Global Knowledge®

Expert Reference Series of White Papers

Managing Hyper-V with PowerShell for the GUI-Bound Administrator

Managing Hyper-V with PowerShell for the GUI-Bound Administrator

Michael Porter, MCSE, MCSA, MCT

The PowerShell Environment for Hyper-V

As Hyper-V becomes more popular as an enterprise-level virtualization platform, the need for understanding and using PowerShell to automate and efficiently manage the Hyper-V environment is imperative. This paper discusses the usage of PowerShell cmdlets specific to installing and configuring Hyper-V. As a general approach to presenting this topic the discussion is directed toward IT professionals who have worked with virtualization platforms such as Hyper-V through the GUI, but have not had much hands-on experience with PowerShell. That said, in this paper we are "baby-stepping" our way through the usage of single cmdlets for Hyper-V management. Cmdlets are the building "blocks" of PowerShell scripts. The expectation is that as the readers' understanding of PowerShell for Hyper-V increases, they will naturally move on to the level of PowerShell usage at a more complex script level. So this paper is for all of the virtualization admins who are tethered to their mouse and the graphical interface. If this is you, we will walk (using cmdlets) before we run (scripts that is).

During our discussion the Hyper-V platform used for specific and general points of interest is Hyper-V version 4 running over a fully installed instance of Server 2012 R2. We examine the PowerShell cmdlets run on the "Host" system.

Cmdlets follow a Verb-Noun convention.

Before we step into managing Hyper-V with PowerShell, let's address the very important usage of PowerShell's built-in Help files. Knowing how to use and interpret these files will go a long way in understanding the purpose of any cmdlet and how to use it as examples are provided with all the necessary details. Also, Microsoft updates the Help files on an ongoing basis as the number of available cmdlets increases so updating the Help files is a best practice. That is accomplished by using the Update-Help –Module Hyper-V cmdlet. There are currently 170 cmdlets just for Hyper-V alone.

PowerShell cmdlets specific to Hyper-V enable Virtualization administrators to fully manage their Hyper-V deployments. As PowerShell is closely integrated with the operating system an administrator can simplify the management and maintenance of their burgeoning and increasingly critical virtualized server platform.

Installation and Initial Configuration of Hyper-V with PowerShell

Hyper-V version 3 and later automatically loads the application-specific module to support the appropriate environment. For example, when a role is installed in Server Manager the PowerShell module to support that role is installed also. Knowing that, is still a good idea the know how to manually import, or install, the modules for trouble-shooting purposes at the very least. As a universal approach to learning anything it is always best to be able to "look under the hood" and have a superior grasp of how the technology works. As an extreme "old school" example of this think "it is best to know how to use a slide rule before learning how to use a calculator."

As with all installations of Hyper-V if functionality beyond the base or default set of PowerShell commands, referred to as “cmdlets,” is required then the module-specific set of cmdlets must be installed. This enables specialized scripting for a particular Server-based application such a Hyper-V. This is accomplished by running the Install-WindowsFeature Hyper-V –Restart command. As a best practice prior to installing any module in PowerShell it is recommended to determine which specific modules or features are already installed. This can be done by running the Get-WindowsFeature “*Hyper-V*”. We now verify successful installation by re-booting the server. After the server restarts, launch PowerShell from the command window, and type Get-WindowsFeature | where {$_.Installed –eq $True} to verify the successful installation of the Hyper-V role.

Securing PowerShell’s Execution Policy

Prior to getting started with PowerShell and creating new guest VMs, it a very good idea to determine the execution policy on the host system for running PowerShell. Given PowerShell’s capability regarding all things configuration, its usage in the wrong hands could result in a real problem. Following is a list of the possible execution policies and their values:

- Restricted: Does not load configuration files or run scripts. "Restricted" is the default execution policy.
- AllSigned: Requires that all scripts and configuration files be signed by a trusted publisher, including scripts that you write on the local computer.
- RemoteSigned: Requires that all scripts and configuration files downloaded from the Internet be signed by a trusted publisher.
- Unrestricted: Loads all configuration files and runs all scripts. If you run an unsigned script that was downloaded from the Internet, you are prompted for permission before it runs.
- Bypass: Nothing is blocked and there are no warnings or prompts.
- Undefined: Removes the currently assigned execution policy from the current scope. This parameter will not remove an execution policy that is set in a Group Policy scope.

It is important that you determine the current execution policy by using the following cmdlets first by “getting” the current policy and then “setting” or modifying the current policy if necessary.

```
Get-ExecutionPolicy
Set-ExecutionPolicy
```

Creating a New VM

Let’s get started.

The following command simply creates a new VM instance called “TestVm1" without any additional configuration of the VM.

```
C:\>New-VM –Name "TestVm1"
```

This cmdlet creates a new virtual machine name to work with but as we want to bring this VM into a test or production environment, we are required to allocate the resources necessary to do so. That being the case we can assign resources to the VM. This includes RAM, a virtual hard-drive, the number of cores to utilize for CPU operation and a virtual network connection. Let’s walk through each of the cmdlets required for resource allocation in a logical sequence or progression.

Configuring Memory for the VM

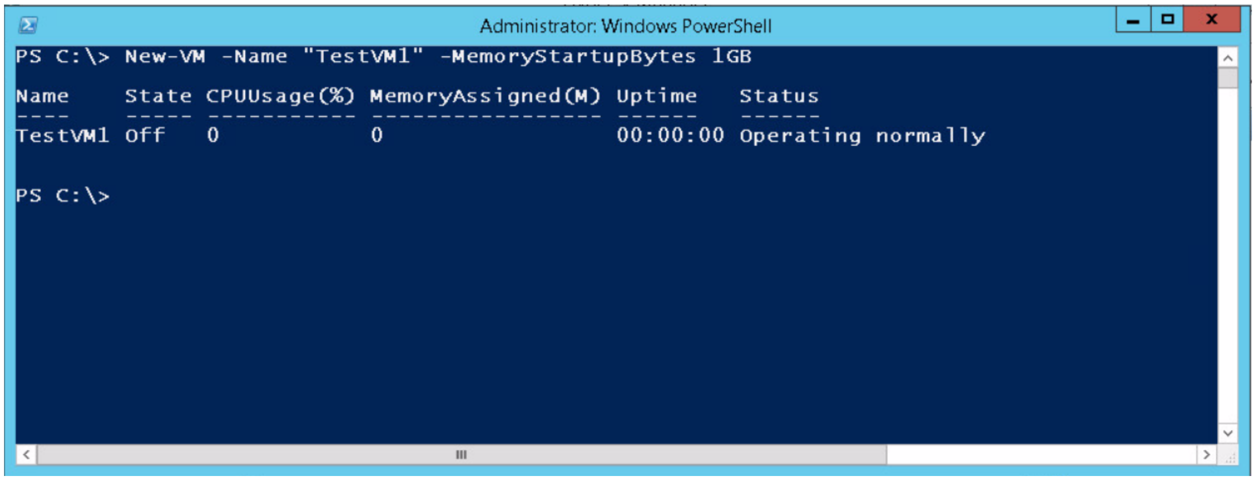
In the process of configuring the new TestVM1 assign the VM the memory it will use at startup or boot.

To do this we use the New-VM cmdlet with the –MemoryStartupBytes switch to configure the amount of Startup RAM for TestVM1.

PS C:\> New-VM –Name “TestVM1” –MemoryStartupBytes 1GB

The amount configured indicates the amount of physical memory that the VM can access and use at boot time. In this example we are assigning 1GB of startup memory.

See cmdlet below:



```
PS C:\> New-VM -Name "TestVM1" -MemoryStartupBytes 1GB

Name    State CPUUsage(%) MemoryAssigned(M) Uptime   Status
----    ----- ----------- ----------------- ------   ------
TestVM1 Off   0           0                 00:00:00 Operating normally


PS C:\>
```

Had we not configured the Startup RAM along with the creation of the new virtual machine, or guest OS, we could have done so after the fact with the Set-VM or SetVMMemory cmdlets.

Once Startup RAM is configured it is time to allocate memory for the VM in general or as it is fully up and running in a test or production environment. At this point the decision is whether or not to enable Dynamic Memory. As we will see, choosing to enable Dynamic Memory configuration is determined by the application workload of that VM and other VMs running on the same host.

Dynamic memory assignment is an allocation of RAM with what is necessary to get the VM up and operational, post startup, in a “just in time” type of scenario. Memory is allocated between a minimum and maximum range. This supports the processes and applications running on the VM as they load, run, terminate and exit. So, if we start with 1GB of minimum RAM it will serve as a minimum amount of memory that the VM will ever use.

The other parameter to configure with this setting is Maximum RAM. This is the maximum amount of memory that the guest VM is allocated. So for the purpose of discussion, if we configure Maximum RAM at 2GB then the VM will “flex” in its usage of RAM from 1GB up to 2GB. In this type of scenario Hyper-V is able to provide a VM additional memory based on the application workload and less memory as the workload fluctuates downward. Allocating memory to VMs on an as needed basis can potentially allow the running of more guest VMs on a single host.

At this point is our discussion, we have TESTVM1 with Startup RAM of 1GB and no Dynamic Memory enabled. That setting or memory allocation can be determined by using the following cmdlet:

Get-VM -Name TestVM1 | select *Memory*

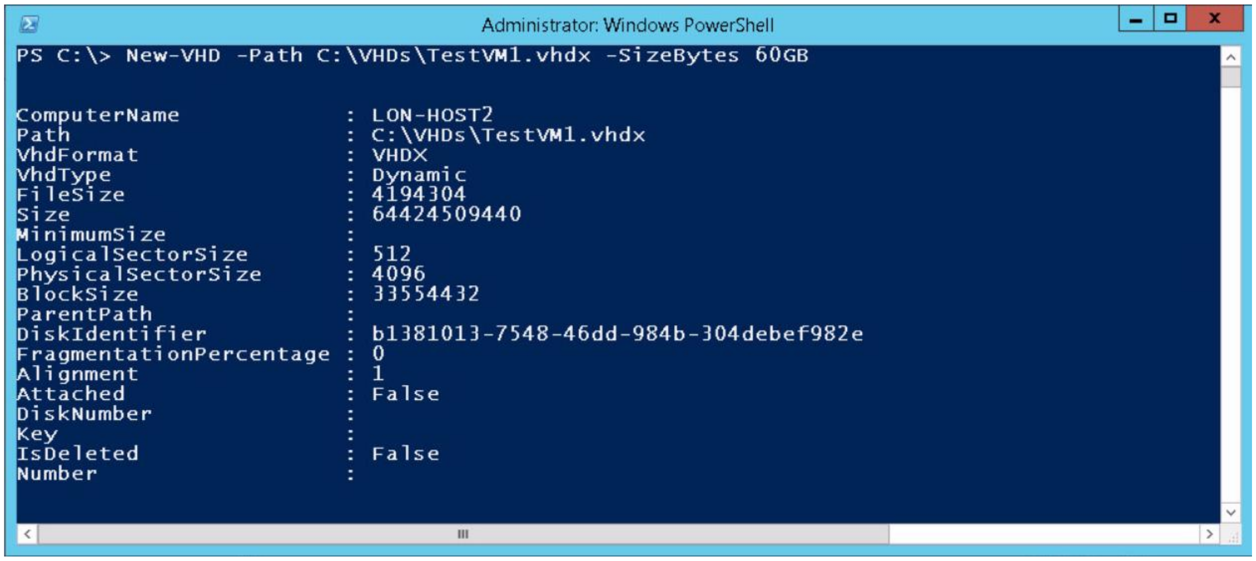
Creating a Virtual Hard Drive for the TestVM1 VM

We now have a guest VM named TestVM1 with a set memory configuration and it is now time to configure the virtual hard drive that TestVM1 will use.

To do so we will use the New-VHD cmdlets with the -Path, -SizeBytes, modifiers. The New-VHD cmdlet creates a new virtual hard disk. It can create the new VHD in either VHD format or the new and improved VHDX format. So when creating the VHD in PowerShell, you specify the extension of the VHD, either VHD or VHDX, in the format you want to create. In our example below we have created a VHD 60 GB in size with a VhdFormat of VHDX. Notice that the VhdType is Dynamic. When created the choices for VhdType are either Fixed or Dynamic. Fixed allocates a set size for the VHD at creation where Dynamic allocates only what is requested by the VM and dynamically allocates disk space as the VM requires up to a maximum of 60GB. (Remember "just-in-case" vs. "just-in-time" from our discussion about memory.) Notice in the screen below that the VhdType is Dynamic which is the default when no type is specified.

PS C:\>New-VHD -Path C:\VHDs\TestVM1.vhdx -SizeBytes 60 GB

See cmdlet in use below:



```
PS C:\> New-VHD -Path C:\VHDs\TestVM1.vhdx -SizeBytes 60GB


ComputerName            : LON-HOST2
Path                    : C:\VHDs\TestVM1.vhdx
VhdFormat               : VHDX
VhdType                 : Dynamic
FileSize                : 4194304
Size                    : 64424509440
MinimumSize             :
LogicalSectorSize       : 512
PhysicalSectorSize      : 4096
BlockSize               : 33554432
ParentPath              :
DiskIdentifier          : b1381013-7548-46dd-984b-304debef982e
FragmentationPercentage : 0
Alignment               : 1
Attached                : False
DiskNumber              :
Key                     :
IsDeleted               : False
Number                  :
```

Now we must attached to the newly created VHD. We accomplish this by using the Add-VMHardDiskDrive cmdlet:

PS C:\>AddVMHardDrive –VMName TestVM1 –Path C:\VHDTestVM1.vhdx

At this point we want to mount an ISO file with the operating system we want running on the guest operating system or VM. So, for example, if we have an ISO of a Windows 8.1 installation media we point to that ISO with a path:

```
PS C:\>Set-VMDvdDrive -VMName TestVM1 -ControllerNumber 1 -Path C:\ISO\windows81.ISO
```

So now we have a guest VM configured to run Windows 8.1 and it is time to start the machine with the following cmdlet:

```
PS C:\>Start-VM –Name TestVM1
```

Configuring the CPU for TestVM1
Configuring CPU use for TestVM1

```
PS C:\>SET-VMProcessor -count 2
```

The will enable the VM to access 2 of the physical processor cores on the Host machine. The default setting is 1.

Configuring the Virtual Network for TestVM1

Assuming we want our TestVM1 machine to be able to communicate to resources over the local physical LAN and beyond (Internet) we need to configure the virtual network adapter for TestVM1 and attach it to a virtual switch. The virtual network adapter virtual switch configuration can be one of the following:

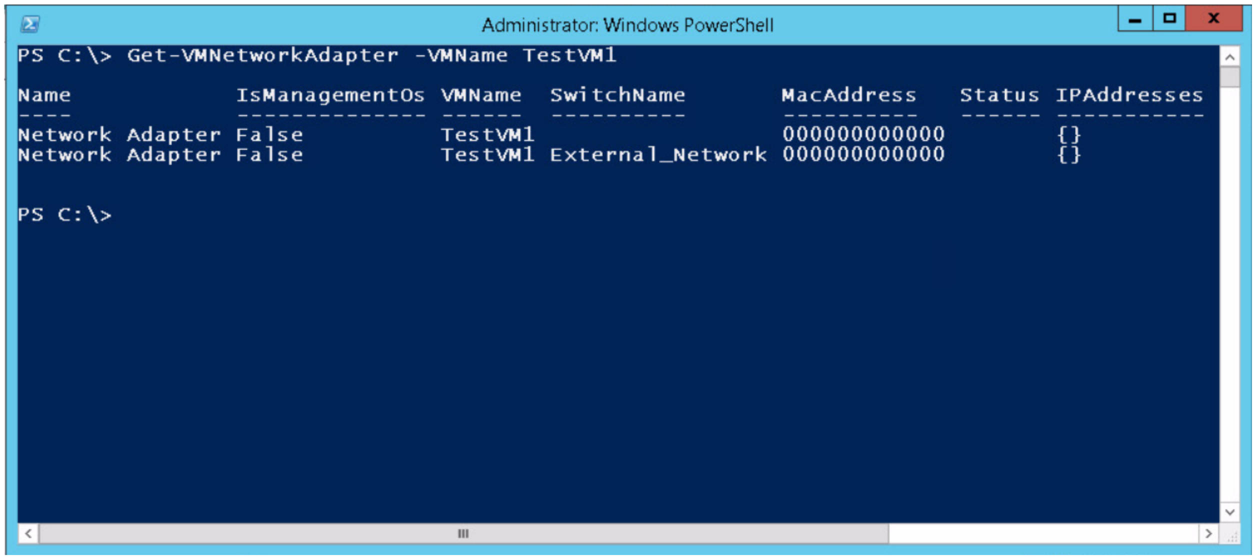
- External: to bind the virtual switch to the physical network adapter of the physical host system allowing for connectivity over the LAN and a fully routable physical network.
- Internal: to connect the virtual switch to other VMs running on the host and to the physical host.
- Private: to connect to only other VMs running on the physical host.

So for our configuration we will use the following:

```
PS C:\>Add-VMNetworkAdapter -VMName "TestVM1" -SwitchName "External_Network"
```

To confirm this configuration we use:

```
PS C:\>Get-VMNetworkAdapter -VMName "TestVM1"
```


Administrator: Windows PowerShell

```
PS C:\> Get-VMNetworkAdapter -VMName TestVM1

Name            IsManagementOs VMName  SwitchName       MacAddress   Status IPAddresses
----            -------------- ------  ----------       ----------   ------ -----------
Network Adapter False          TestVM1                  000000000000        {}
Network Adapter False          TestVM1 External_Network 000000000000        {}


PS C:\>
```

So now we have a new virtual machine configured with memory resources, a virtual hard drive, two CPU cores and a configured network adapter attached to a virtual switch and network.

Management and Maintenance of Hyper-V with PowerShell

Once you have created and configured a VM in PowerShell you can now focus on starting the newly created VM. The Start-VM cmdlet performs that function.

Starting and Stopping a VM

The following command starts the fully configured, for our purposes, TestVM1 guest OS.

PS C:\>Start-VM –Name TestVM1

Conversely, if we want to stop the TestVM1 virtual machine we use the following cmdlet:

PS C:\>Stop-VM TestVM1

If the VM is in the process of running an application or other processes and you want to shutdown to be immediate but as graceful as possible, then you can force the shutdown with the -Force switch.

PS C:\>Stop-VM TestVM1 –Force

At this point let's clear-up some ambiguity surrounding the stopping of a guest VM. The Stop-VM cmdlet send a message to the guest VM and gracefully brings down the VM. Whereas running Stop-VM with the -turnoff option is like pulling the plug on the system.

Additionally, if you wish to have the VM shutdown in a saved state you can include the -save option.

Stop-VM TestVM1 -Save -Force

To take this a step further, if you want to have a restore point or checkpoint to fall back on you can issue the Checkpoint-VM cmdlet and save the checkpoint with a unique identifier such as SnapShotAlpha.

```
CheckPoint-VM –TestVM1 “SnapShotAlpha”
```

Furthermore, if you do not want to completely stop a running VM you can pause it instead by using the Suspend-VM cmdlet.

```
PS C:\>Suspend-VM –Name TestVM1
```

Once you have paused the VM you can “unpause” the VM using the Resume-VM cmdlet.

```
PS C:\>Resume-VM –Name TestVM1
```

Once you have created and named your VM there might be a reason for renaming the VM. For example the naming convention for a grouping of Virtual File Servers is causing host name conflicts with another grouping of Servers over the External network and LAN (plan ahead).

```
PS C:\>Rename-VM TestVM1 –NewName LabVM1
```

Determining Current VM Configuration

Maintenance of a VM generally requires knowing the current configuration state of that VM. The Get-*Noun* cmdlet in is good starting point for learning much about the current configuration of a VM prior to any modifications to the settings of the VM. Below are some examples of this and other cmdlets you might find particularly useful. We will start with a couple of general cmdlets that you can use and then move on to Hyper-V specific cmdlets.

Get-Service is one of the most useful of the GET cmdlets. Running it returns a list of services installed on the system and their status, name, and displayName. This is very similar to the Services applet in Control Panel.

Get-Process is very useful also as the output from the cmdlet displays the number of file Handles a process has as well as the Process Id and the ProcessName. This output is similar to the information you find running Task Manager, Processes Tab.

Get-VM displays all of the installed VMs on a Host system. To take it a step further we use the following cmdlet to return information about a specific VM running on the Host. The output we see here includes valuable information such as the State of the VM, for example “off,” the CPU usage of the VM, the Uptime of the VM and many others.

```
PS C:\>Get-VM Test1VM | Select *
```

From the output found in that cmdlet we can take it further regarding CPU usage of theTestVM1 with this cmdlet:

```
PS C:\>Get-VMProcessor TestVM1
```

In addition to the Get cmdlets there are other useful cmdlets that may come in handy as you are managing your Hyper-V environment.

There a many more Hyper-V focused cmdlets starting with the "Get" verb. For example, you can use the Get-Command with the –Module Hyper-V –Name *Get* to return a list of all Get commands specific to Hyper-V and then you are on your way to "Getting" all things Hyper-V.

Migrating a VM (Live Migrations)

PowerShell can be used to perform Hyper-V live migrations. In this situation you migrate and move a live instance of a VM from one Host system to another. If done properly you have the salient benefit of no user downtime! This is a new feature in Windows Server 2012 R2 called cross-version live migration. For example, it enables moving a running virtual machine from Windows Server 2012 to Windows Server 2012 R2.

Prior to performing live migrations you are required to enable it on the Source and Destination Host Servers. To determine if the Host Server is configured for live migration you run the Get-VMHost cmdlet and determine the value of the VirtualMachineMigrationEnabled *field|attribute* as either true or false. If the value is false you can enable live migration using the Enable-VMMigration cmdlet. Please note that for live migration to be enabled a migration network must be configured first using the Add-VMMigrationNetwork cmdlet. It is recommended that a separate network is designated and configured for migration traffic. If that is not possible, you can configure VM migration to use any configured network for migration through the Set-VMHost–UseAnyNetworkForMigration $True on the Destination Host Server. So once we have configured the Source and Destination hosts for migration we use the Move-VM cmdlet for the actual live migration.

Moving a VM to another Hyper-V Server Host

You may find yourself in a situation where the host server is running low on hardware resources and you need to redistribute one or more VMs to another host server that has an excess of hardware resources. The Move VM cmdlet will accomplish this objective.

```
Move-VM "TestVM1" DestinationServerName
```

Export a VM

Unlike the dynamic process of a live migration the Export-VM and Import-VM cmdlets reference a path to the location of a folder containing the VM. This folder location is indicated in the cmdlet Export-VM and contains three subfolders. Each of these sub folders have data including Snapshots, VHDs and a Virtual Machine configuration file in XML format. Once that data is exported to a folder location it can then be referenced using the Import-VM cmdlet to enable importing a virtual machine from a file. When using the Import-VM cmdlet, be sure to include the filename in the path. The cmdlets and switches for Exporting and Importing a VM are:

```
C:\> Export-VM –Name TestVM1 – C:\VMExport_ImportFolder
C:\> Import-VM –Name TestVM1 – C:\VMExport_ImportFolder\Test1.xml
```

Removing an Existing VM

And finally, if you want to remove a VM you can do so using the Remove-VM cmdlet.

```
C:\>Remove-VM "TestVM1"
```

Conclusion

In this discussion we have explored the basic usage of PowerShell in the pursuit of automating the management of Hyper-V deployments. This paper is targeted to IT professionals who have been "GUI-Bound" to their management interfaces and have been working with or are embarking on the journey of Hyper-V administration. We explored the usage of Hyper-V specific cmdlets with the intent of ultimately being able to include multiple cmdlets in a "script" to allow for pushing one button and accomplishing the combined actions of many multiple mouse clicks in the GUI. No more repetitive tasks!

Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge through training.

PowerShell Essentials

Automating Administration with Windows PowerShell (M10961)

Windows PowerShell Scripting and Toolmaking (M55039)

Visit **www.globalknowledge.com** or call **1-800-COURSES (1-800-268-7737)** to speak with a Global Knowledge training advisor.

About the Author

Michael Porter is a 20-year veteran of the information technology industry. During that time he has served as a consultant to organizations ranging from small to medium size businesses and instructor to Fortune 1000 corporations. He hold a bachelor's degree from Miami University in Oxford, Ohio and a master's degree from The University of Dayton. In addition to consulting Michael has written and edited several technology publications and developed numerous courses.