Mike Nelson – SSV article

Managing XenServer with PowerShell Deep Dive

In a previous article, I wrote about starting out with using powershell with XenServer. Now, we'll take a little deeper look into some sample scripts and how you can utilize PowerShell and the XenServer cmdlets to streamline and automate your administrative tasks.

For those of you that haven't yet installed the XenServer PowerShell cmdlets, as of this writing, there still is an issue when installing them on Windows 7 x64. You have to download the XenServer PowerShell Snap-In from here and extract the PowerShell cmdlets. To get the snap-in working under a x64 system, you need to run the following command from an elevated command prompt to register the DLL with the 64bit .NET Framework –

C:\windows\microsoft.net\frameowkr64\v2.0.50727\installutil.exe "c:\program files (x86)\citrix\xenserverpssnapin\xenserverpssnapin.dll"

You should see a final "The commit phase completed successfully" message meaning it all registered fine. Also, go ahead and download XenServer Snapshots snap-in created by Shannon Ma if you do use Snapshots in your environment. Make sure to add it to your PowerShell profile (see the previous article mentioned above for details and a script) and also don't forget to always do a "Connect-Xenserver" to start out your sessions. Without it, you can't do much and PowerShell will spew error messages back at you. Now, you should be ready to rock!

In the previous article, we went over some of the cmdlets and their basic usage for copying VM's, determining the Pool Master, configuring licensing and host editions, and setting a homeserver for a VM. Let's go deeper and take a look at some more elaborate scripts and one-liners that you can use with XenServer.

Find a XenServer guest by MAC address (Dr. Miru) – To determine a VM based off of it's MAC address, you must obtain the object property for the VIF or Virtual InterFace, as shown below:

```
$vif = get-xenserver:VIF | ? { $_.MAC -match "ce:e2:b7:56:85:7f"};(Get-
XenServer:VIF.VM -VIF $vif.uuid).name_label
```

By applying the "VIF" property to the Get-Xenserver cmdlet, we are distinctly looking for the MAC address of the virtual interface. Now, replace that VIF with another property, such as VM. This example shows how to retrieve the VM information for the selected UUID.

```
Get-XenServer:VM -properties @{ uuid='$uuid' }
```

As you can see, just by changing the property, you can retrieve, manage, and correlate different results based on that property value. The listing of available

properties for just the Get-XenServer cmdlets is pretty elaborate and long It includes things like Role, Pool, Secret, Sessions, Bond, and the list goes on. Run a "get-help Get-XenServer" command and see all of them that are available and create your own query quick and easy.

Besides the well-used Get-XenServer cmdlet, there are many others with many properties that can be explored. You have the ability to utilize functions of XenServer, such as Create, Add, Destroy, Revoke, Set, and Remove. Remember, to list all the XenServer cmdlets and their properties that are available to you, run a "Get-Command *xen*" from the PowerShell prompt and the list will scroll by with its numerous entries.

Let's take a look at some more elaborate scripts that I've found that show more of the inner-workings of the XenServer cmdlets, how complex they can get, and how you could use them. This update script from Maikel Gaedker is one that I have used often in the past and it gives some great examples of how properties are used and stored in variables:

```
# Xenserver-UpdateScript
# V 0.9.2
# 20.07.2012
# written by Maikel Gaedker (maikel@gaedker.de)
#
# Updates XenServer with all patches available within a defined path
# Requirements: PowerShell v2.0; XenServerPSSnapIn

Import-Module XenServerPSSnapIn

$server = read-host "Enter XenServer to patch" # Enter XenServer Hostname or IP-Address when promted
$SecureString = read-host "Enter root-password" -asSecureString
$password =
[Runtime.InteropServices.Marshal]::PtrToStringAuto([Runtime.InteropServices.Marshal]::SecureStringToBSTR($secureString))

# Path to folder with downloaded XenServerpatches
# Only have required patches within these folder
$updatepath = "\\Fileserver\Shared_Folder\xs_updates" # Change to whatever your folder is

#
#Start Script
#

Connect-XenServer -server $server -UserName root -Password $password
write-host "$server will be patched; running VMs will be shut down" -foregroundcolor "green"
write-host "========================================================="
```

```
Get-XenServer:VM | Where-Object { $_.is_a_template-eq $false-and
$_.is_control_domain-eq $false} | foreach-object {Invoke-XenServer:VM.CleanShutdown
-RunAsync -vm $_.name_label}

foreach ($update in Get-ChildItem $updatepath)
{
write-host "Patch: $update will be prepared" -foregroundcolor "green"
Invoke-XenServer:Host.Disable -Host $server
Invoke-XenServer:Host.Reboot -Host $server
write-host "Server will be rebooted" -foregroundcolor "green"
start-sleep -s 300 # test if this time could be shorten for yor Xen Host
Connect-XenServer -server $server -UserName root -Password $password
write-host "$update will be uploaded" -foregroundcolor "green"
xe.exe -s $server -u root -pw $password patch-upload file-name=$updatepath\$update
| out-file $server"_uuid.txt"
$uuid_patch = get-content .\$server"_uuid.txt"
write-host "$update will be installed" -foregroundcolor "green"
xe.exe -s $server -u root -pw $password patch-pool-apply uuid=$uuid_patch
write-host "$server was updated with patch: $update" -foregroundcolor "green"
write-host "======================================================="
}

write-host "$server was updated with all required patches; one last reboot" -
foregroundcolor "yellow"
Invoke-XenServer:Host.Disable -Host $server
Invoke-XenServer:Host.Reboot -Host $server
start-sleep -s 300 # test if this time could be shorten for yor Xen Host
Connect-XenServer -server $server -UserName root -Password $password
Remove-Item .\$server"_uuid.txt"
write-host "VMs will be started" -foregroundcolor "green"
Get-XenServer:VM | Where-Object { $_.is_a_template-eq $false-and
$_.is_control_domain-eq $false} | foreach-object {Invoke-XenServer:VM.Start -vm
$_.name_label}
Disconnect-XenServer
```

Here is another good example that was written earlier this year by Ingmar Verheij. It allows users to connect to a XenServer VM's console via RDP. He previously wrote a script that allowed a direct console connection to the VM, but RDP is much more efficient and you do have console access. This is also a good primer of how to use 3rd party applications in your PowerShell scripts, such as plink.exe, which he uses to invoke the command line version of the SSH client Putty. The last thing to motice, and I find this very intuitive with any PowerShell scripting, is the ability to execute native OS or CLI commands within the script itself. In the Retrieve the IP of the VM section, notice how he uses the "xe vm-list command", which is a XenServer CLI command, to filter the properties of the VM.

```
#
# Name              : RDSXSConsole.ps1
# Description   : Connects to VM hosted on a XenServer hypervisor via
Microsoft Remote Desktop Client
# Author            : Ingmar Verheij - http://www.ingmarverheij.com/
# Version           : 1.0, 13 february 2012
#
```

```
# Requires          : plink (a command-line interface to the puTTY back
ends)
#
http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html
#

function StartProcess([String]$FileName, [String]$Arguments){
    $process = New-Object "System.Diagnostics.Process"
    $startinfo = New-Object "System.Diagnostics.ProcessStartInfo"
    $startinfo.FileName = $FileName
    $startinfo.Arguments = $arguments
    $startinfo.UseShellExecute = $false
    $startinfo.RedirectStandardInput = $true
    $startinfo.RedirectStandardOutput = $true
    $startinfo.WindowStyle =
[System.Diagnostics.ProcessWindowStyle]::Hidden
    $process.StartInfo = $startinfo
    $temp = $process.start()
    return $process
}

#Region PrequisiteCheck
   #Check number of arguments
   If ($args.count -lt 5)
   {
      Write-Host "Usage"
      Write-Host "powershell.exe .\RDSConnect.ps1 (XenServerPoolMaster)
(XenServerUsername) (XenServerPassword) (VMName) (Network ID)
[CustomFieldName] [CustomFieldValue]"
      Write-Host ""
      Write-Host "Example"
      Write-Host "powershell.exe .\RDSConnect.ps1 172.16.1.1 root
Passw0rd WS01 0 STUDENT 1"
      Write-Host ""
      Write-Host "Press any key to continue ..."
      $x = $host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")
      break
   }
#EndRegion
#Region Define variables and read

   #Executables
   $strExecutableMSTSC=((Get-Item
"Env:WinDir").Value)+'\system32\mstsc.exe'
   $strExecutablePLink=(Split-Path -parent
$MyInvocation.MyCommand.Definition) + '\plink.exe'

   #File paths
   $strPathTemp=$Env:TEMP
   $strFileQueryNetworks ='QueryNetworks'

   #Script variables
   $XenServerHost=$args[0]
   $XenServerUsername=$args[1]
   $XenServerPassword=$args[2]
   $VirtualMachineName=$args[3]
   $VirtualMachineNetworkID=$args[4].toString()
   If ($args.count -ge 7) {
             $CustomFieldName=$args[5]
             $CustomFieldValue=$args[6]
   } else {
         $CustomFieldName=""
         $CustomFieldValue=""
```

```
   }
   $strNICInterface=($VirtualMachineNetworkID)+'/ip: '

   #Filter variables
   $strFilterVM='name-label="' + $VirtualMachineName+'"'
   IF ($CustomFieldName) {$strFilterVM+=' other-
config:XenCenter.CustomFields.' + $CustomFieldName + '=' +
$CustomFieldValue}
#EndRegion



#Prevent rsa2 key fingerprint message
#====================================
#The server's host key is not cached in the registry. You have no
guarantee that the server is the computer you #think it is.
#The server's rsa2 key fingerprint is: ssh-rsa 2048
7c:99:f3:31:38:ca:b7:b6:3b:21:53:55:ff:f3:76:1e
#If you trust this host, enter "y" to add the key to PuTTY's cache and
carry on connecting.
#If you want to carry on connecting just once, without adding the key to
the cache, enter "n".
#If you do not trust this host, press Return to abandon the connection.
#
#Run plink and confirm rsa2 key fingerprint with yes
#----------------------------------------------------
$process = StartProcess $strExecutablePLink (' -l '+$XenServerUsername+'
-pw '+$XenServerPassword+' '+$XenServerHost+' exit')
$process.StandardInput.WriteLine('y')


#Retrieve IP of VM
#=================
#
#Create a script to query a XenServer and ask the networks of the VM
#-------------------------------------------------------------------
New-Item $strPathTemp -Name $strFileQueryNetworks -type file -Force  |
Out-Null
Add-Content ($strPathTemp + '\' + $strFileQueryNetworks) -Value ('xe vm-
list '+$strFilterVM+' os-version:distro="windows" params=networks --
minimal')


#Run the script on the specified XenServer
#-----------------------------------------
$process = StartProcess $strExecutablePLink (' -l '+$XenServerUsername+'
-pw '+$XenServerPassword+' '+$XenServerHost+' -m '+($strPathTemp + '\' +
$strFileQueryNetworks))
$VMNetworks = $process.StandardOutput.ReadLine()
Remove-Item ($strPathTemp+'\'+$strFileQueryNetworks)


#Determine if the networks of the virtual machine can be found
#-------------------------------------------------------------
if(!$VMNetworks) {
   Write-Host "The virtual machine '"$VirtualMachineName"' could not be
found."
   Write-Host ""
   Write-Host "Press any key to continue ..."
   $x = $host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")
   break
} else {
```

```
   #Determine the IP address of the NIC can be found
   foreach ($strVMNetwork in $VMNetworks.Split(";")) {
      if ($strVMNetwork.Contains($strNICInterface)) {

$strVMIPaddress=$strVMNetwork.Substring($strVMNetwork.IndexOf($strNICInte
rface) + $strNICInterface.Length)
      }
   }

   #Determine if the IP address of the network ID can be found
   #-------------------------------------------------------------
   if(!$strVMIPaddress) {
      Write-Host "The IP address of network '"$VirtualMachineNetworkID"'
could not be found."
      Write-Host ""
      Write-Host "Press any key to continue ..."
      $x = $host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")
      break
   }
   else {
       Write-Host "The virtual machine '"$VirtualMachineName"' is
connected via IP "$strVMIPaddress
   }
}



#Start MSTSC to the VM
#======================
$processMSTSC=StartProcess $strExecutableMSTSC ('/v:'+$strVMIPaddress+'
/f')
```

Finally, I offer up this very elaborate and extremely functional script that automates provisioning of virtual desktops posted on the Citrix Blogs earlier this year by Ajene' Hall Barrett. If you take a close look at all the sections of it, you will see many variable references, active directory cmdlets, and a mix of XenServer and PVS (Provisioning Server) cmdlets. It really is a well written and educational piece of script. Since it is so large, I will only offer a link to the blog posting to save a few digital trees ☺. You can read it here.

As you can see from all these examples and by trying them out for yoiurself, there are so many things you can automate and administer with the XenServer PowerShell cmdlets. And, as Citrix evolves this toolset even more, which is in their roadmap to do so, it can only get better for all of us.