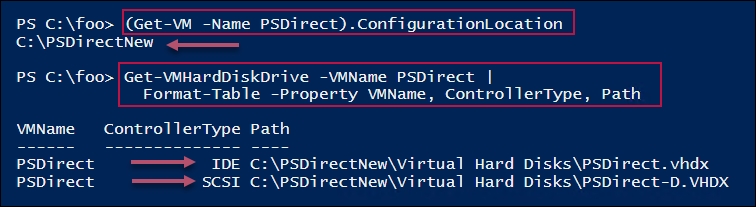
How it works...

With step 3, you view the details of the two virtual disk drives used by the PSDirect VM. You created one drive when you installed the PSDirect virtual machine (using the Creating a virtual machine recipe), and you created the second drive based on the Configuring VM hardware recipe. The output of this step looks like this:

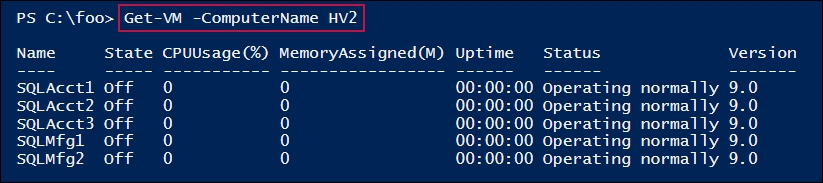


In step 4, you move the PSDirect VM and the VMs hard drives to the C:\PSDirectNew folder. There is no output from this step.

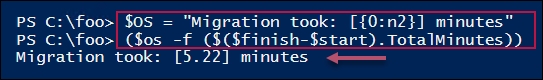
With step 5, you get the new location for the PSDirect VM's configuration files and get the details of the VM's disks, including their new location, which looks like this:



Before you migrate the PSDirect VM, in step 6, you view the VMs currently residing on HV2, which looks like this:

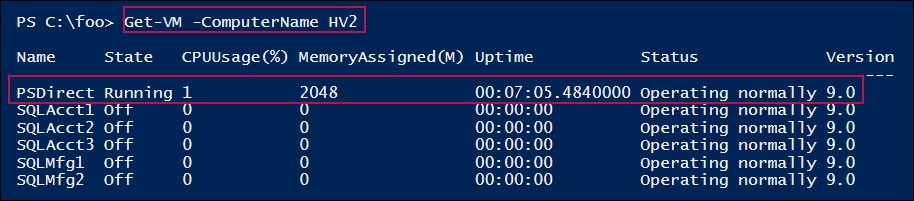


In step 7, you enable VM migration on both HV1 and HV2. In step 8, you configure VM migration on both servers. In step 9, you perform a migration of the PSDirect VM from HV1 to HV2. These three steps produce no output. In step 10, you display how long the migration took, which looks like this:

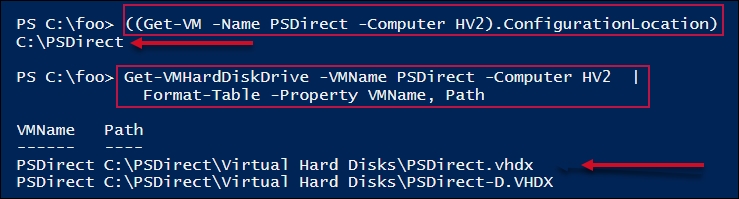


With the migration of the PSDirect VM to HV2 completed, in step 11, you get the VMs on HV1. Since the PSDirect VM has been migrated, you have no VMs running in HV1, thus there is no output from this step.

In step 12, you get the VMs running on HV2, which shows the PSDirect VM is now running on this server (in addition to the five VMs you saw in step 6), as follows:



In the final step, you examine the configuration location for the PSDirect VM (on HV2), and the disk details for this VM's two disks, which looks like this:



## There's more...

In this recipe, you first view the storage location details of a VM, before moving the VM and again after. You first move the PSDirect VM within the HV1 VM, then you move the VM to a different server, HV2.

In step 3, you moved the storage for the PSDirect VM. If you had an RDP connection open to the PSDirect VM, you would have seen the VM functioning normally during the migration. You may have seen a brief flicker as the VM movement completes and Hyper-V begins to use the new VM details.

In this recipe, HV1 and HV2 are two non-clustered systems. In step 8, you move the PSDirect VM from HV1 to HV2. In this case, since there is no shared storage involved with the VMs, Hyper-V needs to perform a full storage migration migrating the storage between the two servers. If you store the VM on shared storage, for example, using an SMB Scale-Out File Server or storing VHDX files on an iSCSI server, moving a VM between cluster nodes is significantly faster.

At the completion of the VM movement in step 8, Hyper-V drops connectivity to the VM on HV1 and establishes it on HV2. This means that for a moment, you lose connectivity to the VM. If you open an RDP connection into the PSDirect VM before you move the VM, you can see that as the movement finishes, the connection stops for a moment, then reappears with the VM running on HV2. Using VMConnect would mean having to reopen the connection on the other Hyper-V host.

You could also open up a PowerShell window on another system, say DC1, and ping the VM continuously during the movement of the VM. You may notice a moment of dropped pings, before they pick up again once the live migration has completed. An application running on the PSDirect VM would be unaffected by the momentary loss of connectivity during the move.

# Configuring VM replication

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

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 it. 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 replicates the PSDirect VM running in HV2 to HV1. This assumes you have used the Creating a virtual machine recipe to create the VM, and used the Configuring VM and storage movement recipe; you need to move the PSDirect VM to HV2.

## How to do it...

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

$SB1 = {

Set-ADComputer -Identity HV1 -TrustedForDelegation $True

}

Invoke-Command -ComputerName DC1 -ScriptBlock $SB1

$SB2 = {

Set-ADComputer -Identity HV2 -TrustedForDelegation $True}

Invoke-Command -ComputerName DC1 -ScriptBlock $SB2

1. Reboot the HV1 and HV2 servers:

Restart-Computer -ComputerName HV1 -Force

Restart-Computer -ComputerName HV2 -Force

1. Once both systems are restarted, log back on to HV2, and set up both servers as a replication server:

$VMRHT = @{

ReplicationEnabled = $true

AllowedAuthenticationType = 'Kerberos'

KerberosAuthenticationPort = 42000

DefaultStorageLocation = 'C:\Replicas'

ReplicationAllowedFromAnyServer = $true

ComputerName = 'HV1', 'HV2'

}

Set-VMReplicationServer @VMRHT

1. Enable PSDirect on HV2 to be a replica source:

$VMRHT = @{

VMName = 'PSDirect'

Computer = 'HV2' # from server

ReplicaServerName = 'HV1' # to server

ReplicaServerPort = 42000

AuthenticationType = 'Kerberos'

CompressionEnabled = $true

RecoveryHistory = 5

}

Enable-VMReplication @VMRHT

1. View the replication status of HV2:

Get-VMReplicationServer -ComputerName HV2

1. Check the PSDirect VM's status on HV2:

Get-VM -ComputerName HV2 -VMName PSDirect

1. Start the initial replication:

Start-VMInitialReplication -VMName PSDirect -ComputerName HV2

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

Measure-VMReplication -ComputerName HV2

1. Wait for replication to finish, then examine the replication status on HV2:

Measure-VMReplication -ComputerName HV2

1. Test PSDirect failover to HV1:

$SB = {

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

Start-VM $VM

}

Invoke-Command -ComputerName HV1 -ScriptBlock $SB

1. View the status of VMs on HV1:

Get-VM -ComputerName HV1

1. Stop the failover test on HV1:

$SB = {

Stop-VMFailover -VMName PSDirect

}

Invoke-Command -ComputerName HV1 -ScriptBlock $SB

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

Get-VM -ComputerName HV1

Get-VM -ComputerName HV2

1. Stop PSDirect on HV2 prior to performing a planned failover:

Stop-VM PSDirect -ComputerName HV2

1. Start VM failover from HV1:

$STHT = @{

VMName = 'PSDirect'

ComputerName = 'HV1'

Confirm = $false

}

Start-VMFailover @STHT

1. Complete the failover:

$CHT = @{

VMName = 'PSDirect'

ComputerName = 'HV1'

Confirm = $false

}

Complete-VMFailover @CHT

1. Start the replicated VM on HV1:

Start-VM -VMname PSDirect -ComputerName HV1

1. View the VMs on HV1 and HV2 after the failover:

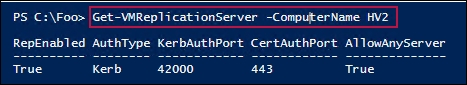
Get-VM -ComputerName HV1

Get-VM -ComputerName HV2

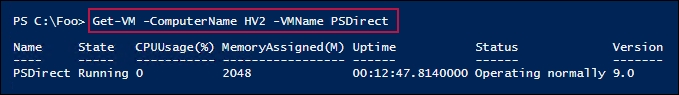
## How it works...

In step 1, you configure HV1 and HV2 to be trusted for delegation in AD on DC1. After configuring delegation, in step 2, you restart both servers to enable the delegation trust. With step 3, you set replication settings, and in step 4, you complete establishing replication between HV2 an HV1. These four steps produce no output.

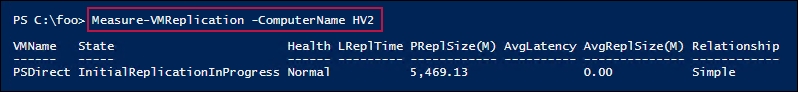
In step 5, you view the replication status on HV2, which looks like this:



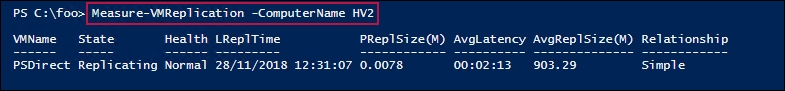
In step 6, you view the VM details of the PSDirect VM, running on HV2, which looks like this:



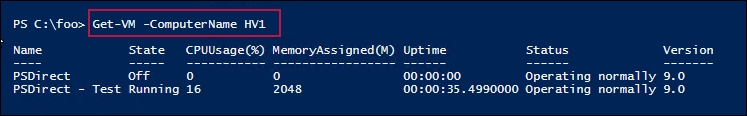
In step 7, you start the replication of PSDirect from HV2 to HV1. This produces no output. In step 8, just after you started replication, you take another look at the replication status, which looks like this:



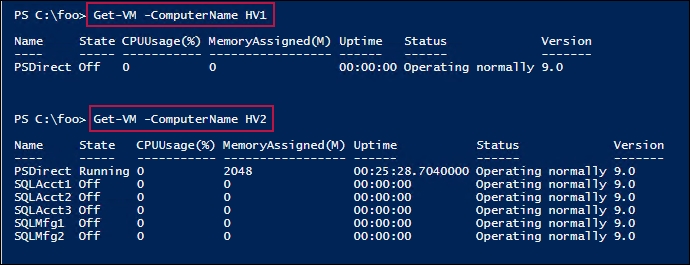
After the VM replication has completed, in step 9, you re-check the replication status, which looks like this:



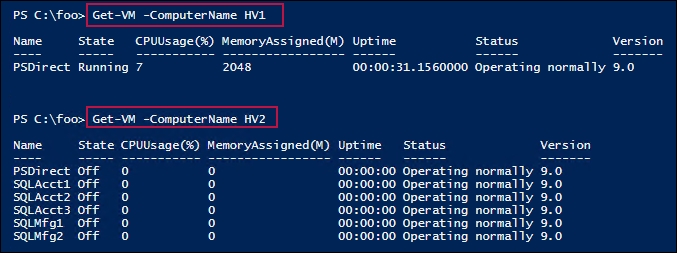
In step 10, you test a failover of PSDirect between the Hyper-V servers. This step produces no output. In step 11, you view the status of the failover test on HV1, which looks like this:



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



In step 14, you shut down the PSDirect VM on HV2 in preparation for a failover. In step 15, you begin the failover. In step 16, you complete the failover. After the VM failover has completed, in step 17, you start the PSDirect VM on HV1. These three steps produce no output. The result of these four steps, which produce no output, is that the PSDirect VM is now running on HV1. You can view this in step 18, where you review the VMs running on both Hyper-V servers. The output of this step looks like this:



## There's more...

In step 2, you restart both Hyper-V hosts (HV1 and HV2). Since you run this recipe from the HV2 machine, the second command in this step reboots the HV2 system you are working on immediately (there is no prompt to ask you if you are sure or to save any work). If you test this recipe, make sure you have any files saved before you reboot.

In step 3, you configure HV1 to accept inbound replication from any Hyper-V system. If you have not configured the host firewalls (or turned them off) you may see errors trying to invoke replication.

You may also wish to configure HV2 to accept replication only from a specific server, such as HV1. To do this, you would have to set up the replication server to not accept replication from any server. Then, you use the Hyper-V cmdlet New-VMReplicationAuthorizationEntry to specify that HV2 can only receive replicas from the HV1 server. To set this up, you would do the following:

$RHT = @{

AllowedPrimaryServer = 'HV1'

ReplicaStorageLocation = 'C:\Replica'

ComputerName = 'HV2'

}

Set-VMReplicationAuthorizationEntry @RHT

In step 12, you view the status of VMs on both HV1 and HV2. As you can see, the PSDirect VM is up and running (on HV2) with a replica VM on HV1, which is turned off.

In step 11, you view the details of the test version of the PSDirect VM running. If you were to open up a VMConnect window on this test VM, you see that both the hostname and the IP address are not the same as the VM running on.

After step 17, if you looked inside PSDirect, running this time on HV1, you would find the same issue. The impact is that after a real-life failover, you may need to reset the hostname and reset the IP configuration details. If you are using Hyper-V replica in production, it would be a great idea to develop a script to fix these two issues in an automated fashion.

In step 18, you see that the PSDirect VM is running on HV1 and stopped on HV2. However, if you look inside the PSDirect VM running on HV1, you see it has a hostname that is not PSDirect and has no networking setup. If you were to fail over and wanted to run the failed-over VM, you would need to deal with these two issues, which typically involves re-booting the PSDirect VM and possibly running a script to ensure the hostname and the IP configuration details are correct for your environment.

# Managing VM checkpoints

With Hyper-V in Server 2019, 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 first provided this feature. With Server 2008, these restore points were called snapshots. With Server 2012, Microsoft also 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.

While 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 multiple checkpoints for a VM.

Checkpoints are great for a variety of scenarios. They can be useful for troubleshooting. For example, if you are troubleshooting, you can get the VM to the point where some bug is triggered, then take a checkpoint. With the checkpoint taken, you can try a fix. If the fix does not 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 PSDirect, then you create a file in the VM. You take a further checkpoint and create a second file, after reverting 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, which you run on HV1, uses the PSDirect VM you created and used earlier in this chapter. The VHDX files for the VM reside in C:\Vm\Vhds\PSDirect. Depending on which other recipes you have run from this chapter, the virtual disks may be in a different folder, but the recipe copes with the disk files being in any folder (known to Hyper-V).

## How to do it...

1. Create 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. Look at C\: in the PSDirect VM at the start:

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

$ICHT = @{

VMName = 'PSDirect'

ScriptBlock = $SB

Credential = $RKCred

}

Invoke-Command @ICHT

1. Create a snapshot of the PSDirect VM on HV1:

$CPHT = @{

VMName = 'PSDirect'

ComputerName = 'HV1'

SnapshotName = 'Snapshot1'

}

Checkpoint-VM @CPHT

1. Look at the files created to support the checkpoints:

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

Select-Object -ExpandProperty HardDrives).Path |

Select -First 1

Get-ChildItem -Path $Parent

1. Create some content in a file on PSDirect and display 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. Take a second checkpoint of the PSDirect VM:

$SNHT = @{

VMName = 'PSDirect'

ComputerName = 'HV1'

SnapshotName = 'Snapshot2'

}

Checkpoint-VM @SNHT

1. Get the VM checkpoint details for PSDirect:

Get-VMSnapshot -VMName PSDirect

1. Look at the files supporting the two checkpoints:

Get-ChildItem -Path $Parent

1. Create and display another file in PSDirect (after you have taken Snapshot2):

$SB = {

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

Get-Date | Out-File -FilePath $FileName2

Get-ChildItem -Path C:\ -File

}

$ICHT = @{

VMName = 'PSDirect'

ScriptBlock = $SB

Credential = $RKCred

}

Invoke-Command @ICHT

1. Restore 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. See what files we now have on PSDirect:

$SB = {

Get-ChildItem -Path C:\

}

$ICHT = @{

VMName = 'PSDirect'

ScriptBlock = $SB

Credential = $RKCred

}

Invoke-Command @ICHT

1. Roll 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. Observe the files you now have in the PSDirect VM:

$SB = {

Get-ChildItem -Path C:\

}

$ICHT = @{

VMName = 'PSDirect'

ScriptBlock = $SB

Credential = $RKCred

}

Invoke-Command @ICHT

1. Restore the VM to 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. Check snapshots and VM data files again:

Get-VMSnapshot -VMName PSDirect

Get-ChildItem -Path $Parent

1. Remove all the snapshots from HV1:

Get-VMSnapshot -VMName PSDirect |

Remove-VMSnapshot

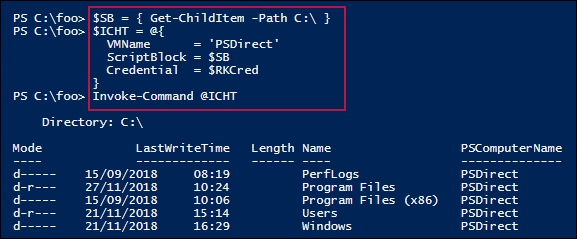
1. Check VM data files again:

Get-ChildItem -Path $Parent

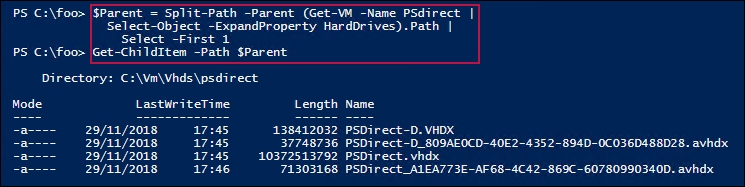
## How it works...

In step 1, you create a credentials object for the PSDirect VM's local administrator account. There is no output from this step.

In step 2, you examine the C:\ drive in the PSDirect VM, which looks like this:



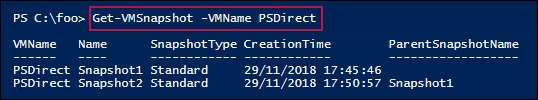
In step 3, you create a checkpoint of PSDirect on HV1, which produces no output. With step 4, examine the files supporting the PSDirect VM after the checkpoint is taken, which looks like this:



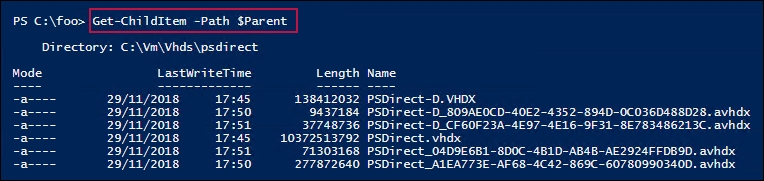
In step 5, you create a file (after the first checkpoint is taken) and file contents, then view the file, which looks like this:



After creating the file, in step 6, you take another checkpoint for the PSDirect VM, which produces no output. With step 7, you view the VM snapshots available for the PSDirect VM, which looks like this:



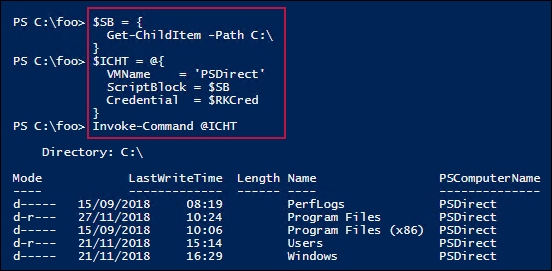
In step 8, you again view the files supporting the PSDirect VM, which looks like this:



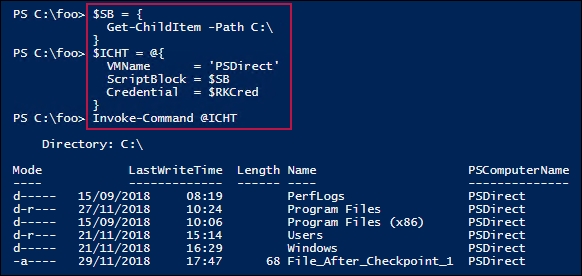
After reverting the PSDirect VM back to the first checkpoint, in step 9, you create a further file in the C:\ folder inside the PSDirect VM, which looks like this:



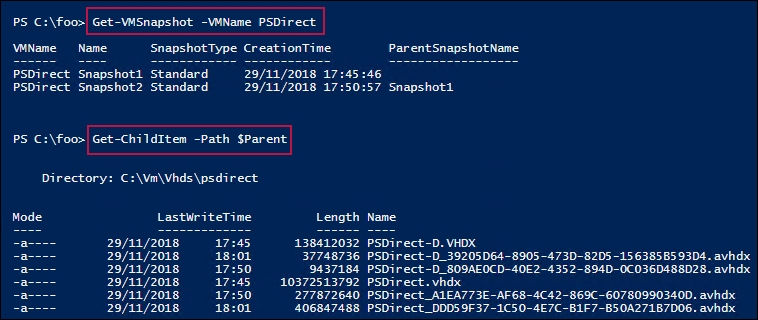
Without saving any work done, in step 10, you revert the PSDirect VM back to the first snapshot, Snapshot1, which creates no output. Once the PSDirect VM is restarted from Snapshot1, in step 11, you look at the files now in the PSDirect VM's C:\ drive, which looks like this:



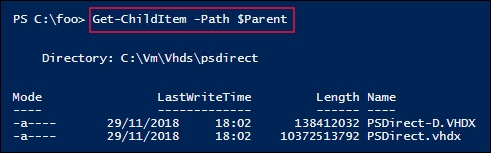
In step 12, you roll the VM forward to the second checkpoint, Snapshot2. This step produces no output. With step 13, you view the files in the root of the C: drive of the PSDirect VM, which looks like this:



In step 14, you re-restore the PSDirect VM back to the first snapshot, Snapshot1, then wait until the VM is restarted. This step produces no output. In step 15, you re-view the PSDirect VM's snapshots and the snapshot files, which looks like this:



In step 16, you remove all the snapshots from the PSDirect virtual machine, which produces no output. In step 17, you view the VHDX files for the PSDirect VM (now that all snapshots are removed), which looks like this:



## There's more...

In step 10, you revert the PSDirect VM back to an earlier snapshot. Any file activity in the VM after you took the second snapshot, and before reverting to the earlier snapshot, is now lost. In step 12, you roll forward the PSDirect VM to the second snapshot. This loses all work you might have done after reverting to Snapshot1 and before rolling forward to Snapshot2. Be very careful when reverting to earlier snapshots or rolling forward.