5 EASY WAYS TO SAVE TIME ON HYPER-V MANAGEMENT TASKS USING **POWERSHELL**

ALTARO

NEWOUTLOOK.IT

CONTENTS

INTRODUCTION

Unless your datacenter is in a cave you have certainly heard of Windows PowerShell. Perhaps you have even opened up a PowerShell session and tried running a few basic commands. Although if you haven't you are probably not alone. I still encounter many IT Pros just getting started with PowerShell even though it has been out for 10 years and we're now on version 5.0.

PowerShell is more than scripting language or a set of commands. It is a management paradigm that can fundamentally change (hopefully for the better) the way you do your job. If you have Hyper-V management duties, PowerShell should be in your toolbox.

I've written a great deal about managing Hyper-V with PowerShell in the past. You can find many of those articles on the Altaro blog. This eBook intends to take a slightly different approach. My goal is to demonstrate that PowerShell is worth the investment to learn.

I hear from people all the time at conferences or classes I teach that they wish they had learned PowerShell earlier or don't know why they put it off.

Once you understand the fundamentals you'll realize that managing "things" with PowerShell isn't that difficult.

It doesn't matter if you are managing a bunch of Active Directory user accounts or Hyper-V virtual machines. The fundamental principles don't change and the rewards can be significant.

PowerShell doesn't mean you have to write a script to do everything. A great PowerShell feature is that there is really no difference between running PowerShell commands interactively at a prompt or running a PowerShell script. The benefit of a script is that you only have to type the commands once, it can be documented and it will run the same way every time.

The concept of this eBook is to demonstrate how PowerShell automation can help you save time when it comes to managing Hyper-V. Even though my PowerShell "answers" will be shown as PowerShell scripts, you could also run many of the commands interactively in the console. Actually, I hope you'll take my code examples and build your own Hyper-V management tools and scripts.

This eBook will go through a list of common Hyper-V management tasks and demonstrate how and why you might consider using PowerShell instead of manually accomplishing the task through the Hyper-V management console. I hope to show you how much time you can save.

Some of the time savings may not seem like much, but over the course of a year, or even a week, they can really add up. Plus, if you employ re-usable scripts and functions, anyone can run them even if they don't know PowerShell. They simply need access to the script files, the necessary admin permissions, and some basic training in how to run a PowerShell script or command.

My approach for all of this follows 2 principles:

1. First, everything is done from the desktop. There is no need to logon to a server. There is no need to open up a remote desktop connection.

If that is the way you are used to managing servers, you need to break yourself of that habit.

2. The 2nd principle is "If you can do it for one you can do it for many."

PowerShell makes it very easy to manage things at scale. By that I mean instead of managing one thing at a time, like a virtual machine, it is just as easy to manage 10, 100 or 1000 of them. When I am creating PowerShell tools I'm always thinking about working remotely and managing multiple items at once.

REQUIREMENTS

For many of my examples **I will be using PowerShell 5.0 on Windows 10**, although PowerShell 4.0 should suffice. **I am also using version 2.0 of the PowerShell Hyper-V module**. More on that in a moment.

I am also **logged in with an account that has administrator privileges in my domain**. In fact, all of my Hyper-V hosts and most of my virtual machines are in the domain. Depending on your available version you may or may not be able to run some of my examples with alternate credentials.

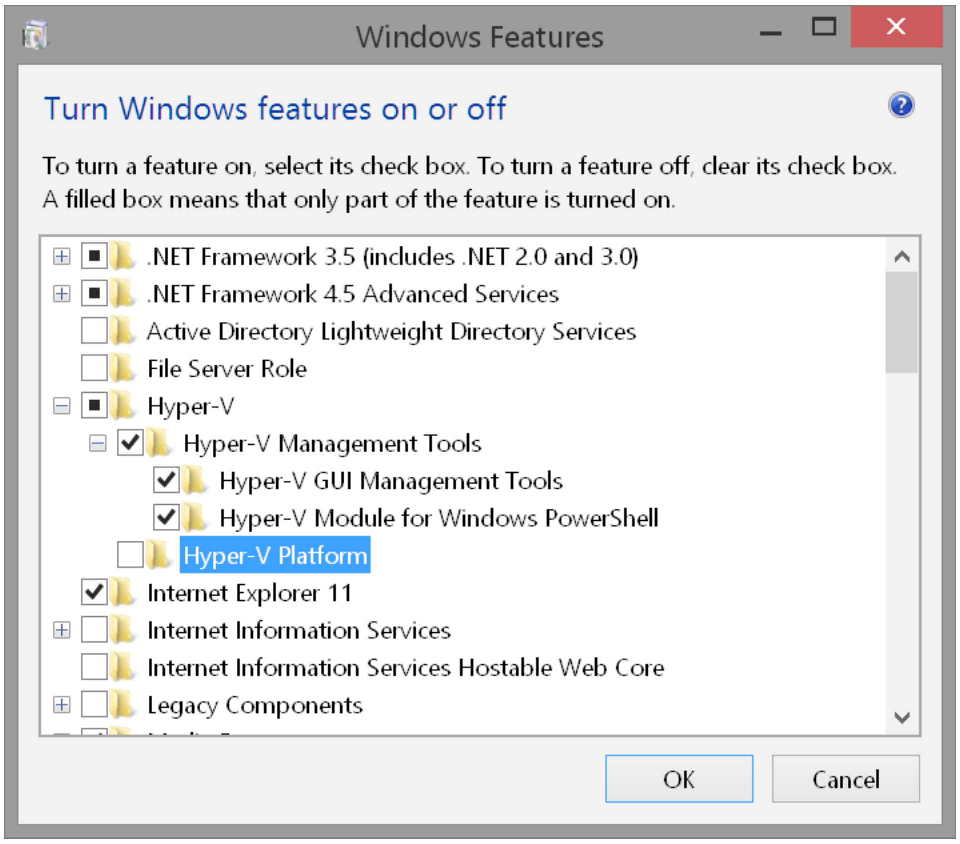
You should always read full help and examples for any PowerShell command that you don't recognize like this:

```
Help get-vmhost –full
```

I have a few Hyper-V hosts running Window Server 2012 R2 and Windows Server 2016 Technical Preview 5. As long as you are running Windows Server 2012 or later I think you should be fine.

As I mentioned, I am going to be demonstrating from the client, which is a practice you should be following. This means you need the PowerShell module for managing Hyper-V. I'm assuming you have either Windows 8.1 or Windows 10. It is possible to set up implicit remoting and use the PowerShell commands from a Hyper-V host but that is an advanced topic that is out of scope for this eBook.

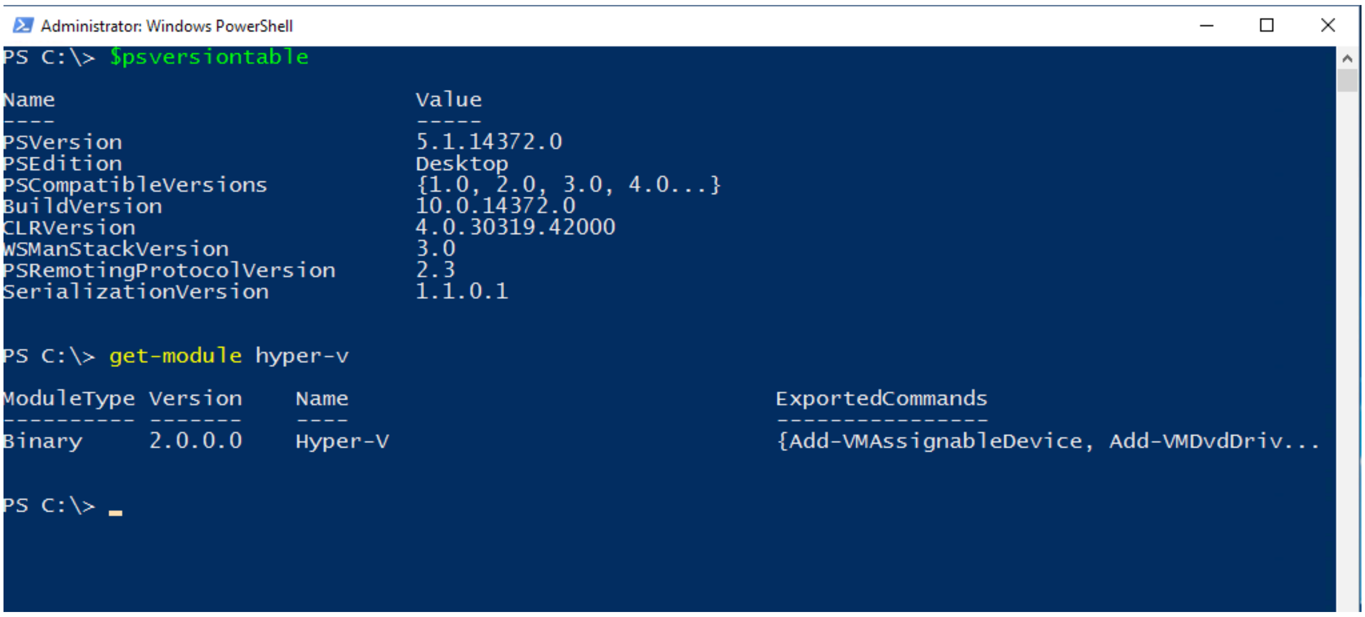
You ***do not*** need to install the hypervisor on the client desktop. But you may need to enable the management tools. To do so, open Control Panel – Programs – Turn Windows Features On or Off and expand Hyper-V. Check the boxes for the Hyper-V management tools.



To turn a feature on, select its check box. To turn a feature off, clear its check box. A filled box means that only part of the feature is turned on.

- ■ .NET Framework 3.5 (includes .NET 2.0 and 3.0)
- ■ .NET Framework 4.5 Advanced Services
- ☐ Active Directory Lightweight Directory Services
- ☐ File Server Role
- ■ Hyper-V
  - ✓ Hyper-V Management Tools
    - ✓ Hyper-V GUI Management Tools
    - ✓ Hyper-V Module for Windows PowerShell
  - ☐ Hyper-V Platform
- ✓ Internet Explorer 11
- ☐ Internet Information Services
- ☐ Internet Information Services Hostable Web Core
- ☐ Legacy Components

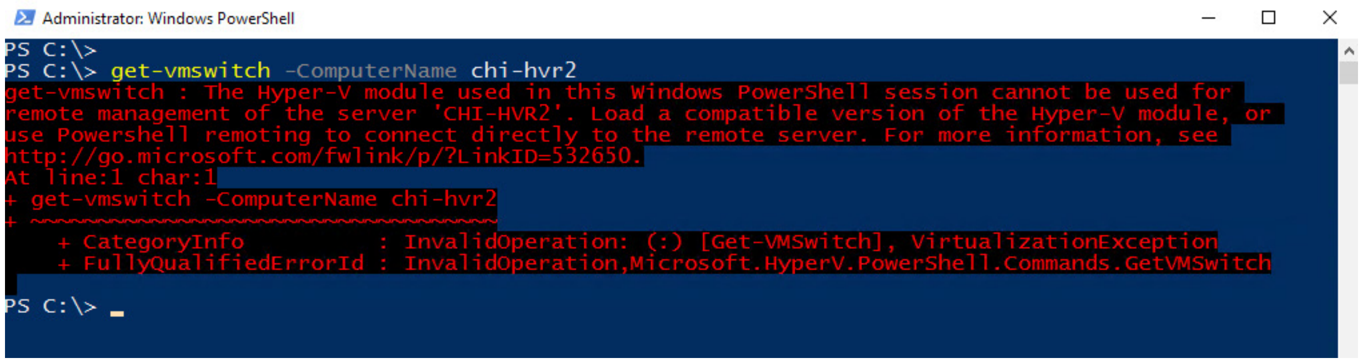
You can always compare my versions with yours:



```
PS C:\> $psversiontable

Name                           Value
----                           -----
PSVersion                      5.1.14372.0
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.14372.0
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1


PS C:\> get-module hyper-v

ModuleType Version    Name                                ExportedCommands
---------- -------    ----                                ----------------
Binary     2.0.0.0    Hyper-V                             {Add-VMAssignableDevice, Add-VMDvdDriv...


PS C:\> _
```

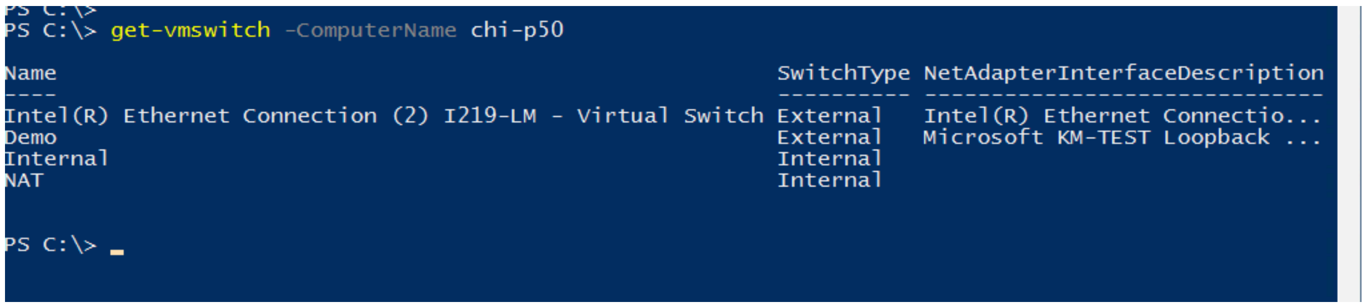
MANAGING MULTIPLE VERSIONS

As I write this, Microsoft has 2 versions of their PowerShell commands for managing Hyper-V. Which version you have depends the version of your operating system and PowerShell. Today, there are potential issues using the commands, especially if you have a mix of Hyper-V servers. In my test environment I have Hyper-V servers running Windows Server 2016 Technology Preview 5 and Window Hyper-V Server 2012. If you try to run commands from a newer system to an older system, you might see an error like this:



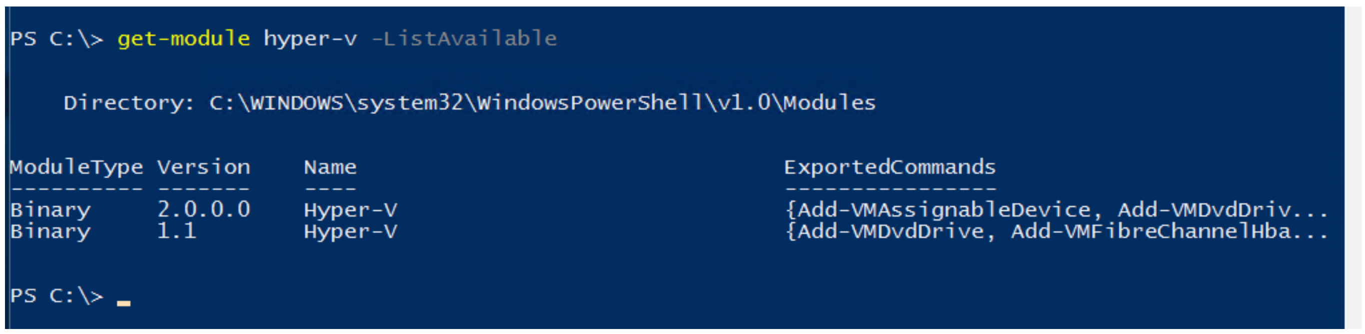
```
PS C:\>
PS C:\> get-vmswitch -ComputerName chi-hvr2
get-vmswitch : The Hyper-V module used in this Windows PowerShell session cannot be used for
remote management of the server 'CHI-HVR2'. Load a compatible version of the Hyper-V module, or
use Powershell remoting to connect directly to the remote server. For more information, see
http://go.microsoft.com/fwlink/p/?LinkID=532650.
At line:1 char:1
+ get-vmswitch -ComputerName chi-hvr2
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [Get-VMSwitch], VirtualizationException
    + FullyQualifiedErrorId : InvalidOperation,Microsoft.HyperV.PowerShell.Commands.GetVMSwitch

PS C:\> _
```

Although the command works just fine against a newer server.



```
PS C:\>
PS C:\> get-vmswitch -ComputerName chi-p50

Name                                              SwitchType NetAdapterInterfaceDescription
----                                              ---------- ------------------------------
Intel(R) Ethernet Connection (2) I219-LM - Virtual Switch External   Intel(R) Ethernet Connectio...
Demo                                              External   Microsoft KM-TEST Loopback ...
Internal                                          Internal
NAT                                               Internal

PS C:\> _
```

I am going to try and avoid doing anything that has to cross this type of "version boundary", and hopefully all your servers will be running the same version of Hyper-V. If not, you might have to take a few extra steps to load a compatible version. Look at my Windows 10 client:



```
PS C:\> get-module hyper-v -ListAvailable

    Directory: C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules

ModuleType Version    Name                                ExportedCommands
---------- -------    ----                                ----------------
Binary     2.0.0.0    Hyper-V                             {Add-VMAssignableDevice, Add-VMDvdDriv...
Binary     1.1        Hyper-V                             {Add-VMDvdDrive, Add-VMFibreChannelHba...

PS C:\> _
```

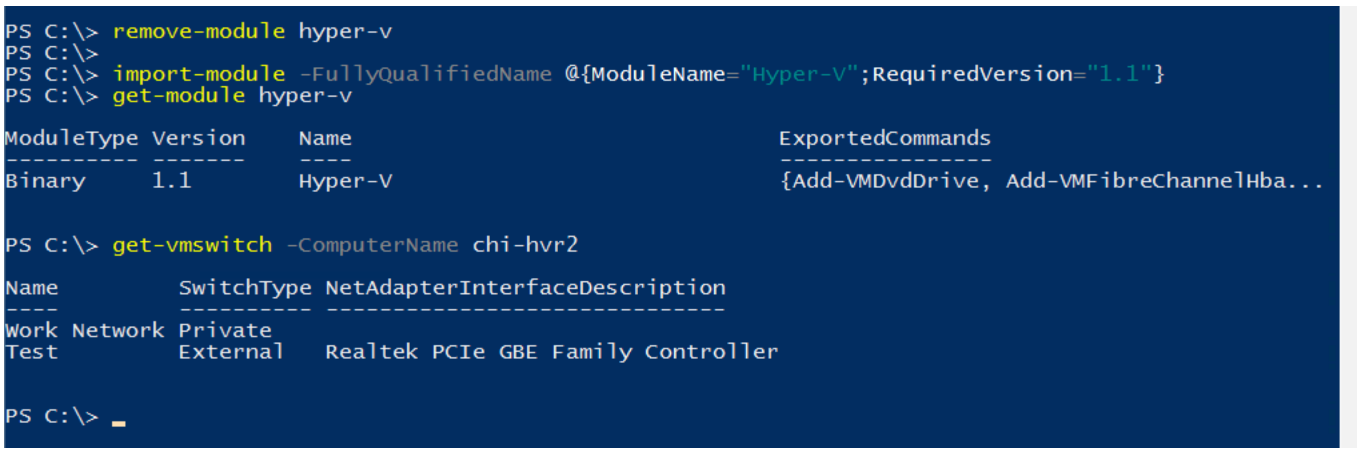
I have two versions of the Hyper-V module. By default, PowerShell uses the commands from the newest version. But instead I want to load the older version as it will be able to manage both older and newer servers. There are few new cmdlets in version 2 of the module as well as a few new parameters, but for what we will be looking at I don't think it will make much difference.

First, because I've already loaded v2 of the Hyper-V module, I'll unload it.

```
Remove-Module Hyper-V
```

Now I can explicitly load the required version using either of these commands:

```
Import-Module -FullyQualifiedName @{ModuleName="Hyper-V";
RequiredVersion="1.1"}
Import-Module Hyper-V –maximumversion 1.1
```



```
PS C:\> remove-module hyper-v
PS C:\>
PS C:\> import-module -FullyQualifiedName @{ModuleName="Hyper-V";RequiredVersion="1.1"}
PS C:\> get-module hyper-v

ModuleType Version    Name                                ExportedCommands
---------- -------    ----                                ----------------
Binary     1.1        Hyper-V                             {Add-VMDvdDrive, Add-VMFibreChannelHba...

PS C:\> get-vmswitch -ComputerName chi-hvr2

Name         SwitchType NetAdapterInterfaceDescription
----         ---------- ------------------------------
Work Network Private
Test         External   Realtek PCIe GBE Family Controller

PS C:\> _
```

This version works with the older Hyper-V server. I'm hoping this won't be an issue for you.

Finally, to see all of the available commands in the module, use Get-Command:

```
Get-Command –module Hyper-V
```

CAVEATS

If you are also running System Center Virtual Machine Manager or have those PowerShell cmdlets installed, be careful. There are a number of cmdlets that share the same name between the SCVMM and Hyper-V modules. Even though the name is the same, the commands are different and if the SCVMM version gets picked instead you might run into problems. If this becomes a problem, you can modify my code examples to explicitly reference the Hyper-V module:

```
hyper-v\get-vm –computername chi-hvr2
```

I will also be running commands from the traditional PowerShell console. You can certainly load my script files into the PowerShell ISE and run them from there but that is not required. And speaking of the scripts, you can download a zip file with sample scripts here:

DOWNLOAD SAMPLE SCRIPTS

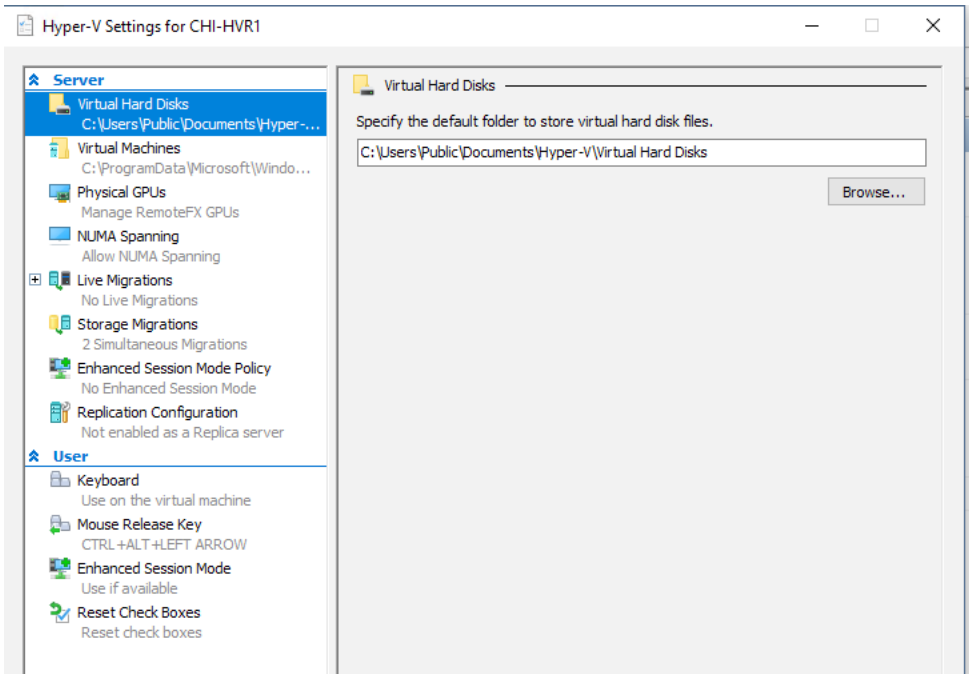
The script files are provided for educational and demonstration purposes only. They are not necessarily production ready and I can't guarantee they will work in your environment. You should test everything and build your own PowerShell solutions in a non-production environment.

TASK # 1 CONFIGURE A HYPER-V HOST

Let's get started with managing Hyper-V at the host itself. Typically, I don't expect you to have to make many changes to a Hyper-V host. Generally, once it is set up I think most admins leave it alone. On one hand since you only have to do it once, using the graphical Hyper-V Manager isn't necessarily a bad choice. But it can be time consuming and certainly doesn't scale.

THE TRADITIONAL WAY

There are a number of settings you might want to configure on a per host basis from default locations, to NUMA Spanning to virtual machine migration. Not to mention tasks such as creating new virtual switches. Generally, I'm talking about the settings you see when viewing server properties in the Hyper-V Manager.



Hyper-V Settings for CHI-HVR1

Server
- Virtual Hard Disks — C:\Users\Public\Documents\Hyper-...
- Virtual Machines — C:\ProgramData\Microsoft\Windo...
- Physical GPUs — Manage RemoteFX GPUs
- NUMA Spanning — Allow NUMA Spanning
- Live Migrations — No Live Migrations
- Storage Migrations — 2 Simultaneous Migrations
- Enhanced Session Mode Policy — No Enhanced Session Mode
- Replication Configuration — Not enabled as a Replica server

User
- Keyboard — Use on the virtual machine
- Mouse Release Key — CTRL+ALT+LEFT ARROW
- Enhanced Session Mode — Use if available
- Reset Check Boxes — Reset check boxes

Virtual Hard Disks

Specify the default folder to store virtual hard disk files.

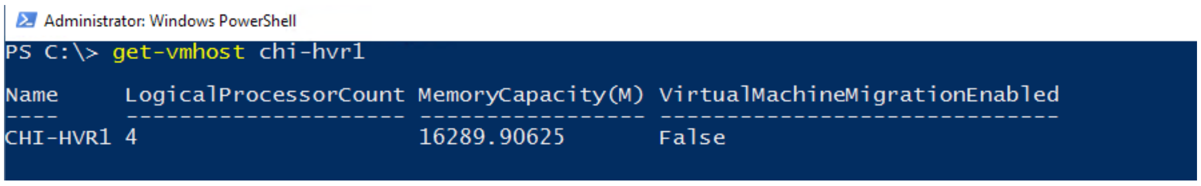
C:\Users\Public\Documents\Hyper-V\Virtual Hard Disks

Browse...

Depending on what you need to configure this could take as long as 5 to 20 minutes per host. That seems like a long time to me. Plus, anything you do manually is prone to human error.

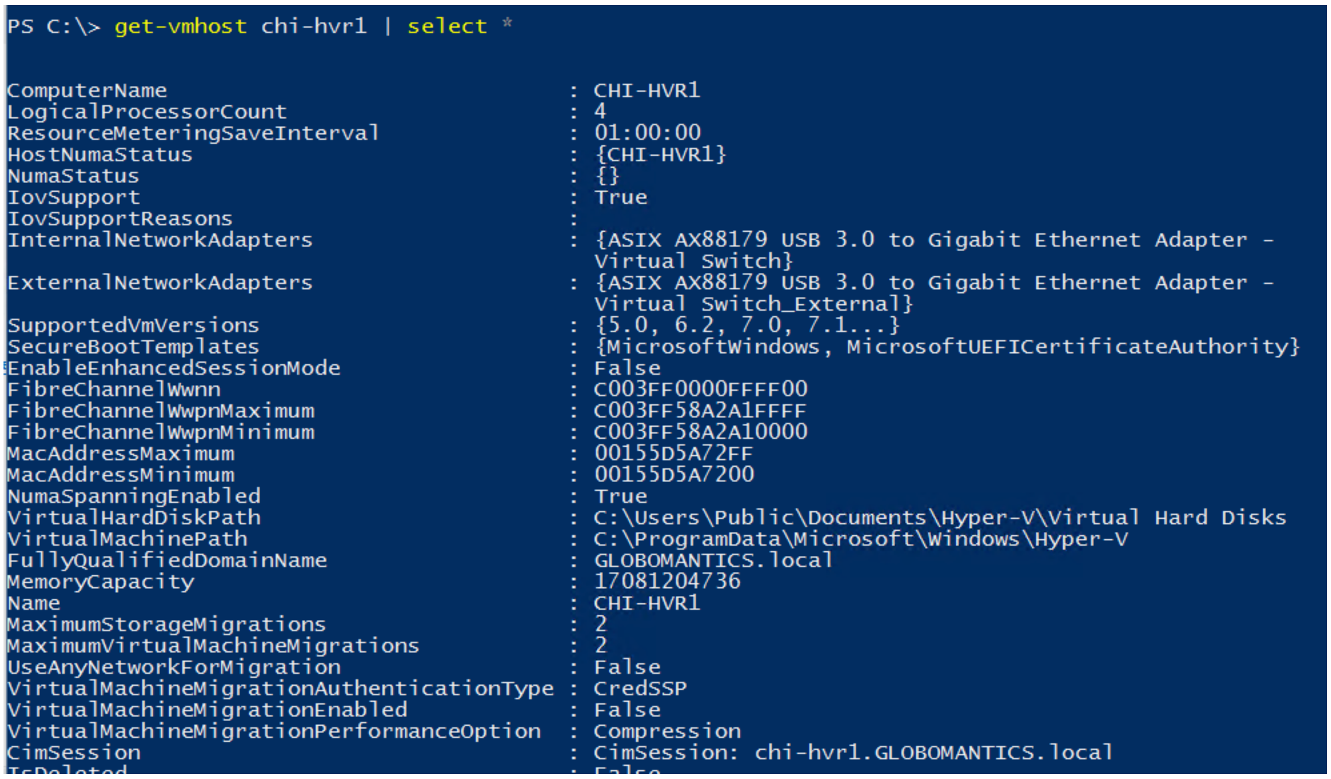
THE POWERSHELL WAY

You can the Get-VMHost cmdlet to view the current configuration.



```
PS C:\> get-vmhost chi-hvr1

Name     LogicalProcessorCount MemoryCapacity(M) VirtualMachineMigrationEnabled
----     --------------------- ----------------- ------------------------------
CHI-HVR1 4                     16289.90625       False
```

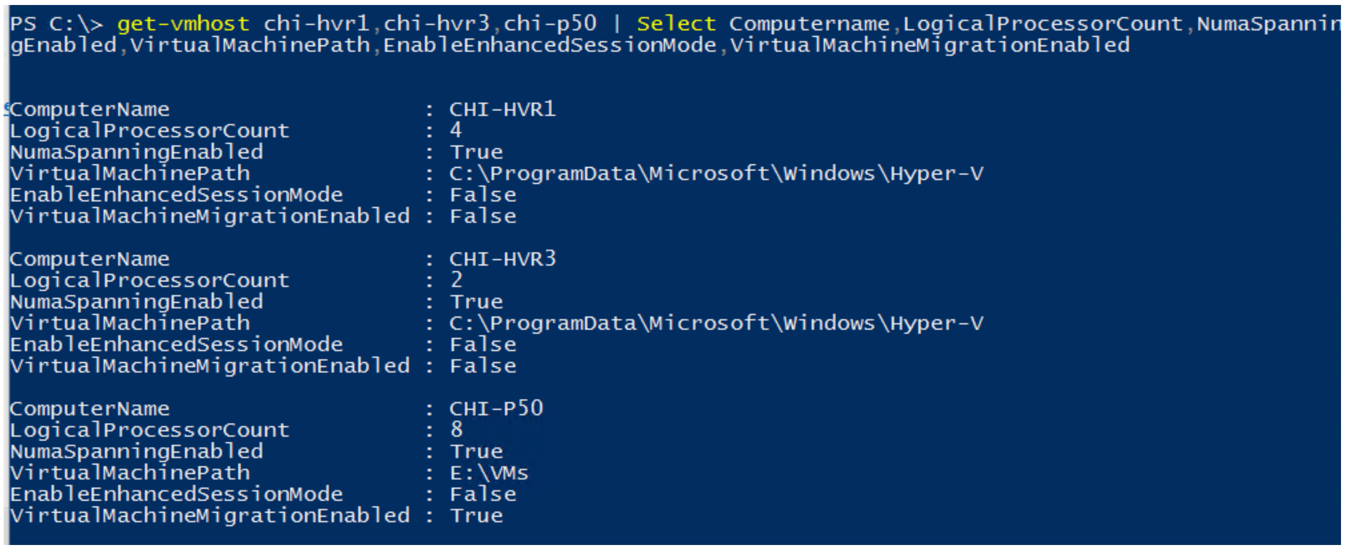
Remember, in PowerShell everything is an object and what you see by default isn't necessarily everything. Use Select-Object to view all properties.



```
PS C:\> get-vmhost chi-hvr1 | select *


ComputerName                              : CHI-HVR1
LogicalProcessorCount                     : 4
ResourceMeteringSaveInterval              : 01:00:00
HostNumaStatus                            : {CHI-HVR1}
NumaStatus                                : {}
IovSupport                                : True
IovSupportReasons                         :
InternalNetworkAdapters                   : {ASIX AX88179 USB 3.0 to Gigabit Ethernet Adapter -
                                            Virtual Switch}
ExternalNetworkAdapters                   : {ASIX AX88179 USB 3.0 to Gigabit Ethernet Adapter -
                                            Virtual Switch_External}
SupportedVmVersions                       : {5.0, 6.2, 7.0, 7.1...}
SecureBootTemplates                       : {MicrosoftWindows, MicrosoftUEFICertificateAuthority}
EnableEnhancedSessionMode                 : False
FibreChannelWwnn                          : C003FF0000FFFF00
FibreChannelWwpnMaximum                   : C003FF58A2A1FFFF
FibreChannelWwpnMinimum                   : C003FF58A2A10000
MacAddressMaximum                         : 00155D5A72FF
MacAddressMinimum                         : 00155D5A7200
NumaSpanningEnabled                       : True
VirtualHardDiskPath                       : C:\Users\Public\Documents\Hyper-V\Virtual Hard Disks
VirtualMachinePath                        : C:\ProgramData\Microsoft\Windows\Hyper-V
FullyQualifiedDomainName                  : GLOBOMANTICS.local
MemoryCapacity                            : 17081204736
Name                                      : CHI-HVR1
MaximumStorageMigrations                  : 2
MaximumVirtualMachineMigrations           : 2
UseAnyNetworkForMigration                 : False
VirtualMachineMigrationAuthenticationType : CredSSP
VirtualMachineMigrationEnabled            : False
VirtualMachineMigrationPerformanceOption  : Compression
CimSession                                : CimSession: chi-hvr1.GLOBOMANTICS.local
IsDeleted                                 : False
```

Once you know the property names, you can select them. In fact, you can select them for multiple servers at the same time.

```
get-vmhost chi-hvr1,chi-hvr3,chi-p50 |
Select Computername,LogicalProcessorCount,
NumaSpanningEnabledVirtualMachinePath,
EnableEnhancedSessionMode,VirtualMachineMigrationEnabled
```



```
PS C:\> get-vmhost chi-hvr1,chi-hvr3,chi-p50 | Select Computername,LogicalProcessorCount,NumaSpannin
gEnabled,VirtualMachinePath,EnableEnhancedSessionMode,VirtualMachineMigrationEnabled


ComputerName                   : CHI-HVR1
LogicalProcessorCount          : 4
NumaSpanningEnabled            : True
VirtualMachinePath             : C:\ProgramData\Microsoft\Windows\Hyper-V
EnableEnhancedSessionMode      : False
VirtualMachineMigrationEnabled : False

ComputerName                   : CHI-HVR3
LogicalProcessorCount          : 2
NumaSpanningEnabled            : True
VirtualMachinePath             : C:\ProgramData\Microsoft\Windows\Hyper-V
EnableEnhancedSessionMode      : False
VirtualMachineMigrationEnabled : False

ComputerName                   : CHI-P50
LogicalProcessorCount          : 8
NumaSpanningEnabled            : True
VirtualMachinePath             : E:\VMs
EnableEnhancedSessionMode      : False
VirtualMachineMigrationEnabled : True
```

Let's say that the company standard is for Hyper-V servers to have enhanced session mode and virtual machine migration enabled and the virtual machine path to be C:\VMs. Using the Hyper-V Manager, which was already open, it took me about 45 seconds per server to complete.

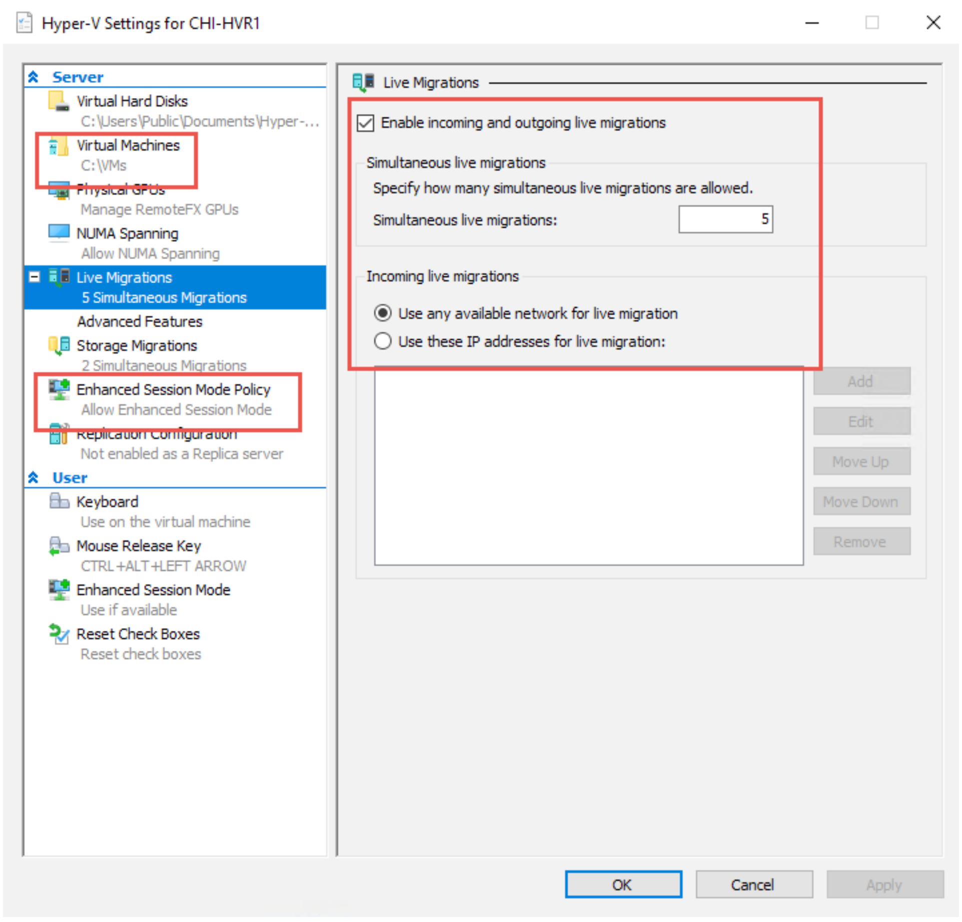
Instead let's use Set-VMHost to make the same changes. I've already created and verified C:\VMs separately, although you could certainly incorporate it into your PowerShell solution. Remember, read full help and examples for Set-VMHost.

```
Set-VMHost -ComputerName chi-hvr1,chi-hvr3,chi-p50-VirtualMachinePath
C:\VMs -EnableEnhancedSessionMode $True
-UseAnyNetworkForMigration $True -MaximumVirtualMachineMigrations
5 -VirtualMachineMigrationAuthenticationType Kerberos
-VirtualMachineMigrationPerformanceOption Compression
```

With this one-line command, which took about 1 second to run to configure 3 servers configured the servers the way I wanted. Although, there is actually one more step:

```
Enable-VMMigration -ComputerName chi-hvr1,chi-hvr3,chi-p50
```

I needed to run a command to tick the check box.


Hyper-V Settings for CHI-HVR1

Server
- Virtual Hard Disks: C:\Users\Public\Documents\Hyper-...
- Virtual Machines: C:\VMs
- Physical GPUs: Manage RemoteFX GPUs
- NUMA Spanning: Allow NUMA Spanning
- Live Migrations: 5 Simultaneous Migrations
  - Advanced Features
- Storage Migrations: 2 Simultaneous Migrations
- Enhanced Session Mode Policy: Allow Enhanced Session Mode
- Replication Configuration: Not enabled as a Replica server

User
- Keyboard: Use on the virtual machine
- Mouse Release Key: CTRL+ALT+LEFT ARROW
- Enhanced Session Mode: Use if available
- Reset Check Boxes: Reset check boxes

Live Migrations

[✓] Enable incoming and outgoing live migrations

Simultaneous live migrations

Specify how many simultaneous live migrations are allowed.

Simultaneous live migrations: 5

Incoming live migrations

[● Use any available network for live migration]

○ Use these IP addresses for live migration:

You can run the command interactively, or put it in PowerShell script file and run it that way. The benefit is that anyone can run it at any time to ensure Hyper-V hosts are properly configured. The script also serves as a record of what was done. Once you gain experience creating PowerShell tools you can even turn it into a re-usable function that takes computer names as parameters, logs activity to a file or database, and perhaps even allows for different migration settings.

VERDICT

It is pretty hard to argue that taking a few seconds to configure multiple servers with a single command is not a compelling argument for PowerShell. Sure, you'll take a little time up front to learn the commands and syntax but the more experience you gain, the shorter amount of time you will require. And when you are finished, if you save your command to a script file you have a re-usable tool which is much more effective than giving someone a manual with a bunch of screen shots and click instructions. Unless you like that sort of work!

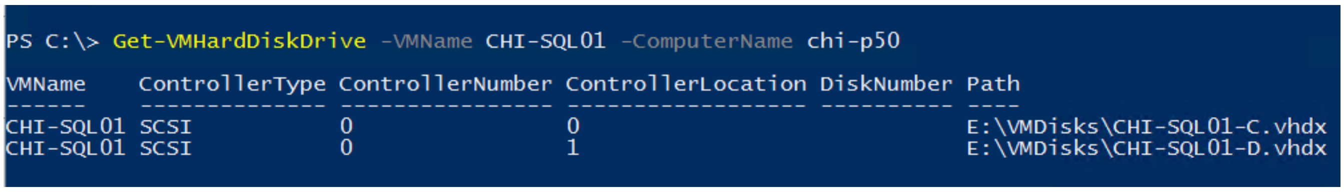
TASK # 2 MANAGE VIRTUAL MACHINE STORAGE

Another task that probably should be done more often but probably isn't is managing a virtual machine's storage. By that I mean the VHD or VHDX files associated with a given virtual machine. I'm betting this isn't done often because it is a tedious process to do manually.

REPORTING

Sure, it isn't too difficult to check a single VHD file. In the GUI this might take 20-30 seconds per disk. Even longer if you need to prepare some type of report for customers or managers. Instead, PowerShell makes this a snap.

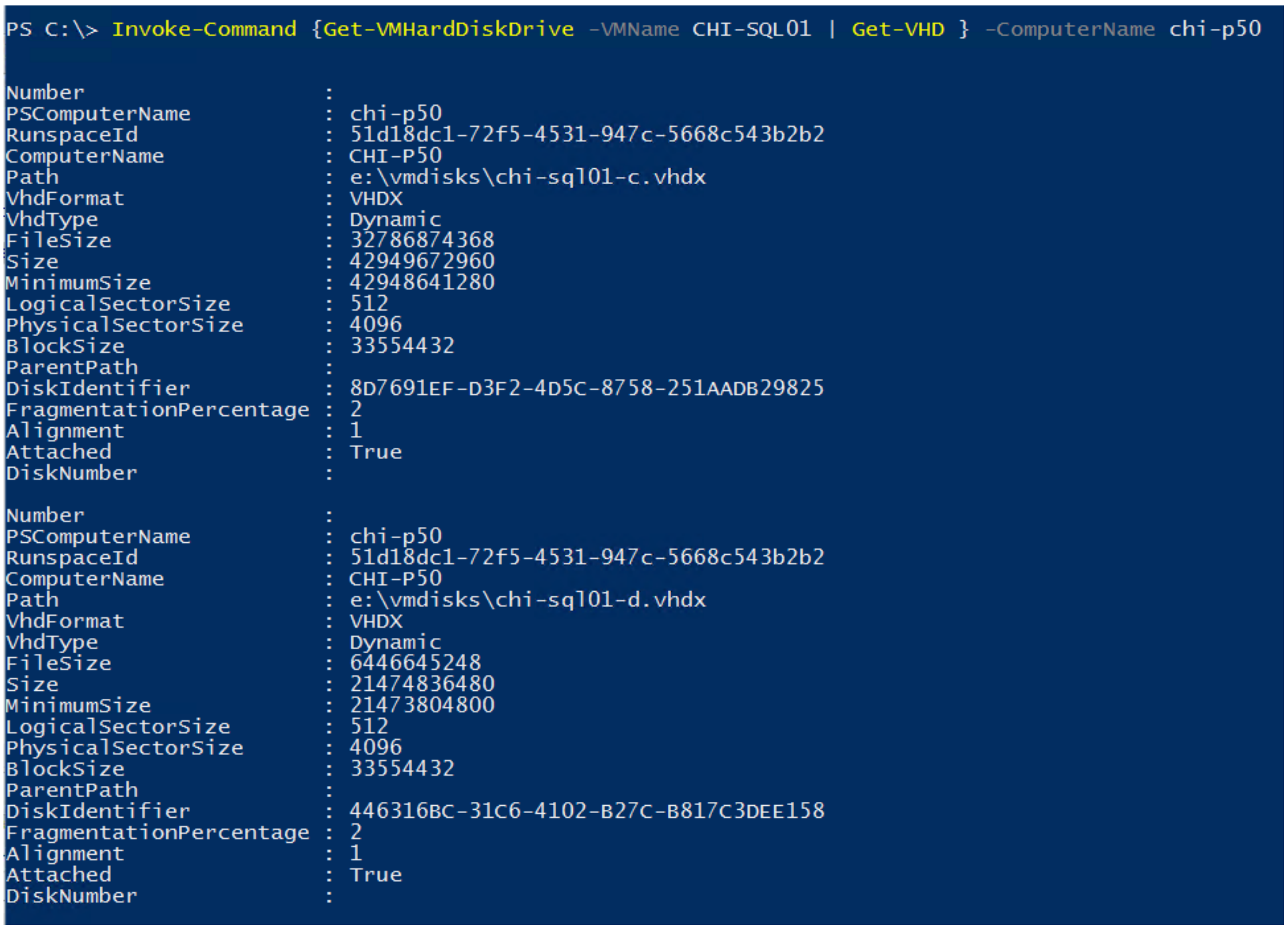
```
Get-VMHardDiskDrive -VMName CHI-SQL01 -ComputerName chi-p50
```



```
PS C:\> Get-VMHardDiskDrive -VMName CHI-SQL01 -ComputerName chi-p50

VMName    ControllerType ControllerNumber ControllerLocation DiskNumber Path
------    -------------- ---------------- ------------------ ---------- ----
CHI-SQL01 SCSI           0                0                             E:\VMDisks\CHI-SQL01-C.vhdx
CHI-SQL01 SCSI           0                1                             E:\VMDisks\CHI-SQL01-D.vhdx
```

A command like this takes less than half a seconds and can at least see the disk files. However, if we want to examine the VHDX file directly that will take an extra step or two. There is a cmdlet called Get-VHD which will be very useful but it needs access to the files. In my example above I am running the command from my client desktop but the file paths are relative to the server. This means I need to run Get-VHD on the Hyper-V host. Fortunately, this is pretty easy to do using PowerShell remoting.

```
Invoke-Command {Get-VMHardDiskDrive -VMName CHI-SQL01 | Get-VHD }
-ComputerName chi-p50
```

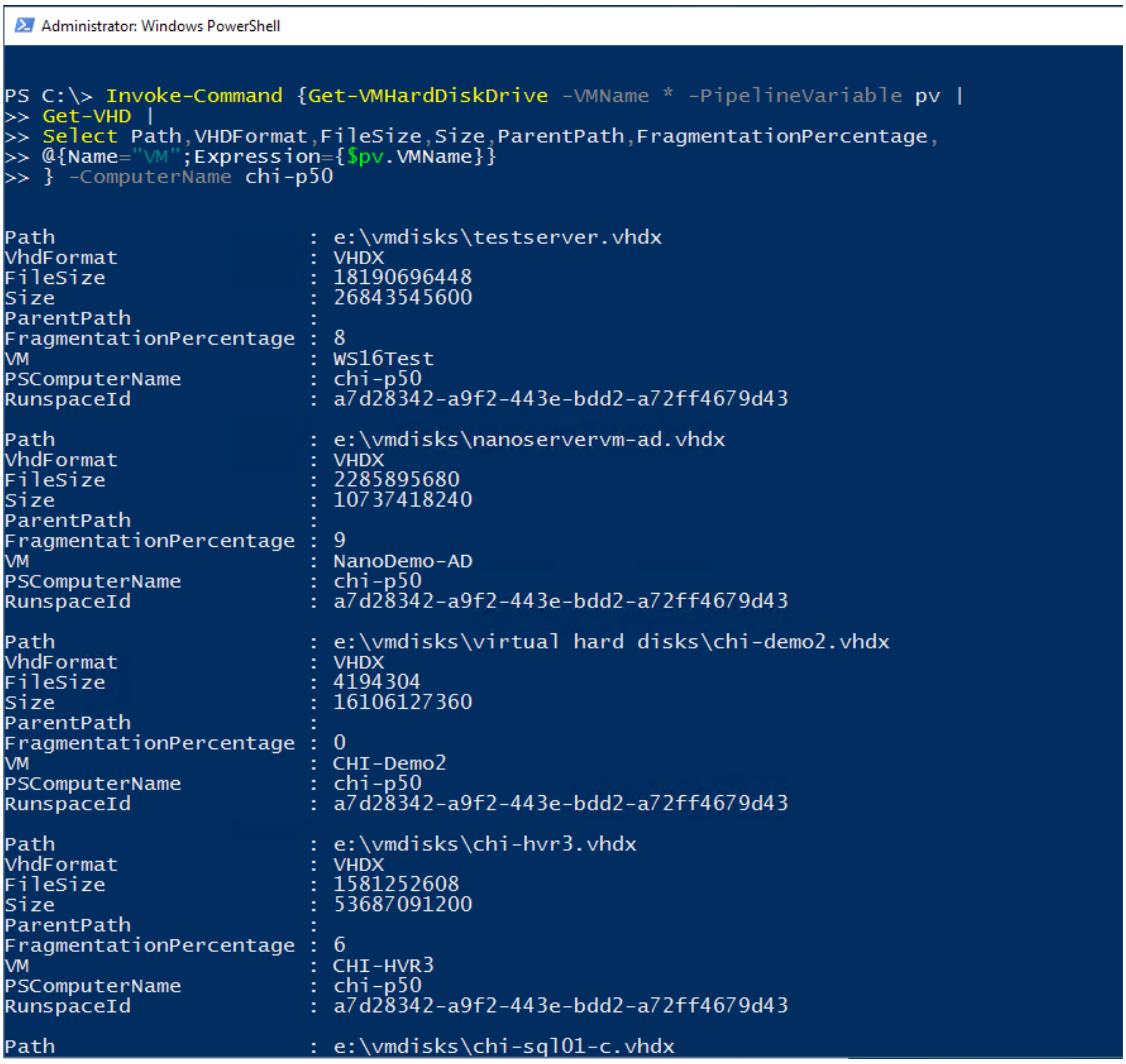


```
PS C:\> Invoke-Command {Get-VMHardDiskDrive -VMName CHI-SQL01 | Get-VHD } -ComputerName chi-p50


Number                  :
PSComputerName          : chi-p50
RunspaceId              : 51d18dc1-72f5-4531-947c-5668c543b2b2
ComputerName            : CHI-P50
Path                    : e:\vmdisks\chi-sql01-c.vhdx
VhdFormat               : VHDX
VhdType                 : Dynamic
FileSize                : 32786874368
Size                    : 42949672960
MinimumSize             : 42948641280
LogicalSectorSize       : 512
PhysicalSectorSize      : 4096
BlockSize               : 33554432
ParentPath              :
DiskIdentifier          : 8D7691EF-D3F2-4D5C-8758-251AADB29825
FragmentationPercentage : 2
Alignment               : 1
Attached                : True
DiskNumber              :

Number                  :
PSComputerName          : chi-p50
RunspaceId              : 51d18dc1-72f5-4531-947c-5668c543b2b2
ComputerName            : CHI-P50
Path                    : e:\vmdisks\chi-sql01-d.vhdx
VhdFormat               : VHDX
VhdType                 : Dynamic
FileSize                : 6446645248
Size                    : 21474836480
MinimumSize             : 21473804800
LogicalSectorSize       : 512
PhysicalSectorSize      : 4096
BlockSize               : 33554432
ParentPath              :
DiskIdentifier          : 446316BC-31C6-4102-B27C-B817C3DEE158
FragmentationPercentage : 2
Alignment               : 1
Attached                : True
DiskNumber              :
```

This doesn’t take much longer and provides some useful information. In fact, since I can do this for a single virtual machine, I can do it for all of them.

```
Invoke-Command {Get-VMHardDiskDrive -VMName * -PipelineVariable pv |
Get-VHD |
Select Path,VHDFormat,FileSize,Size,ParentPath,FragmentationPercentage,
@{Name=”VM”;Expression={$pv.VMName}}
} -ComputerName chi-p50
```

This is technically a one-line command that took about 5 seconds to run against 22 disk files.



```
PS C:\> Invoke-Command {Get-VMHardDiskDrive -VMName * -PipelineVariable pv |
>> Get-VHD |
>> Select Path,VHDFormat,FileSize,Size,ParentPath,FragmentationPercentage,
>> @{Name="VM";Expression={$pv.VMName}}
>> } -ComputerName chi-p50


Path                    : e:\vmdisks\testserver.vhdx
VhdFormat               : VHDX
FileSize                : 18190696448
Size                    : 26843545600
ParentPath              :
FragmentationPercentage : 8
VM                      : WS16Test
PSComputerName          : chi-p50
RunspaceId              : a7d28342-a9f2-443e-bdd2-a72ff4679d43

Path                    : e:\vmdisks\nanoservervm-ad.vhdx
VhdFormat               : VHDX
FileSize                : 2285895680
Size                    : 10737418240
ParentPath              :
FragmentationPercentage : 9
VM                      : NanoDemo-AD
PSComputerName          : chi-p50
RunspaceId              : a7d28342-a9f2-443e-bdd2-a72ff4679d43

Path                    : e:\vmdisks\virtual hard disks\chi-demo2.vhdx
VhdFormat               : VHDX
FileSize                : 4194304
Size                    : 16106127360
ParentPath              :
FragmentationPercentage : 0
VM                      : CHI-Demo2
PSComputerName          : chi-p50
RunspaceId              : a7d28342-a9f2-443e-bdd2-a72ff4679d43

Path                    : e:\vmdisks\chi-hvr3.vhdx
VhdFormat               : VHDX
FileSize                : 1581252608
Size                    : 53687091200
ParentPath              :
FragmentationPercentage : 6
VM                      : CHI-HVR3
PSComputerName          : chi-p50
RunspaceId              : a7d28342-a9f2-443e-bdd2-a72ff4679d43

Path                    : e:\vmdisks\chi-sql01-c.vhdx
```

I had to do a little advanced trick with the common PipelineVariable parameter so that I could temporarily capture the virtual machine name from the first part and use it as a custom property in the last part. Invoke-Command supports multiple computer names so I could run this command at the same time against all my Hyper-V hosts and export the information to a CSV file.

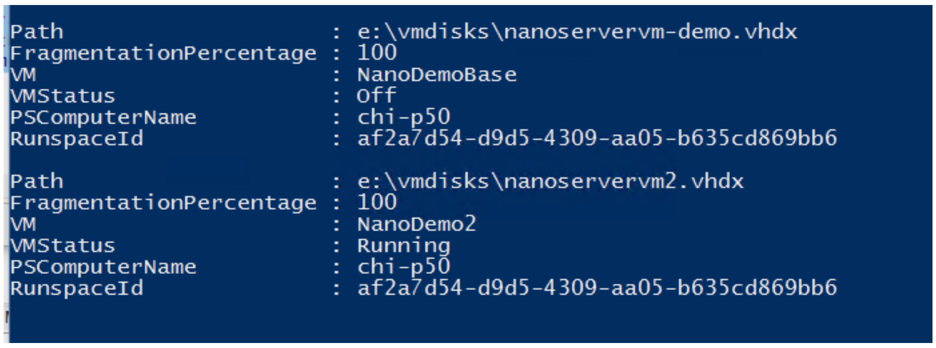
```
Invoke-Command {Get-VMHardDiskDrive -VMName * -PipelineVariable pv |
Get-VHD |
Select Path,VHDFormat,FileSize,Size,ParentPath,FragmentationPercentage,
@{Name=”VM”;Expression={$pv.VMName}}
} -ComputerName chi-p50,chi-hvr1,chi-hvr2,chi-hvr3 |
Select * -ExcludeProperty RunspaceID |
Export-Csv -Path c:\work\VMDisks.csv –NoTypeInformation
```

There’s no way with the Hyper-V Manager to do anything like this.

COMPACTING

Once we have a way to look at storage, we can then do something such as compacting or resizing. Again, in the GUI this will probably take you about 60-90 seconds per disk, not counting time for the actual operation. Let's find all disks on a Hyper-V server that are at least 80% fragmented and optimize them. We can re-use some of the code we just created.

```
Invoke-Command {Get-VMHardDiskDrive -VMName * -PipelineVariable pv |
Get-VHD | Where {$_.FragmentationPercentage -ge 80} |
Select Path,FragmentationPercentage,@{Name=”VM”;Expression={$pv.
VMName}},
@{Name=”VMStatus”;Expression = {(Get-VM $pv.VMname).State}}
} -computername chi-p50
```



```
Path                    : e:\vmdisks\nanoservervm-demo.vhdx
FragmentationPercentage : 100
VM                      : NanoDemoBase
VMStatus                : Off
PSComputerName          : chi-p50
RunspaceId              : af2a7d54-d9d5-4309-aa05-b635cd869bb6

Path                    : e:\vmdisks\nanoservervm2.vhdx
FragmentationPercentage : 100
VM                      : NanoDemo2
VMStatus                : Running
PSComputerName          : chi-p50
RunspaceId              : af2a7d54-d9d5-4309-aa05-b635cd869bb6
```

This gives us a good idea of what needs to be addressed. The Optimize-VHD cmdlet has a number of options so you'll have to decide which is appropriate for your situation. The other important factor is the disk file must be unattached or in read-only mode. Generally, this means the virtual machine must be offline. But you can automate that as well.

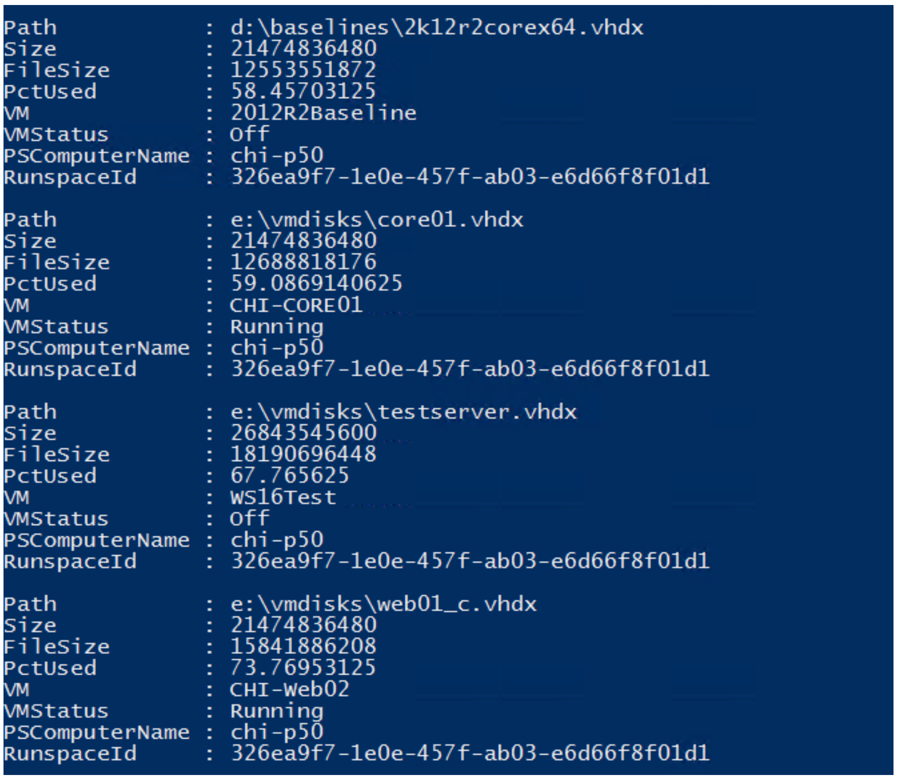
```
Invoke-Command {Get-VMHardDiskDrive -VMName * -PipelineVariable pv |
Get-VHD | Where {$_.FragmentationPercentage -ge 80} |
Select Path,FragmentationPercentage,@{Name=”VM”;Expression={$pv.
VMName}},
@{Name=”VMStatus”;Expression = {(Get-VM $pv.VMname).State}} |
Foreach {
if ($_.State -eq “running”) {
Stop-VM -Name $_.VM
#save the stopped VM name
$stopped = $_.VM
}
Optimize-VHD -Path $_.Path -Mode Full

#restart VM if it was stopped
if ($Stopped) {
Start-VM -VM $_.VM
remove-variable stopped
}
}
} -computername chi-p50
```

This is a little more complicated and makes more sense, at least to me, as a PowerShell script. I could run this script once a quarter if I wanted and it would only take me a few seconds. Or I could take advantage of PowerShell scheduled jobs and setup it up to ran like a scheduled task and never have to worry about it again!

RESIZING

Or I might need to do the opposite and expand a disk file. You've already seen that we can get a VHDs configured size and the actual file size. Personally, because I use dynamic disks most of the time, I find that if the file size is getting close to the configured size, I need to expand the VHDX file. First, let's see what disks might need this treatment.

```
Invoke-Command {Get-VMHardDiskDrive -VMName * -PipelineVariable pv |
Get-VHD |
Select Path,Size,FileSize,
@{Name="PctUsed";Expression = { ($_.filesize/$_.size)*100 }},
@{Name="VM";Expression={$pv.VMName}},
@{Name="VMStatus";Expression = {(Get-VM $pv.VMname).State}}
} -computername chi-p50 | Sort PctUsed
```



```
Path           : d:\baselines\2k12r2corex64.vhdx
Size           : 21474836480
FileSize       : 12553551872
PctUsed        : 58.45703125
VM             : 2012R2Baseline
VMStatus       : Off
PSComputerName : chi-p50
RunspaceId     : 326ea9f7-1e0e-457f-ab03-e6d66f8f01d1

Path           : e:\vmdisks\core01.vhdx
Size           : 21474836480
FileSize       : 12688818176
PctUsed        : 59.0869140625
VM             : CHI-CORE01
VMStatus       : Running
PSComputerName : chi-p50
RunspaceId     : 326ea9f7-1e0e-457f-ab03-e6d66f8f01d1

Path           : e:\vmdisks\testserver.vhdx
Size           : 26843545600
FileSize       : 18190696448
PctUsed        : 67.765625
VM             : WS16Test
VMStatus       : Off
PSComputerName : chi-p50
RunspaceId     : 326ea9f7-1e0e-457f-ab03-e6d66f8f01d1

Path           : e:\vmdisks\web01_c.vhdx
Size           : 21474836480
FileSize       : 15841886208
PctUsed        : 73.76953125
VM             : CHI-Web02
VMStatus       : Running
PSComputerName : chi-p50
RunspaceId     : 326ea9f7-1e0e-457f-ab03-e6d66f8f01d1
```

Again, I am only checking a single Hyper-V host but could easily have checked 10 of them with the same command. To manually resize disk files will take at least 30 seconds per file. I have identified 3 files that can be resized so doing this manually would take me 90-120 seconds. Or I can run a PowerShell command like this:

```
Invoke-Command {Get-VMHardDiskDrive -VMName * -PipelineVariable pv |
Get-VHD | where { (($_.filesize/$_.size)*100 -ge 80) -AND $_.MinimumSize
-gt 0 } |Select Path,Size,FileSize,
@{Name=”PctUsed”;Expression = { ($_.filesize/$_.size)*100 }},
@{Name=”VM”;Expression={$pv.VMName}},
@{Name=”VMStatus”;Expression = {(Get-VM $pv.VMname).State}}  |
Foreach {
if ($_.State -eq “running”) {
Stop-VM -Name $_.VM
#save the stopped VMName
$stopped = $_.VM
}
#calculate a new size as 20% larger
$NewSize = $s_.size + ($_.Size * .20)
Resize-VHD -Path $_.Path -SizeBytes $NewSize
```

```
                                #restart VM if it was stopped
                                       if ($Stopped) {
                                        Start-VM -VM $_.VM
                                     remove-variable stopped
                                               }
                                             }
                                 } -computername chi-p50
```

This took a matter of seconds. There are some considerations when it comes to resizing files. If the disk file is being treated as a dynamic disk as opposed to a basic disk, then this type of operation will most likely fail. If the MinimumSize property is empty, this is usually a good indication that a dynamic disk is being used so my script filters those disk files out. The new size is calculated for each disk by adding 20% of the current size.

VERDICT

If you want to do any type of reporting or storage management across multiple Hyper-V hosts or virtual machines, you have no choice but to use PowerShell. But it isn't difficult if you take your time, start slow and practice in a non-production environment. Needless to say mucking about with storage is a potentially dangerous task so make sure you have good backups in place.

TASK # 3 CREATE A NEW VIRTUAL MACHINE

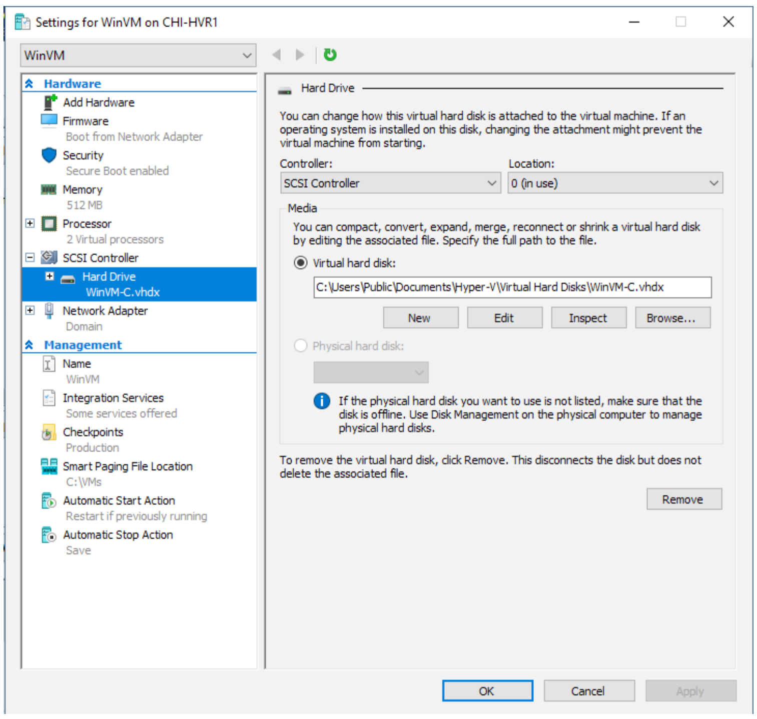
Perhaps no task is a better candidate for automation than creating a new virtual machine. Of course, you can create one in the Hyper-V Manager but it will most likely take 1 to 2 minutes of clicking and typing, not counting time to install any operating system. PowerShell provides a number of tools that you can mix and match that can spin up a new virtual machine in seconds. And when we think about managing at scale that matters.

ALL IN ONE

The primary cmdlets we will use are New-VM and Set-VM. The former will create a virtual machine and the latter will let you fine tune it. You can create a fully configured virtual machine in moments.

```
$vm = New-VM -Name WinVM -MemoryStartupBytes 512MB
-SwitchName Domain -NewVHDPath WinVM-C.vhdx -NewVHDSizeBytes
25GB -Generation 2 -ComputerName chi-hvr1
Set-VM -VM $vm -ProcessorCount 2 -DynamicMemory -MemoryMinimumBytes 512MB
-MemoryMaximumBytes 2GB -AutomaticStartAction StartIfRunning
-AutomaticStopAction Save -passthru
```

These commands took about 3.5 seconds to run and created a fully configured virtual machine, although an operating system still needs to be installed.


Settings for WinVM on CHI-HVR1

WinVM

Hardware
- Add Hardware
- Firmware
  Boot from Network Adapter
- Security
  Secure Boot enabled
- Memory
  512 MB
- Processor
  2 Virtual processors
- SCSI Controller
  - Hard Drive
    WinVM-C.vhdx
- Network Adapter
  Domain

Management
- Name
  WinVM
- Integration Services
  Some services offered
- Checkpoints
  Production
- Smart Paging File Location
  C:\VMs
- Automatic Start Action
  Restart if previously running
- Automatic Stop Action
  Save

Hard Drive

You can change how this virtual hard disk is attached to the virtual machine. If an operating system is installed on this disk, changing the attachment might prevent the virtual machine from starting.

Controller: SCSI Controller

Location: 0 (in use)

Media

You can compact, convert, expand, merge, reconnect or shrink a virtual hard disk by editing the associated file. Specify the full path to the file.

(●) Virtual hard disk:

C:\Users\Public\Documents\Hyper-V\Virtual Hard Disks\WinVM-C.vhdx

New | Edit | Inspect | Browse...

( ) Physical hard disk:

If the physical hard disk you want to use is not listed, make sure that the disk is offline. Use Disk Management on the physical computer to manage physical hard disks.

To remove the virtual hard disk, click Remove. This disconnects the disk but does not delete the associated file.

Remove

OK | Cancel | Apply

DISK FIRST

Another approach is to create the disk separately. This too takes only a few seconds.

```
$Path = Join-Path -Path (Get-VMHost -ComputerName CHI-HVR1).
VirtualHardDiskPath
-ChildPath WinVM2.vhdx
$vhdx = New-VHD -Path $Path -ParentPath D:\2K12R2Corex64.vhdx
-Differencing
-SizeBytes 40GB -ComputerName CHI-HVR1
```

I just created a new 40GB differencing disk. Then I can create a new virtual machine using this disk.

```
$vm = New-VM -Name WinVM2 -MemoryStartupBytes 512MB -SwitchName Domain
-VHDPath $Path -Generation 1 -ComputerName chi-hvr1
Set-VM -VM $vm -ProcessorCount 2 -DynamicMemory -MemoryMinimumBytes
512MB
-MemoryMaximumBytes 2GB -AutomaticStartAction StartIfRunning
-AutomaticStopAction Save –passthru
Start-vm $vm
```

The parent disk is from an older version so I have to create a generation 1 virtual machine. But within literally seconds I have a virtual machine up and running. I can use PowerShell remoting to connect to the guest operating system and reconfigure as necessary.

VERDICT

There's no question that using PowerShell automation to create new virtual machines is the smart and only choice. I can just as easily create 10 new virtual machines in not much more time than it takes to bring up 1. Most likely this is something you will create some PowerShell tooling around so that other admins can run the commands. Or you might integrate the commands into some automated monitoring system that starts up new virtual machines when circumstances warrant.

TASK # 4 WORKING WITH VIRTUAL SWITCHES

While there are cmdlets for creating new virtual switches, this isn't a task I think you do often and you aren't likely to create more than a few at a time. In these cases, using the graphical manager may be just as easy. However, managing virtual switches, especially when configuring virtual machines is definitely a candidate for automation.

Let's say you have created a new virtual switch, or have an existing one, and you want to configure a virtual machine to start using it. In the Hyper-V Server Manager, changing a VM's switch can take 20-30 seconds, depending on the number of virtual machines you have to wade through. If you had to make the change on 10 virtual machines that would take you at least 5 minutes of pretty tedious work. And sure, you could assign the task to an intern, hoping he wouldn't make a mistake along the way.

THE POWERSHELL WAY

Or you can assign a new virtual switch with PowerShell in less than half a second.

```
Get-VMNetworkAdapter –vmname VMDemo01 –computername chi-hvr1 | Connect-
                VMNetworkAdapter -SwitchName Internal
```

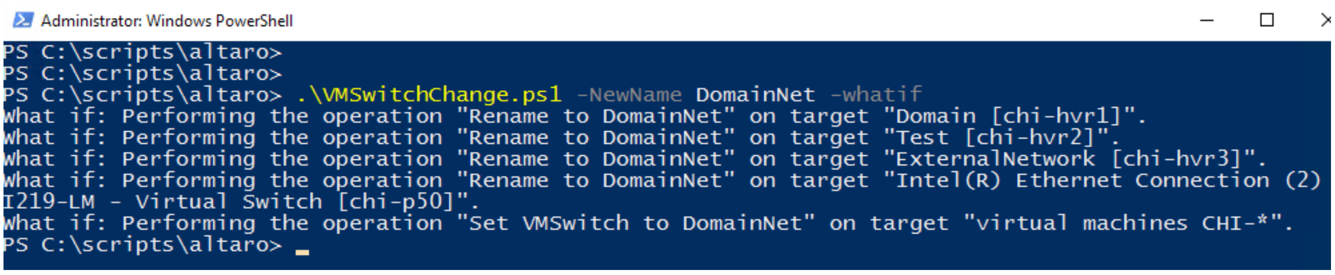
Yes, this might take a few seconds to type, but you could easily turn it into a PowerShell function or script. In fact, I needed to automate some changes in my Hyper-V environment. As I added Hyper-V servers, I wasn't very careful about naming my virtual switches. This meant that I needed to take extra steps when it came time to move or live migrate virtual machines between hosts.

My solution was to rename all of the switches bound to the host's external adapter so that they all shared the same name. Then I needed to make sure that all my virtual machines that started with CHI (I was at least consistent there) were connected to this switch. If the virtual machine was already connected to the switch under the old name, it would automatically use the new name. But just in case I have some odd machines that are misconfigured, this should catch those as well.

I could have typed the commands out interactively, but I wrote a short script primarily to include support for –WhatIf.

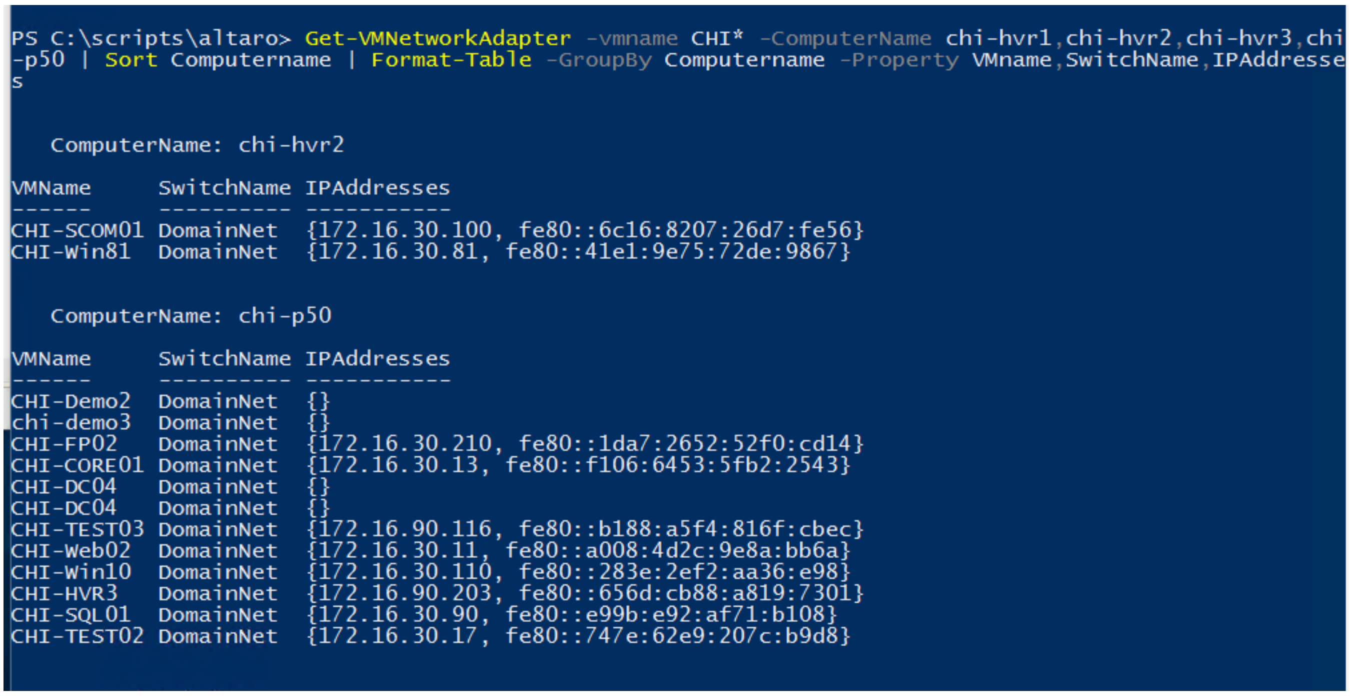
```
#requires -version 4.0
#requires -module Hyper-V

[cmdletbinding(SupportsShouldProcess)]
Param(
[Parameter(Mandatory,HelpMessage="Enter the new name for the switch")]
[ValidateNotNullorEmpty()]
[string]$NewName,

[ValidateNotNullorEmpty()]
[string[]]$VMHosts = @("chi-hvr1","chi-hvr2","chi-hvr3","chi-p50")

)

#rename switch
Get-VMSwitch -SwitchType External -ComputerName $VMHosts |
Where {$_.NetAdapterInterfaceDescription -notmatch "loopback"} |
foreach {
#whatIf code
If ($PSCmdlet.ShouldProcess("$($_.name) [$($_.
computername)]","Rename to $NewName")) {
Rename-VMSwitch -Name $_.Name -NewName $NewName -ComputerName
$_.Computername
}
}
#change virtual machines
#I added my own code for -WhatIf because Connect-VMNetworkAdapter
complains
#if there is no existing switch
If ($PSCmdlet.ShouldProcess("virtual machines CHI-*","Set VMSwitch to
$NewName")) {
Get-VMNetworkAdapter -VMName CHI-* -ComputerName $VMHosts |
foreach { Connect-VMNetworkAdapter -SwitchName $NewName -VMName
$_.VMName }
}
```

This lets me safely test the process.


Administrator: Windows PowerShell

```
PS C:\scripts\altaro>
PS C:\scripts\altaro>
PS C:\scripts\altaro> .\VMSwitchChange.ps1 -NewName DomainNet -whatif
What if: Performing the operation "Rename to DomainNet" on target "Domain [chi-hvr1]".
What if: Performing the operation "Rename to DomainNet" on target "Test [chi-hvr2]".
What if: Performing the operation "Rename to DomainNet" on target "ExternalNetwork [chi-hvr3]".
What if: Performing the operation "Rename to DomainNet" on target "Intel(R) Ethernet Connection (2)
I219-LM - Virtual Switch [chi-p50]".
What if: Performing the operation "Set VMSwitch to DomainNet" on target "virtual machines CHI-*".
PS C:\scripts\altaro> _
```

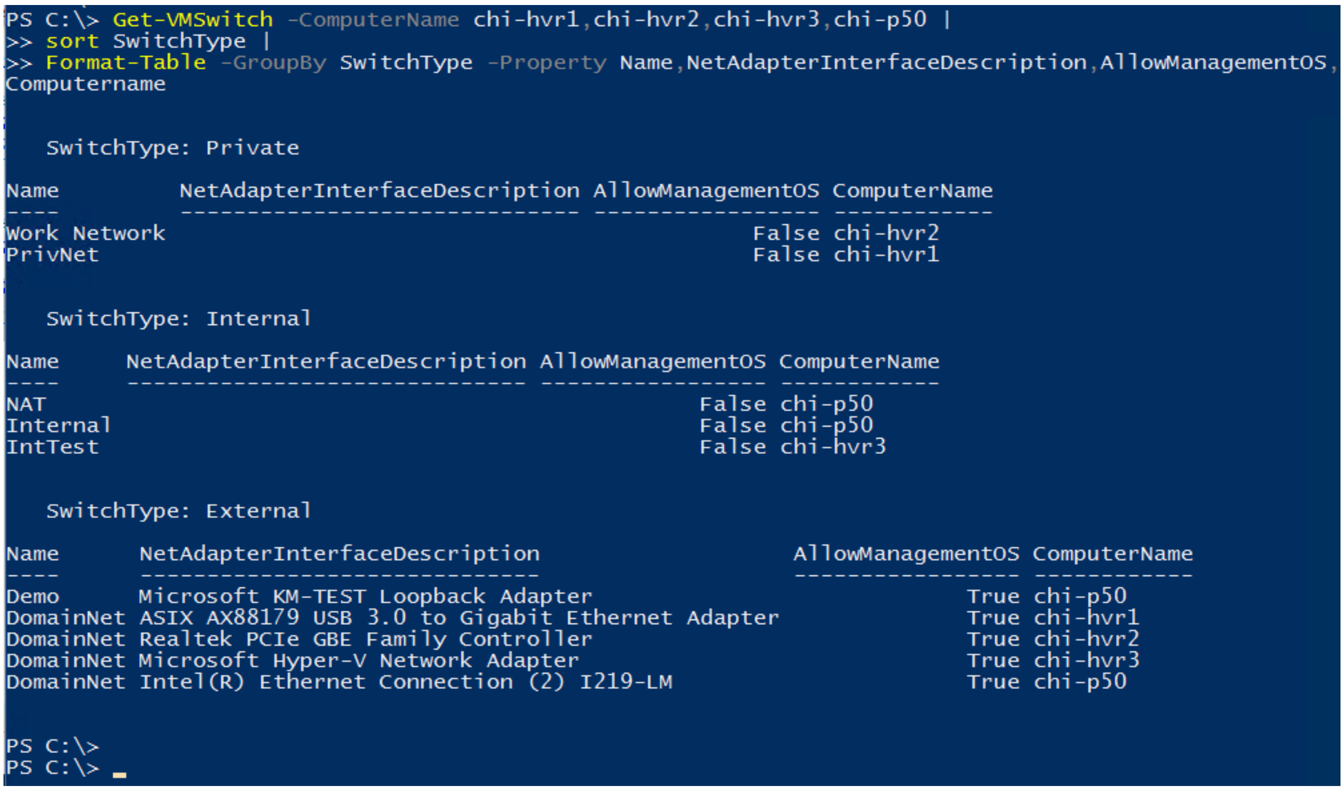
Now I can run it for real without –WhatIf and in 6 seconds I reconfigure switches on 4 servers and 14 virtual machines.



```
PS C:\scripts\altaro> Get-VMNetworkAdapter -vmname CHI* -ComputerName chi-hvr1,chi-hvr2,chi-hvr3,chi
-p50 | Sort Computername | Format-Table -GroupBy Computername -Property VMname,SwitchName,IPAddresse
s


   ComputerName: chi-hvr2

VMName      SwitchName IPAddresses
------      ---------- -----------
CHI-SCOM01 DomainNet  {172.16.30.100, fe80::6c16:8207:26d7:fe56}
CHI-Win81  DomainNet  {172.16.30.81, fe80::41e1:9e75:72de:9867}


   ComputerName: chi-p50

VMName      SwitchName IPAddresses
------      ---------- -----------
CHI-Demo2  DomainNet  {}
chi-demo3  DomainNet  {}
CHI-FP02   DomainNet  {172.16.30.210, fe80::1da7:2652:52f0:cd14}
CHI-CORE01 DomainNet  {172.16.30.13, fe80::f106:6453:5fb2:2543}
CHI-DC04   DomainNet  {}
CHI-DC04   DomainNet  {}
CHI-TEST03 DomainNet  {172.16.90.116, fe80::b188:a5f4:816f:cbec}
CHI-Web02  DomainNet  {172.16.30.11, fe80::a008:4d2c:9e8a:bb6a}
CHI-Win10  DomainNet  {172.16.30.110, fe80::283e:2ef2:aa36:e98}
CHI-HVR3   DomainNet  {172.16.90.203, fe80::656d:cb88:a819:7301}
CHI-SQL01  DomainNet  {172.16.30.90, fe80::e99b:e92:af71:b108}
CHI-TEST02 DomainNet  {172.16.30.17, fe80::747e:62e9:207c:b9d8}
```

I can also use PowerShell to easily report on all of my virtual switches. Something that is impossible in the management console.

```
Get-VMSwitch -ComputerName chi-hvr1,chi-hvr3,chi-p50 |
sort SwitchType |
Format-Table -GroupBy SwitchType -Property
Name,NetAdapterInterfaceDescription,AllowManagementOS,Computername
```

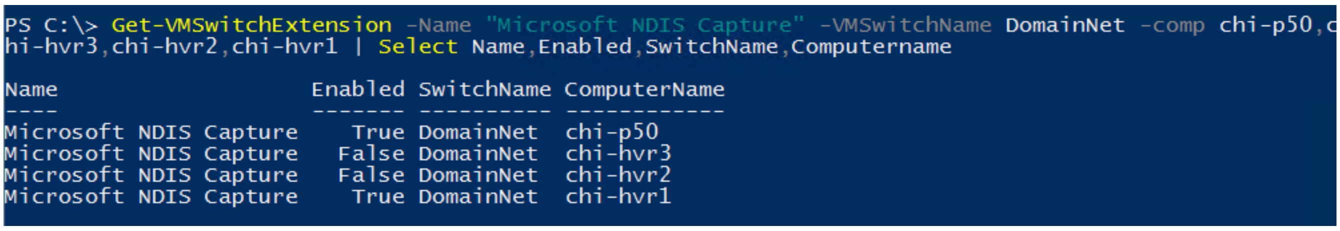


```
PS C:\> Get-VMSwitch -ComputerName chi-hvr1,chi-hvr2,chi-hvr3,chi-p50 |
>> sort SwitchType |
>> Format-Table -GroupBy SwitchType -Property Name,NetAdapterInterfaceDescription,AllowManagementOS,
Computername


   SwitchType: Private

Name         NetAdapterInterfaceDescription AllowManagementOS ComputerName
----         ------------------------------ ----------------- ------------
Work Network                                            False chi-hvr2
PrivNet                                                 False chi-hvr1


   SwitchType: Internal

Name     NetAdapterInterfaceDescription AllowManagementOS ComputerName
----     ------------------------------ ----------------- ------------
NAT                                                 False chi-p50
Internal                                            False chi-p50
IntTest                                             False chi-hvr3


   SwitchType: External

Name      NetAdapterInterfaceDescription                    AllowManagementOS ComputerName
----      ------------------------------                    ----------------- ------------
Demo      Microsoft KM-TEST Loopback Adapter                             True chi-p50
DomainNet ASIX AX88179 USB 3.0 to Gigabit Ethernet Adapter               True chi-hvr1
DomainNet Realtek PCIe GBE Family Controller                             True chi-hvr2
DomainNet Microsoft Hyper-V Network Adapter                              True chi-hvr3
DomainNet Intel(R) Ethernet Connection (2) I219-LM                       True chi-p50


PS C:\>
PS C:\> _
```

And of course I can just as easily modify the switches, something that takes about 20 seconds per switch to accomplish.

Let's work with switch extensions.

```
Get-VMSwitchExtension -Name "Microsoft NDIS Capture" -VMSwitchName
DomainNet -comp chi-p50,chi-hvr3,chi-hvr2,chi-hvr1 |
Select Name,Enabled,SwitchName,Computername
```



```
PS C:\> Get-VMSwitchExtension -Name "Microsoft NDIS Capture" -VMSwitchName DomainNet -comp chi-p50,c
hi-hvr3,chi-hvr2,chi-hvr1 | Select Name,Enabled,SwitchName,Computername

Name                   Enabled SwitchName ComputerName
----                   ------- ---------- ------------
Microsoft NDIS Capture    True DomainNet  chi-p50
Microsoft NDIS Capture   False DomainNet  chi-hvr3
Microsoft NDIS Capture   False DomainNet  chi-hvr2
Microsoft NDIS Capture    True DomainNet  chi-hvr1
```

I want to enable this extension.

```
Get-VMSwitchExtension -Name “Microsoft NDIS Capture” -VMSwitchName
DomainNet -comp chi-p50,chi-hvr3,chi-hvr2,chi-hvr1 |
Where {-Not $_.Enabled} |
foreach {
Enable-VMSwitchExtension -VMSwitchName $_.SwitchName -ComputerName
$_.ComputerName -Name $_.Name
}
```

This is technically a one line command that took 2.5 seconds to complete. It finds the NDIS Capture extension on Hyper-V hosts where it is not enable, and enables it.

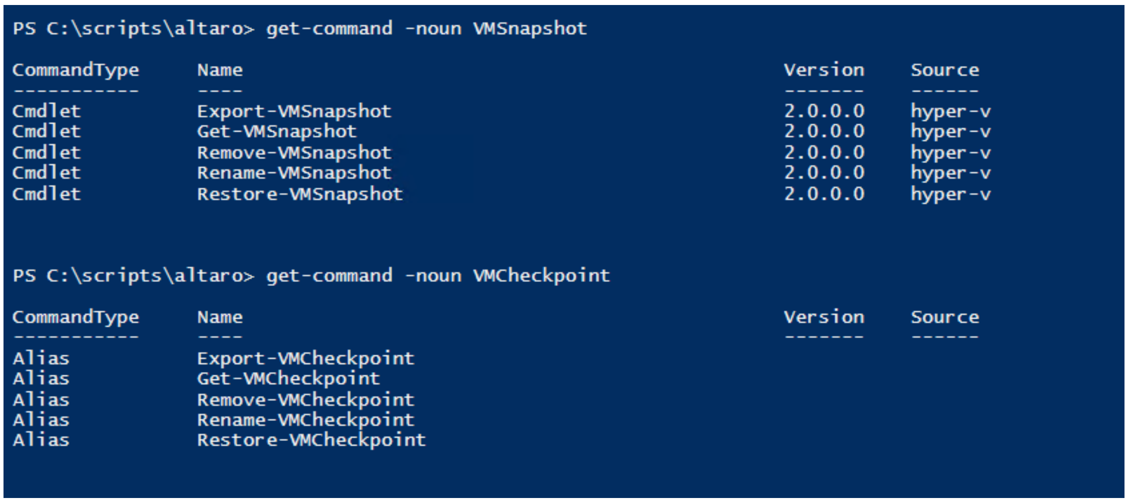
VERDICT

I will be the first to admit that some of the commands in the Hyper-V module don’t behave that way I would expect based on my previous experience with PowerShell. I wish some commands worked better in a pipelined expression without having to resort to ForEach-Object as I did above. That is why it is important that you take the time to read cmdlet and examples.

However, even with this learning curves, managing virtual switches and virtual machines is an order of magnitude faster than using the Hyper-V Manager.

TASK # 5 MANAGING VIRTUAL MACHINE CHECKPOINTS

The last management task we'll look at is working with virtual machine snapshots, which Microsoft now refers to as checkpoints. In fact, in starting with version 2.0 of the Hyper-V module, all of the cmdlets that use VMSnapshot as the noun have aliases using VMCheckpoint.



```
PS C:\scripts\altaro> get-command -noun VMSnapshot

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Cmdlet          Export-VMSnapshot                                  2.0.0.0    hyper-v
Cmdlet          Get-VMSnapshot                                     2.0.0.0    hyper-v
Cmdlet          Remove-VMSnapshot                                  2.0.0.0    hyper-v
Cmdlet          Rename-VMSnapshot                                  2.0.0.0    hyper-v
Cmdlet          Restore-VMSnapshot                                 2.0.0.0    hyper-v


PS C:\scripts\altaro> get-command -noun VMCheckpoint

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           Export-VMCheckpoint
Alias           Get-VMCheckpoint
Alias           Remove-VMCheckpoint
Alias           Rename-VMCheckpoint
Alias           Restore-VMCheckpoint
```

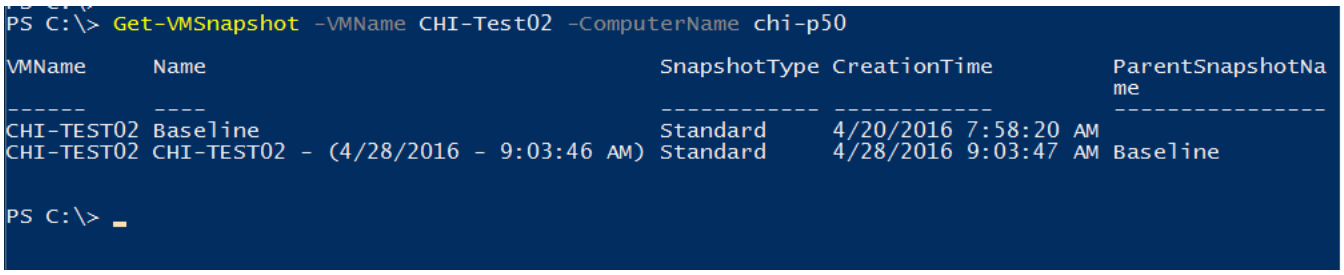
But for consistency I'll use the VMSnapshot commands.

If you needed to manually create a checkpoint it isn't that difficult. In fact, you can even select multiple virtual machines and create a checkpoint for all of them. Although removing checkpoints for multiple virtual machines is bit more complicated and takes about 30-60 seconds per virtual machine, not counting disk operations.

THE POWERSHELL WAY

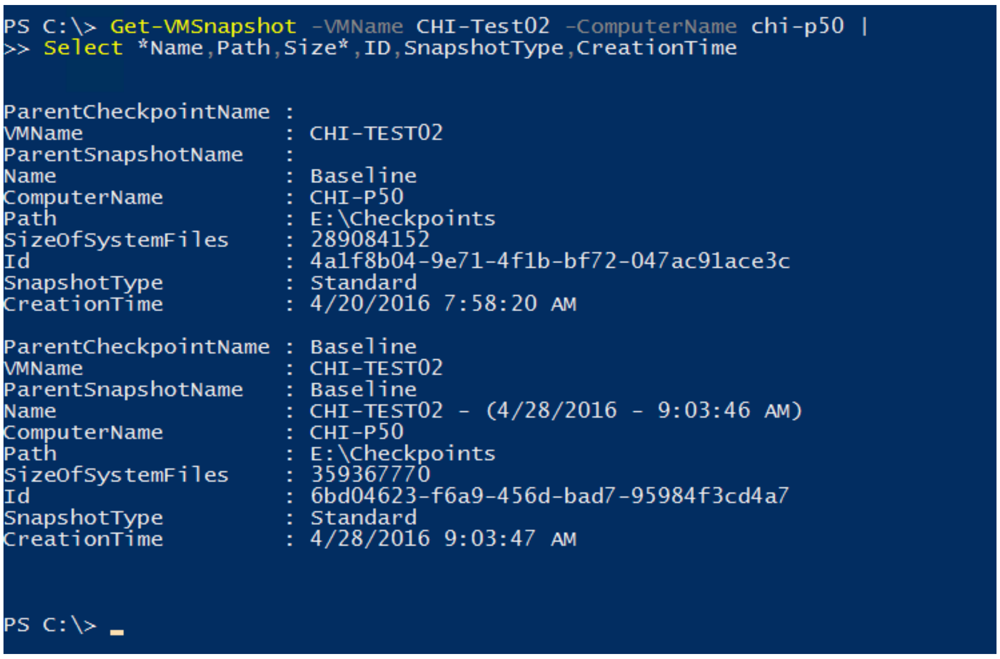
First, let's look at reporting on the current state of checkpoints, primarily because PowerShell brings some added value that would take you a long time to duplicate manually. Let's look at snapshots for a single virtual machine.

```
Get-VMSnapshot -VMName CHI-Test02 -ComputerName chi-p50
```


```
PS C:\> Get-VMSnapshot -VMName CHI-Test02 -ComputerName chi-p50

VMName     Name                                     SnapshotType CreationTime         ParentSnapshotNa
                                                                                      me
------     ----                                     ------------ ------------         ----------------
CHI-TEST02 Baseline                                 Standard     4/20/2016 7:58:20 AM
CHI-TEST02 CHI-TEST02 - (4/28/2016 - 9:03:46 AM)    Standard     4/28/2016 9:03:47 AM Baseline

PS C:\> _
```

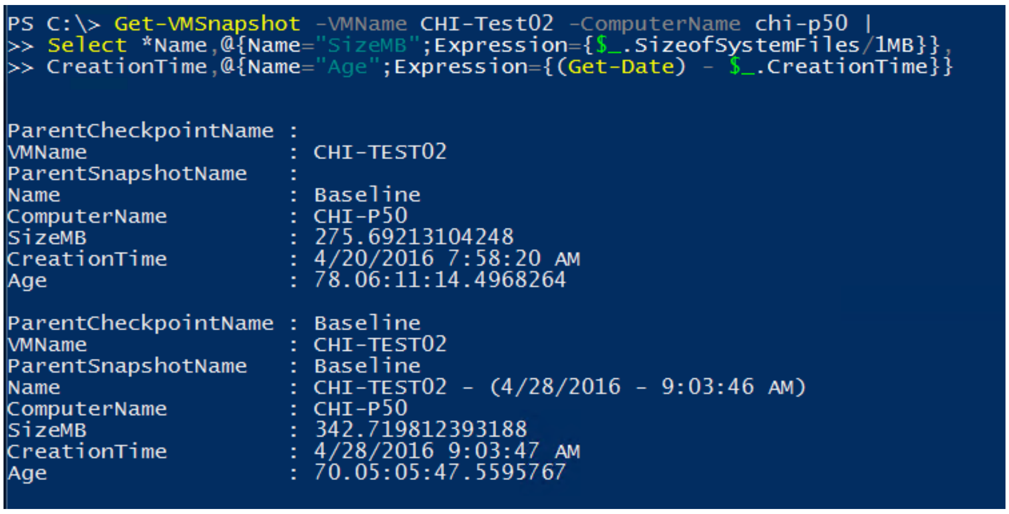
If you pipe this to Get-Member you'll discover there are many more useful properties.

```
Get-VMSnapshot -VMName CHI-Test02 -ComputerName chi-p50 |
Select *Name,Path,Size*,ID,SnapshotType,CreationTime
```


```
PS C:\> Get-VMSnapshot -VMName CHI-Test02 -ComputerName chi-p50 |
>> Select *Name,Path,Size*,ID,SnapshotType,CreationTime


ParentCheckpointName :
VMName               : CHI-TEST02
ParentSnapshotName   :
Name                 : Baseline
ComputerName         : CHI-P50
Path                 : E:\Checkpoints
SizeOfSystemFiles    : 289084152
Id                   : 4a1f8b04-9e71-4f1b-bf72-047ac91ace3c
SnapshotType         : Standard
CreationTime         : 4/20/2016 7:58:20 AM

ParentCheckpointName : Baseline
VMName               : CHI-TEST02
ParentSnapshotName   : Baseline
Name                 : CHI-TEST02 - (4/28/2016 - 9:03:46 AM)
ComputerName         : CHI-P50
Path                 : E:\Checkpoints
SizeOfSystemFiles    : 359367770
Id                   : 6bd04623-f6a9-456d-bad7-95984f3cd4a7
SnapshotType         : Standard
CreationTime         : 4/28/2016 9:03:47 AM


PS C:\> _
```

One thing I especially like is that SizeOfSystemFiles property. This indicates how much disk space the snapshot is consuming. Although I should note that in earlier versions of the Hyper-V module, this value does not appear to be accurate. With a little tweaking and some custom properties, we can even get a more meaningful display in practically no time.

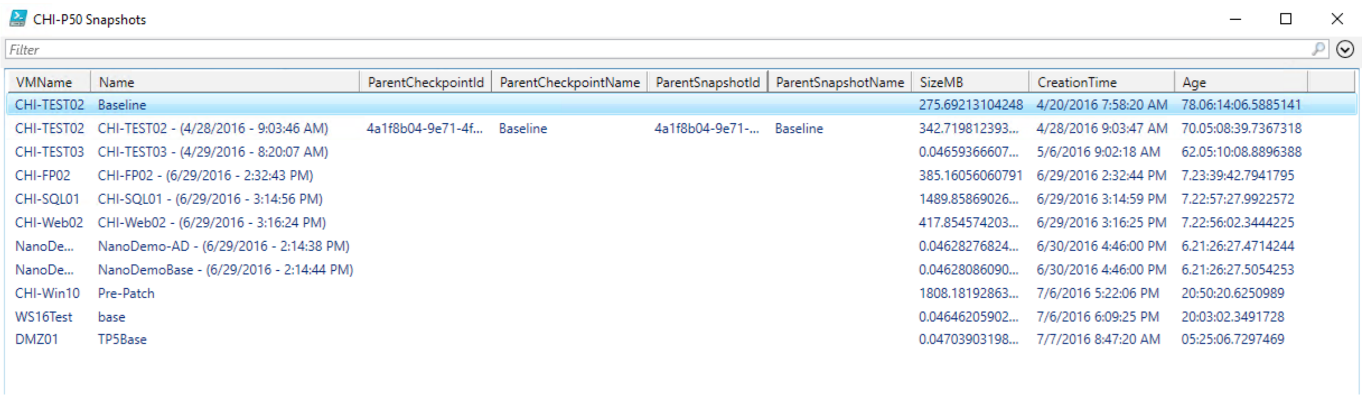
```
Get-VMSnapshot -VMName CHI-Test02 -ComputerName chi-p50 |
Select *Name,@{Name="SizeMB";Expression={$_.SizeofSystemFiles/1MB}},
CreationTime,@{Name="Age";Expression={(Get-Date) - $_.CreationTime}}
```


```
PS C:\> Get-VMSnapshot -VMName CHI-Test02 -ComputerName chi-p50 |
>> Select *Name,@{Name="SizeMB";Expression={$_.SizeofSystemFiles/1MB}},
>> CreationTime,@{Name="Age";Expression={(Get-Date) - $_.CreationTime}}

ParentCheckpointName :
VMName               : CHI-TEST02
ParentSnapshotName   :
Name                 : Baseline
ComputerName         : CHI-P50
SizeMB               : 275.69213104248
CreationTime         : 4/20/2016 7:58:20 AM
Age                  : 78.06:11:14.4968264

ParentCheckpointName : Baseline
VMName               : CHI-TEST02
ParentSnapshotName   : Baseline
Name                 : CHI-TEST02 - (4/28/2016 - 9:03:46 AM)
ComputerName         : CHI-P50
SizeMB               : 342.719812393188
CreationTime         : 4/28/2016 9:03:47 AM
Age                  : 70.05:05:47.5595767
```

If we can do it for one virtual machine, we can do it for all of them.

```
Get-VMSnapshot -VMName * -ComputerName chi-p50 |
Select VMName,Name,Parent*,@{Name=”SizeMB”;Expression={$_.
SizeofSystemFiles/1MB}},
CreationTime,@{Name=”Age”;Expression={(Get-Date) - $_.CreationTime}} |
Out-Gridview -Title “CHI-P50 Snapshots”
```

In this version I revised the Select-Object part of the command to better control the property display order.


CHI-P50 Snapshots

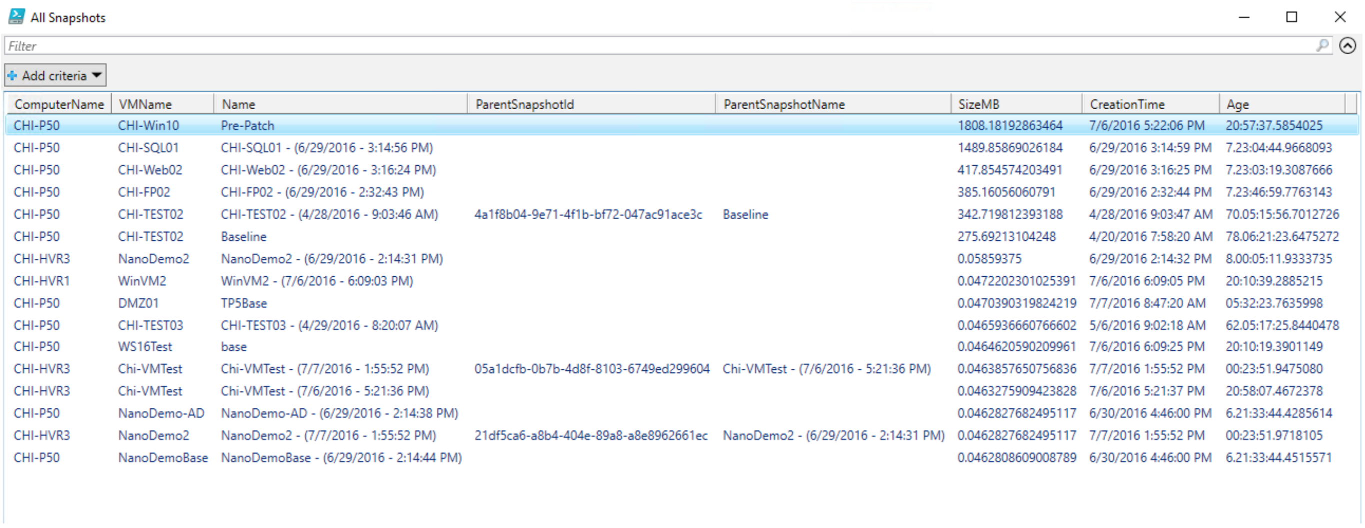
Filter

VMName	Name	ParentCheckpointId	ParentCheckpointName	ParentSnapshotId	ParentSnapshotName	SizeMB	CreationTime	Age
CHI-TEST02	Baseline					275.69213104248	4/20/2016 7:58:20 AM	78.06:14:06.5885141
CHI-TEST02	CHI-TEST02 - (4/28/2016 - 9:03:46 AM)	4a1f8b04-9e71-4f...	Baseline	4a1f8b04-9e71-...	Baseline	342.719812393...	4/28/2016 9:03:47 AM	70.05:08:39.7367318
CHI-TEST03	CHI-TEST03 - (4/29/2016 - 8:20:07 AM)					0.04659366607...	5/6/2016 9:02:18 AM	62.05:10:08.8896388
CHI-FP02	CHI-FP02 - (6/29/2016 - 2:32:43 PM)					385.16056060791	6/29/2016 2:32:44 PM	7.23:39:42.7941795
CHI-SQL01	CHI-SQL01 - (6/29/2016 - 3:14:56 PM)					1489.85869026...	6/29/2016 3:14:59 PM	7.22:57:27.9922572
CHI-Web02	CHI-Web02 - (6/29/2016 - 3:16:24 PM)					417.854574203...	6/29/2016 3:16:25 PM	7.22:56:02.3444225
NanoDe...	NanoDemo-AD - (6/29/2016 - 2:14:38 PM)					0.04628276824...	6/30/2016 4:46:00 PM	6.21:26:27.4714244
NanoDe...	NanoDemoBase - (6/29/2016 - 2:14:44 PM)					0.04628086090...	6/30/2016 4:46:00 PM	6.21:26:27.5054253
CHI-Win10	Pre-Patch					1808.18192863...	7/6/2016 5:22:06 PM	20:50:20.6250989
WS16Test	base					0.04646205902...	7/6/2016 6:09:25 PM	20:03:02.3491728
DMZ01	TP5Base					0.04703903198...	7/7/2016 8:47:20 AM	05:25:06.7297469

Finally, let's scale this out and check for snapshots on multiple Hyper-V hosts.

```
$vmhosts = "chi-hvr1","chi-hvr3","chi-p50"
Get-VMSnapshot -VMName * -ComputerName $VMHosts |
Select Computername,VMName,Name,ParentSnapshotID,ParentSnapshotName,
@{Name="SizeMB";Expression={$_.SizeofSystemFiles/1MB}},
CreationTime,@{Name="Age";Expression={(Get-Date) - $_.CreationTime}} |
Sort SizeMB -Descending |
Out-Gridview -Title "All Snapshots"
```

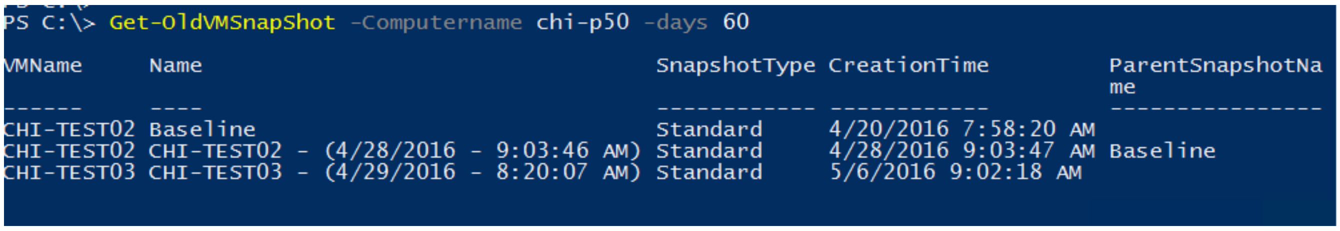
The changes to the command were minor.



ComputerName	VMName	Name	ParentSnapshotId	ParentSnapshotName	SizeMB	CreationTime	Age
CHI-P50	CHI-Win10	Pre-Patch			1808.18192863464	7/6/2016 5:22:06 PM	20:57:37.5854025
CHI-P50	CHI-SQL01	CHI-SQL01 - (6/29/2016 - 3:14:56 PM)			1489.85869026184	6/29/2016 3:14:59 PM	7.23:04:44.9668093
CHI-P50	CHI-Web02	CHI-Web02 - (6/29/2016 - 3:16:24 PM)			417.854574203491	6/29/2016 3:16:25 PM	7.23:03:19.3087666
CHI-P50	CHI-FP02	CHI-FP02 - (6/29/2016 - 2:32:43 PM)			385.16056060791	6/29/2016 2:32:44 PM	7.23:46:59.7763143
CHI-P50	CHI-TEST02	CHI-TEST02 - (4/28/2016 - 9:03:46 AM)	4a1f8b04-9e71-4f1b-bf72-047ac91ace3c	Baseline	342.719812393188	4/28/2016 9:03:47 AM	70.05:15:56.7012726
CHI-P50	CHI-TEST02	Baseline			275.69213104248	4/20/2016 7:58:20 AM	78.06:21:23.6475272
CHI-HVR3	NanoDemo2	NanoDemo2 - (6/29/2016 - 2:14:31 PM)			0.05859375	6/29/2016 2:14:32 PM	8.00:05:11.9333735
CHI-HVR1	WinVM2	WinVM2 - (7/6/2016 - 6:09:03 PM)			0.0472202301025391	7/6/2016 6:09:05 PM	20:10:39.2885215
CHI-P50	DMZ01	TP5Base			0.0470390319824219	7/7/2016 8:47:20 AM	05:32:23.7635998
CHI-P50	CHI-TEST03	CHI-TEST03 - (4/29/2016 - 8:20:07 AM)			0.0465936660766602	5/6/2016 9:02:18 AM	62.05:17:25.8440478
CHI-P50	WS16Test	base			0.0464620590209961	7/6/2016 6:09:25 PM	20:10:19.3901149
CHI-HVR3	Chi-VMTest	Chi-VMTest - (7/7/2016 - 1:55:52 PM)	05a1dcfb-0b7b-4d8f-8103-6749ed299604	Chi-VMTest - (7/6/2016 - 5:21:36 PM)	0.0463857650756836	7/7/2016 1:55:52 PM	00:23:51.9475080
CHI-HVR3	Chi-VMTest	Chi-VMTest - (7/6/2016 - 5:21:36 PM)			0.0463275909423828	7/6/2016 5:21:37 PM	20:58:07.4672378
CHI-P50	NanoDemo-AD	NanoDemo-AD - (6/29/2016 - 2:14:38 PM)			0.0462827682495117	6/30/2016 4:46:00 PM	6.21:33:44.4285614
CHI-HVR3	NanoDemo2	NanoDemo2 - (7/7/2016 - 1:55:52 PM)	21df5ca6-a8b4-404e-89a8-a8e8962661ec	NanoDemo2 - (6/29/2016 - 2:14:31 PM)	0.0462827682495117	7/7/2016 1:55:52 PM	00:23:51.9718105
CHI-P50	NanoDemoBase	NanoDemoBase - (6/29/2016 - 2:14:44 PM)			0.0462808609008789	6/30/2016 4:46:00 PM	6.21:33:44.4515571

This is all but impossible to do in the server manager GUI. In fact, you could create some nice tooling to find old snapshots.

```
Function Get-OldVMSnapShot {
[cmdletbinding()]
Param(
[string]$VMName = "*",
[Parameter(Mandatory)]
[string]$Computername,
[int]$Days = 30
)
Get-VMSnapshot -VMName $VMName -ComputerName $Computername |
Where {((Get-Date) - $_.CreationTime).TotalDays -ge $Days}
}
```



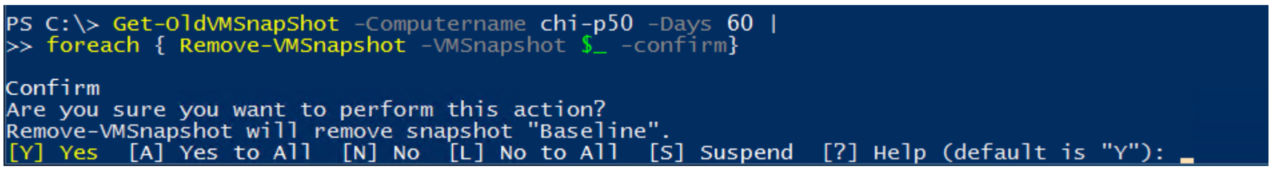
```
PS C:\> Get-OldVMSnapShot -Computername chi-p50 -days 60

VMName     Name                                          SnapshotType CreationTime          ParentSnapshotNa
                                                                                            me
------     ----                                          ------------ ------------          ----------------
CHI-TEST02 Baseline                                      Standard     4/20/2016 7:58:20 AM
CHI-TEST02 CHI-TEST02 - (4/28/2016 - 9:03:46 AM)         Standard     4/28/2016 9:03:47 AM  Baseline
CHI-TEST03 CHI-TEST03 - (4/29/2016 - 8:20:07 AM)         Standard     5/6/2016 9:02:18 AM
```

Within milliseconds I discovered all snapshots on the given Hyper-V server that were created over 60 days ago.

Why is this useful? How about cleaning them up.

```
Get-OldVMSnapShot -Computername chi-p50 -Days 60 |
foreach { Remove-VMSnapshot -VMSnapshot $_ -confirm}
```

As I mentioned earlier, some of these cmdlets don't behave the way they probably should so I've had to resort to using ForEach-Object. But it works.



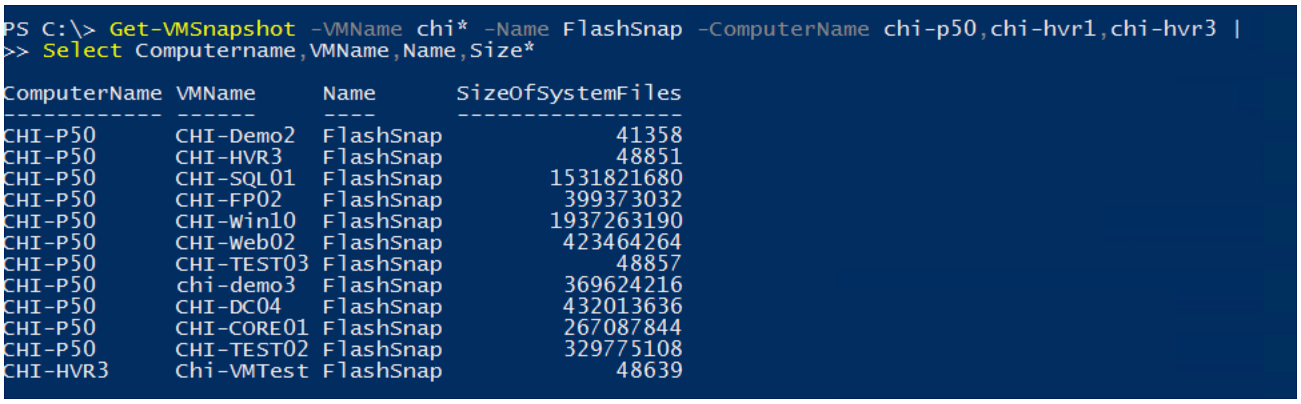
```
PS C:\> Get-OldVMSnapShot -Computername chi-p50 -Days 60 |
>> foreach { Remove-VMSnapshot -VMSnapshot $_ -confirm}

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

Finally, let's look at the opposite situation.

Suppose you are about to undertake a process and want to have a snapshot handy as a fallback. With a single command I can create a checkpoint on all virtual machines that start with CHI.

```
Checkpoint-VM -Name CHI* -ComputerName chi-p50,chi-hvr1,chi-hvr3
-SnapshotName "FlashSnap" -AsJob
```

All of the checkpoints will have the same name. I am also taking advantage of PowerShell's background job feature. The checkpoints will be created in the background and I can get my PowerShell prompt back immediately to do other things. I can check later to see that the snapshots are complete.


```
PS C:\> Get-VMSnapshot -VMName chi* -Name FlashSnap -ComputerName chi-p50,chi-hvr1,chi-hvr3 |
>> Select Computername,VMName,Name,Size*

ComputerName VMName      Name      SizeOfSystemFiles
------------ ------      ----      -----------------
CHI-P50      CHI-Demo2   FlashSnap             41358
CHI-P50      CHI-HVR3    FlashSnap             48851
CHI-P50      CHI-SQL01   FlashSnap        1531821680
CHI-P50      CHI-FP02    FlashSnap         399373032
CHI-P50      CHI-Win10   FlashSnap        1937263190
CHI-P50      CHI-Web02   FlashSnap         423464264
CHI-P50      CHI-TEST03  FlashSnap             48857
CHI-P50      chi-demo3   FlashSnap         369624216
CHI-P50      CHI-DC04    FlashSnap         432013636
CHI-P50      CHI-CORE01  FlashSnap         267087844
CHI-P50      CHI-TEST02  FlashSnap         329775108
CHI-HVR3     Chi-VMTest  FlashSnap             48639
```

That's 12 snapshots created probably in seconds where as I would have spent at least 15-20 seconds per virtual machine creating the snapshot and renaming it in the Hyper-V Manager. That's only 4-5 minutes. But my Hyper-V environment is really quite small.

Suppose you had 100 virtual machines on multiple hosts. More than likely it would take close to a half hour just to generate the snapshot.

I don't know about you, but that's not a task I would enjoy.

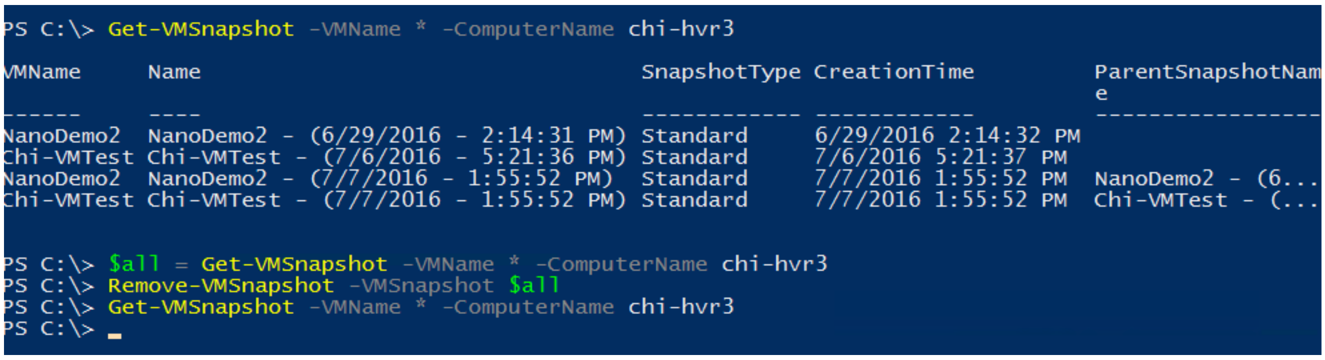
And as you've already seen, cleanup is a snap!

```
Remove-VMSnapshot -VMName * -Name FlashSnap
-ComputerName chi-p50,chi-hvr1,chi-hvr3
```

This too you could run with the –AsJob parameter.

Finally, let's get rid of all snapshots on a Hyper-V host.

```
$all = Get-VMSnapshot -VMName * -ComputerName chi-hvr3
Remove-VMSnapshot -VMSnapshot $all
```


```
PS C:\> Get-VMSnapshot -VMName * -ComputerName chi-hvr3

VMName     Name                                 SnapshotType CreationTime         ParentSnapshotNam
                                                                                  e
------     ----                                 ------------ ------------         -----------------
NanoDemo2  NanoDemo2 - (6/29/2016 - 2:14:31 PM) Standard     6/29/2016 2:14:32 PM
Chi-VMTest Chi-VMTest - (7/6/2016 - 5:21:36 PM) Standard     7/6/2016 5:21:37 PM
NanoDemo2  NanoDemo2 - (7/7/2016 - 1:55:52 PM)  Standard     7/7/2016 1:55:52 PM  NanoDemo2 - (6...
Chi-VMTest Chi-VMTest - (7/7/2016 - 1:55:52 PM) Standard     7/7/2016 1:55:52 PM  Chi-VMTest - (...

PS C:\> $all = Get-VMSnapshot -VMName * -ComputerName chi-hvr3
PS C:\> Remove-VMSnapshot -VMSnapshot $all
PS C:\> Get-VMSnapshot -VMName * -ComputerName chi-hvr3
PS C:\> _
```

I only had to remove 4 small snapshots but it could just as easily been 40 or 400. The commands are the same.

VERDICT

If your organization relies on VM snapshots, and you have more than a few virtual machines, I don't see you can you effectively manage them without some form of automation. As with many of the other tasks we've looked at, what can take 20-30 seconds per virtual machine can be accomplished in milliseconds from a PowerShell prompt.

SUMMARY

If you've made your way through the entire eBook then I'm assuming you recognize the value of automation when it comes to managing Hyper-V. I also hope that you realized PowerShell can play a critical role in your daily and weekly tasks and save you a great deal of time. It may be in small increments, but these add up over the course of a year. Not to mention the benefits of consistency, documentation and sheer fun.

ADDITIONAL RESOURCES

If you are looking for additional resources on Hyper-V as well as managing Hyper-V with PowerShell, you should of course keep an eye on the Altaro blog at http://www.altaro.com/hyper-v. I also maintain a pretty active blog at http://blog.jdhitsolutions.com

If you are just getting started with PowerShell or feel you could use a refresher, you might consider the following books which I've co-authored with veteran PowerShell expert Don Jones:

- *Learn Windows PowerShell in a Month of Lunches*
- *Learn PowerShell Toolmaking in a Month of Lunches*

If you are a visual learner, you'll find a number of video training courses at Pluralsight.com. Or catch me at a conference like TechMentor or IT/Dev Connections where I am most likely presenting something PowerShell related.

Finally, I would also encourage you to use the forums at PowerShell.org which is part of the non-profit DevOps Collective (https://devopscollective.org). The forums are very active and cover a wide range of PowerShell-related content. You'll also find free eBooks on the site and information about the PowerShell+DevOps Global Summit (https://powershell.org/summit) which is the meeting for learning about what PowerShell, and other DevOps related technologies, can do for you.

YOUR ACTION PLAN

I'll leave you with some recommendations. First, learning how to use PowerShell to manage anything, is like learning a foreign language. **The best way to learn is to immerse yourself and use it every day**. If you really want to master many of the examples I've shared with you, ditch the Hyper-V Manager and begin to learn how to do everything from the PowerShell prompt. It will be a struggle at first and definitely take longer than you expect. But at some point things will click, especially when you start managing Hyper-V hosts and virtual machines at scale.

Once you have worked out some basic PowerShell expressions, you can begin building scripts and functions and before you know it you'll have a custom management suite of PowerShell tools that will make you more efficient. In fact, you'll probably become the go-to person in your organization which is not always a bad thing career-wise.

I hope you found this book informative and inspiring. If our paths cross at training class or conference, I hope you'll let me know.

Good luck and happy scripting!

ABOUT ALTARO

Altaro Software (www.altaro.com) is a fast growing developer of easy to use backup solutions used by over 30,000 customers to back up and restore both Hyper-V and VMware-based virtual machines, built specifically for Small and mid-market business with up to 50 host servers. 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 30,000 satisfied customers worldwide and are a Gold Microsoft Partner for Application Development and Technology Alliance VMware Partner.

ABOUT ALTARO VM BACKUP

Altaro VM Backup is an easy to use backup software solution used by over 30,000 Small and mid-market business customers to back up and restore both Hyper-V and VMware-based virtual machines. Eliminate hassle and headaches with an easy-to-use interface, straightforward setup and a backup solution that gets the job done every time.

Altaro VM Backup is intuitive, feature-rich and you get outstanding support as part of the package. 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!

ALTARO

VM BACKUP

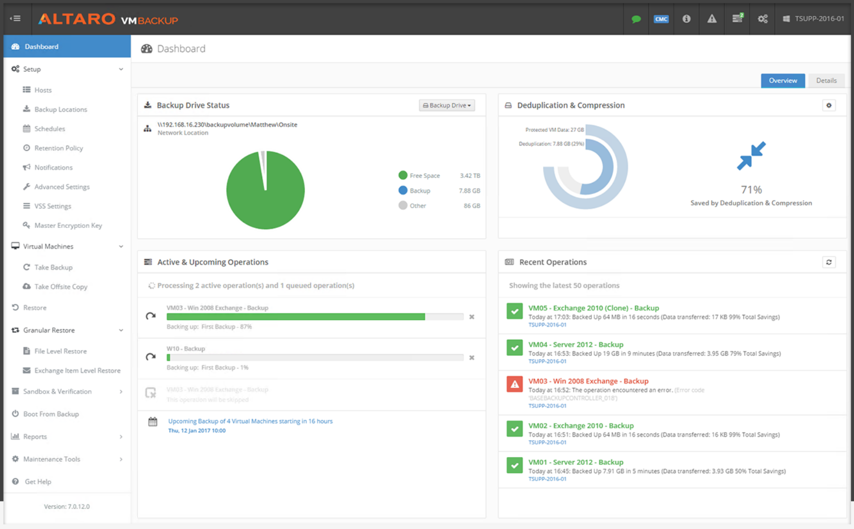
Altaro VM Backup - Trusted by over 30,000 SMBs

New v7! Altaro VM Backup for Hyper-V & VMware. Hassle-free and affordable VM backup software.
Grab your free copy for 2 VMs now!

✓ Hassle-free and effective
✓ Unbeatable Value
✓ Outstanding Support

Free for 2 VMs, forever.

Back up unlimited VMs for 30 -days.
After 30-days you get 2 VMs for free, forever. Download now!



\\192.168.16.230\backupvolume\Matthew\Onsite
Network Location

● Free Space	3.42 TB
● Backup	7.88 GB
● Other	86 GB

Protected VM Data: 27 GB
Deduplication: 7.88 GB (29%)

71%
Saved by Deduplication & Compression

Processing 2 active operation(s) and 1 queued operation(s)

VM03 - Win 2008 Exchange - Backup
Backing up: First Backup - 87%

W10 - Backup
Backing up: First Backup - 1%

VM03 - Win 2008 Exchange - Backup
This operation will be skipped

Upcoming Backup of 4 Virtual Machines starting in 16 hours
Thu, 12 Jan 2017 10:00

Showing the latest 50 operations

VM05 - Exchange 2010 (Clone) - Backup
Today at 17:03: Backed Up 64 MB in 16 seconds (Data transferred: 17 KB 99% Total Savings)
TSUPP-2016-01

VM04 - Server 2012 - Backup
Today at 16:53: Backed Up 19 GB in 9 minutes (Data transferred: 3.95 GB 79% Total Savings)
TSUPP-2016-01

VM03 - Win 2008 Exchange - Backup
Today at 16:52: The operation encountered an error. (Error code 'BASEBACKUPCONTROLLER_018')
TSUPP-2016-01

VM02 - Exchange 2010 - Backup
Today at 16:51: Backed Up 64 MB in 16 seconds (Data transferred: 16 KB 99% Total Savings)
TSUPP-2016-01

VM01 - Server 2012 - Backup
Today at 16:45: Backed Up 7.91 GB in 5 minutes (Data transferred: 3.93 GB 50% Total Savings)
TSUPP-2016-01

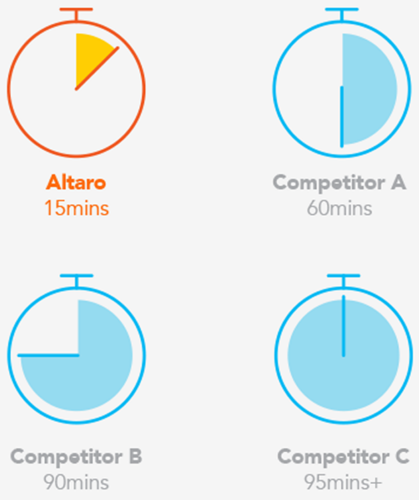
Backup Now!

Up and running quickly, without the need for complex configurations!

With Altaro VM Backup, you can install and run your first virtual machine (VM) backup in less than 15 minutes. Get up and running quickly, without the need for complex configurations or software dependencies.

Altaro VM Backup is designed to give you the power you need, without the hassle and steep learning curve.

- **Easy to use, intuitive UI** - making it easy to implement a rock solid backup strategy
- **Managing and configuring backup/restore jobs across multiple hosts has never been simpler**
- **Full control & scalability** - Monitor and manage all your Hyper-V and VMware hosts from a single console



Virtual machine backup software packed with powerful features for Hyper-V and VMware.

View Features

NEWOUTLOOK.IT

ABOUT JEFF HICKS

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. He works today as an independent author, trainer and consultant. Jeff has taught and presented on PowerShell and the benefits of automation to IT Pros worldwide. Jeff has authored and co-authored a number of books, writes for numerous online sites and print publications, is a contributing editor at Petri.com, a Pluralsight author, and a frequent speaker at technology conferences and user groups.

You can keep up with Jeff on Twitter http://www.twitter.com/JeffHicks and on his blog http://www.blog.jdhitsolutions.com

FOLLOW ALTARO

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

Subscribe to our Hyper-V blog http://www.altaro.com/hyper-v/ 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 RESOURSE!

Liked the eBook? Share it now on: