



```
Windows PowerShell
PS C:\>
PS C:\> get-vm web01 -computername chi-hvr2 | select -ex
hvr2

ComputerName      : chi-hvr2
Path              : E:\Users\Public\Documents\Hyper-V\
VhdFormat        : VHDX
VhdType          : Dynamic
FileSize         : 4194304
Size             : 10737418240
MinimumSize      :
LogicalSectorSize : 512
PhysicalSectorSize : 4096
BlockSize       : 33554432
ParentPath       :
DiskIdentifier    : bb4e49df-8d54-4624-86eb-4cef0
FragmentationPercentage : 0
Alignment        : 1
Attached         : False
DiskNumber       :
Key              :
IsDeleted        : False
Number           :
```

THE ALTARO POWERSHELL HYPER-V COOKBOOK

Brought to you by **Altaro Software**,
developers of [Altaro VM Backup](#)

by **Jeffery Hicks**

Table of contents

INTRODUCTION	3
REQUIREMENTS AND SETUP	3
HYPER-V CMDLET BASICS	5
CREATING A VIRTUAL MACHINE	5
Using a Template	6
Using an ISO File	12
VIRTUAL MACHINE INVENTORY	20
Virtual Machines	20
Get-Newest Virtual Machines	24
Hard Disk Report	24
Memory Usage	29
Get VM Last Use	34
Get VM Operating System	37
GET MOUNTED ISO FILES	41
IDENTIFYING ORPHANED VHD/VHDX FILES	42
DELETING OBSOLETE SNAPSHOTS	45
QUERYING HYPER-V EVENT LOGS	49
A HYPER-V HEALTH REPORT	57
TIPS AND TRICKS	60
ADDITIONAL RESOURCES	61
ABOUT THE AUTHOR	62
ABOUT ALTARO	62

Introduction

For most Windows administrators, the Hyper-V management console is more than sufficient. However you are limited to what the console is designed to handle. For more complex tasks, or those that might extend beyond the scope of the management console, you need an alternative. That alternative is Windows PowerShell.

With PowerShell, if you can work with one item, such as a virtual machine, VHD or snapshot, you can work with 10, 100 or 1000 with practically very little extra effort. This e-book will include a number of recipes for managing Hyper-V using Windows PowerShell. Some of the recipes are one or two line commands. Others, are more complicated scripts. You don't have to be a PowerShell expert to use the recipes, although certainly the greater your PowerShell experience, the more you will be able to take this recipes and customize them to meet your requirements. All of the recipes are included in an accompanying ZIP file. Many of these recipes were originally published on the Altaro Hyper-V Backup blog (<http://www.altaro.com/hyper-v/>) but I have revised and/or made some minor adjustments to many of the scripts.

Accompanying ZIP with scripts

To get to work with the recipes described in this eBook you can download all the scripts in a ZIP file here:



Requirements and Setup

Most of these recipes will require at least PowerShell 3.0, the Hyper-V PowerShell module and a Hyper-V server running Windows Server 2012. You are welcome to try any recipe using a Hyper-V server running an older operating system, but there are no guarantees they will work. All of the recipes were tested from a Windows 8.1 client running the Hyper-V role, which will give you the PowerShell cmdlets, and a Hyper-V server running Windows Server 2012 R2. Both the client and server are running PowerShell 4.0.

Ideally, you will want to do as much as you can from your client desktop. You don't have to have a local hypervisor, just the PowerShell tools. In Control Panel – Programs, click on “Turn Windows features on or off” and navigate down to “Hyper-V”. Be sure to check the box for the “Hyper-V module for Windows PowerShell” as shown in Figure 1.

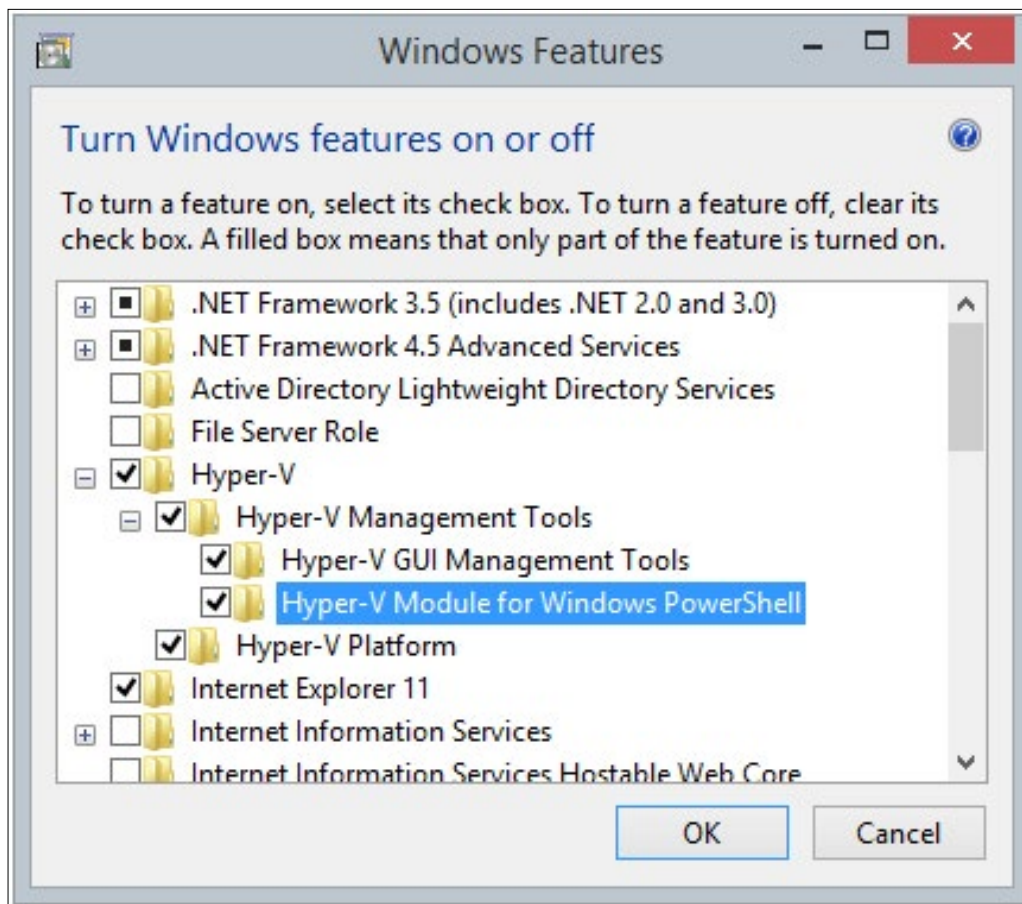


Figure 1

An alternative is to install the Remote Server Administration Tools and specify the Hyper-V role. Just remember to include the Hyper-V PowerShell module. <http://www.microsoft.com/en-us/search/DownloadResults.aspx?q=remote%20server%20administration%20tools>

I also recommend that your Hyper-V servers have PowerShell remoting enabled. The Hyper-V commands don't require it, but there will be some recipes where it is more efficient to run the command through a PSSession.

Many of the recipes consist of a PowerShell function that resides in a PowerShell script. In order to execute the function, you must first have a script execution policy that will allow you to run scripts. Then you will need to dot source the script file into your PowerShell session.

```
PS C:\> . .\scripts\New-VMFromTemplate.ps1
```

Now you can invoke any functions defined inside the script.

Please take note that none of these scripts or functions require anything other than the Hyper-V module, unless noted. These commands have not been tested in an environment with System Center Virtual Machine Manager, which has similar command names to the Hyper-V PowerShell module.

Finally, and I'd like to think it goes without saying, but you should test all of these recipes in a non-production environment. Neither I nor Altaro Software make any claims, warranties or guarantees about how these recipes will perform or behave in your environment. This is definitely "use at your own risk".

Hyper-V Cmdlet Basics

When you add the Hyper-V PowerShell module you get a number of commands. These commands follow the same syntax and structure as other PowerShell commands. The commands all have help and examples so learning how to use them is not that much different than any other PowerShell command. You can see all of the available cmdlets with this expression:

```
Get-Command -Module Hyper-V
```

Many of the cmdlets can be used in pairs such as Get-VM and Start-VM. For example, you can get a set of virtual machines that meet some criteria and then start them.

```
Get-VM chi* | where {$_.state -eq 'Off'} | Start-VM -AsJob
```

This command gets all virtual machines that start with CHI and that are not running and then starts them. The startup process runs as a background job so you get your PowerShell prompt back immediately. The Hyper-V cmdlets are using the local computer. Most of the Hyper-V cmdlets let you specify a remote Hyper-V server using the -Computername parameter.

```
Get-VM chi* -ComputerName chi-hvr2 | where {$_.state -eq 'Off'} | Start-VM -asjob
```

This is the exact same command except it will use virtual machines running on the server CHI-HVR2. The recipes throughout this book will use many of the Hyper-V cmdlets in the Hyper-V module. Sometimes as simple one-line commands and other times in more complex scripts or functions. Don't forget that to use any of the Hyper-V cmdlets, you must be running PowerShell in an elevated session as an administrator. If you feel that your PowerShell skills could use a little improving, I have some resources at the end of the book.

Let's start cooking.

Creating a Virtual Machine

One of the best uses of PowerShell is to provision a new virtual machine. The Hyper-V module has a cmdlet called New-VM. There is also a cmdlet for creating new virtual disks, New-VHD. There's nothing wrong with using these commands interactively, but I think you will get more out of them by using scripts and functions that utilize these commands.

Using a Template

Often you may have virtual machines that fall into different categories. The first recipe is a function that will create a virtual machine based on a pre-defined type of Small, Medium or Large. You will have to edit the script if you would like to change any of the settings.

New-VMFromTemplate.ps1

```
#requires -version 3.0
```

```
Function New-VMFromTemplate {
```

```
<#
```

```
.Synopsis
```

```
Provision a new Hyper-V virtual machine based on a template
```

```
.Description
```

```
This script will create a new Hyper-V virtual machine based on a template or hardware profile. You can create a Small, Medium or Large virtual machine. You can specify the virtual switch and paths for the virtual machine and VHDX files.
```

```
All virtual machines will be created with dynamic VHDX files and dynamic memory. The virtual machine will mount the specified ISO file so that you can start the virtual machine and load an operating system.
```

```
VM Types
```

```
Small (default)
```

```
    MemoryStartup=512MB
```

```
    VHDSIZE=10GB
```

```
    ProcCount=1
```

```
    MemoryMinimum=512MB
```

```
    MemoryMaximum=1GB
```

```
Medium
```

```
    MemoryStartup=512MB
```

```
    VHDSIZE=20GB
```

```
    ProcCount=2
```

```
    MemoryMinimum=512MB
```

```
    MemoryMaximum=2GB
```

```
Large
```

```
    MemoryStartup=1GB
```

```
    VHDSIZE=40GB
```

```
    ProcCount=4
```

```
    MemoryMinimum=512MB
```

```
    MemoryMaximum=4GB
```

```
This script requires the Hyper-V 3.0 PowerShell module.
```

```
.Parameter Path
```

```
The path for the virtual machine.
```

```
.Parameter VHDRoot
```

```
The folder for the VHDX file.
```

```
.Parameter ISO
```

```
The path to an install ISO file.
```

```
.Parameter VMSwitch
```

```
The name of the Hyper-V switch to connect the virtual machine to.
```

```
.Parameter Computername
```

```
The name of the Hyper-V server. If you specify a remote server, the command will attempt to make a remote PSSession and use that. Any paths will be relative to the remote computer.
```

```
Parameter Start
```

Start the virtual machine immediately.

.Example

```
PS C:\> New-VMFromTemplate WEB2012-01 -VMType Small -passthru
```

Name	State	CPUUsage(%)	MemoryAssigned(M)	Uptime	Status
WEB2012-01	off	0	0	00:00:00	Operating normally

.Example

```
PS C:\> New-VMFromTemplate -name DBTest01 -VMType Medium -ISO G:\ISO\win2k12R2.iso  
-computername SERVER02 -VHRoot F:\VHDS -start
```

This will create a Medium sized virtual machine on SERVER01 called DBTest. The VHDX file will be created in F:\VHDS. The virtual machine will be stored in the default location. An ISO file will also be mounted. After the virtual machine is created, it will be started.

.Notes

Version 2.0

Last Updated June 17, 2014

```
*****  
* DO NOT USE IN A PRODUCTION ENVIRONMENT UNTIL YOU HAVE TESTED *  
* THOROUGHLY IN A LAB ENVIRONMENT. USE AT YOUR OWN RISK. IF *  
* YOU DO NOT UNDERSTAND WHAT THIS SCRIPT DOES OR HOW IT WORKS, *  
* DO NOT USE IT OUTSIDE OF A SECURE, TEST SETTING. *  
*****
```

.Link

New-VM

Set-VM

#>

```
[cmdletbinding(SupportsShouldProcess)]
```

Param(

```
[Parameter(Position=0,Mandatory,HelpMessage="Enter the name of your new virtual machine")]
```

```
[ValidateNotNullOrEmpty()]
```

```
[string]$Name,
```

```
[ValidateSet("Small","Medium","Large")]
```

```
[string]$VMType="Small",
```

```
[ValidateNotNullOrEmpty()]
```

```
[string]$Path = (Get-VMHost).VirtualMachinePath,
```

```
[ValidateNotNullOrEmpty()]
```

```
[string]$VHRoot=(Get-VMHost).VirtualHardDiskPath,
```

```
[Parameter(HelpMessage="Enter the path to an install ISO file")]
```

```
[string]$ISO,
```

```
[string]$VMSwitch = "work Network",
```

```
[ValidateNotNullOrEmpty()]
```

```
[string]$Computername = $env:COMPUTERNAME,
```

```
[switch]$Start,
```

```
[switch]$Passthru
```

```
)
```

```
if ($Computername -eq $env:computername) {
```

```

#validate parameters here
if (-Not (Test-Path $Path)) {
    write-warning "Failed to verify VM path $path"
    #bail out
    Return
}
if (-Not (Test-Path $VHDRoot)) {
    write-warning "Failed to verify VHDRoot $VHDRoot"
    #bail out
    Return
}
if ($ISO -AND (-Not (Test-Path $ISO))) {
    write-warning "Failed to verify ISO path $ISO"
    #bail out
    Return
}
if (-Not (Get-VMSwitch -Name $VMSwitch -ErrorAction SilentlyContinue)) {
    write-warning "Failed to find VM Switch $VMSwitch on $computername"
    Return
}
write-verbose "Running locally on $Computername"
#path for the new VHDX file. All machines will use the same path.
$VHDPATH= Join-Path $VHDRoot "$($name)_C.vhdx"

write-verbose "Creating new $VMType virtual machine"

#define parameter values based on VM Type
Switch ($VMType) {
    "Small" {
        write-verbose "Setting Small values"
        $MemoryStartup=512MB
        $VHDSIZE=10GB
        $ProcCount=1
        $MemoryMinimum=512MB
        $MemoryMaximum=1GB
        Break
    }
    "Medium" {
        write-verbose "Setting Medium values"
        $MemoryStartup=512MB
        $VHDSIZE=20GB
        $ProcCount=2
        $MemoryMinimum=512MB
        $MemoryMaximum=2GB
        Break
    }
    "Large" {
        write-verbose "Setting Large values"
        $MemoryStartup=1GB
        $VHDSIZE=40GB
        $ProcCount=4
        $MemoryMinimum=512MB
        $MemoryMaximum=4GB
        Break
    }
    Default {

```



```

        write-verbose "why are you here?"
    }
} #end switch

Write-Verbose "Mem: $MemoryStartup"
Write-verbose "VHD: $VHDSize"
Write-Verbose "Proc: $ProcCount"

#define a hash table of parameters for New-VM
$newParam = @{
    Name=$Name
    SwitchName=$VMSwitch
    MemoryStartupBytes=$MemoryStartup
    Path=$Path
    NewVHDPATH=$VHDPATH
    NewVHDSIZEBytes=$VHDSIZE
    ErrorAction="Stop"
}
#define a hash table of parameters for Set-VM
$setParam = @{
    ProcessorCount=$ProcCount
    DynamicMemory=$True
    MemoryMinimumBytes=$MemoryMinimum
    MemoryMaximumBytes=$MemoryMaximum
    ErrorAction="Stop"
}
if ( $PSBoundParameters.ContainsKey("Passthru")) {
    Write-Verbose "Adding Passthru to Set parameters"
    $setParam.Add("Passthru",$True)
    Write-Verbose ($setParam | out-string)
}
Try {
    Write-Verbose "Creating new virtual machine $name"
    Write-Verbose ($newParam | out-string)
    $VM = New-VM @newparam
}
Catch {
    Write-Warning "Failed to create virtual machine $Name"
    Write-Warning $_.Exception.Message
    #bail out
    Return
}
if ($VM) {
    If ($ISO) {
        #mount the ISO file
        Try {
            Write-Verbose "Mounting DVD $iso"
            Set-VMVDvdDrive -vmname $vm.name -Path $iso -ErrorAction Stop
        }
        Catch {
            Write-Warning "Failed to mount ISO for $Name"
            Write-Warning $_.Exception.Message
            #don't bail out but continue to try and configure virtual machine
        }
    } #if iso
}

```

```

Try {
    write-verbose "Configuring new virtual machine $name"
    write-verbose ($setParam | out-string)
    $VM | Set-VM @setparam
}
Catch {
    write-warning "Failed to configure virtual machine $Name"
    write-warning $_.Exception.Message
    #bail out
    Return
}
If ($Start) {
    write-verbose "Starting the virtual machine"
    Start-VM -VM $VM
}
} #if $VM
} #if local
else {
    write-verbose "Running Remotely"

    #create a PSSession
    Try {
        $sess = New-PSSession -ComputerName $Computername -ErrorAction Stop

        write-verbose "copy the function to the remote session"
        $thisFunction = ${function:New-VMFromTemplate}
        Invoke-Command -ScriptBlock {
            Param($content)
            New-Item -Path Function:New-VMFromTemplate -value $using:thisfunction -Force |
            Out-Null
        } -Session $sess -ArgumentList $thisFunction

        write-verbose "invoke the function with these parameters"
        write-verbose ($PSBoundParameters | Out-String)

        Invoke-Command -ScriptBlock {
            Param([hashtable]$Params)
            New-VMFromTemplate @params
        } -session $sess -ArgumentList $PSBoundParameters
    } #Try
    Catch {
        write-warning "Failed to create a remote PSSession to $computername. $($_.Exception.
message)"
    }
    #remove the PSSession
    write-verbose "Removing pssession"
    $sess | Remove-PSSession -WhatIf:$False
}
write-verbose "Ending command"
} #end function

```

This command can be run locally or you can specify a computer name. If you specify a remote computer name, the command will create a temporary PSSession and invoke necessary commands remotely. The function will use the default location for the new VHDX file and

virtual machine unless you specify alternate locations. Optionally, you can also specify an ISO file. This file will be “loaded” into the virtual machine’s DVD drive which will be configured to boot from the media. The end result is that when you start the virtual machine, the setup process will kick off immediately. Be aware that any paths are relative to the computer.

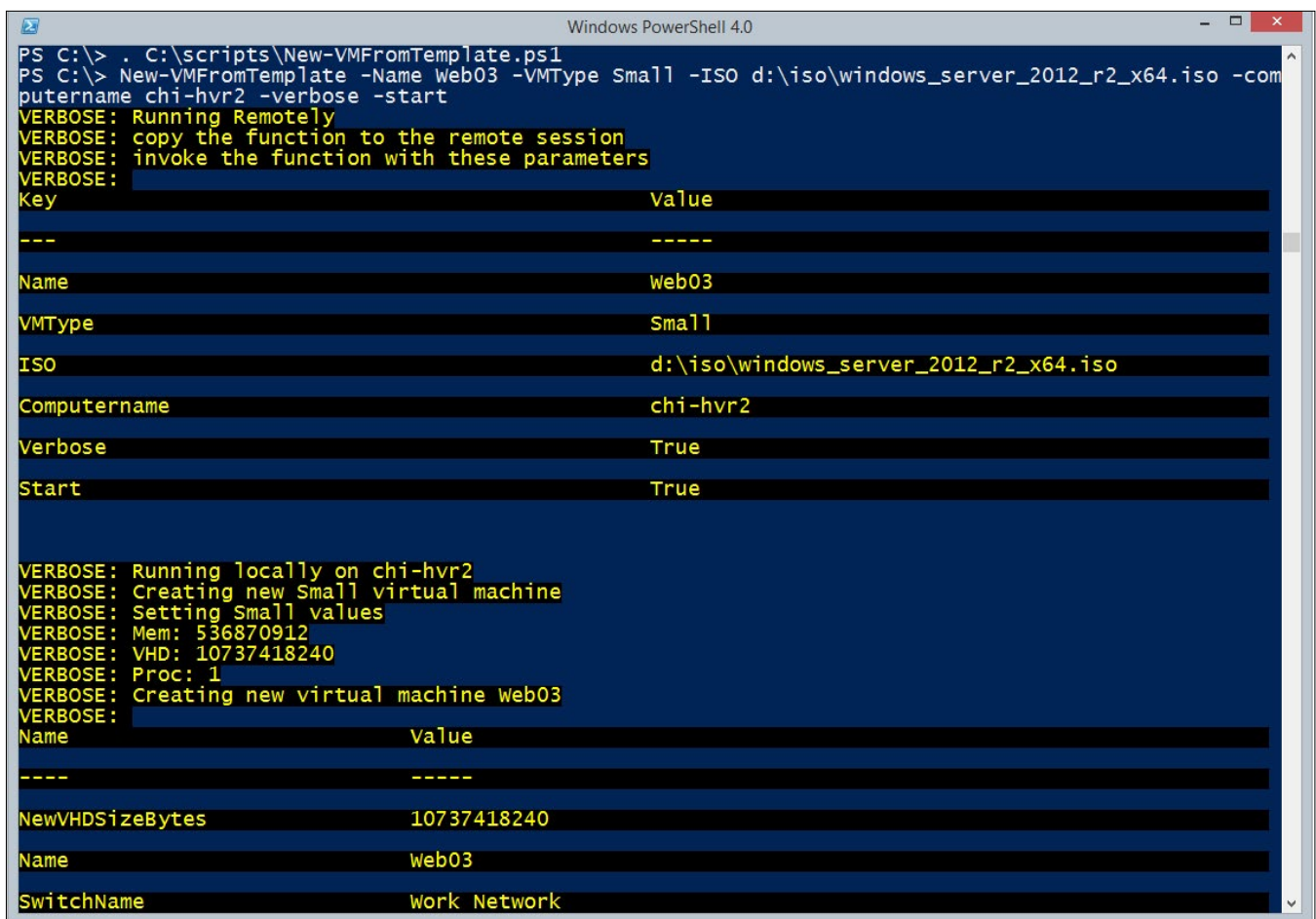
Here’s the command in action. Because this is the first time using the function, I need to dot source it.

```
PS C:\> . C:\scripts\New-VMFromTemplate.ps1
```

The function has complete help like any other cmdlet, including examples. Here is how I created a “small” virtual machine on a remote server that also automatically mounted the Windows Server 2012 ISO file that is on the remote Hyper-V server. After the virtual machine is created it is automatically started.

```
PS C:\> New-VMFromTemplate -Name web03 -VMType Small -ISO d:\iso\windows_server_2012_r2_x64.iso -computername chi-hvr2 -verbose -start
```

Like many of the recipes, this command has verbose output if you want to trace the command as you can see in Figure 2.



```
Windows PowerShell 4.0
PS C:\> . C:\scripts\New-VMFromTemplate.ps1
PS C:\> New-VMFromTemplate -Name web03 -VMType Small -ISO d:\iso\windows_server_2012_r2_x64.iso -computername chi-hvr2 -verbose -start
VERBOSE: Running Remotely
VERBOSE: copy the function to the remote session
VERBOSE: invoke the function with these parameters
VERBOSE:
Key Value
---
Name Web03
VMType Small
ISO d:\iso\windows_server_2012_r2_x64.iso
Computername chi-hvr2
Verbose True
Start True

VERBOSE: Running locally on chi-hvr2
VERBOSE: Creating new Small virtual machine
VERBOSE: Setting Small values
VERBOSE: Mem: 536870912
VERBOSE: VHD: 10737418240
VERBOSE: Proc: 1
VERBOSE: Creating new virtual machine Web03
VERBOSE:
Name Value
----
NewVHDSizeBytes 10737418240
Name web03
SwitchName Work Network
```

Figure 2

Using an ISO File

A potential drawback to the last recipe is that you still need to setup Windows in the virtual machine. Another option is to create a virtual machine directly from an ISO file. In order to do this, you will need to know name of the Windows edition on the ISO file. Here is a PowerShell function that uses the Storage and DISM modules in PowerShell 4 available on Windows 8/ Windows Server 2012 and later.

Get-ImageFromISO.ps1

```
#requires -version 4.0
#requires -module Storage,DISM

Function Get-ImageFromISO {
<#
.Synopsis
List windows editions from an ISO file.
.Description
This command will list the available windows images from an ISO file.
.Example
PS C:\> Get-ImageFromISO G:\iso\en_windows_server_2012_x64_dvd_915478.iso

ISOPath      : G:\iso\en_windows_server_2012_x64_dvd_915478.iso
Name         : windows Server 2012 SERVERSTANDARDCORE
Description  : windows Server 2012 SERVERSTANDARDCORE
Index        : 1
SizeMB       : 6862

ISOPath      : G:\iso\en_windows_server_2012_x64_dvd_915478.iso
Name         : windows Server 2012 SERVERSTANDARD
Description  : windows Server 2012 SERVERSTANDARD
Index        : 2
SizeMB       : 11444

ISOPath      : G:\iso\en_windows_server_2012_x64_dvd_915478.iso
Name         : windows Server 2012 SERVERDATACENTERCORE
Description  : windows Server 2012 SERVERDATACENTERCORE
Index        : 3
SizeMB       : 6844

ISOPath      : G:\iso\en_windows_server_2012_x64_dvd_915478.iso
Name         : windows Server 2012 SERVERDATACENTER
Description  : windows Server 2012 SERVERDATACENTER
Index        : 4
SizeMB       : 11440

Generally, the only part of the image name you need is what is in upper case.
.Notes
Last Updated:
Version      : 1.0

.Link
Get-WindowsImage
#>
```

```

[cmdletbinding()]
Param(
[Parameter(Position=0,Mandatory,HelpMessage="Enter the path to the ISO file.",
ValueFromPipeline,ValueFromPipelineByPropertyName)]
[ValidateScript({ Test-Path -path $_})]
[Alias("FullName")]
[string]$Path
)

Process {
write-verbose "Mounting $path as read-only"

$iso = Mount-DiskImage -ImagePath $path -Access ReadOnly -PasThru -StorageType ISO

$drive = "{0}:\` -f ($iso | Get-DiskImage | Get-Volume).DriveLetter

$wimPath = Join-Path -Path $drive -ChildPath "sources\install.wim"

write-verbose "Reading image information from $wimPath"
#add the ISO path to the output and make sure to sort by index.
#The image size is also formatted in MB.
Get-WindowsImage -ImagePath $wimPath |
Add-Member -MemberType NoteProperty -Name ISOPath -Value $path -PasThru |
Select ISOPath,@{Name="Name";Expression={$_.ImageName}},
@{Name="Description";Expression={$_.ImageDescription}},
@{Name="Index";Expression={$_.ImageIndex}},
@{Name="SizeMB";Expression={$_.ImageSize /1MB -as [int]}} | Sort Index

write-verbose "Dismounting disk image"

$iso | Dismount-DiskImage

} #end process

} #end function

```

In the help example you can see how to use this command. Next, you will need to download a script from Microsoft called Convert-WindowsImage.ps1. The most recent version supports Windows 8 and later (<http://gallery.technet.microsoft.com/scriptcenter/Convert-WindowsImageps1-0fe23a8f>). My script will call this script to apply a Windows image from an ISO to a new VHD or VHDX file.

New-VMFromISO.ps1

```

#requires -version 3.0
<#
.Synopsis
  Create a Hyper-V virtual machine from an ISO file.
.Description
  This script This script requires the Convert-windowsImage.ps1 script which you can download
  from Microsoft:

  http://gallery.technet.microsoft.com/scriptcenter/Convert-WindowsImageps1-0fe23a8f

```

The default location for the script is C:\Scripts or edit this script file accordingly.

The script will create a virtual memory with disk, memory and processor specifications. The VHDX file will be created and Convert-windowsImage will apply the specified windows image from the ISO. You can use the -ShowUI parameter for a GUI to create the VHDX file, select the ISO and apply the image.

.Parameter Name

The name of your new virtual machine.

.Parameter Path

The path to store your new virtual machine. The default is the server default location.

.Parameter ISOPath

The name and path to the ISO file.

.Parameter DiskName

The name of your new virtual disk. Include the VHD or VHDX extension.

.Parameter DiskPath

The folder for the new virtual disk. The default is the server default location for disks.

.Parameter Size

The size of the new virtual disk file.

.Parameter Memory

The amount of memory for the new virtual machine.

.Parameter Switch

The name of the virtual switch for the new virtual machine.

.Parameter ProcessorCount

The number of processors for the new virtual machine.

.Parameter Edition

The name of the Windows image from the ISO.

.Parameter Unattend

The filename and path of an unattend.xml file to be inserted into new virtual disk.

.Parameter ShowUI

Run the Convert script using the ShowUI parameter. You will be prompted to re-enter the path you specified for the new virtual disk. You might also need to manually remove the virtual drive that is created. DO NOT use this parameter if running this script in a remote PSSession.

.Example

```
PS C:\> $iso = "G:\iso\en_windows_server_2008_r2_standard_enterprise_datacenter_and_web_x64_dvd_x15-59754.iso"
```

```
PS C:\> $newParams=@{
```

```
  Name = '2008Web'
```

```
  DiskName = 'webDemo-01.vhdx'
```

```
  ISOPath = $Iso
```

```
  Edition = "ServerWeb"
```

```
  Size = 10GB
```

```
  Memory = 1GB
```

```
  Switch = "Work Network"
```

```
  ProcessorCount = 2
```

```
}
```

```
PS C:\> c:\scripts\new-vmfromiso.ps1 @newparams
```

This example creates a hashtable of parameters to splat to the script which will create a new virtual machine running the web edition of windows Server 2008.

.Example

```
PS C:\> c:\scripts\new-vmfromiso.ps1 -name "DemoVM" -ShowUI
```

This command will launch the script and create a virtual machine called DemoVM using all

of the default settings. The convert GUI will displayed.

.Example

```
PS C:\> invoke-command {c:\scripts\new-vmfromiso.ps1 -name Dev01 -diskname Dev01_C.vhdx  
-isopath f:\iso\windows2012-x64.iso -edition serverstandardcore -verbose} -comp SERVER01
```

This command will remotely run this script on SERVER01 to create the desired virtual machine. This script and the Microsoft script have to reside on the remote server.

.Notes

Last Updated: June 18, 2014

Version : 2.0

```
*****  
* DO NOT USE IN A PRODUCTION ENVIRONMENT UNTIL YOU HAVE TESTED *  
* THOROUGHLY IN A LAB ENVIRONMENT. USE AT YOUR OWN RISK. IF *  
* YOU DO NOT UNDERSTAND WHAT THIS SCRIPT DOES OR HOW IT WORKS, *  
* DO NOT USE IT OUTSIDE OF A SECURE, TEST SETTING. *  
*****
```

.Link

New-VM

#>

```
[cmdletbinding(DefaultParameterSetName="Manual",SupportsShouldProcess)]
```

Param(

```
[Parameter (Position = 0,Mandatory,  
HelpMessage = "Enter the name of the new virtual machine")]
```

```
[ValidateNotNullorEmpty()]
```

```
[string]$Name,
```

```
[ValidateNotNullorEmpty()]
```

```
[string]$Path = (Get-VMHost).VirtualMachinePath,
```

```
[Parameter(ParameterSetName="Manual",Mandatory,HelpMessage="Enter the path to the ISO file")]
```

```
[ValidateScript({Test-Path $_ })]
```

```
[string]$ISOPath,
```

```
[Parameter(ParameterSetName="Manual",Mandatory,HelpMessage="Enter the name of the install  
edition, e.g. windows Server 2012 R2 SERVERSTANDARD")]
```

```
[ValidateNotNullorEmpty()]
```

```
[string]$Edition,
```

```
[Parameter(ParameterSetName="Manual",Mandatory,HelpMessage="Enter the file name of the new  
VHD or VHDX file including the extension.")]
```

```
[ValidateNotNullorEmpty()]
```

```
[string]$DiskName,
```

```
[Parameter(ParameterSetName="Manual",HelpMessage="Enter the directory name of the new VHD  
file.")]
```

```
[ValidateNotNullorEmpty()]
```

```
[string]$DiskPath=(Get-VMHost).VirtualHardDiskPath,
```

```
[Parameter(ParameterSetName="Manual")]
```

```
[ValidateScript({$_ -ge 10GB})]
```

```

[int64]$Size = 20GB,

[Parameter(ParameterSetName="Manual")]
[ValidateScript({Test-Path $_ })]
[string]$Unattend,

[ValidateScript({$_ -ge 256MB})]
[int64]$Memory = 1GB,

[ValidateNotNullorEmpty()]
#set your default switch
[string]$Switch = "Work Network",

[ValidateScript({$_ -ge 1})]
[int]$ProcessorCount = 2,

[Parameter(ParameterSetName="UI")]
[switch]$ShowUI
)
### DEFINE THE PATH TO THE CONVERT-WINDOWSIMAGE.PS1 SCRIPT !!!
$convert = "c:\scripts\Convert-windowsImage.ps1"

if (-Not (Test-Path -Path $convert)) {
    write-warning "Failed to find $convert which is required."
    write-warning "Please download from:"
    write-warning " http://gallery.technet.microsoft.com/scriptcenter/Convert-windowsImageps1-0fe23a8f"
    write-warning "and try again."
    #bail out
    Return
}

#region creating the VHD or VHDX file
if ($pscmdlet.ParameterSetName -eq 'UI') {

if ($pscmdlet.ShouldProcess("ShowUI")) {
    &$convert -showUI

    Write-Warning "You may need to manually use Dismount-DiskImage to remove the mounted ISO file."
    $ok= $False
    do {
        [string]$vhdPath = Read-Host "`nwhat is the complete name and path of the virtual disk you just created? Press enter without anything to abort"

        if ($vhdPath -notmatch "\w+") {
            Write-Warning "No path specified. Exiting."
            Return
        }

        if (Test-Path -Path $vhdPath) {
            $ok = $True
        }
    } else {

```



```

        write-warning $_.Exception.Message
    }
} #should process
#endregion

#endregion
}
#endregion

#region Creating the virtual machine
write-verbose "Creating virtual machine $Name"
write-verbose "VHDPATH = $VHDPATH"
write-verbose "MemoryStartup = $Memory"
write-verbose "Switch = $Switch"
write-verbose "ProcessorCount = $ProcessorCount"
write-verbose "Path = $Path"

#new vm parameters
$newParam = @{
    Path = $Path
    Name = $Name
    VHDPATH = $VHDPATH
    MemoryStartupBytes = $Memory
    SwitchName = $Switch
}
New-VM @NewParam | Set-VM -DynamicMemory -ProcessorCount $ProcessorCount -Passthru

#dismount the disk image if still mounted
if ($ISOPATH -AND (Get-DiskImage -ImagePath $ISOPATH).Attached) {
    write-verbose "dismounting $isopath"
    Dismount-DiskImage -ImagePath $ISOPATH
}
write-verbose "New VM from ISO complete"

#endregion

```

My script assumes you put the Convert-WindowsImage script in C:\Scripts. Otherwise, you will need to change this line:

```
$convert = "c:\scripts\Convert-windowsImage.ps1"
```

To reflect the new location.

To use the script you will specify the name of a new virtual machine, the name of the virtual disk to create (including the VHD or VHDX extension), the path to the ISO file and the name of the Windows edition to apply. The script will use default locations, unless you specify otherwise. You can accept the defaults for disk size, memory and processor count or modify them via parameters. Finally, if you have an unattend.xml file, you can specify that path and the script will copy that to the virtual machine so that the first time you boot, the settings will be applied.

```
PS C:\> c:\scripts\new-vmfromiso.ps1 -name Dev01 -diskname Dev01_C.vhdx -isopath f:\iso\windows2012-x64.iso -edition serverstandardcore -processor 4 -memory 4GB -DiskPath D:\DevDisks
```

This is a script, not a function, so you need to specify the full name to run it. In this example I am creating locally a new virtual machine called Dev01 with a new virtual disk that will be created in D:\DevDisks\Dev01_C.vhdx from the SERVERSTANDARD CORE image on the specified ISO. This may take about 5 minutes to complete, but when finished the virtual machine is ready to go. By the way, my script has a default setting for the virtual switch. I recommend modifying the script to set a new default or remember to specify a switch when running the command.

As an alternative, you Convert-WindowImage.ps1 script has a -ShowUI parameter. I have added that to my script so that you can run a command like this:

```
PS C:\> C:\scripts\New-VMfromISO.ps1 -Name Test01 -ShowUI
```

You will get a form which makes it easier to browse for the ISO, select the Windows edition and create the virtual disk.

The screenshot shows a Windows dialog box titled "Convert-WindowImage UI". The dialog contains the following elements:

- Header: "Convert-WindowImage UI" with a close button (X).
- Instruction: "You can use the fields below to configure the VHD or VHDX that you want to create!"
- Section 1: "1. Choose a source" with a text input field containing "G:\iso\9200.16384.WIN8_RTM.120725-1247_X64FRE_SERVER_EVAL_EN-I" and a browse button (...).
- Section 2: "2. Choose a SKU from the list" with a dropdown menu showing "ServerDataCenterEval".
- Section 3: "3. Choose configuration options" with three dropdown menus: "VHD Format" (VHDX), "VHD Type" (Dynamic), and "VHD Size" (30 GB). Below these are text input fields for "Working Directory" (g:\vhds), "VHD Name (Optional)" (dceval.vhdx), and "Unattend File (Optional)" with a browse button (...).
- Section 4: "4. Make the VHD!" with a large button labeled "Make my VHD".

Figure 3

The working directory is where I want to create the VHDX file. When you create the name, be sure to include the extension that matches your format type. Otherwise, everything else is essentially the same.

This script doesn't have a provision to connect to a remote server. However, if you copy the necessary scripts to the remote server, you can invoke it with PowerShell remoting from a client.

```
PS C:\> invoke-command {c:\scripts\new-vmfromiso.ps1 -name Dev02 -diskname Dev02_C.vhdx -isopath d:\iso\windows2012-x64.iso -edition serverstandardcore -memory 4GB} -computername chi-hvr2
```

The remote server must have the Storage and DISM modules which if it is running Windows Server 2012 or later should not be an issue. Of course, another option is to create the virtual machine locally on a Windows 8 client with Hyper-V enabled, export the virtual machine and then import it on the server.

Virtual Machine Inventory

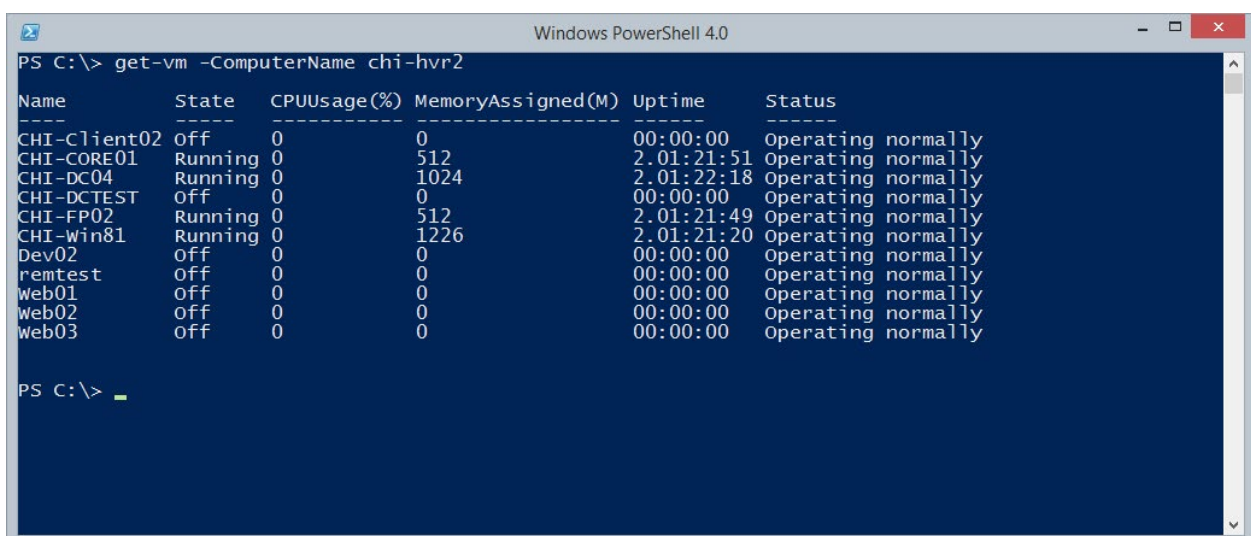
One of the best uses of PowerShell with Hyper-V is to discover or display information about virtual machines or the Hyper-V host. Yes, you have a graphical management console, but PowerShell allows you to get at information that isn't as readily accessible in the management console, and rarely for a group of virtual machines.

Virtual Machines

The easiest way to see your virtual machines from PowerShell is to run Get-VM. From a client you can specify the name of a Hyper-V host.

```
PS C:\> get-vm -ComputerName chi-hvr2
```

You will get output like Figure 4.



Name	State	CPUUsage(%)	MemoryAssigned(M)	Uptime	Status
CHI-Client02	Off	0	0	00:00:00	Operating normally
CHI-CORE01	Running	0	512	2.01:21:51	Operating normally
CHI-DC04	Running	0	1024	2.01:22:18	Operating normally
CHI-DCTEST	Off	0	0	00:00:00	Operating normally
CHI-FP02	Running	0	512	2.01:21:49	Operating normally
CHI-win81	Running	0	1226	2.01:21:20	Operating normally
Dev02	Off	0	0	00:00:00	Operating normally
remtest	Off	0	0	00:00:00	Operating normally
Web01	Off	0	0	00:00:00	Operating normally
Web02	Off	0	0	00:00:00	Operating normally
Web03	Off	0	0	00:00:00	Operating normally

Figure 4

Unfortunately, the Get-VM cmdlet doesn't have any filtering parameters. So if you wanted to see only running virtual machines you would need a command like this.

```
PS C:\> get-vm -ComputerName chi-hvr2 | where { $_.state -eq "running"}
```

```
Windows PowerShell 4.0
PS C:\>
PS C:\> get-vm -ComputerName chi-hvr2 | where { $_.state -eq "running"}
Name           State   CPUUsage(%) MemoryAssigned(M) Uptime           Status
-----
CHI-CORE01     Running 0             512                2.01:26:05      Operating normally
CHI-DC04       Running 0             1024               2.01:26:32      Operating normally
CHI-FP02       Running 0             512                2.01:26:03      Operating normally
CHI-Win81      Running 0             1226               2.01:25:34      Operating normally
PS C:\> _
```

Figure 5

To simplify things, here is a function that combines these steps into a single command.

Get-MyVM.ps1

```
#requires -version 3.0
#requires -module Hyper-v

Function Get-MyVM {
<#
.Synopsis
Get VM by state
.Description
This command is a proxy function for Get-VM. The parameters are identical to that command
with the addition of a parameter to filter virtual machines by their state. The default is
to only show running virtual machines. Use * to see all virtual machines.
.Example
PS C:\> get-myvm -computername chi-hvr2

Name           State   CPUUsage(%) MemoryAssigned(M) Uptime           Status
-----
CHI-CORE01     Running 0             512                2.01:47:42      Operating normally
CHI-DC04       Running 0             1024               2.01:48:10      Operating normally
CHI-FP02       Running 0             512                2.01:47:40      Operating normally
CHI-Win81      Running 0             1226               2.01:47:11      Operating normally

Get all running virtual machines on server CHI-HVR2.
.Example
PS C:\scripts> get-myvm -State saved -computername chi-hvr2

Name           State   CPUUsage(%) MemoryAssigned(M) Uptime           Status
-----
CHI-FP01     Saved 0             0                 00:00:00         Operating normally
```

```

CHI-win8 Saved 0          0          00:00:00 operating normally

Get saved virtual machines on server CHI-HVR2.
.Notes
Last Updated: June 20, 2014
Version      : 2.0
.Link
Get-VM
#>
[CmdletBinding(DefaultParameterSetName='Name')]
param(
    [Parameter(ParameterSetName='Id', Position=0, ValueFromPipeline=$true, ValueFromPipelineByPropertyName=$true)]
    [ValidateNotNull()]
    [System.Nullable[guid]] $Id,

    [Parameter(ParameterSetName='Name', Position=0, ValueFromPipeline=$true)]
    [Alias('VMName')]
    [ValidateNotNullOrEmpty()]
    [string[]] $Name="*",

    [Parameter(ParameterSetName='ClusterObject', Mandatory=$true, Position=0, ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [PSTypeName('Microsoft.FailoverClusters.PowerShell.ClusterObject')]
    [psobject] $ClusterObject,

    [Parameter(ParameterSetName='Id')]
    [Parameter(ParameterSetName='Name')]
    [ValidateNotNullOrEmpty()]
    [string[]] $ComputerName = $env:computername,

    [Microsoft.Hyperv.PowerShell.VMState] $State="Running"
)

begin
{
write-verbose "Getting virtual machines on $($computername.ToUpper()) with a state of $state"
    try {
        $outBuffer = $null
        if ($PSBoundParameters.TryGetValue('OutBuffer', [ref]$outBuffer))
        {
            $PSBoundParameters['OutBuffer'] = 1
        }

        $wrappedCmd = $ExecutionContext.InvokeCommand.GetCommand('Get-VM', [System.Management.Automation.CommandTypes]::Cmdlet)
        #remove my custom parameter because Get-VM won't recognize it.
        $PSBoundParameters.Remove('State') | Out-Null

        $scriptCmd = {& $wrappedCmd @PSBoundParameters | where {$_.state -like "$state"} }
        $steppablePipeline = $scriptCmd.GetSteppablePipeline($myInvocation.CommandOrigin)
        $steppablePipeline.Begin($PSCmdlet)
    } catch {

```

```

        throw
    }
}
process
{
    try {
        $steppablePipeline.Process($_)
    } catch {
        throw
    }
}
end
{
    try {
        $steppablePipeline.End()
    } catch {
        throw
    }
}
} #end function

```

Without getting too deep into the technical details, this function is a proxy function based on Get-VM. I've inserted a Where-Object command so that you can run it like this:

```
PS C:\> get-myvm -ComputerName chi-hvr2
```

Name	State	CPUUsage(%)	MemoryAssigned(M)	Uptime	Status
CHI-CORE01	Running	0	512	2.01:59:23	operating normally
CHI-DC04	Running	0	1024	2.01:59:51	operating normally
CHI-FP02	Running	0	512	2.01:59:22	operating normally
CHI-Win81	Running	0	1226	2.01:58:52	operating normally

The default is to display running virtual machines. But you can specify different states.

```
PS C:\> get-myvm -State saved -computername chi-hvr2 | select Name
```

```

Name
----
CHI-FP01
CHI-Win8

```

Use a value of * for the -State parameter to get all virtual machines, or use the original Get-VM.

Get-Newest Virtual Machines

If you manage a Hyper-V server where other people might be creating virtual machines, you'll probably want to keep an eye on newly created virtual machines. This is pretty easy because the virtual machine object includes a `CreationTime` property, which means you can filter with it:

```
PS C:\> get-vm -computer chi-hvr2 | where {$_.CreationTime -ge (Get-Date).AddDays(-7)}
```

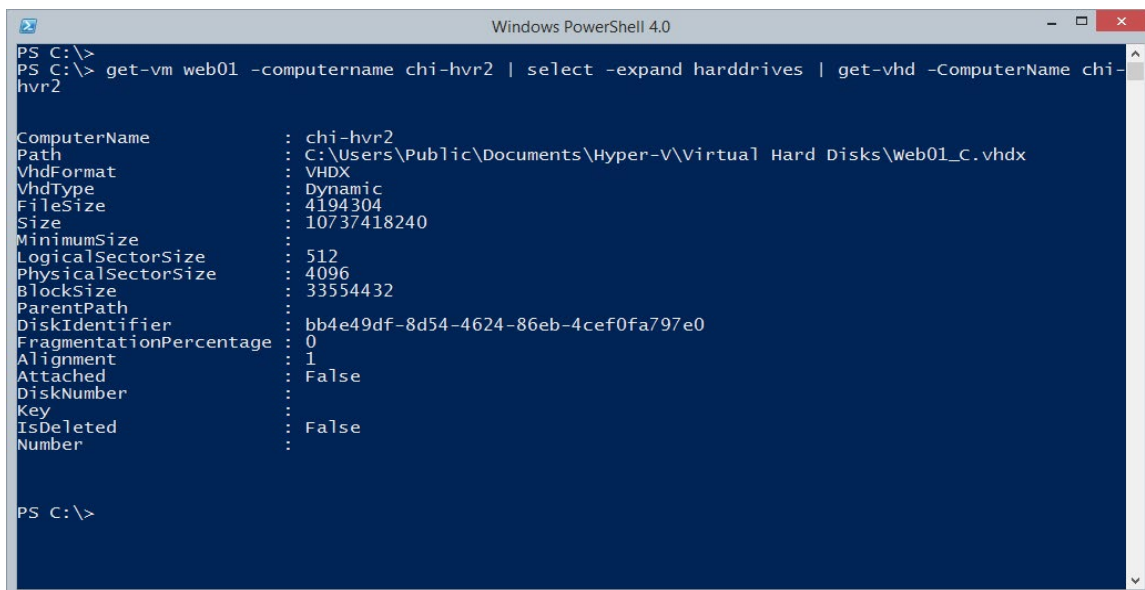
Name	State	CPUUsage(%)	MemoryAssigned(M)	Uptime	Status
Dev02	off	0	0	00:00:00	operating normally
web02	off	0	0	00:00:00	operating normally
web03	off	0	0	00:00:00	operating normally

These are the virtual machines on CHI-HVR2 that have been created in the last 7 days. There is one caveat with this technique: it appears that virtual machines that have been imported will have a `CreationTime` of 12/31/1600 7:00:00 PM, or something very similar depending on your time zone. But anything created with the management console, the `New-VM` cmdlet, and presumably the associated APIs should have a valid `CreationTime` value.

Hard Disk Report

Another typical management task should be to keep track of virtual disks associated with each virtual machine. The Hyper-V module includes a `Get-VHD` cmdlet which only really needs the path to a VHD file.

```
PS C:\> get-vm web01 -computername chi-hvr2 | select -expand harddrives | get-vhd -ComputerName chi-hvr2
```



```
Windows PowerShell 4.0
PS C:\>
PS C:\> get-vm web01 -computername chi-hvr2 | select -expand harddrives | get-vhd -ComputerName chi-hvr2
ComputerName      : chi-hvr2
Path              : C:\Users\Public\Documents\Hyper-V\Virtual Hard Disks\Web01_C.vhdx
VhdFormat         : VHDX
VhdType           : Dynamic
FileSize          : 4194304
Size              : 10737418240
MinimumSize       :
LogicalSectorSize : 512
PhysicalSectorSize : 4096
BlockSize         : 33554432
ParentPath        :
DiskIdentifier     : bb4e49df-8d54-4624-86eb-4cef0fa797e0
FragmentationPercentage : 0
Alignment         : 1
Attached          : False
DiskNumber        :
Key               :
IsDeleted         : False
Number           :
```

Figure 6

If for some reason the file doesn't exist you will get an error. Getting an error about a missing file on one hand is useful, but it makes it a bit more difficult to get a feel for the big picture. Also, from my testing I believe that when you use Get-VHD remotely, it translates the path to use the hidden administrative shares. That could be problematic as many admins are disabling those shares and it is likely not to be very cloud friendly. So here is my solutions for you.

Get-VHDInfo.ps1

```
#requires -version 4.0
#requires -modules Hyper-V
#requires -RunAsAdministrator

Function Get-VHDInfo {
<#
.Synopsis
Get virtual disk information.
.Description
This command will get virtual disk information for a given set of virtual machines on a
Hyper-V server. The default is all virtual machines.

The command uses PowerShell remoting to verify that the virtual disk file exists.

.Example
PS C:\> Get-VHDInfo -VMName chi-fp02 -Computername chi-hvr2
Getting disk file information on chi-hvr2 for virtual machine chi-fp02

VM                : CHI-FP02
Path              : C:\VM\CHI-FP02\Virtual Hard Disks\CHI-FP02_C.vhdx
VhdFormat        : VHDX
VhdType          : Dynamic
Size             : 21474836480
FileSize         : 20069744640
FragmentationPercentage : 5
ParentPath       :
Attached         : True
Verified         : True
ComputerName     : CHI-HVR2

VM                : CHI-FP02
Path              : C:\vhd\chi-fp02-disk2.vhdx
VhdFormat        : VHDX
VhdType          : Dynamic
Size             : 10737418240
FileSize         : 306184192
FragmentationPercentage : 7
ParentPath       :
Attached         : True
Verified         : True
ComputerName     : CHI-HVR2
.Example
PS C:\> Get-VHDInfo -computername chi-hvr2 | export-csv c:\work\DiskReport.csv -notype
```

Get virtual disk information for all VMs on server CHI-HVR2 and export to a CSV file.

.Example

```
PS C:\scripts> Get-VHDInfo -Computername chi-hvr2 | sort FragmentationPercentage -Descending  
| select -first 3 -Property VM,Path,Frag*,*size
```

Getting disk file information on chi-hvr2 for virtual machines *

```
VM           : Dev02  
Path         : C:\Users\Public\Documents\Hyper-V\Virtual Hard Disks\Dev02_C.vhdx  
FragmentationPercentage : 33  
Size        : 21474836480  
FileSize    : 5641338880
```

```
VM           : CHI-FP02  
Path         : C:\vhd\chi-fp02-disk2.vhdx  
FragmentationPercentage : 27  
Size        : 10737418240  
FileSize    : 306184192
```

```
VM           : CHI-FP02  
Path         : C:\VM\CHI-FP02\Virtual Hard Disks\CHI-FP02_C.vhdx  
FragmentationPercentage : 15  
Size        : 21474836480  
FileSize    : 20069744640
```

Get the 3 most fragmented VHD files.

.Example

```
PS C:\scripts> Get-VHDInfo -comp chi-hvr3 | where {! $_.verified}
```

Getting disk file information on chi-hvr3 for virtual machines *

```
VM           : Dev01  
Path         : D:\VHD\Dev01_C.vhdx  
VHDFormat    : VHDX  
VHDType      : UNKNOWN  
Size        : 0  
FileSize    : 0  
FragmentationPercentage :  
ParentPath   :  
Attached     : False  
Verified     : False  
Computername : CHI-HVR3
```

Identify VHD files that are referenced but missing.

.Notes

Last Updated: June 20, 2014

Version : 2.0

.Link

Get-VHD

#>

[cmdletbinding()]

Param(

```

[Parameter(Position=0)]
[ValidateNotNullorEmpty()]
[alias("Name")]
[string[]]$VMName="*",
[ValidateNotNullorEmpty()]
[string]$Computername = $env:computername
)
write-Host "Getting disk file information on $computername for virtual machines $VMName"
-ForegroundColor Cyan

Try {
$disks = Get-VM -name $VMname -computername $computername -ErrorAction Stop |
Select-Object -ExpandProperty harddrives | Select-Object VMName,Path,Computername
}
Catch {
    Throw $_
}
#continue if there are some disks
if ($disks) {

#create a temporary PSSession to the remote computer so we can test the path
Try {
    if ($computername -ne $env:computername) {
        write-verbose "Creating a temporary PSSession top $computername"
        $sess = New-PSSession -ComputerName $Computername -ErrorAction Stop
    }
}
Catch {
    #failed to create PSSession
    Throw $_
    #bail out
    Return
}
write-verbose "Processing disks..."
foreach ($disk in $disks) {

write-verbose ("VM {0} : {1}" -f $disk.VMName,$disk.path)

Try {
    $disk | Get-VHD -ComputerName $computername -ErrorAction Stop |
Select-Object -property @{Name="VM";Expression={$disk.vmname}},
Path,VHDFormat,VHDType,Size,FileSize,FragmentationPercentage,ParentPath,Attached,
@{Name="Verified";Expression={
if ($computername -eq $env:computername) {
    Test-Path -path $_.path
}
else {
    $diskpath = $_.path
    Invoke-command -ScriptBlock {Test-Path -path $using:diskpath} -session $sess
}
}},Computername
} #Try
Catch {

```

```

write-warning "Failed to find $($disk.path)"
#write a mostly empty custom object for the missing file
$hash=[ordered]@{
    VM = $disk.VMName
    Path = $disk.path
    VhdFormat = (split-path $disk.path -Leaf).split(".")[1].ToUpper()
    VhdType = "UNKNOWN"
    Size = 0
    FileSize = 0
    FragmentationPercentage=$null
    ParentPath=$null
    Attached=$False
    Verified=$False
    Computername = $disk.Computername
}
[pscustomobject]$hash
} #catch
} #foreach disk
}

#clean up
if ($sess) {
    Write-Verbose "Removing PSSession"
    Remove-PSSession $sess
}
} #end function

```

The script wraps up the one line command I showed earlier and adds some code to use Test-Path to verify if the file exists. If you are querying a remote computer, the function will create a temporary PSSession. You can use it like this:

```
PS C:\> get-vhdfinfo chi* -Computername chi-hvr2 | out-gridview -title "Chicago VMs"
```

This example gets all of the virtual machines that start with "chi" on the server CHI-HVR2 and sends the results to Out-GridView (figure 7).

VM	Path	VhdFormat	VhdType	Size	FileSize	Fragment...	ParentPath	Attached	Verified	ComputerName
CHI-Win81	C:\Users\Public\Documents\Hyper-V\Vir...	VHDX	Differencing	21,474,836,480	6,795,821,056		C:\Users\Pub...	True	True	chi-hvr2
CHI-FP02	C:\VM\CHI-FP02\Virtual Hard Disks\CHI-F...	VHDX	Dynamic	21,474,836,480	20,069,744,...	5		True	True	chi-hvr2
CHI-FP02	e:\vhd\chi-fp02-disk2.vhdx	VHDX	Dynamic	10,737,418,240	306,184,192	7		True	True	chi-hvr2
CHI-DCTEST	C:\VHD\CHI-DCTest_870939AE-57E6-45E...	VHDX	Differencing	21,474,836,480	4,194,304		C:\VHD\CHI-...	False	True	chi-hvr2
CHI-DC04	C:\Users\Public\Documents\Hyper-V\Vir...	VHDX	Differencing	42,949,672,960	14,594,080,...		C:\Users\Pub...	True	True	chi-hvr2
CHI-DC04	C:\Users\Public\Documents\Hyper-V\Vir...	VHDX	Dynamic	21,474,836,480	12,419,334,...	0		True	True	chi-hvr2
CHI-CORE01	C:\vhd\Core01_441AE074-6C97-4757-91...	VHDX	Differencing	21,474,836,480	2,145,386,496		C:\vhd\Core...	True	True	chi-hvr2
CHI-Client02	C:\VM\CHI-Client02\Virtual Hard Disks\W...	VHD	Dynamic	26,843,545,600	26,768,343,...	4		False	True	chi-hvr2
CHI-Client02	C:\VM\CHI-Client02\Virtual Hard Disks\C...	VHDX	Dynamic	4,294,967,296	2,889,875,456	3		False	True	chi-hvr2

Figure 7

Now I can easily find virtual machines with missing files.

```
PS C:\> get-vhdfinfo -computername win81-ent-01 | where {-Not $_.verified}
Getting disk file information on win81-ent-01 for virtual machines *
WARNING: Failed to find D:\VHD\Dev01_C.vhdx

VM                : Dev01
Path              : D:\VHD\Dev01_C.vhdx
VHDFormat        : VHDX
VHDType          : UNKNOWN
Size             : 0
FileSize         : 0
FragmentationPercentage :
ParentPath       :
Attached         : False
Verified         : False
Computername     : win81-ent-01
```

I intentionally renamed one of the disk files on a Windows 8.1 box to test. Because PowerShell is writing an object to the pipeline, you can do just about anything you want with it. Here is a one-line command to get the total file size in GB for all virtual disks.

```
PS C:\> get-vhdfinfo -computername chi-hvr2 | measure Filesize -sum | Format-Table
Count,@{Name="SizeGB";Expression={$_.Sum/1gb}} -AutoSize
```

Which gives me this result:

```
Count          SizeGB
-----
14 85.4192204475403
```

There is really no end to what you can do with this information.

Memory Usage

Another useful metric to monitor is memory. The Hyper-V module includes a cmdlet that will display memory information.

```
PS C:\> Get-VMemory -VMName CHI-FP02 -ComputerName chi-hvr2

VMName   DynamicMemoryEnabled Minimum(M) Startup(M) Maximum(M)
-----
CHI-FP02 True                512       512       2048
```

As with most things in PowerShell, there is more to this object than what you see on the screen.

```
PS C:\> Get-VMMemory -VMName CHI-FP02 -ComputerName chi-hvr2 | select *

Startup           : 536870912
DynamicMemoryEnabled : True
Minimum           : 536870912
Maximum           : 2147483648
Buffer            : 20
Priority           : 50
MaximumPerNumaNode : 5952
ResourcePoolName  : Primordial
ComputerName      : chi-hvr2
Name              : Memory
Id                : Microsoft:5FE62AF7-CE0A-477C-8DB4-E133CBC31C8F\4764334d-
                  e001-4176-82ee-5594ec9b530e
IsDeleted         : False
VMId              : 5fe62af7-ce0a-477c-8db4-e133cbc31c8f
VMName            : CHI-FP02
VMSnapshotId     : 00000000-0000-0000-0000-000000000000
VMSnapshotName    :
Key               :
```

To provide a more meaningful report, I created this PowerShell function.

Get-VMMemoryReport.ps1

```
#requires -version 3.0
#requires -module Hyper-V
Function Get-VMMemoryReport {
<#
.Synopsis
Get a VM memory report
.Description
This command gets memory settings for a given Hyper-V virtual machine. All memory values
are in MB. The command requires the Hyper-V module.
.Parameter VMName
The name of the virtual machine or a Hyper-V virtual machine object. This parameter has an
alias of "Name."
.Parameter VM
A Hyper-V virtual machine object. See examples.
.Parameter Computername
The name of the Hyper-V server to query. The default is the local host.
.Example
PS C:\> Get-VMMemoryReport chi-dc04 -ComputerName chi-hvr2

Computername : CHI-HVR2
Name         : CHI-DC04
Dynamic      : True
Assigned     : 1024
Demand      : 849
Startup     : 1024
```

```
Minimum      : 1024
Maximum      : 2048
Buffer       : 20
Priority      : 50
```

Get a memory report for a single virtual machine.

.Example

```
PS C:\> Get-VM -computer chi-hvr2 | where {$_.state -eq 'running'} | Get-VMMemoryReport |
format-table -autosize
```

Computername	Name	Dynamic	Assigned	Demand	Startup	Minimum	Maximum	Buffer	Priority
CHI-HVR2	CHI-CORE01	True	512	332	512	512	1024	20	50
CHI-HVR2	CHI-DC04	True	1024	849	1024	1024	2048	20	50
CHI-HVR2	CHI-FP02	True	512	389	512	512	2048	20	50
CHI-HVR2	CHI-Win81	True	1216	1021	1024	1024	1048576	20	50

Get a memory report for all running virtual machines formatted as a table.

.Example

```
PS C:\> get-content d:\MyVMs.txt | get-vmmemoryreport | Export-CSV c:\work\VMMemReport.csv
-notypeinformation
```

Get virtual machine names from the text file MyVMs.txt and pipe them to Get-VMMemoryReport. The results are then exported to a CSV file.

.Example

```
PS C:\> get-vm -computer chi-hvr2 | get-vmmemoryreport | Sort Maximum | convertto-html
-title "VM Memory Report" -css c:\scripts\blue.css -PreContent "<H2>Hyper-V Memory Report</
H2>" -PostContent "<br>An assigned value of 0 means the virtual machine is not running." |
out-file c:\work\vmmemreport.htm
```

Get a memory report for all virtual machines, sorted on the maximum memory property. This command then creates an HTML report.

.Notes

Last Updated: June 20, 2014

Version : 2.0

.Link

Get-VM

Get-VMMemory

.Inputs

Strings

Hyper-V virtual machines

.Outputs

Custom object

#>

```
[cmdletbinding(DefaultParameterSetName="Name")]
```

```
Param(
```

```
[Parameter(Position=0,HelpMessage="Enter the name of a virtual machine",
```

```
ValueFromPipeline,ValueFromPipelineByPropertyName,
```

```
ParameterSetName="Name")]
```

```
[alias("Name")]
```

```
[ValidateNotNullorEmpty()]
```

```
[string]$VMName="*",
```

```
[Parameter(Position=0,Mandatory,HelpMessage="Enter the name of a virtual machine",
```

```
ValueFromPipeline,ValueFromPipelineByPropertyName,
```

```

ParameterSetName="VM")]
[ValidateNotNullorEmpty()]
[Microsoft.HyperV.PowerShell.VirtualMachine[]]$VM,
[ValidateNotNullorEmpty()]
[Parameter(ValueFromPipelinebyPropertyName)]
[ValidateNotNullorEmpty()]
[string]$Computername=$env:COMPUTERNAME
)
Begin {
    Write-Verbose "Starting $($MyInvocation.Mycommand)"
} #begin

Process {

    if ($PSCmdlet.ParameterSetName -eq "Name") {
        Try {
            $VMs = Get-VM -name $VMName -ComputerName $computername -ErrorAction Stop
        }
        Catch {
            Write-warning "Failed to find VM $vmname on $computername"
            #bail out
            Return
        }
    }
    else {
        $VMs = $VM
    }
    foreach ($V in $VMs) {
        #get memory values
        Try {
            Write-Verbose "Querying memory for $($V.name) on $($computername.ToUpper())"
            $memorysettings = Get-VMMemory -VMName $v.name -ComputerName $Computername
            -ErrorAction Stop

            if ($MemorySettings) {
                #all values are in MB
                $hash=[ordered]@{
                    Computername = $v.ComputerName.ToUpper()
                    Name = $V.Name
                    Dynamic = $V.DynamicMemoryEnabled
                    Assigned = $V.MemoryAssigned/1MB
                    Demand = $V.MemoryDemand/1MB
                    Startup = $V.MemoryStartup/1MB
                    Minimum = $V.MemoryMinimum/1MB
                    Maximum = $V.MemoryMaximum/1MB
                    Buffer = $memorysettings.buffer
                    Priority = $memorysettings.priority
                }
                #write the new object to the pipeline
                New-Object -TypeName PObject -Property $hash
            } #if $memorySettings found
        } #Try
        Catch {

```



```

        Throw $_
    } #Catch
    } #foreach $v in $VMs
} #process
End {
    Write-Verbose "Ending $($MyInvocation.Mycommand)"
} #end
} #end Get-VMMemoryReport

```

To use you can specify either the name or names of a virtual machine or pipe the results of a Get-VM comma

```

PS C:\scripts> Get-VMMemoryReport chi-dc04 -Computername chi-hvr2

Computername : CHI-HVR2
Name          : CHI-DC04
Dynamic       : True
Assigned      : 1024
Demand        : 849
Startup       : 1024
Minimum       : 1024
Maximum       : 2048
Buffer        : 20
Priority       : 50

```

All of the memory values are formatted as MB. Here's another example that takes the output and creates an HTML report:

```

PS C:\> get-vm -computer chi-hvr2 | where { $_.state -eq "running" } | get-vmmemoryreport
| Sort Maximum | convertto-html -title "VM Memory Report" -css c:\scripts\blue.
css -PreContent "<H2>Hyper-V Memory Report</H2>" -PostContent "<i>report created by
$env:username</i>" | out-file c:\work\vmmemreport.htm

```

This command is getting all of the running virtual machines on CHI-HVR2, getting memory information, sorting on the Maximum size and then creating an HTML report which you can see in Figure 8.

Computername	Name	Dynamic	Assigned	Demand	Startup	Minimum	Maximum	Buffer	Priority
CHI-HVR2	CHI-CORE01	True	512	337	512	512	1024	20	50
CHI-HVR2	CHI-DC04	True	1024	849	1024	1024	2048	20	50
CHI-HVR2	CHI-FP02	True	512	389	512	512	2048	20	50
CHI-HVR2	CHI-Win81	True	1214	1019	1024	1024	1048576	20	50

report created by Jeff

Figure 8

Get VM Last Use

Another task that I have is to discover when a virtual machine was last used. There is nothing in the virtual machine object that will provide that information. But the next best thing appears to be getting the last write time property of the associated disk file. Here is a PowerShell function that does just that.

Get-VMLastUse.ps1

```
#requires -version 3.0
#requires -modules Hyper-V
```

```
Function Get-VMLastUse {
```

```
<#
```

```
.Synopsis
```

```
Find a virtual machine last use date.
```

```
.Description
```

```
This command will write a custom object to the pipeline which should indicate when the virtual machine was last used. The command finds all hard drives that are associated with a Hyper-V virtual machine and selects the first one.
```

```
The assumption is that if the virtual machine is running the hard drive file will be changed. The function retrieves the last write time property from the first VHD or VHDX file.
```

```
You can pipe a collection of Hyper-V virtual machines or specify a virtual machine name. Wildcards are supported. The default is to display last use data for all virtual machines. This command must be run on the Hyper-V server in order to get file system information from the disk file. Therefore, it uses PowerShell remoting to query remote servers.
```

```
The command requires the Hyper-V module running in PowerShell 3.0 or later.
```

```
.Example
```

```
PS C:\> get-vmLastUse chi* -computername chi-hvr2 | sort LastUseAge -Descending
```

VMName	LastUse	LastUseAge	Computername
CHI-Client02	4/8/2014 12:17:03 PM	76.01:47:30.2752590	chi-hvr2
CHI-DCTEST	6/17/2014 1:32:52 PM	6.00:31:40.7218754	chi-hvr2
CHI-Win81	6/23/2014 2:03:41 PM	00:00:52.3668297	chi-hvr2
CHI-FP02	6/23/2014 2:04:03 PM	00:00:30.2643841	chi-hvr2
CHI-DC04	6/23/2014 2:04:28 PM	00:00:04.9566880	chi-hvr2
CHI-CORE01	6/23/2014 2:04:30 PM	00:00:03.0382539	chi-hvr2

```
Get last use information for any virtual machine starting with CHI and sort by LastUseAge in descending order.
```

```
.Example
```

```
PS C:\> get-vm -computer chi-hvr2 | where {$_.state -eq 'off'} | get-vmLastUse
```

VMName	LastUse	LastUseAge	Computername
CHI-Client02	4/8/2014 12:17:03 PM	76.01:47:50.6724602	chi-hvr2
CHI-DCTEST	6/17/2014 1:32:52 PM	6.00:32:01.7385359	chi-hvr2
Dev02	6/19/2014 2:04:18 PM	4.00:00:36.5491423	chi-hvr2

web01	6/17/2014 2:05:27 PM	5.23:59:28.6523976	chi-hvr2
web02	6/17/2014 9:28:46 PM	5.16:36:10.5266729	chi-hvr2
web03	6/17/2014 9:28:44 PM	5.16:36:12.6050236	chi-hvr2

Get last use information for any virtual machine that is currently off on server CHI-HVR2.

.Notes

Last Updated: June 23, 2014

Version : 2.0

.Inputs

String or Hyper-V Virtual Machine

.Outputs

custom object

.Link

Get-VM

#>

[cmdletbinding()]

Param (

[Parameter(Position=0,

HelpMessage="Enter a Hyper-V virtual machine name",

ValueFromPipeline,ValueFromPipelinebyPropertyName)]

[ValidateNotNullorEmpty()]

[alias("vm")]

[object]\$Name="*",

[Parameter(ValueFromPipelineByPropertyName)]

[ValidateNotNullorEmpty()]

[string]\$Computername=\$env:COMPUTERNAME

)

Begin {

Write-Verbose -Message "Starting \$(\$MyInvocation.Mycommand)"

} #begin

Process {

if (\$name -is [string]) {

Write-Verbose -Message "Getting virtual machine(s)"

Try {

\$vms = Get-VM -Name \$name -ComputerName \$computername -ErrorAction Stop

}

Catch {

Write-Warning "Failed to find a VM or VMS with a name like \$name on \$(\$Computername.ToUpper())"

#bail out

Return

}

}

else {

#otherwise we'll assume \$Name is a virtual machine object

Write-Verbose "Found one or more virtual machines matching the name"

\$vms = \$name

}

if (\$vms) {

```

if ($vms[0].ComputerName -ne $env:computername) {
    #create a temporary PSSession
    Try {
        Write-Verbose "Creating a temporary session to $($vms[0].ComputerName)"
        New-PSSession -ComputerName $vms[0].ComputerName -Name $vms[0].
ComputerName -ErrorAction Stop | Out-Null
    }
    Catch {
        Write-warning "Failed to create a PSSession to $($vms[0].ComputerName)"
        Throw $_
        #bail out
        Return
    }
}
foreach ($vm in $vms) {
    Write-Verbose "Processing $($vm.name)"
    if ($vm.harddrives) {
        $sb = {
            Param($v)
            #get first drive file

            Try {
                $diskFile = Get-Item -Path $v.path -ErrorAction Stop
                Write-Verbose "..found $($diskFile.fullname)"
                $diskfile | Select-Object -property @{Name="VMName";Expression={$v.
vmname}},
                @{Name="LastUse";Expression={$DiskFile.LastWriteTime}},
                @{Name="LastUseAge";Expression={(Get-Date) - $diskFile.LastWriteTime}},
                @{Name="Computername";Expression={$v.computername}}
            }
            Catch {
                Write-warning "Failed to find $($v.Path) for $($v.vmname) on $($v.
computername)"
            }
        } #scriptblock

        if ($vm.computername -eq $env:computername) {
            Invoke-Command -ScriptBlock $sb -ArgumentList $vm.HardDrives[0]
        }
        else {
            Invoke-Command -ScriptBlock $sb -ArgumentList $vm.HardDrives[0]
            -HideComputerName -session (Get-PSSession -Name $vm.computername) |
            Select VMName,LastUse,LastUseAge,Computername
        }
    } #if VM has hard drive files
    else {
        Write-warning "Failed to find any hard drive files for $($vm.vmname) on $($vm.
computername)"
    }
} #foreach
} #if $vms
else {
    #this should never happen
    Write-Warning "No virtual machines."
}

```

```

    }
    #clean up any PSSessions
    Remove-PSSession -Name $vm.computername -ErrorAction SilentlyContinue
} #process

End {

    Write-Verbose -Message "Ending $($MyInvocation.Mycommand)"
} #end

} #end function

```

The function selects the first virtual disk file associated with a virtual machine, using the assumption that this disk holds the operating system and is likely to change when the virtual machine is powered on. You can use it like this:

```
PS C:\> get-vm\lastuse -computer chi-hvr2
```

Or you can pipe the results of a Get-VM expression:

```
PS C:\> get-vm chi* -computer chi-hvr2 | where {$_.state -eq 'off'} | get-vm\lastuse |
sort LastUseAge -descending
```

Name	LastUse	LastUseAge	Computername
-----	-----	-----	-----
CHI-Client02	4/8/2014 12:17:03 PM	76.01:49:37.8312351	chi-hvr2
CHI-DCTEST	6/17/2014 1:32:52 PM	6.00:33:47.7675333	chi-hvr2

You could even remove very old virtual machines.

```
PS C:\> get-vm\lastuse chi* -computer chi-hvr2 | where {$_.lastuseage.totalDays -gt 75} |
Select -expand vmname | remove-vm -ComputerName chi-hvr2 -whatif
```

what if: remove-vm will remove virtual machine "CHI-Client02".

If I had wanted, I could have removed any virtual machine starting with CHI on server CHI-HVR2 that hasn't been used in more than 75 days.

Get VM Operating System

Another piece of information you might find useful is to know what operating system is running on your virtual machines. This information is buried deep in WMI (Windows Management Instrumentation) and for right now I can only retrieve information for running virtual machines and only if they are running a Windows operating system.

Get-VMOS.ps1

```
#requires -version 3.0
```

```
Function Get-VMOS {
```

```
<#
```

```
.Synopsis
```

```
Get the installed windows operating system on a virtual machine.
```

```
.Description
```

```
This command will display the installed windows operating system on a Hyper-V virtual machine. The virtual machine must be running.
```

```
The function uses WMI to query remote computers.
```

```
.Example
```

```
PS C:\> Get-VMOS CHI-core01 -Computersname chi-hvr2
```

VMName	OperatingSystem	Computersname
CHI-CORE01	Windows Server 2012 R2 Datacenter	CHI-HVR2

```
Get a single virtual machine operating system.
```

```
.Example
```

```
PS C:\> get-vm -computersname chi-hvr2 | where {$_.state -eq 'running'} | get-vmos
```

VMName	OperatingSystem	Computersname
CHI-CORE01	Windows Server 2012 R2 Datacenter	CHI-HVR2
CHI-DC04	Windows Server 2012 Datacenter	CHI-HVR2
CHI-FP02	Windows Server 2012 R2 Standard	CHI-HVR2
CHI-Win81	Windows 8.1 Pro	CHI-HVR2

```
Get operating system information for all running virtual machines.
```

```
.Notes
```

```
Last Updated: June 23, 2014
```

```
Version      : 2.0
```

```
.Link
```

```
Get-WMIObject
```

```
#>
```

```
[cmdletbinding()]
```

```
Param(
```

```
[Parameter(Position=0,HelpMessage="Enter the name of a virtual machine",  
ValueFromPipeline,ValueFromPipelinebyPropertyName)]
```

```
[ValidateNotNullOrEmpty()]
```

```
[Alias("Name")]
```

```
[string]$VMName="*",
```

```
[Parameter(ValueFromPipelinebyPropertyName)]
```

```
[string]$Computersname=$env:COMPUTERNAME
```

```
)
```

```
Begin {
```

```
    Write-Verbose "Starting $($MyInvocation.Mycommand)"
```

```
} #begin
```

```
Process {
```

```

Write-verbose "Querying virtual machines on $($Computername.ToUpper())"

$wmiParam=@{
Namespace= "root/virtualization/v2"
ClassName= "Msvm_VirtualSystemManagementService"
ComputerName= $Computername
errorAction= "Stop"
errorVariable= "myErr"
}

Try {
    $vsm = Get-WmiObject @wmiParam
}
Catch {
    $myerr.errorrecord.exception.message
}

#modify the parameter hash
$wmiParam.ClassName= "MSVM_Computersystem"
if ($VMName -eq "*") {
    $filter = "Caption='Virtual Machine'"
}
elseif ($VMName -match "\*") {
    #replace * with %
    $elementname = $VMName.Replace("*", "%")
    $filter = "elementname LIKE '$elementname'"
}
else {
    $filter = "elementname='$VMName'"
}

$wmiParam.filter= $filter
Write-verbose "Querying virtual machine $VMName"
write-verbose ($wmiParam | Out-String)

Try {
    $vm = Get-WmiObject @wmiParam
}
Catch {
    $myerr.errorrecord.exception.message
}

if ($vm) {

#get virtual system data and filter out checkpoints
$vsd = $vm.GetRelated("MSVM_VirtualSystemSettingData") | where {$_.Description -notmatch
"^Checkpoint"}

#an array of items to get
#http://msdn.microsoft.com/en-us/library/hh850062(v=vs.85).aspx
[uint32[]]$requested = @(1,106)

$result = $vsm.GetSummaryInformation($vsd,$requested)

#display the result
$result.summaryinformation |
select @{Name="VMName";Expression={$_.Elementname}},
@{Name="OperatingSystem";Expression={$_.GuestOperatingSystem}},
@{Name="Computername";Expression={$vsm.pscomputername}}

```

```

}
else {
    write-warning "Failed to find virtual machine $VMName"
}

} #Process

End {
    write-verbose -Message "Ending $($MyInvocation.Mycommand)"
} #end

} #end function

```

This function relies on Get-WMIObject which means you must have WMI access to any remote computer, which you probably already do. The default behavior is to return information for all virtual machines on the local host, but you can limit your query to a single virtual machine:

```
PS C:\> get-vmos chi-dc04 -computername chi-hvr2
```

VMName	OperatingSystem	Computername
CHI-DC04	windows Server 2012 Datacenter	CHI-HVR2

Or you can query multiple virtual machines:

```
PS C:\> get-vmos chi* -computername chi-hvr2
```

VMName	OperatingSystem	Computername
CHI-Win81	windows 8.1 Pro	CHI-HVR2
CHI-DCTEST		CHI-HVR2
CHI-FP02	windows Server 2012 R2 Standard	CHI-HVR2
CHI-Client02		CHI-HVR2
CHI-DC04	windows Server 2012 Datacenter	CHI-HVR2
CHI-CORE01	windows Server 2012 R2 Datace...	CHI-HVR2

Notice that some virtual machines have no operating system, because the VM is not running. You'll want to do something like this:

```
PS C:\> get-vm chi* -computername chi-hvr2 | where {$_.state -eq 'running'} | get-vmos | format-table -auto
```

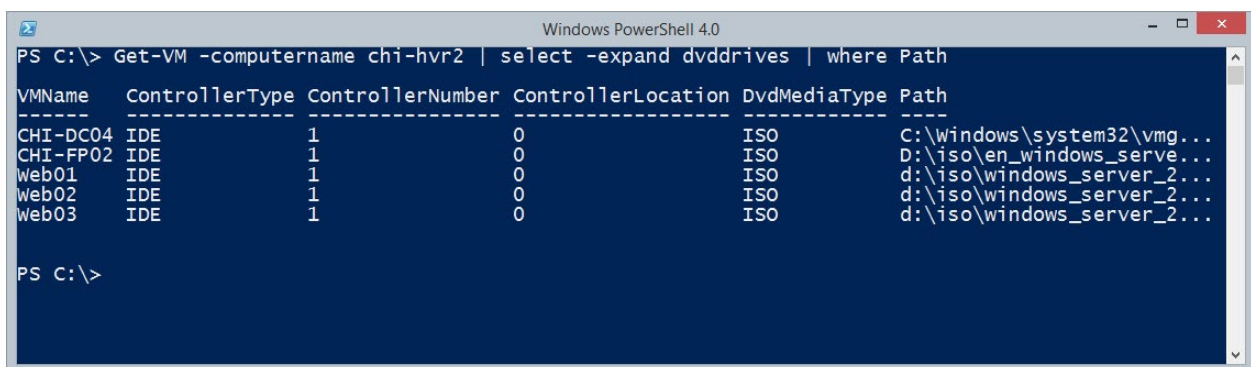
VMName	OperatingSystem	Computername
CHI-Win81	windows 8.1 Pro	CHI-HVR2
CHI-FP02	windows Server 2012 R2 Standard	CHI-HVR2
CHI-DC04	windows Server 2012 Datacenter	CHI-HVR2
CHI-CORE01	windows Server 2012 R2 Datacenter	CHI-HVR2

Get Mounted ISO files

I love being able to mount ISO files in virtual DVD drives. However, sometimes they can be left mounted which could cause problems. Or perhaps you just like knowing how your virtual machines are currently configured. This is a very simple one-line PowerShell command.

```
PS C:\> Get-VM -computername chi-hvr2 | select -expand dvd drives | where Path
```

There is plenty of useful information as you see in Figure 9.



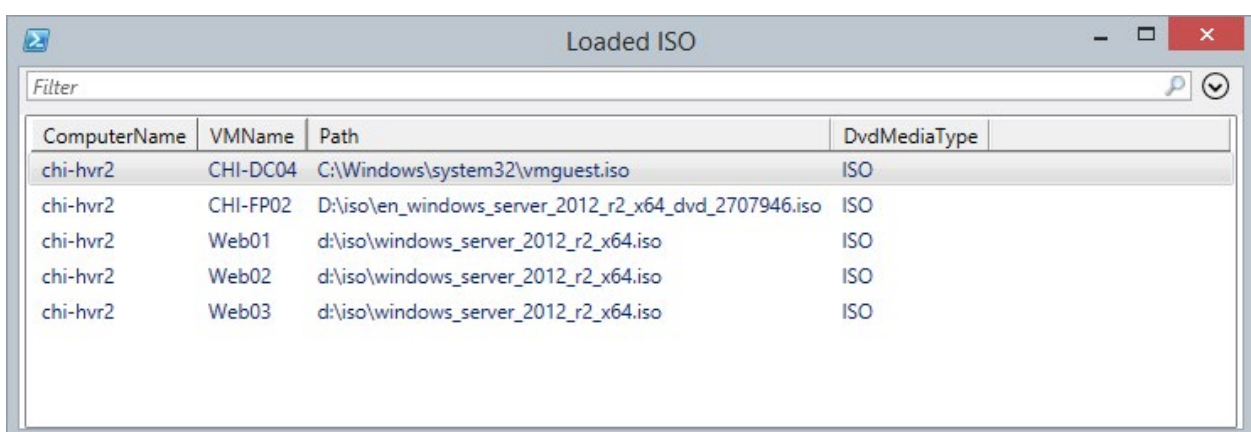
```
Windows PowerShell 4.0
PS C:\> Get-VM -computername chi-hvr2 | select -expand dvd drives | where Path
VMName      ControllerType ControllerNumber ControllerLocation DvdMediaType Path
-----
CHI-DC04    IDE           1              0                  ISO          C:\windows\system32\vmg...
CHI-FP02    IDE           1              0                  ISO          D:\iso\en_windows_serve...
Web01       IDE           1              0                  ISO          d:\iso\windows_server_2...
Web02       IDE           1              0                  ISO          d:\iso\windows_server_2...
Web03       IDE           1              0                  ISO          d:\iso\windows_server_2...
```

Figure 9

Although most likely you will want to select a subset of properties.

```
PS C:\> Get-VM -computername chi-hvr2 | select -expand dvd drives | where Path | select
ComputerName,VMName,Path,DVDMediaType | out-gridview -title "Loaded ISO".
```

The result is shown in Figure 10.



ComputerName	VMName	Path	DvdMediaType
chi-hvr2	CHI-DC04	C:\Windows\system32\vmguest.iso	ISO
chi-hvr2	CHI-FP02	D:\iso\en_windows_server_2012_r2_x64_dvd_2707946.iso	ISO
chi-hvr2	Web01	d:\iso\windows_server_2012_r2_x64.iso	ISO
chi-hvr2	Web02	d:\iso\windows_server_2012_r2_x64.iso	ISO
chi-hvr2	Web03	d:\iso\windows_server_2012_r2_x64.iso	ISO

Figure 10

Identifying Orphaned VHD/VHDX Files

In an earlier recipe I showed you how to list virtual disk information. In that situation we were verifying the virtual disks from the virtual machine's perspective. If you are like me, you may store all of your virtual disks in one or more locations. How about checking those locations for files that are not associated with any virtual machines? That's what this recipe is all about.

Get-ObsoleteVHD.ps1

```
#requires -version 3.0
#requires -module Hyper-V

Function Get-ObsoleteVHD {
<#
.Synopsis
Get orphaned or obsolete virtual disk files.
.Description
This command will search a directory for VHD or VHDX files that are not attached to any
existing Hyper-V virtual machines. The default behavior is to search the default virtual
hard disk path on the local computer.

The function uses PowerShell remoting to query paths on remote computers.
.Example
PS C:\> get-obsoletevhd -computer chi-hvr2

    Directory: C:\Users\Public\Documents\Hyper-V\Virtual Hard Disks

Mode                LastWriteTime         Length Name
----                -
-a---             6/26/2013   8:51 PM 9399435264 win8.1PreviewBase.vhdx

Getting unused virtual disk files on server CHI-HVR2 in the default location.
.Example
PS C:\> get-obsoletevhd -computer chi-hvr2 -Path c:\vhd

    Directory: C:\vhd

Mode                LastWriteTime         Length Name
----                -
-a---             5/9/2014   1:59 PM 10842275840 Essentials.vhdx

An unused file in a different location on server CHI-HVR2.
.Example
PS C:\> get-obsoletevhd -path g:\vhds -computer Server01 | measure -sum Length | Select
Count,@{Name="SizeGB";Expression={$_.sum/1GB}}

Count                SizeGB
----                -
    11                82.3568634986877

This example is finding all unused virtual disk files in G:\VHDS on SERVER01 and then
calculating how much disk space they are consuming.

.Notes
Last Updated: June 25, 2014
Version      : 1.0
```

```

.Link
Get-VHD
#>

[cmdletbinding()]

Param(
[Parameter(Position=0)]
[ValidateNotNullorEmpty()]
#use the value for -Computername is specified, otherwise the local computer
[string]$Path=(Get-VMHost -computername ( &{if ($computername) { $computername} else {
$env:computername}})).VirtualHardDiskPath,
[Alias("CN")]
[ValidateNotNullorEmpty()]
[string]$Computername=$env:computername
)

Write-Verbose -Message "Starting $($MyInvocation.Mycommand)"
Write-Verbose "Searching for obsolete virtual disk files in $Path on $($Computername.
ToUpper())"

#initialize an array to hold file information
$files = @()

Try {
#get currently used virtual disk files
Get-VM -computername $computername -ErrorAction Stop | Select -ExpandProperty
HardDrives |
Get-VHD -ComputerName $computername |
foreach {
$files+=$_path
if ($_parentPath) {
$files+=$_parentPath
} #if path
} #foreach
} #try
Catch {
Throw $_
#bail out
}

if ($files) {
#filter out duplicates
$diskfiles = $files | Sort | Get-Unique -AsString

write-verbose "Attached files"
$diskfiles | Write-Verbose

write-verbose "Orphaned files in $path"
$sb = {
Param($path)
if (Test-Path -path $Path) {
dir -Path $path -file -filter *.vhd? -Recurse
}
else {
write-warning "Failed to find path $path on $($env:computername)"
}
}
}

```

```

    $found = if ($Computername -ne $env:computername) {
        Invoke-Command -ScriptBlock $sb -ComputerName $computername -HideComputerName
    }
    else {
        &$sb $path
    }
    if ($found) {
        write-verbose "Found $($found.count) files"
        $found.fullname | write-verbose
        $found | where {$files -notcontains $_.fullname}
    }
    else {
        write-host "No files found in $path on $computername" -ForegroundColor Red
    }
} #if files were found
write-verbose -Message "Starting $($MyInvocation.Mycommand)"
} #end function

```

This function, by default, will search the local computer using the default location for virtual disk files. When you query a remote computer, the function will use a temporary PSSession with Invoke-Command in order to get a directory listing on the remote computer. You may be wondering why I don't use Get-VHD to identify obsolete or unused files. It is true that Get-VHD results will show an Attached property, but that will be false if the virtual machine is not running, so that isn't a valid property to use in this situation.

```
PS C:\> get-obsoletevhd -computer server01
```

```
Directory: D:\VHD
```

Mode	LastWriteTime	Length	Name
-a---	4/27/2014 2:19 PM	102760448	Demo2.vhdx
-a---	6/17/2014 12:37 PM	4194304	small2_C.vhdx
-a---	6/17/2014 1:23 PM	4194304	small3_C.vhdx
-a---	6/17/2014 1:28 PM	4194304	small4_C.vhdx
-a---	6/17/2014 1:29 PM	4194304	small5_C.vhdx
-a---	6/17/2014 12:32 PM	4194304	small_C.vhdx
-a---	4/9/2014 8:22 AM	4194304	temp.vhdx
-a---	5/3/2014 12:54 PM	5362417664	test2.vhdx
-a---	6/18/2014 4:29 PM	5574230016	xDev01_C.vhdx

This example shows the unused virtual disk files in the default location on Hyper-V server SERVER01. Remember, the output you see is on the remote server. If you wanted to delete these files, you could use commands like this:

```
PS C:\> $old = get-obsoletevhd -computer server01
```

This saves the results to a variable. Then, use Invoke-Command to delete them remotely.

```
PS C:\> invoke-command { $using:old | del -whatif } -computername server01

what if: Performing the operation "Remove File" on target "D:\VHD\Demo2.vhdx".
what if: Performing the operation "Remove File" on target "D:\VHD\small2_C.vhdx".
what if: Performing the operation "Remove File" on target "D:\VHD\small3_C.vhdx".
what if: Performing the operation "Remove File" on target "D:\VHD\small4_C.vhdx".
what if: Performing the operation "Remove File" on target "D:\VHD\small5_C.vhdx".
what if: Performing the operation "Remove File" on target "D:\VHD\small_C.vhdx".
what if: Performing the operation "Remove File" on target "D:\VHD\temp.vhdx".
what if: Performing the operation "Remove File" on target "D:\VHD\test2.vhdx".
what if: Performing the operation "Remove File" on target "D:\VHD\xDev01_C.vhdx".
```

Deleting Obsolete Snapshots

Another typical management task is to clean up old or obsolete snapshots. The Get-VMSnapshot cmdlet can easily retrieve snapshots for one or more virtual machines. Wrapping it up in PowerShell, it doesn't take much to get a quick report of snapshots. This is a one-line PowerShell expression.

```
Get-VMSnapshot -VMName * -ComputerName chi-hvr2 | Select ComputerName,VMName,Name,SnapshotType,CreationTime,@{Name="Age";Expression={ (Get-Date) - $_.CreationTime }}
```

You can see my results in Figure 11.

```
Windows PowerShell 4.0

PS C:\> Get-VMSnapshot -VMName * -ComputerName chi-hvr2 |
>> Select ComputerName,VMName,Name,SnapshotType,CreationTime,
>> @{Name="Age";Expression={ (Get-Date) - $_.CreationTime }}
>>

ComputerName : chi-hvr2
VMName       : CHI-win81
Name         : Profile Cleanup Test
SnapshotType : Standard
CreationTime : 5/27/2014 10:52:10 AM
Age          : 29.01:06:12.6239968

ComputerName : chi-hvr2
VMName       : CHI-CORE01
Name         : CHI-CORE01 - (6/17/2014 - 1:32:40 PM)
SnapshotType : Standard
CreationTime : 6/17/2014 1:32:42 PM
Age          : 7.22:25:40.8557781

ComputerName : chi-hvr2
VMName       : CHI-DCTEST
Name         : CHI-DCTEST - (6/17/2014 - 1:32:52 PM)
SnapshotType : Standard
CreationTime : 6/17/2014 1:32:52 PM
Age          : 7.22:25:30.3759554

PS C:\>
```

Figure 11

You might also want to know how much space the snapshot files are consuming. This will require PowerShell remoting to query a remote server, but really isn't that much more difficult.

```
Invoke-Command {
Get-VMSnapshot -VMName * |Select -ExpandProperty HardDrives | Get-Item |
Measure-Object -Property Length -sum |
Select Count,@{Name="SizeGB";Expression={$_.Sum/1GB}}
} -ComputerName chi-hvr2 | Select * -ExcludeProperty runspaceId
```

In this command, I'm creating a custom property called SizeGB that takes the sum of all the associated disk files and converts the value to GB, which you can see in Figure 12.

```
PS C:\> Invoke-Command {
>> Get-VMSnapshot -VMName * |
>> Select -ExpandProperty HardDrives | Get-Item |
>> Measure-Object -Property Length -sum |
>> Select Count,@{Name="SizeGB";Expression={$_.Sum/1GB}}
>> } -ComputerName chi-hvr2 | Select * -ExcludeProperty runspaceId
>>

          Count                               SizeGB PSComputerName
-----
          3                               26.6259765625 chi-hvr2

PS C:\>
```

Figure 12

In fact, why don't we combine the two?

Get-VMSnapshotUsage.ps1

```
#requires -version 3.0
#requires -module Hyper-V

Param(
[string[]] $VMName="*",
[string] $Computername=$env:computername)

Invoke-Command -scriptblock {
Get-VMSnapshot -VMName $using:VMName |
Select Computername,VMName,Name,SnapshotType,CreationTime,
@{Name="Age";Expression={ (Get-Date) - $_.CreationTime }},
@{Name="SizeGB";
Expression = { ($_.HardDrives | Get-Item | Measure-Object -Property length -sum).sum/1GB
}}
} -computername $computername -HideComputerName | Select * -ExcludeProperty RunspaceID
```

This is a quick script but you could easily turn it into a function or expand upon it. Figure 13 depicts the script in action.

```

Windows PowerShell 4.0
PS C:\> C:\scripts\Get-VMSnapshotUsage.ps1 -Computername chi-hvr2

ComputerName : CHI-HVR2
VMName       : CHI-win81
Name         : Profile Cleanup Test
SnapshotType : Standard
CreationTime : 5/27/2014 10:52:10 AM
Age          : 29.01:20:56.4007760
SizeGB       : 15.814453125

ComputerName : CHI-HVR2
VMName       : CHI-CORE01
Name         : CHI-CORE01 - (6/17/2014 - 1:32:40 PM)
SnapshotType : Standard
CreationTime : 6/17/2014 1:32:42 PM
Age          : 7.22:40:24.9110940
SizeGB       : 8.9716796875

ComputerName : CHI-HVR2
VMName       : CHI-DCTEST
Name         : CHI-DCTEST - (6/17/2014 - 1:32:52 PM)
SnapshotType : Standard
CreationTime : 6/17/2014 1:32:52 PM
Age          : 7.22:40:14.6531322
SizeGB       : 1.83984375

PS C:\>

```

Figure 13

If you look at help examples for `Remove-VMSnapshot`, you'll see an example for removing old snapshots. I took that idea and expanded it into a more full-fledged Powershell function.

Remove-OldVMSnapshot.ps1

```

#requires -version 3.0
#requires -module Hyper-V

Function Remove-OldVMSnapshot {

<#
.Synopsis
Remove old Hyper-V snapshots.
.Description
This command will find and remove snapshots older than a given number of days, the default
is 90, on a Hyper-V server. You can limit the removal process to specific virtual machines
as well as specific types of VM snapshots.

This command will remove all child snapshots as well so use with caution. The command
supports -whatif and -Confirm.
.Example
PS C:\> Remove-OldVMSnapshot -VMName Ubuntu-Demo -computername SERVER01

This command removed all snapshots for the Ubuntu-Demo virtual machine on SERVER01 that
is older than 90 days.
.Example
PS C:\> Remove-OldVMSnapshot -computername chi-hvr2 -days 14 -whatif
what if: Remove-VMSnapshot will remove snapshot "Profile Cleanup Test".

These are the snapshots older than 14 days on server CHI-HVR2 that would be removed.
.Example
PS C:\> Remove-OldVMSnapshot -computer chi-hvr2 -days 14 -confirm

```

```

Confirm
Are you sure you want to perform this action?
Remove-VMSnapshot will remove snapshot "Profile Cleanup Test".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Y"):Y

Answering Y to the prompt will delete the snapshot.

.Notes
Last Updated: June 25, 2014
Version : 1.0

.Link
Remove-VMSnapshot
#>
[cmdletbinding(SupportsShouldProcess,ConfirmImpact="High",DefaultParameterSetName="All")]

Param (
[Parameter(Position=0)]
[ValidateNotNullorEmpty()]
[string]$VMName="*",

[Parameter(Position=1)]
[ValidateScript({$_ -ge 1})]
[Alias("days")]
[int]$Age=90,

[Parameter(Position=1,ParameterSetName="ByType")]
[ValidateNotNullorEmpty()]
[Alias("type")]
[Microsoft.HyperV.PowerShell.SnapshotType]$SnapshotType = "Standard",

[ValidateNotNullorEmpty()]
[Alias("CN")]
[string]$computername = $env:computername
)

Write-Verbose -Message "Starting $($MyInvocation.Mycommand)"

#parameters for Get-VMSnapshot
$getParams= @{
Computername = $computername
ErrorAction = "Stop"
VMName = $VMName
}

if ($PSCmdlet.ParameterSetName -eq 'ByType') {
Write-Verbose "Limiting snapshots to type $SnapshotType"
$getParams.Add("SnapshotType",$SnapshotType)
}

Try {
[datetime]$Cutoff = ((Get-Date).Date).AddDays(-$Age)
Write-Verbose "Searching for snapshots equal to or older than $Cutoff on $computername"
$snaps = Get-VMSnapshot @getParams | Where {$_.CreationTime -le $Cutoff }
}

```



```

Catch {
    Throw $_
}

if ($snaps) {
    Write-Verbose "Found $($snaps.count) snapshots to be removed"
    $snaps | Remove-VMSnapshot -IncludeAllChildSnapshots
}

Write-Verbose -Message "Ending $($MyInvocation.Mycommand)"

} #end function

```

By default the function will return all snapshots for all virtual machines older than 90 days. You can specify a different number of days, filter the virtual machines by name or limit the snapshots to a certain snapshot type. Because the command could potentially remove a lot of files, it has support for `-Whatif` and `-Confirm`.

```

PS C:\> remove-oldvmsnapshot -days 7 -computername chi-hvr2 -whatif
what if: Remove-VMSnapshot will remove snapshot "Profile Cleanup Test".
what if: Remove-VMSnapshot will remove snapshot "CHI-CORE01 - Check 1".
what if: Remove-VMSnapshot will remove snapshot "DCTEST - Check 1".

```

If I re-ran the command without `-Whatif`, these snapshots, including any child snapshots, would be removed.

Querying Hyper-V Event Logs

Of course, managing a Hyper-V server means keeping an eye on logged events. With PowerShell this is pretty simple once you know what you are looking for.

```

PS C:\> Get-EventLog -LogName system -Source "Microsoft-Windows-Hyper-V*" -new-est 100 -ComputerName Chi-Hvr2 | Sort Source | Format-Table -GroupBy Source -property TimeGenerated,EntryType,Message -Wrap -AutoSize

```

This command will use the standard `Get-EventLog` cmdlet to search the System event log for anything logged from a Hyper-V source and select the first 100 entries. In this example, I formatted the results as a table grouped by the source. You can see my results in Figure 14.

```

Windows PowerShell 4.0
PS C:\> Get-EventLog -LogName system -Source "Microsoft-Windows-Hyper-V*" -newest 100 -ComputerName Chi-Hvr2 | Sort Source | Format-Table -GroupBy Source -property TimeGenerated,EntryType,Message -Wrap -AutoSize

Source: Microsoft-Windows-Hyper-V-Hypervisor
-----
TimeGenerated      EntryType  Message
-----
6/18/2014 6:58:10 AM Information The hypervisor enabled I/O remapping. IOV may be available if the
system hardware and BIOS support it.
6/13/2014 9:14:36 AM Information The hypervisor enabled I/O remapping. IOV may be available if the
system hardware and BIOS support it.
5/16/2014 9:53:46 AM Information The hypervisor enabled I/O remapping. IOV may be available if the
system hardware and BIOS support it.
6/18/2014 6:58:10 AM Information Hypervisor successfully started.
6/13/2014 2:59:41 PM Information Hypervisor successfully started.
6/13/2014 2:59:41 PM Information The hypervisor enabled I/O remapping. IOV may be available if the
system hardware and BIOS support it.
5/19/2014 3:24:28 PM Information Hypervisor successfully started.
5/19/2014 3:24:28 PM Information The hypervisor enabled I/O remapping. IOV may be available if the
system hardware and BIOS support it.
6/13/2014 9:14:36 AM Information Hypervisor successfully started.
5/16/2014 9:53:46 AM Information Hypervisor successfully started.

Source: Microsoft-Windows-Hyper-V-VmSwitch
-----
TimeGenerated      EntryType  Message
-----
5/19/2014 3:24:29 PM Information Miniport NIC F4841C36-D393-45E2-9447-A6093A4F25D5 (Friendly
Name: Test) successfully enabled
5/19/2014 3:24:29 PM Information Miniport NIC 7F1657B7-96A4-41F0-B43A-4566BB25736A (Friendly
Name: ) successfully enabled
5/19/2014 3:24:29 PM Information Miniport NIC 7F1657B7-96A4-41F0-B43A-4566BB25736A (Friendly
Name: work Network) successfully enabled
5/27/2014 12:29:23 PM Information Networking driver in CHI-Win81 is loaded and the protocol
version is negotiated to the most recent version (Virtual
machine ID 149E1B91-2B52-43E1-BDA6-BD6D89504BE1).
5/19/2014 3:24:29 PM Information Switch DCEA40B6-1133-4273-BABE-D7EE91970A62 (Friendly Name:
Test) successfully initialized.

```

Figure 14

Or, you might want to limit your search.

```

PS C:\> Get-EventLog -LogName system -Source "Microsoft-Windows-Hyper-V*" -EntryType
Error,Warning -ComputerName Chi-Hvr2 -After (Get-Date).AddDays(-3)

```

This command will find all Hyper-V related errors and warnings on server CHI-HVR2 that have been recorded in the last 3 days. Get-Eventlog searches "classic" event logs. Hyper-V also has its own set of operational logs which you can query with Get-WinEvent.

```

PS C:\> Get-WinEvent -ListLog *Hyper-V*

```

LogMode	MaximumSizeInBytes	RecordCount	LogName
Circular	1052672	146	Microsoft-Windows-Hyper-V-Config-Admin
Circular	1052672	0	Microsoft-Windows-Hyper-V-Config-Operational
Circular	1052672	0	Microsoft-Windows-Hyper-V-EmulatedNic-Admin
Circular	1052672	0	Microsoft-Windows-Hyper-V-Hypervisor-Admin
Circular	1052672	283	Microsoft-Windows-Hyper-V-Hypervisor-Operational
Circular	1052672	887	Microsoft-Windows-Hyper-V-Integration-Admin

Circular	1052672	0	Microsoft-Windows-Hyper-V-SynthFc-Admin
Circular	1052672	654	Microsoft-Windows-Hyper-V-SynthNic-Admin
Circular	1052672	1	Microsoft-Windows-Hyper-V-SynthStor-Admin
Circular	1052672	0	Microsoft-Windows-Hyper-V-SynthStor-Operational
Circular	1052672	0	Microsoft-Windows-Hyper-V-VID-Admin
Circular	1052672	2277	Microsoft-Windows-Hyper-V-VMMS-Admin
Circular	1052672	38	Microsoft-Windows-Hyper-V-VMMS-Networking
Circular	1052672	60	Microsoft-Windows-Hyper-V-VMMS-Operational
Circular	1052672	40	Microsoft-Windows-Hyper-V-VMMS-Storage
Circular	1052672	0	Microsoft-Windows-Hyper-V-VmSwitch-Operational
Circular	1052672	1187	Microsoft-Windows-Hyper-V-Worker-Admin

In order to query a remote computer, you must configure a firewall exception for remote event log management or use PowerShell remoting with Invoke-Command.

```
PS C:\> invoke-command {Get-WinEvent -ListLog *Hyper-V*} -ComputerName chi-hvr2
```

LogMode	MaximumSizeInBytes	RecordCount	LogName	PSComputerName
Circular	1052672	0	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	0	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	0	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	0	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	246	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	1278	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	0	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	871	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	0	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	0	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	0	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	685	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	30	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	36	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	4	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	0	Microsoft-Windows-Hyper-V-...	chi-hvr2
Circular	1052672	817	Microsoft-Windows-Hyper-V-...	chi-hvr2

You may want to limit your search to those logs that have entries.

```
PS C:\> invoke-command {Get-WinEvent -ListLog *Hyper-V* | where {$_.recordcount -gt 0} | Select Logname,RecordCount} -ComputerName chi-hvr2
```

```

LogName      : Microsoft-Windows-Hyper-V-Hypervisor-Operational
RecordCount  : 246
PSComputerName : chi-hvr2
RunspaceId   : b1ce1c35-4476-4684-9991-b9487f60bb51

LogName      : Microsoft-Windows-Hyper-V-Integration-Admin
RecordCount  : 1278
PSComputerName : chi-hvr2
RunspaceId   : b1ce1c35-4476-4684-9991-b9487f60bb51
...

```

Once you know the name of the log to query, you can run a simple command like this:

```

PS C:\> Invoke-Command {Get-WinEvent -LogName Microsoft-Windows-Hyper-V-Hypervisor-Operational -MaxEvents 10 } -computername chi-hvr2 | format-list

TimeCreated      : 6/19/2014 2:04:48 PM
ProviderName     : Microsoft-Windows-Hyper-V-Hypervisor
Id               : 16642
Message          : Hyper-V successfully deleted a partition (partition 8).
PSComputerName   : chi-hvr2

TimeCreated      : 6/19/2014 8:42:40 AM
ProviderName     : Microsoft-Windows-Hyper-V-Hypervisor
Id               : 16641
Message          : Hyper-V successfully created a new partition (partition 8).
PSComputerName   : chi-hvr2
...

```

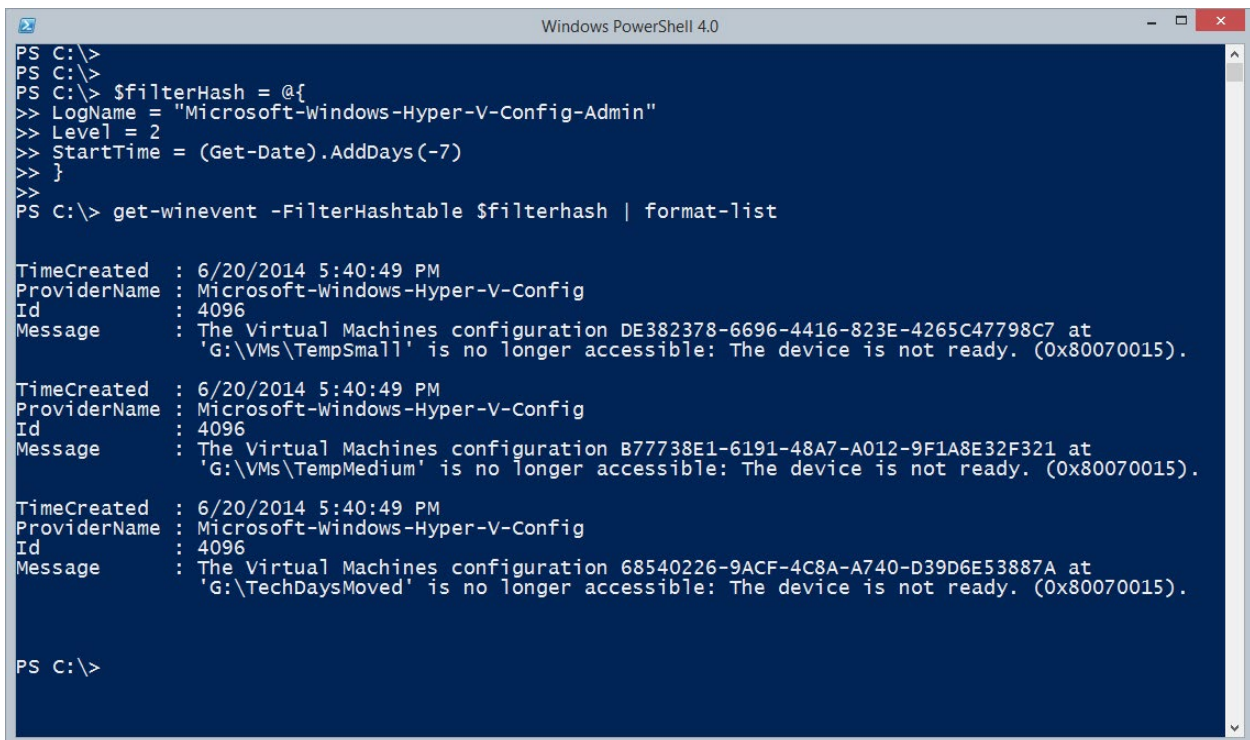
Here, I've retrieved the 10 most recent entries from the Microsoft-Windows-Hyper-V-Hypervisor-Operational log. If you need to get more granular, you have a few options, but I think using a hashtable of filtering options is the easiest approach. You can use code like this.

```

$filterHash = @{
LogName = "Microsoft-Windows-Hyper-V-Config-Admin"
Level = 2
StartTime = (Get-Date).AddDays(-7)
}
get-winevent -FilterHashtable $filterhash | format-list

```

My result, which is all errors logged in the specified log over the last 7 days for the local host, is shown in Figure 15.



```
Windows PowerShell 4.0
PS C:\>
PS C:\>
PS C:\> $filterHash = @{
>> LogName = "Microsoft-Windows-Hyper-V-Config-Admin"
>> Level = 2
>> StartTime = (Get-Date).AddDays(-7)
>> }
PS C:\> get-winevent -FilterHashtable $filterhash | format-list

TimeCreated      : 6/20/2014 5:40:49 PM
ProviderName     : Microsoft-Windows-Hyper-V-Config
Id               : 4096
Message          : The Virtual Machines configuration DE382378-6696-4416-823E-4265C47798C7 at
                  'G:\VMs\TempSmall' is no longer accessible: The device is not ready. (0x80070015).

TimeCreated      : 6/20/2014 5:40:49 PM
ProviderName     : Microsoft-Windows-Hyper-V-Config
Id               : 4096
Message          : The Virtual Machines configuration B77738E1-6191-48A7-A012-9F1A8E32F321 at
                  'G:\VMs\TempMedium' is no longer accessible: The device is not ready. (0x80070015).

TimeCreated      : 6/20/2014 5:40:49 PM
ProviderName     : Microsoft-Windows-Hyper-V-Config
Id               : 4096
Message          : The Virtual Machines configuration 68540226-9ACF-4C8A-A740-D39D6E53887A at
                  'G:\TechDaysMoved' is no longer accessible: The device is not ready. (0x80070015).

PS C:\>
```

Figure 15

Information messages are type 4 and warnings are type 3. I've wrapped all of this into a PowerShell function.

Get-HyperVEvents.ps1

```
#requires -version 3.0
```

```
Function Get-HyperVEvents {
```

```
<#
```

```
.Synopsis
```

```
Get errors and warnings from Hyper-V Operational logs.
```

```
.Description
```

```
This command will search a specified server for all Hyper-V related windows operational logs and get all errors and warnings that have been recorded in the specified number of days which is 7 by default.
```

```
The command uses PowerShell remoting to query event logs and resolve SIDs to account names. The remote event log management firewall exception is not required to use the command.
```

```
.Example
```

```
PS C:\> Get-HyperVEvents -Days 30 -computer CHI-HVR2 | Select LogName,TimeCreated,Type, ID,Message,Username | Out-GridView -title "Events"
```

```
Get all errors and warnings within the last 30 days on server CHI-HVR2 and display with Out-GridView.
```

```

.Notes
Last Updated: June 25, 2014
Version      : 2.0

.Link
Get-WinEvent
Get-Eventlog
.Inputs
[String]
.Outputs
[System.Diagnostics.Eventing.Reader.EventLogRecord]
Technically this will be a deserialized version of this object.
#>

[cmdletbinding()]

Param(
[Parameter(Position=0,HelpMessage="Enter the name of a Hyper-V host")]
[ValidateNotNullOrEmpty()]
[Alias("CN","PSComputername")]
[string]$Computername=$env:COMPUTERNAME,
[ValidateScript({$_ -ge 1})]
[int]$Days=7,
[Alias("RunAs")]
[System.Management.Automation.Credential()]$Credential = [System.Management.Automation.
PSCredential]::Empty
)

write-verbose "Starting $($MyInvocation.MyCommand)"
write-verbose "Querying Hyper-V logs on $($computername.ToUpper())"

#define a hash table of parameters to splat to Invoke-Command
$icmParams=@{
ErrorAction="Stop"
ErrorVariable="MyErr"
Computername=$Computername
HideComputername=$True
}

if ($credential.username) {
write-verbose "Adding a credential for $($credential.username)"
$icmParams.Add("Credential",$credential)
}

#define the scriptblock to run remotely and get events using Get-WinEvent
$sb = {

Param([string]$Verbose="SilentlyContinue")

#set verbose preference in the remote scriptblock
$VerbosePreference=$Verbose

#calculate the cutoff date
$start = (Get-Date).AddDays(-$using:days)
write-verbose "Getting errors since $start"

#construct a hash table for the -FilterHashTable parameter in Get-WinEvent
$filter= @{
Logname= "Microsoft-windows-Hyper-V*"

```

```

Level=2,3
StartTime= $start
}

#search logs for errors and warnings
#turn off errors to ignore exceptions about no matching records, which would be ok.
Try {
    #add a property for each entry that translates the SID into
    #the account name
    Get-WinEvent -filterHashTable $filter -ErrorAction Stop | foreach {
        #add some custom properties
        $_ | Add-Member -MemberType AliasProperty -Name "Type" -Value "LevelDisplayName"

        $_ | Add-Member -MemberType ScriptProperty -Name Username -Value {
            [WMI]$Resolved = "root\cimv2:win32_SID.SID=' $($this.UserID)'"
            #write the resolved name to the pipeline
            "$($Resolved.ReferencedDomainName)\ $($Resolved.Accountname)"
        } -PassThru
    }
}
Catch {
    Write-Warning "No matching events found."
}

} #close scriptblock

#add the scriptblock to the parameter hashtable for Invoke-Command
$icmParams.Add("Scriptblock", $sb)

if ($VerbosePreference -eq "Continue") {
    #if this command was run with -Verbose, pass that to the scriptblock
    #which will be running remotely.
    Write-Verbose "Adding verbose scriptblock argument"
    $sbArgs="Continue"
    $icmParams.Add("Argumentlist", $sbArgs)
}

Try {
    #invoke the scriptblock remotely and pass properties to the pipeline, except
    #for the RunspaceID from the temporary remoting session which we don't need.
    Invoke-Command @icmParams
}
Catch {
    #Invoke-Command failed
    Write-Warning "Failed to connect to $($computername.ToUpper())"
    Write-Warning $MyErr.errorRecord
    #bail out of the function and don't do anything else
    Return
}

#All done here
Write-Verbose "Ending $($MyInvocation.MyCommand)"

} #end function

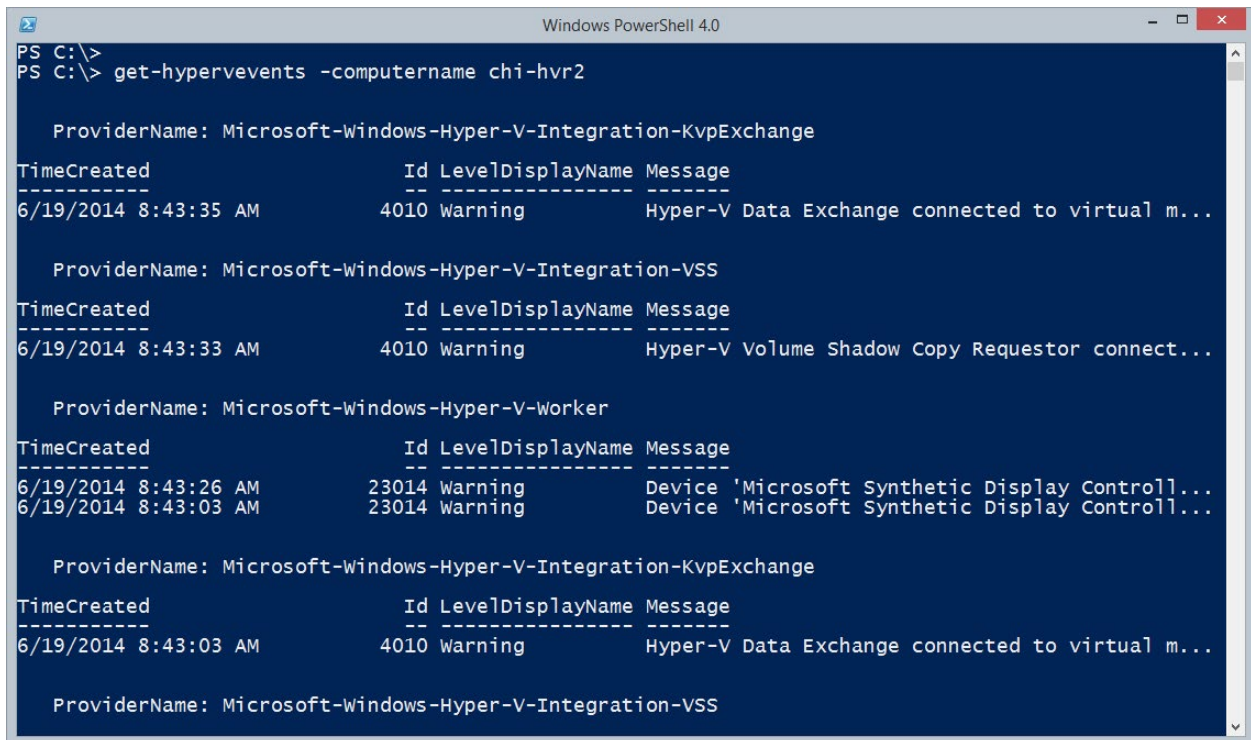
```

This function assumes you will use PowerShell remoting to query the operational event logs. By default, it gets all errors and warnings recorded in the last 7 days. I have included code to

add an alias of Type (e.g. error) instead of having to know to use LogDisplayName, which isn't exactly intuitive. There is also code to convert the user SID to a friendly name.

```
PS C:\> get-hypervevents -computername chi-hvr2
```

Figure 16 shows the default output.



```
Windows PowerShell 4.0
PS C:\>
PS C:\> get-hypervevents -computername chi-hvr2

ProviderName: Microsoft-Windows-Hyper-V-Integration-KvpExchange
TimeCreated           Id LevelDisplayName Message
-----
6/19/2014 8:43:35 AM  4010 Warning      Hyper-V Data Exchange connected to virtual m...

ProviderName: Microsoft-Windows-Hyper-V-Integration-VSS
TimeCreated           Id LevelDisplayName Message
-----
6/19/2014 8:43:33 AM  4010 Warning      Hyper-V Volume Shadow Copy Requestor connect...

ProviderName: Microsoft-Windows-Hyper-V-Worker
TimeCreated           Id LevelDisplayName Message
-----
6/19/2014 8:43:26 AM  23014 Warning      Device 'Microsoft Synthetic Display Controll...
6/19/2014 8:43:03 AM  23014 Warning      Device 'Microsoft Synthetic Display Controll...

ProviderName: Microsoft-Windows-Hyper-V-Integration-KvpExchange
TimeCreated           Id LevelDisplayName Message
-----
6/19/2014 8:43:03 AM  4010 Warning      Hyper-V Data Exchange connected to virtual m...

ProviderName: Microsoft-Windows-Hyper-V-Integration-VSS
```

Figure 16

Here's an example that takes advantage of my username addition.

```
PS C:\> get-hypervevents -computername chi-hvr2 -Days 180 | where {$_.username -match "globomantics"} | Select Logname,TimeCreated,Username,Type,Message

LogName      : Microsoft-Windows-Hyper-V-VMMS-Storage
TimeCreated  : 6/17/2014 2:07:49 PM
Username     : GLOBOMANTICS\Administrator
Type         : Error
Message      : The system failed to create 'C:\Users\Public\Documents\Hyper-V\Virtual Hard
              Disks\web01_C.vhdx': The file exists. (0x80070050).
```

So, it seems I only have 1 entry recorded by a domain user as opposed to a system account. This is what I'm referring to:


```
PS C:\> get-hypervevents -computername chi-hvr2 -days 180 | group Username -NoElement |
sort Count | format-table -AutoSize
```

```
Count Name
```

```
-----
```

```
1 GLOBOMANTICS\Administrator
4 NT VIRTUAL MACHINE\5FE62AF7-CE0A-477C-8DB4-E133CBC31C8F
4 NT VIRTUAL MACHINE\6164B819-D828-425C-823A-561D96EC0975
5 NT VIRTUAL MACHINE\E5099F2C-6489-4646-BD27-D0D519D6B0ED
5 NT VIRTUAL MACHINE\62E6858B-3B73-410E-8AF1-BA2B6F93ACA3
6 NT VIRTUAL MACHINE\60423BAE-36A4-441A-93B6-F2B2ABD9DBBC
18 NT VIRTUAL MACHINE\149E1B91-2B52-43E1-BDA6-BD6D89504BE1
40 NT AUTHORITY\SYSTEM
```

A Hyper-V Health Report

By now you can see that PowerShell is a fantastic Hyper-V reporting tool. In fact, I have a reporting script that I run weekly that generates a Hyper-V health report an HTML file. The script is called [New-HVHealthReport.ps1](#). I'll admit it is a rather lengthy and complex PowerShell script. In fact, it is really too long to list here in its entirety. [The script is included in the accompanying zip file download.](#)

The script can be run locally to query remote servers. You will need the Hyper-V, Storage and NetAdapter PowerShell modules.

The syntax isn't that complicated:

```
<path>\New-HVHealthReport.ps1 [[-Computername] <String>] [-Path <String>]
[-RecentCreated <Int32>] [-LastUsed <Int32>] [-Hours <Int32>] [-Performance]
[-Metering] [<CommonParameters>]
```

The script includes full help.

```
PS C:\> help c:\scripts\new-hvhealthreport.ps1 -full
```

Specify the full path to the script file. By default, the script will create an HTML report for the local computer using the specified path. The `-RecentCreated` parameter is used to find virtual machines that have been created in the last X number of days. The default is 30. The value for `-LastUsed` will show you virtual machines that have not been used in that number of days.

The script will display all errors and warnings from all Hyper-V event logs recorded in the last X number of hours. The default is 24.

The script will optionally return Hyper-V performance counter information (-Performance). If you have metering enabled you can use -Metering to display that information as well. Even though the intent of metering is for monitoring usage, I think it offers another glimpse into the overall health of the Hyper-V server and virtual machines.

Using the script, it is as easy as this to create the report:

```
PS C:\> C:\scripts\New-HVHealthReport.ps1 -Computername chi-hvr2 -path c:\work\hvr2.htm -performance
```

Figure 17 shows my report.

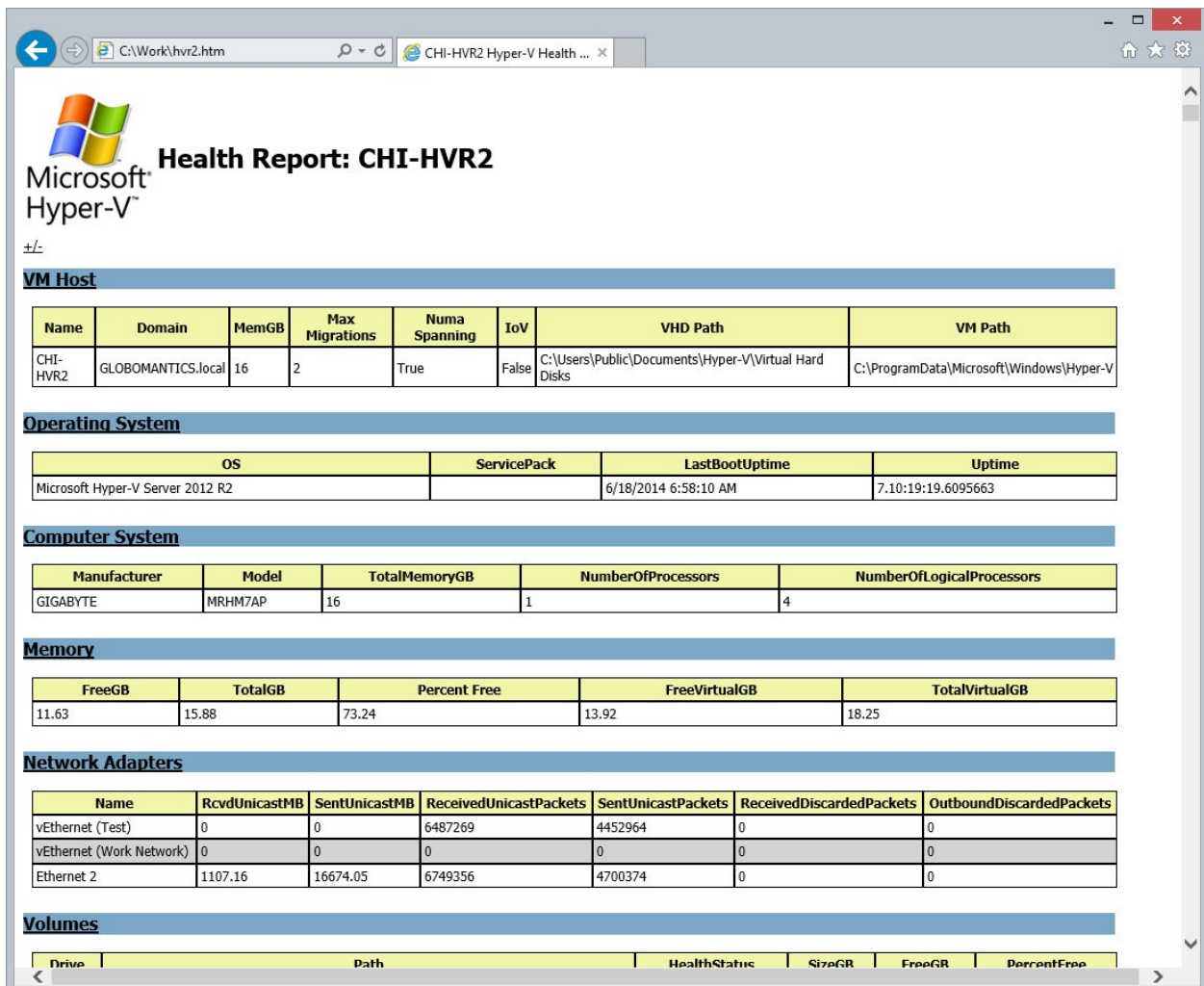


Figure 17

If you use Internet Explorer, you might be prompted to allow blocked content. Go ahead and do so, as the page includes some code to collapse sections. You can click on a heading title to toggle that section or on the +/- link at the top to toggle all sections.

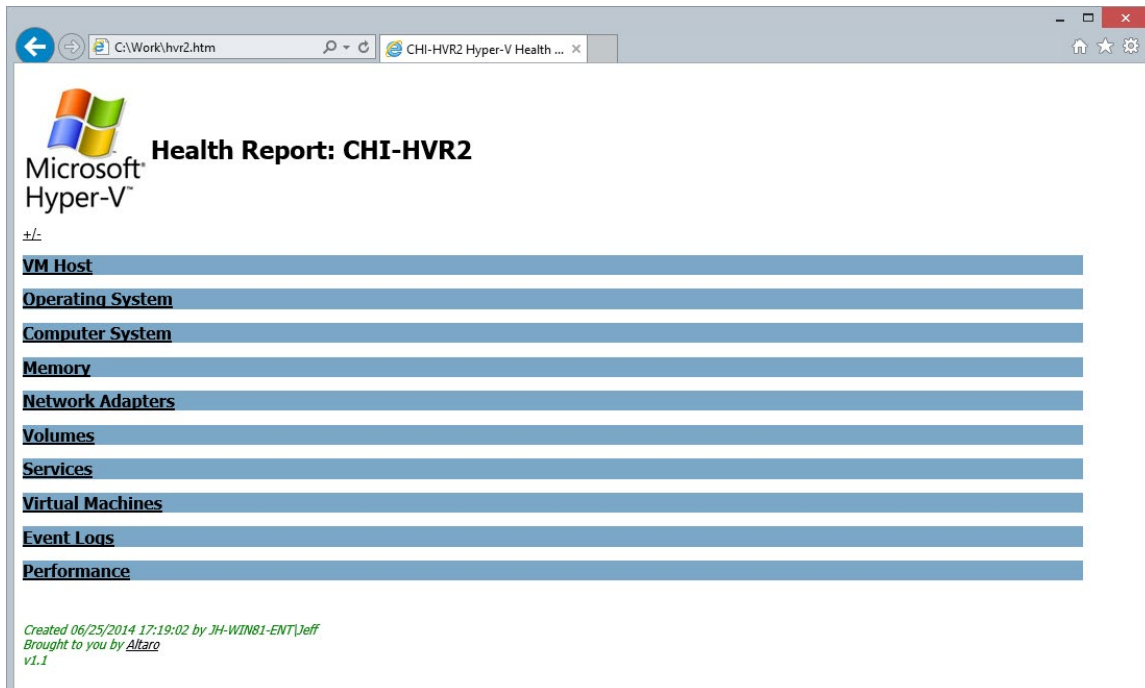


Figure 18

Once collapsed, you can click on any heading to expand.

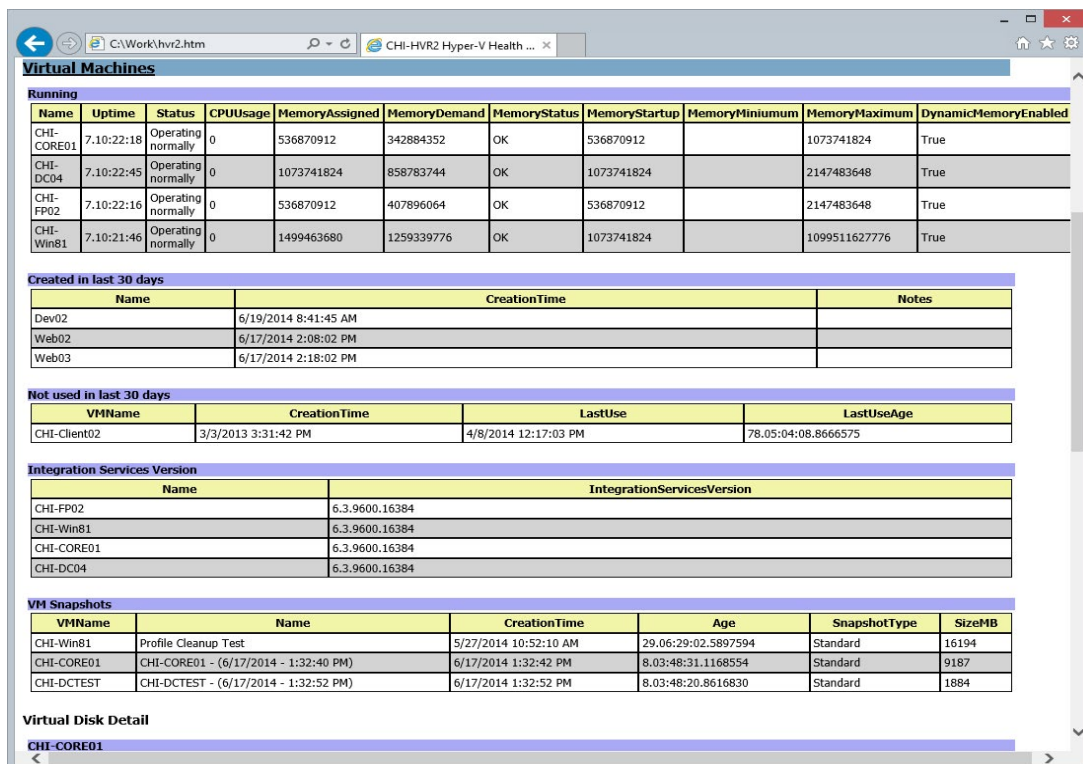


Figure 19

The report primarily shows data based on running virtual machines as you can see in Figure 19. There is a lot of information in the report and you will have to run it for yourself.

Tips and Tricks

Let me wrap up by leaving you with a few tips and tricks for getting the most out of PowerShell. First, set a default value for the ComputerName parameter of the Hyper-V cmdlets. You'll find this very useful if you are running the cmdlets from a client against a remote Hyper-V server. So instead of having to remember to always type:

```
Get-VM -ComputerName chi-hvr2
```

You could define a normal variable in your profile and use that in your commands:

```
$hv = "chi-hvr2"  
Get-VM -ComputerName $hv
```

Or you could use the new preference variable technique:

```
$PSDefaultParameterValues.Add("*-VM:Computername", "chi-hvr2")
```

Now, when I run Get-VM or any cmdlet that ends in -VM, the ComputerName parameter will automatically use the default value. You can always specify a different value.

This default parameter value only lasts for as long as your PowerShell session is open, so add this command to your PowerShell profile if you always want this default.

Another tip, especially if you are new to PowerShell, is to expand nested properties or collections of objects. You've seen this in some recipes already. You might run a command like this:

```
PS C:\> get-vm chi-win81 -computername chi-hvr2 | select VMIntegrationService  
VMIntegrationService  
-----  
{Time Synchronization, Heartbeat, Key-Value Pair Exchange, Shutdown...}
```

The value in side {} is your clue that this can be expanded.

```
PS C:\> get-vm chi-win81 -computername chi-hvr2 | select -expandproperty VMIntegration-  
Service
```

PowerShell will expand the property and write each nested object to the pipeline.

```
Windows PowerShell 4.0

PS C:\> get-vm chi-win81 -computername chi-hvr2 | select VMIntegrationService
VMIntegrationService
-----
{Time Synchronization, Heartbeat, Key-Value Pair Exchange, Shutdown...}

PS C:\> get-vm chi-win81 -computername chi-hvr2 | select -expandproperty VMIntegrationService
VMName      Name                Enabled PrimaryStatusDescription SecondaryStatusDescription
-----
CHI-Win81   Time Synchronization  True    OK
CHI-Win81   Heartbeat              True    OK
CHI-Win81   Key-Value Pair Exchange True    OK
CHI-Win81   Shutdown              True    OK
CHI-Win81   VSS                   True    OK
CHI-Win81   Guest Service Interface True    OK

PS C:\>
```

Figure 20

Finally, even though you can directly modify virtual machines and other objects using the Set-* cmdlets, I prefer to first use a Get-* command to verify I am selecting the right objects.

```
PS C:\> get-vm -Name Demo*
```

Once I am satisfied, I can press the up arrow to bring the last command back and append the necessary Set, Start, Stop or whatever command.

```
PS C:\> get-vm -Name Demo* | Start-VM
```

And as an added level of safety, don't forget to see if the cmdlet supports the -What If parameter.

Additional Resources

I hope the first place you go to for additional PowerShell and Hyper-V resources is the Altaro VM Backup blog at <http://www.altaro.com/vm-backup/>. As you might expect, my own blog (<http://jdhitsolutions.com/blog>) includes a great deal of PowerShell content. If you are struggling to get started with PowerShell you can find my most current listing of essential books and training material at <http://jdhitsolutions.com/blog/essential-powershell-resources/>.



About the Author

Jeffery Hicks is an IT veteran with over 25 years of experience, much of it spent as an IT infrastructure consultant specializing in Microsoft server technologies with an emphasis in automation and efficiency. He is a multi-year recipient of the Microsoft MVP Award in Windows PowerShell. He works today as an independent author, trainer and consultant. Jeff has written for numerous online sites and print publications, is a contributing editor at Petri.com, and is a frequent speaker at technology conferences and user groups. His latest book is PowerShell In Depth: An Administrator's Guide 2nd Ed. You can keep up with Jeff at his blog <http://jdhitsolutions.com/blog>, on Twitter ([@jeffhicks](https://twitter.com/jeffhicks)) and on Google Plus (<https://plus.google.com/+JefferyHicks>).

About Altaro

Altaro Software (www.altaro.com) is a fast growing developer of easy to use backup solutions targeted towards SMBs and focused on Microsoft Hyper-V. Altaro take pride in their software and their high level of personal customer service and support, and it shows; Founded in 2009, Altaro already service over 15,000 satisfied customers worldwide and are a Gold Microsoft Partner for Application Development.

About VM Backup

Altaro VM Backup is an easy to use, yet powerful backup solution for Microsoft Hyper-V, which takes the guesswork out of backing up VMs and does all the complex Hyper-V backup configuration for the admin. This means best in class technology at the most competitive price on the market.

Demonstrating Altaro's dedication to Hyper-V, they were the first backup provider for Hyper-V to support Windows Server 2012 and 2012 R2 and also continues support Windows Server 2008 R2.

For more information on features and pricing, please visit:
<http://www.altaro.com/vm-backup/>

Don't take our word for it – Take it for a spin!

[DOWNLOAD YOUR FREE COPY OF ALTARO VM BACKUP](#)

and enjoy unlimited functionality for 30 days. After your 30-day trial expires you can continue using the product for up to 2 VMs for free, forever. No catch!

Follow Altaro

Like our eBook? **There's more!**

Subscribe to our Hyper-V blog <http://www.altaro.com/vm-backup/> and receive best practices, tips, free Hyper-V PowerShell scripts and more here: <http://www.altaro.com/hyper-v/sign-up/>

Follow Altaro at:



Share this resource!

Share now on:

