1
Configuring the Basic Settings of an ESXi Host with PowerCLI

In this chapter, you will cover the following topics:

- Connecting to an ESXi host or vCenter instance
- Getting the VMware host object
- Joining an ESXi host into Active Directory
- Enabling services and setting security profiles
- Setting network configuration
- Creating datastores on an ESXi host
- Configuring syslog settings on a host
- Joining an ESXi host to vCenter
- Creating a configuration script to set all properties uniformly

Introduction

Initially automation doesn't save time. To really get the benefits of automation, you must invest the time upfront to create scripts that you'll use time and again. In this chapter, you'll take your first ESXi host that has been installed and has an IP address configured on it and you will build continually from there. This chapter will take an administrator through the basic configuration tasks needed to perform initial configuration, join them to vCenter, and get it into an operational state. At the end of this chapter, all these steps build into a scripted configuration that can be executed against new hosts in the future.

Connecting to an ESXi host or vCenter instance

To begin working with PowerCLI, you must first have PowerShell installed and available on the system you want to run PowerCLI. PowerShell is part of the Windows Management Framework and it ships with Windows client and server versions. PowerCLI extends PowerShell with commands to administer VMware environments. With PowerShell installed, you will need to obtain PowerCLI from vmware.com. The specific link is listed in the See also... section of this recipe.

Once you have installed PowerCLI, you will need an ESXi host built for this recipe. All that is required is a fresh ESXi installation from the ISO or DVD image distributed by VMware. Once installed, set an IP address on an accessible network using the console screens of the new ESXi host. The network address should be accessible from your PowerCLI workstation.

With the assumption that your ESXi host is built, the first step to administering VMware environments in PowerCLI is to connect to the ESXi host or to a vCenter server. In this chapter, you will focus on configuring a single ESXi host and in the next chapter you will focus on configuring a vCenter Server and vSphere cluster of ESXi hosts.

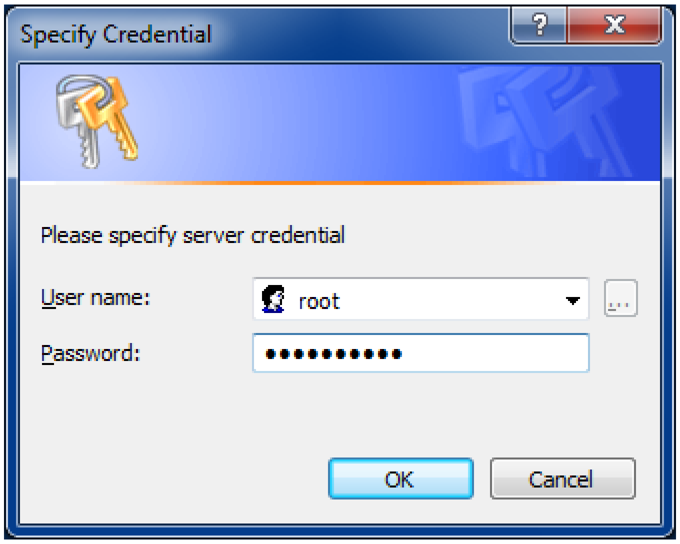
Getting ready

To begin, you only need to launch PowerCLI from its shortcut on the desktop or from the Start Menu. If you already had PowerCLI previously installed, you will want to check the version number to ensure that the cmdlets references throughout the book are available to you. Each version of PowerCLI builds additional native cmdlets and functionality. To check the version you are running, open a PowerCLI prompt and run `Get-PowerCLIVersion`.

The recipes in this book are built and tested using VMware PowerCLI version 5.5 Release 1 and have also been tested with PowerCLI version 5.8 Release 1.

How to do it...

1. At the PowerCLI prompt, you will execute the `Connect-VIServer` cmdlet.
   ```
   Connect-viserver <hostname or IP>
   ```
2. When executed, the code will attempt to single sign-on into the ESXi host, but unless your username is root and you set the same password locally and on ESX, single sign-in will fail. You will be prompted with a normal Windows login window, displayed below, and you should login with the root username and password you specified during your ESXi installation.



~~Insert Image 3724EN_01_01.png~~

3. Once you successfully log into the ESXi host, a confirmation message will be displayed with the name or IP address of the ESXi host you connected to, the port, and the user you've connected as shown in the following example:

```
Name                              Port  User
----                              ----  ----
192.168.0.241                     443   root
```

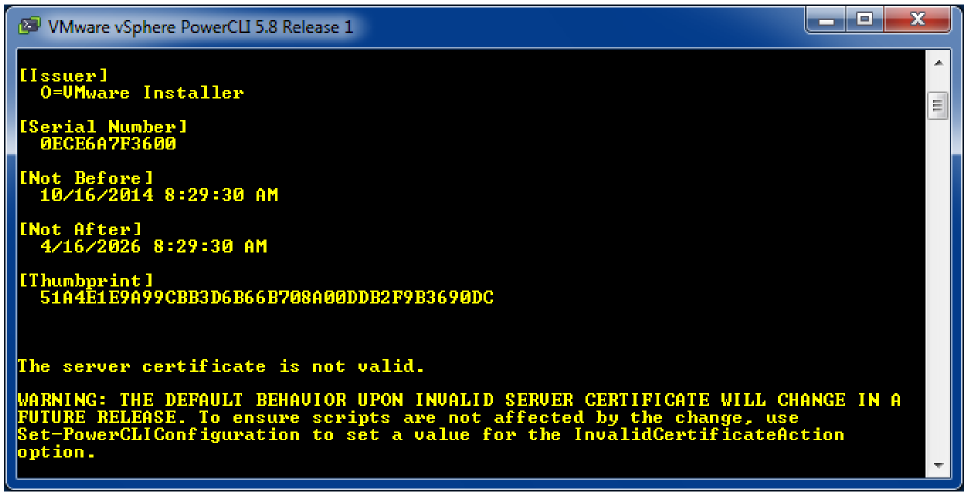
4. At this point, the PowerCLI session is connected to a host and ready to execute work.

How it works…

The `Connect-VIServer` cmdlet is the simplest kind of cmdlet in PowerCLI. The cmdlet initiates a connection to the vCenter or ESXi web services to allow for additional commands to be passed to the server and executed.

The `Connect-VIServer` cmdlet requires only the name of the host to which you want to connect. There are additional parameters that you may pass to the cmdlet, such as the protocol (HTTP or HTTPS), the username, and the password. If you prefer not to keep your password in plain text, you can also pass a `PSCredentials` object. The `PSCredentials` object contains login data to authenticate. For more information about PSCredential objects, type `get-help about_server_authentication`.

Once you execute the cmdlet, a warning will be displayed in yellow like the one below.



```
[Issuer]
  O=VMware Installer

[Serial Number]
  0ECE6A7F3600

[Not Before]
  10/16/2014 8:29:30 AM

[Not After]
  4/16/2026 8:29:30 AM

[Thumbprint]
  51A4E1E9A99CBB3D6B66B708A00DDB2F9B3690DC


The server certificate is not valid.

WARNING: THE DEFAULT BEHAVIOR UPON INVALID SERVER CERTIFICATE WILL CHANGE IN A
FUTURE RELEASE. To ensure scripts are not affected by the change, use
Set-PowerCLIConfiguration to set a value for the InvalidCertificateAction
option.
```

~~Insert Image 3724EN_01_02.png~~

The warning is displayed because the certificate installed on the ESXi host is self-signed and untrusted by the computer you are connecting from. Changing SSL certificate on ESXi hosts will be covered later in the book, but the warning may be ignored at this time. The cmdlet will continue to execute even though the warning is displayed.

You may also prevent the invalid certificate errors by running the following PowerCLI cmdlet, which changes the action when an invalid certificate is encountered.

```
Set-PowerCLIConfiguration -InvalidCertificateAction Ignore -Scope
Session -Confirm:$false
```

There's more…

If you choose to join the ESXi host to Active Directory, your PowerCLI session performs a single sign-in. PowerCLI uses the credentials of your current Windows session to login against the ESXi host or vCenter server, if your account has access. If your account does not have access to the server it is attempting to connect, a login box will be presented like our example in this recipe.

See also

- Joining an ESXi host into Active Directory, Chapter 1
- Setting permissions on vCenter objects, Chapter 2
- VMware PowerCLI Documentation Center & Installation Download https://www.vmware.com/support/developer/PowerCLI/

Getting the VMware host object

After connecting to a host to manage, other cmdlets become available. The first concept that you will need to become aware of are PowerShell objects. Objects are defined data obtained from commands run in PowerShell and PowerCLI. To perform configuration on an ESXi host, the commands you run will need a host object specified. In this recipe, you will learn how to obtain a VMHost object.

Getting ready

To begin with, open a PowerCLI window and connect to an ESXi host or vCenter instance.

How to do it...

1. To retrieve an ESXi host object, PowerCLI is straightforward.
   ```
   Get-VMHost
   ```
2. After running the `Get-VMHost` cmdlet, an object containing one or more ESXi hosts is returned. You are connecting to a single ESXi host in this example, running `Get-VMHost` returns host object with a single host. If you were connecting against a vCenter instance, `Get-VMHost` with no other arguments would return an object containing all of the hosts managed by vCenter. When running against vCenter, you may specify a filter with the `Get-VMHost` cmdlet in order to find one or more hosts that match the specified pattern.
   ```
   Get-VMHost esxhost*
   Get-VMHost VMHOST1
   ```
3. Instead of having to call the `Get-VMHost` cmdlet each time you need to get the ESXi host, you can store the host object in a variable. PowerShell variables are specified as a `$` followed by a name. Below is an example for our ESXi host:
   ```
   $esxihost = Get-VMHost
   ```

How it works…

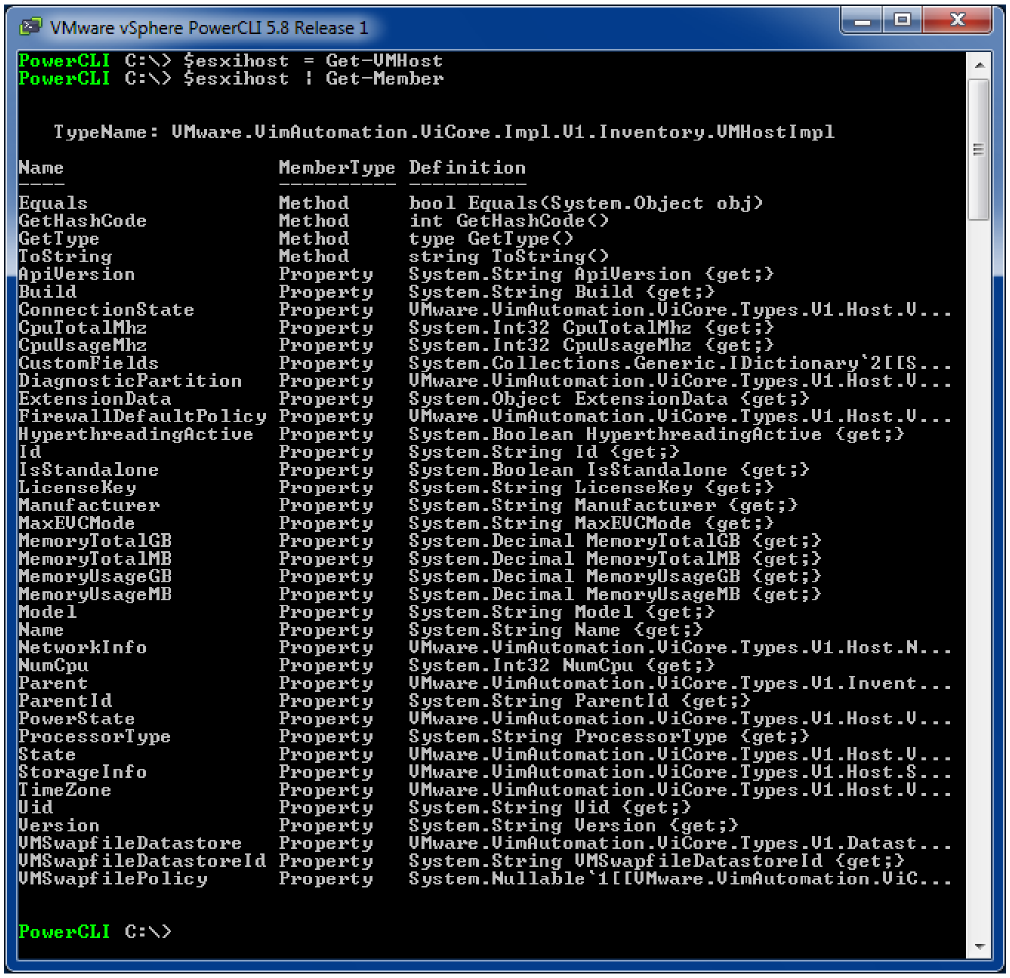
To examine more about the `VMHost` object, you can use the `Get-Member` cmdlet with the variable you have just defined. To use `Get-Member`, you will call the `VMHost` object by typing the `$esxihost` variable. Then, you pipe the object into the `Get-Member` cmdlet.

```
$esxihost | Get-Member
```

PowerCLI is an extension of PowerShell that is used specifically for VMware product management. PowerShell is an object-based language meaning that it uses the concept of encapsulating both data and operations within an object data type, which is a familiar object-oriented programming concept. Objects have defined data areas and may include functions that perform operations on the data in the object.

The output from the cmdlet shows all of the data contained in the `Property` elements in the object. The object also includes a number of methods. The methods are used to manipulate the data in the object.

```
PowerCLI> $esxihost | Get-Member
```



```
PowerCLI C:\> $esxihost = Get-VMHost
PowerCLI C:\> $esxihost | Get-Member


   TypeName: VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl

Name                     MemberType Definition
----                     ---------- ----------
Equals                   Method     bool Equals(System.Object obj)
GetHashCode              Method     int GetHashCode()
GetType                  Method     type GetType()
ToString                 Method     string ToString()
ApiVersion               Property   System.String ApiVersion {get;}
Build                    Property   System.String Build {get;}
ConnectionState          Property   VMware.VimAutomation.ViCore.Types.V1.Host.V...
CpuTotalMhz              Property   System.Int32 CpuTotalMhz {get;}
CpuUsageMhz              Property   System.Int32 CpuUsageMhz {get;}
CustomFields             Property   System.Collections.Generic.IDictionary`2[[S...
DiagnosticPartition      Property   VMware.VimAutomation.ViCore.Types.V1.Host.V...
ExtensionData            Property   System.Object ExtensionData {get;}
FirewallDefaultPolicy    Property   VMware.VimAutomation.ViCore.Types.V1.Host.V...
HyperthreadingActive     Property   System.Boolean HyperthreadingActive {get;}
Id                       Property   System.String Id {get;}
IsStandalone             Property   System.Boolean IsStandalone {get;}
LicenseKey               Property   System.String LicenseKey {get;}
Manufacturer             Property   System.String Manufacturer {get;}
MaxEVCMode               Property   System.String MaxEVCMode {get;}
MemoryTotalGB            Property   System.Decimal MemoryTotalGB {get;}
MemoryTotalMB            Property   System.Decimal MemoryTotalMB {get;}
MemoryUsageGB            Property   System.Decimal MemoryUsageGB {get;}
MemoryUsageMB            Property   System.Decimal MemoryUsageMB {get;}
Model                    Property   System.String Model {get;}
Name                     Property   System.String Name {get;}
NetworkInfo              Property   VMware.VimAutomation.ViCore.Types.V1.Host.N...
NumCpu                   Property   System.Int32 NumCpu {get;}
Parent                   Property   VMware.VimAutomation.ViCore.Types.V1.Invent...
ParentId                 Property   System.String ParentId {get;}
PowerState               Property   VMware.VimAutomation.ViCore.Types.V1.Host.V...
ProcessorType            Property   System.String ProcessorType {get;}
State                    Property   VMware.VimAutomation.ViCore.Types.V1.Host.V...
StorageInfo              Property   VMware.VimAutomation.ViCore.Types.V1.Host.S...
TimeZone                 Property   VMware.VimAutomation.ViCore.Types.V1.Host.V...
Uid                      Property   System.String Uid {get;}
Version                  Property   System.String Version {get;}
VMSwapfileDatastore      Property   VMware.VimAutomation.ViCore.Types.V1.Datast...
VMSwapfileDatastoreId    Property   System.String VMSwapfileDatastoreId {get;}
VMSwapfilePolicy         Property   System.Nullable`1[[VMware.VimAutomation.ViC...


PowerCLI C:\>
```

~~Insert Image 3724EN_01_03.png~~

You can call a method by using dot notation and by calling the method name followed by parenthesis, like the example below.

```
PowerCLI> $esxihost.ConnectionState.ToString()
Connected
```

In this example, the `State` property is an object inside of the `VMHost` object but the `ToString()` method converts the output to a string.

Now that the ESXi host object is stored in a variable, you can proceed with other cmdlets for configuration and run them using the host object to perform configuration.

There's more…

`Get-VMHost` has other applications beyond just returning the `VMHost` object to use. Like all other `Get-` cmdlets, this cmdlet can be used to find a host in a particular configuration or state. You can use `Get-VMHost` to find hosts assigned to a particular location in vCenter using the `-Location` parameter, you may want to find hosts that have been assigned a particular Tag in vSphere using the `-Tag` parameter or you may want to find the host running a particular VM with the `-VM` parameter. Another interesting use case is specifying the `-Datastore` parameter to find all the hosts that have a particular datastore connected.

`Get-VMHost` is just one of many cmdlets that work with `VMHost` objects. Others will be explored in the "Configuring vCenter and computer clusters," Chapter 2.

See also

- Setting up folders to organize objects in vCenter, Chapter 2
- Creating basic reports of VM properties from VMware Tools and PowerCLI, Chapter 3

Joining an ESXi host into Active Directory

As mentioned in the connecting section, joining an ESXi host to Active Directory offers the ability to connect without entering credentials for administrators. Active Directory is a Windows implementation of Lightweight Directory Access Protocol (LDAP). Active Directory contains accounts for users, computers, and groups. Active Directory runs on a Windows Server that has the Active Directory role installed and that has been "promoted" to become a domain controller. To perform this recipe, you will need at least one Active Directory server available on the network with the ESXi host.

Seamless authentication is one of the biggest reasons to join a host to Active Directory. But beyond single sign-on, once the ESXi host is connected to Active Directory, groups in the directory can be leveraged to grant permissions to the ESXi host. If you do not have Active Directory installed and do not wish to, you may skip this recipe and move on to other topics of host configuration without any impact to future recipes.

Getting ready

PowerCLI has `Get-VMHostAuthentication` and `Set-VMHostAuthentication`, two cmdlets to deal with host authentication. To get ready to setup authentication, open a PowerCLI window and connect to a single ESXi host.

How to do it...

1. Because the cmdlets require a `VMHost` object, you'll again be using the `Get-VMHost` to either populate a variable or to pipe the object to the next object. The first step is to obtain a `VMHost` object for our target ESXi host.

    ```
    $esxihost = Get-VMHost 192.168.0.241
    ```

2. Once you have your `VMHost` object, you can look at setting authentication. The `Set-VMHostAuthentication` cmdlet needs to be executed. The cmdlet requires several parameters to join a ESXi host to domain. The syntax needed is displayed follows:

    ```
    $esxihost | Get-VMHostAuthentication | Set-
    VMHostAuthentication -JoinDomain -Domain domain.local -user
    username -password *****
    ```

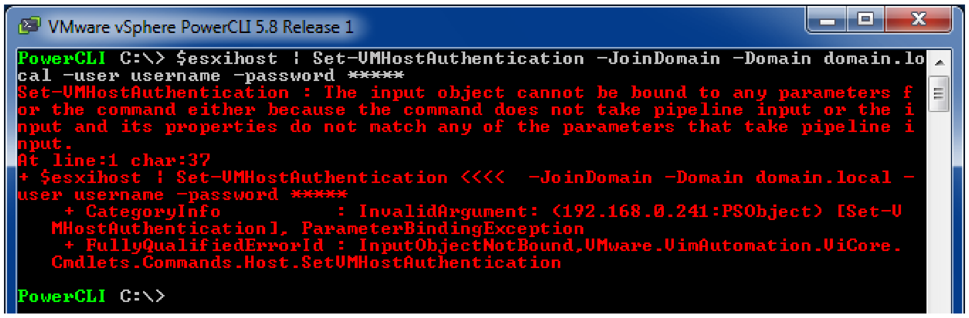
3. Executing the cmdlet will prompt you to confirm that you want to join this host to the domain specified. If you answer `Y`, the cmdlet will continue and execute the operation.

    ```
    Perform operation?
    Joining VMHost '192.168.0.241' to Windows Domain
    'domain.local'.
    [Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend
    [?] Help
    (default is "Y"):Y

    Domain          DomainMembershipStatus    TrustedDomains
    ------          ----------------------    --------------
    DOMAIN.LOCAL    Ok
    ```

How it works…

One of the first things you notice about this recipe is that there is an extra `Get-VMHostAuthentication` cmdlet in the middle of the command line. Why does it need to perform `Get` before performing the `Set`? It would seem that you can simply pipe the `VMHost` object into cmdlet to specify your target host and the cmdlet will execute, but as you try that below, PowerCLI displays an error.

```
$esxihost | Set-VMHostAuthentication -JoinDomain -Domain
domain.local -user username -password *****
```

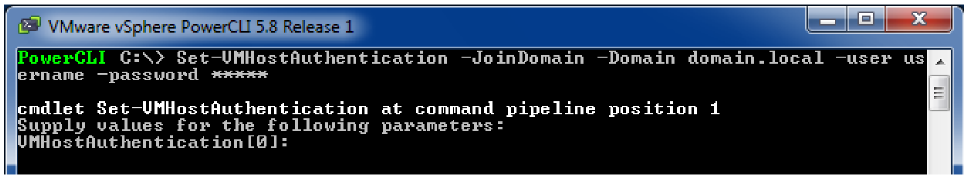


```
PowerCLI C:\> $esxihost | Set-VMHostAuthentication -JoinDomain -Domain domain.lo
cal -user username -password *****
Set-VMHostAuthentication : The input object cannot be bound to any parameters f
or the command either because the command does not take pipeline input or the i
nput and its properties do not match any of the parameters that take pipeline i
nput.
At line:1 char:37
+ $esxihost | Set-VMHostAuthentication <<<<  -JoinDomain -Domain domain.local -
user username -password *****
    + CategoryInfo          : InvalidArgument: (192.168.0.241:PSObject) [Set-V
   MHostAuthentication], ParameterBindingException
    + FullyQualifiedErrorId : InputObjectNotBound,VMware.VimAutomation.ViCore.
   Cmdlets.Commands.Host.SetVMHostAuthentication

PowerCLI C:\>
```

~~Insert Image 3724EN_01_05.png~~

In this case, the cmdlet is looking for a `VMHostAuthentication` object and not a `VMHost` object, so an error is displayed. If you go back and simply execute the `Set-VMHostAuthentication` cmdlet, it will prompt for a `VMHostAuthentication` object and wait for input.

```
Set-VMHostAuthentication -JoinDomain -Domain domain.local -user
username -password *****
```



```
PowerCLI C:\> Set-VMHostAuthentication -JoinDomain -Domain domain.local -user us
ername -password *****

cmdlet Set-VMHostAuthentication at command pipeline position 1
Supply values for the following parameters:
VMHostAuthentication[0]:
```

~~Insert Image 3724EN_01_06.png~~

This is where the `Get-VMHostAuthentication` cmdlet gets added – retrieving the `VMHostAuthentication` object from the host you targeted since this cmdlet accepts the `VMHost` object as piped input.

The `Get-Help` for the `Set-VMHostAuthentication` also shows that the cmdlet expects a `VMHostAuthentication` object to be passed as a parameter for the cmdlet. By executing the cmdlet with all of its parameters and no pipe input, you also learned that you can debug and learn what input the cmdlet is expecting and is missing.

There's more…

The same cmdlets can also be used to remove a host from a domain, if needed. The `-LeaveDomain` parameter is a part of the `Set-VMHostAuthentication` cmdlet and allows for this need.

In addition to setting an ESXi host to accept Active Directory authentication, PowerCLI also provides a number of cmdlets to add local users, groups and permissions inside of a single ESXi host. The `New-VMHostAccount` cmdlet is used to create new users on an ESXi system. The same cmdlet previously allowed the creation of groups, but this functionality was removed with ESXi 5.1. There is a `Set-VMHostAccount` cmdlet to change accounts and group memberships and a `Remove-VMHostAccount` cmdlet to remove a user or group.

See also

- Setting permissions on vCenter objects, Chapter 2

Enabling services and setting security profiles

ESXi hosts enable a few services by default, but there are some additional services that are installed but blocked. In some cases, you may want to enable SSH on the host. But since VMware does not recommend having SSH enabled and will display a warning, however, you can set an advanced setting to disable this warning.

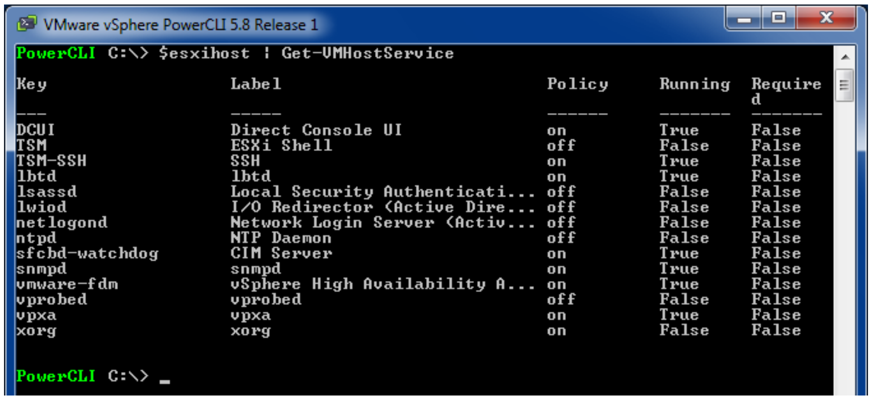
Getting ready

To begin, you should open a PowerCLI prompt and connect to an ESXi or vCenter host. You will also want to store a `VMHost` object in a variable called `$esxihost`.

How to do it…

1. The first step for this process is to get the list of available services from a VMware host. To do this, you use the `Get-VMHostService` cmdlet and pass the `VMHost` object into the cmdlet.

```
$esxihost | Get-VMHostService
```



VMware vSphere PowerCLI 5.8 Release 1

```
PowerCLI C:\> $esxihost | Get-VMHostService

Key                  Label                          Policy     Running  Require
                                                                        d
---                  -----                          ------     -------  -------
DCUI                 Direct Console UI              on         True     False
TSM                  ESXi Shell                     off        False    False
TSM-SSH              SSH                            on         True     False
lbtd                 lbtd                           on         True     False
lsassd               Local Security Authenticati... off        False    False
lwiod                I/O Redirector (Active Dire... off        False    False
netlogond            Network Login Server (Activ... off        False    False
ntpd                 NTP Daemon                     off        False    False
sfcbd-watchdog       CIM Server                     on         True     False
snmpd                snmpd                          on         True     False
vmware-fdm           vSphere High Availability A... on         True     False
vprobed              vprobed                        off        False    False
vpxa                 vpxa                           on         True     False
xorg                 xorg                           on         False    False


PowerCLI C:\> _
```

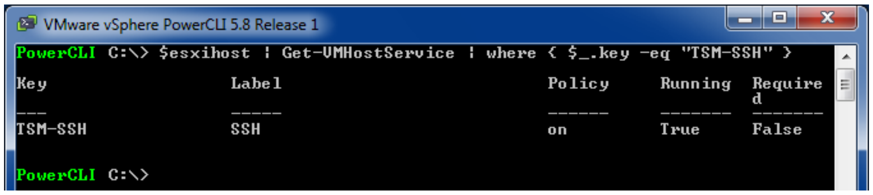
~~**Insert Image 3724EN_01_07.png**~~

2. The output of the cmdlet will show a list of the available services on the ESXi host along with its policy (whether it is set to on or off by default) if it

is running. The label is a friendly identifier to find the service you want to configure, but the key is the piece of data you will use to return the single service you want.

3. In this case, we're looking to configure the service with the `TSM-SSH` key. To scope the results down to that one service in the object, you use a PowerShell `where` clause.

```
$esxihost | Get-VMHostService | where { $_.key -eq "TSM-SSH" }
```

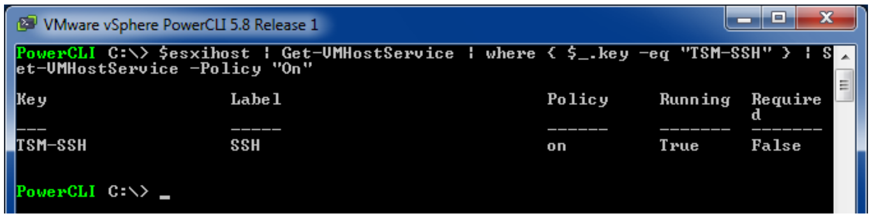


```
PowerCLI C:\> $esxihost | Get-VMHostService | where { $_.key -eq "TSM-SSH" }

Key                 Label                         Policy     Running  Require
                                                                      d
---                 -----                         ------     -------  -------
TSM-SSH             SSH                           on         True     False


PowerCLI C:\>
```

~~Insert Image 3724EN_01_08.png~~

4. Now that you have it scoped down to a single service, you pass this object into the `Set-VMHostService` cmdlet with the desired policy of `"On"`.

```
$esxihost | Get-VMHostService | where { $_.key -eq "TSM-SSH" } | Set-VMHostService -Policy "On"
```

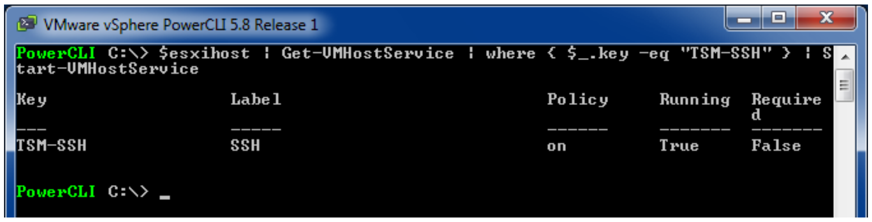


```
PowerCLI C:\> $esxihost | Get-VMHostService | where { $_.key -eq "TSM-SSH" } | S
et-VMHostService -Policy "On"

Key                 Label                         Policy     Running  Require
                                                                      d
---                 -----                         ------     -------  -------
TSM-SSH             SSH                           on         True     False


PowerCLI C:\> _
```

~~Insert Image 3724EN_01_09.png~~

5. At this point, you have configured the host to autostart the service on boot, but the service is still not running in the current boot. To do this, you will instead use the `Start-VMHostService` cmdlet. Again, you have to pass in the `VMHostService` object for SSH (or any other service that you choose).

```
$esxihost | Get-VMHostService | where { $_.key -eq "TSM-SSH" } | Start-VMHostService
```



```
PowerCLI C:\> $esxihost | Get-VMHostService | where { $_.key -eq "TSM-SSH" } | S
tart-VMHostService

Key                 Label                         Policy     Running  Require
                                                                      d
---                 -----                         ------     -------  -------
TSM-SSH             SSH                           on         True     False


PowerCLI C:\> _
```

~~Insert Image 3724EN_01_10.png~~

6. With the service running, vSphere displays the warning that you have enabled SSH. This will leave your host showing in a warning state as long as the service is running, however, VMware does allow you to suppress this warning, but this is set through an advanced setting. To set this, you need to execute the following cmdlet:

   ```
   $esxihost | Get-AdvancedSetting -Name
   UserVars.SuppressShellWarning | Set-AdvancedSetting
   -value 1
   ```

7. When executed, this will prompt to confirm the settings. This confirmation can be suppressed using the – `Confirm:$false` common parameter, which is useful in scripts.

   ```
   $esxihost | Get-AdvancedSetting -Name
   UserVars.SuppressShellWarning | Set-AdvancedSetting
   -value 1 -Confirm:$false
   ```

How it works…

For configuring host services, there native cmdlets follow the expected pattern of Get and Set functionality in PowerCLI. `Get-VMHostService` expects a `VMHost` object as input and this is logical since these host services exist within the scope of a host. Once you get the host service by name and store it in a variable or pass it as an object in the pipeline, you can easily set the settings to the desired configuration. In addition to Get and Set cmdlets, you also have Start and Stop cmdlets. These are more specific to this use case since we're dealing with host services and there is a specific need to start or stop them in addition to configuring them. The Start and Stop cmdlets also accept `HostService` objects as input just like the `Set-VMHostService` cmdlet.

In the specific use case of the SSH Server service, it causes a warning to display in the client. To disable this warning, you used an Advanced Setting called UserVars.SupressShellWarning to disable this from displaying. While not recommended for production systems, there are plenty of use cases where SSH is needed and helpful in lab environments, where you may want to configure the setting.

There's more…

The cmdlet to start the SSH service can be easily adapted beyond the illustrated use case with the use of a `ForEach` loop. For troubleshooting and configuration, you may need to enable SSH in order to tail a log file or to install a custom module. In these cases, starting SSH in bulk may be handy. To do this, you take the code above and wrap it in the loop. An example of this is below with a connection to a vCenter host, a variable with multiple VMHost objects returned and a loop to step through and start SSH on each.

```
Connect-VIServer vcenterhost.domain.local
$esxihosts = Get-VMHost
foreach ($esxihost in $esxihosts) {
```

```
$esxihost | Get-VMHostService | where { $_.key -eq
"TSM-SSH" } | Start-VMHostService
}
```

This quickly allows you to turn on SSH for temporary use. Following a reboot, the service would no longer be running and you could easily change the code above to be a `Stop-VMHostService` cmdlet and turn off the service in bulk.

Setting network configuration

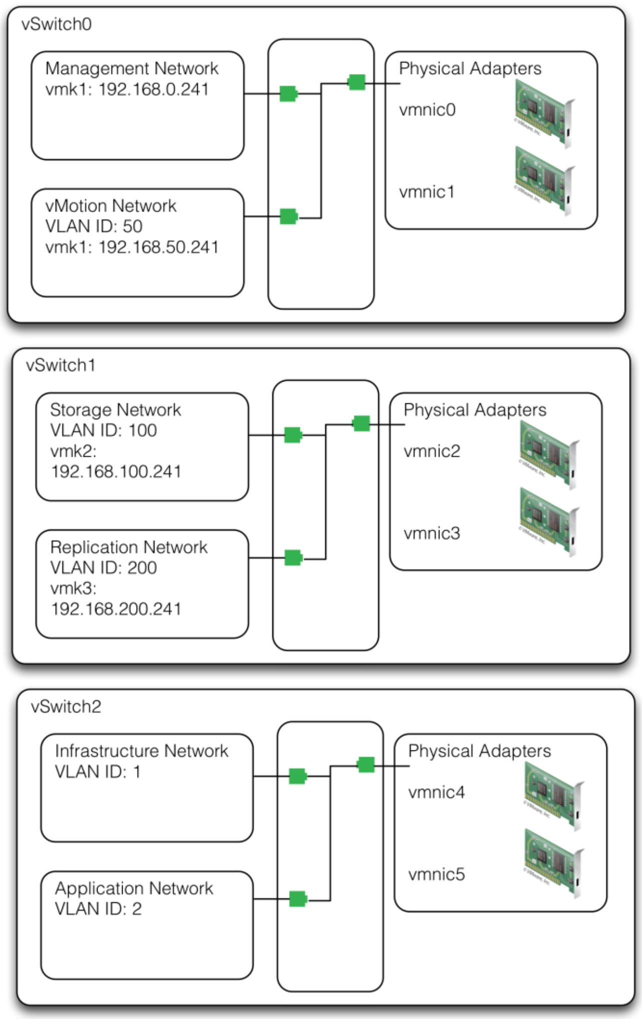
One of the first things to be completed against a new ESXi installation is network configuration. Network configuration consists of several things on an ESXi host – first would be configuring the additional management interfaces of the host for VMotion, Fault Tolerance logging, vSphere Replication, and VSAN traffic.

Getting ready

To begin this recipe, you will need to open a PowerCLI window, connect to an ESXi host, and load a `VMHost` object into a variable. The example uses `$esxihost` as the variable for the `VMHost` object.

On installation, ESXi has a single **network interface card** (**NIC**) labeled eth0 connected to a VMware Standard vSwitch. The vSwitch has two port groups created – one labeled "Management Network" for management traffic and one labeled "VM Network." The "Management Network" is a vmkernel port with the IP defined at the console attached to it.

In this example, our host contains six 10 Gigabit NICs that will connect the host with the network. You will define two additional vSwitches with two physical ports attached to each for redundancy. The additional vSwitches will handle storage and replication traffic on one and VM traffic on the other.



~~**Insert Image 3742EN_01_11.png**~~

Best practices of vSphere networking is far beyond the scope of this book. The network layout above is not an endorsement of a particular layout and is for illustration purposes to show the PowerCLI cmdlets used to configure networking on ESXi.

How to do it...

1. To begin, let's get an idea of the network layout that is in place by default. From a default install, there is a single virtual switch named vSwitch0. The first cmdlet shows you the properties of this virtual switch and the second shows the port groups associated with that vSwitch. To do this, review the output of two PowerCLI cmdlets:

   ```
   $esxihost | Get-VirtualSwitch
   $esxihost | Get-VirtualPortGroup -VirtualSwitch vSwitch0
   ```

2. The first thing to be completed is to remove the default "VM Network" port group since it is not a best practice to have Virtual Machine traffic on the management ports and this default port group is not part of the design you outlined for this configuration.

   ```
   $esxihost | Get-VirtualPortGroup -Name "VM Network" |
   Remove-VirtualPortGroup -Confirm:$false
   ```

3. This command combines the `Get-VirtualPortGroup` and `Remove-VirtualPortGroup` cmdlets to change the confirmation. When executed, you will receive either a confirmation or error. If the port group is connected to or in use by a VM, you will receive an error message. Once you remove the "VM Network" port group, the next step is to add an additional vmkernel port that will be used for vMotion.

> While it is outside of the scope of this book, there are many different ideas for best design of VMware networking. Most administrators agree that Management traffic and vMotion traffic should be separated but with increasing speeds and capabilities of NICs today, it is common to see them sharing the same virtual switch. Administrators will set the management traffic to be active on the first NIC and vMotion to be active on the second NIC. The two traffic streams would only be on the same NIC in a failover situation.

4. In our design, you will set Management and vMotion to be collocated on the same switch. To do this, use the `New-VMHostNetworkAdapter` cmdlet and pass in the name of the port group, the virtual switch and IP information. You also pass in a parameter to specify that this vmkernel port should be used for VMotion.

   ```
   $esxihost | New-VMHostNetworkAdapter -PortGroup "VMotion
   Network" -VirtualSwitch vSwitch0 -IP 192.168.50.241 -
   SubnetMask 255.255.255.0 -VMotionEnabled $true
   ```

5. In our design, although vMotion and Management traffic exist on the same vSwitch, but the traffic will be separated by using active and standby links on each port group. This is done by changing the NIC Teaming Policy with the `Set-NicTeamingPolicy` cmdlet. Notice in the next two commands that the

active and standby NIC assignments are opposite between the two port groups.

```
$esxihost | Get-VirtualPortGroup -Name “Management Network”
| Get-NicTeamingPolicy | Set-NicTeamingPolicy –
MakeNicActive vmnic0 –MakeNicStandby vmnic1

$esxihost | Get-VirtualPortGroup -Name “VMotion Network” |
Get-NicTeamingPolicy | Set-NicTeamingPolicy –MakeNicActive
vmnic1 –MakeNicStandby vmnic0
```

6. The port group is automatically created and the vmkernel/host port is created for our vMotion network, but it is on the wrong VLAN. Our vMotion traffic is on a different VLAN, so you need to set this on the port group.

    ```
    $esxihost | Get-VirtualPortGroup -Name "VMotion Network" |
    Set-VirtualPortGroup –VlanID 50
    ```

7. The next step is to create a new virtual switch with its own uplinks on vmnic2 and vmnic3, as outlined in our diagram. To confirm the physical NICs exist, you can run the following cmdlet.

    ```
    $esxihost | Get-VMHostNetworkAdapter
    ```

    The `Get-VMHostNetworkAdapter` cmdlet displays all the vmkernel ports along with all the physical NICs present on the host.

8. After confirming the NIC, you will run the `New-VirtualSwitch` cmdlet to provision the new virtual switch. This cmdlet provisions the vSwitch with its uplinks, but it is currently an island with no connectivity for management or virtual servers.

    ```
    $esxihost | New-VirtualSwitch -Name vSwitch1 -Nic
    vmnic2,vmnic3
    ```

9. The next step is to create vmkernel ports for storage traffic and for replication traffic. These are created the same as the VMotion Network provisioned earlier.

    ```
    $esxihost | New-VMHostNetworkAdapter -PortGroup "Storage
    Network" -VirtualSwitch vSwitch1 -IP 192.168.100.241 -
    SubnetMask 255.255.255.0 -VsanTrafficEnabled $true

    $esxihost | Get-VirtualPortGroup -Name "Storage Network" |
    Set-VirtualPortGroup –VlanID 100

    $esxihost | New-VMHostNetworkAdapter -PortGroup "FT Logging
    Network" -VirtualSwitch vSwitch1 -IP 192.168.200.241 -
    SubnetMask 255.255.255.0 -FaultToleranceLoggingEnabled
    $true

    $esxihost | Get-VirtualPortGroup -Name "FT Logging Network"
    | Set-VirtualPortGroup –VlanID 200
    ```

10. Again, you want to make sure that our Storage Traffic and Fault Tolerence traffic don’t end up competing for bandwidth, so you will assign one port

group to be active on one uplink and the other port group to be active on the second uplink. This is done again with the `Set-NicTeamingPolicy` cmdlet.

```
$esxihost | Get-VirtualPortGroup -Name “Storage Network” |
Get-NicTeamingPolicy | Set-NicTeamingPolicy –MakeNicActive
vmnic2 –MakeNicStandby vmnic3

$esxihost | Get-VirtualPortGroup -Name “FT Logging Network”
| Get-NicTeamingPolicy | Set-NicTeamingPolicy
–MakeNicActive vmnic3 –MakeNicStandby vmnic2
```

11. The final step of our network provisioning is to create new port groups for virtual machine traffic. You have set all the virtual machine traffic to its own vSwitch and uplinks in the design outlined. The first step is to create the virtual switch like you did for vSwitch1.

    ```
    $esxihost | New-VirtualSwitch -Name vSwitch2 -Nic
    vmnic4,vmnic5
    ```

12. Once the virtual switch is created, you can create two port groups on the virtual switch. In this case, however, `New-VirtualPortGroup` doesn't allow any pipeline input, so you will need to specify the server as a parameter instead of passing it through the pipeline.

    ```
    New-VirtualPortGroup -Name "Infrastructure Network"
    -VirtualSwitch vSwitch2 -VLanId 1 -Server 192.168.0.241

    New-VirtualPortGroup -Name "Application Network"
    -VirtualSwitch vSwitch2 -VLanId 2 -Server 192.168.0.241
    ```

How it works…

In this example, you work with the `VMHost` object to enumerate and identify the existing configuration that is put in place during the installation. From there, you remove the default VM networking configuration, you provision new virtual switches and vmkernel ports to segment traffic and enable certain management functions across the vmkernel ports.

While most of the configuration covered in this section deals with the initial configuration of a host, some of the concepts are repeated more often. For instance, if you have a multi-node cluster and you're adding a new virtual machine network, you'll use the `New-VirtualPortGroup` cmdlet often. Like you have seen in previous examples, you can easily create an array of ESXi hosts - either by using `Get-VMHost` in vCenter or by manually specifying a list of hosts - and then connect and provision the same port group on many hosts, quickly. This would mean a big time savings and less potential for manual error when compared to manually clicking through the GUI to configure the new port group on each host in the cluster.

By also using the `Set-NicTeamingPolicy` cmdlet, you can set a preferred uplink port for each port group and put the other NIC into standby mode. This allows us to keep the Management and vMotion separated and the Storage and Fault Tolerance traffic separated so that they do not cause degraded performance of one another.

There's more…

In this recipe you focus on VMware Standard vSwitches. Users with Enterprise Plus licensing also have the option of using VMware Distributed vSwitches which have their own set of cmdlets to manage and configure these advanced virtual switches.

See also

- Network Management with vSphere Distributed Switches, VMware vSphere 5.5 Documentation Center - http://bit.ly/1wJs1JP (http://pubs.vmware.com/vsphere-55/topic/com.vmware.powercli.ug.doc/GUID-D2C0E491-A0CB-4799-A80D-19EA9114B682.html)

Creating datastores on an ESXi host

With networking, VMware has done a lot of work to ease administration with the VMware Distributed Virtual Switch. In vSphere 5.5, VMware introduced Datastore Clusters which alleviate some of the management of datastores, but from a provisioning standpoint, the initial setup of storage is still manual and can take a lot of manual steps. Scripting this makes a lot of sense in large environments.

Datastore and storage under vSphere is also different since some operations must be performed on the raw storage device and these steps are not repeated on every host. There are three types of storage connectivity that you may need to provision – NFS, iSCSI, and Fibre Channel. For this example, you will focus on iSCSI and NFS and will work on provisioning storage from both. Along the way, Fibre Channel will also be discussed since its concepts overlap with iSCSI, from a vSphere perspective.

Getting ready

For this example, you will need to open a PowerCLI window and connect to an ESXi host. You will also want to make sure you have the VMHost object stored in a variable called `$esxihost`, covered in the 'Getting the VMware host object' section.

How to do it…

1. The simplest of all datastores to provision is an NFS datastore. A single PowerCLI cmdlet will provision an NFS datastore. The `New-Datastore` cmdlet will take all the input needed to provision the new datastore and make it available for use. Since NFS does not use the VMFS filesystem, there are no filesystem properties that need to be passed. To connect NFS, you just need to provide a name for vSphere to identify the datastore, a path (the export) and the host that is providing the NFS.

   ```
   $esxihost | New-Datastore -Nfs -Name DataStoreName -Path
   /data1/export -NfsHost nfsserver.domain.local
   ```

2. With that, you've got your first datastore presented and ready to host virtual machines. For NFS, this is all that is required.
3. iSCSI and Fibre Channel storage is a bit more complex to provision from a PowerCLI and vSphere perspective. Provisioning storage on either of these protocols will require additional decisions to be made when creating the datastore. iSCSI also requires additional configuration steps that are not needed with Fiber Channel. You will focus on iSCSI for this example and I will note where the concepts overlap with Fibre Channel.
4. iSCSI is an IP-based storage protocol and as such, you will need to do a bit of network configuration to set up iSCSI to work in our environment. The first thing that needs to be done is to enable iSCSI and to create a software iSCSI target.

   ```
   $esxihost | Get-VMHostStorage
   ```

5. By default, there isn't a software iSCSI target created. To create this, you expand upon the previous cmdlet and set this value to true.

   ```
   $esxihost | Get-VMHostStorage | Set-VMHostStorage -
   SoftwareIScsiEnabled $true
   ```

6. The next step is to set the iSCSI targets using the `New-IscsiHbaTargets` cmdlet. This cmdlet requires that you pass in the iSCSI HBA as an object, so first you retrieve the iSCSI HBA using `Get-VMHostHba` and store it in a variable and then use it with `New-IscsiHbaTargets`:

   ```
   $iSCSIhba = $esxihost | Get-VMHostHba -Type iScsi

   New-IScsiHbaTarget -IScsiHba $iSCSIhba -Address $target
   -ChapType Required -ChapName vSphere -ChapPassword
   Password1
   ```

In the example, there are additional parameters for authentication. iSCSI uses **Challenge-Handshake Authentication Protocol** (**CHAP**) to authenticate sessions to the target storage. Authentication is not required and if the storage system is not configured for authentication, these parameters could be omitted, however, its bad practice to deploy a production storage array without authentication.

7. The final step of iSCSI initial configuration is to bind the iSCSI HBA to a specific port. Since you created a "Storage Network" management port, this is the port that you want to use. To make this change and to remove any other ports, you have to use the ESXCLI interface within PowerCLI. There isn't a native PowerCLI cmdlet for this function.
   ```
   $esxcli = Get-ESXCLI -VMHost $esxihost
   $esxcli.iscsi.networkportal.add($iscsihba, $true,"vmk2")
   ```
8. In our case, the vmkernel port assigned to the Storage Network port group is vmk2. Using the ESXCLI interface, you can assign it to the iSCSI HBA. To confirm the change, you can use the `list()` method as follows:
   ```
   $esxcli.iscsi.networkportal.list()
   ```
9. If you notice there are other vmkernel ports listed, as in my case `vmk0`, you can remove them with a simple `remove()` method.
   ```
   $esxcli.iscsi.networkportal.remove($iscsihba,$true,"vmk0")
   ```

Now that the system has its targets configured, if the iSCSI array has provisioned storage to the host, it should be visible. This is the point where iSCSI and Fibre Channel converge. Since iSCSI uses the host bus adapter model that Fibre Channel invented, they work the same after initial configuration. Where you must run the NFS mount on each server and you must set up iSCSI initial configuration on each host, scanning and formatting VMFS datastores only needs to be done from a single host for iSCSI and Fibre Channel disks since they are shared resources. This means that when scripting the steps on each host, the next few steps only need to be done on a single host in the cluster and then every host needs to be a rescan.

```
$esxihost | Get-VMHostStorage -RescanAllHBA -RescanVmfs
```

Starting with a rescan is a good place so that your system recognizes all storage changes and sees all disks that have been presented. Whether you're using software or hardware iSCSI, Fibre Channel, or converged network adapters, this is the point where your hosts see its SAN disks.

10. At this point, your ESXi system doesn't have iSCSI or Fibre Channel datastores that it can use. Even though the disk is visible, it is unformatted and not ready to host VMs. To discover your disks and to enumerate the data you need to configure it, you will need to use the `Get-ScsiLun` cmdlet.
    ```
    $esxihost | Get-ScsiLun
    ```
11. This returns a list of disks available to the SCSI subsystem under ESXi. The list may contain a lot of objects. You can use various properties returned by the `ScsiLun` object to identify and leverage the list for provisioning. For instance, you can scope the list using the Vendor property or by the model.

For the purposes of this example, you will assume that you have a disk identified by the model "iSCSIDisk" and use that for scoping. To create a new datastore on the disk, you need the canonical name, which is also a property in the ScsiLun object.

```
$LUN = $esxihost | Get-ScsiLun | Where {$_.Model -like
"iSCSIDisk"}
```

12. In situations where you have many disks presented to a host, identification by model may not be the best. Another method would be to use the `RuntimeName` property which enumerates the HBA, controller, Target, and LUN number. For instance, if you know the LUN number want to prepare is LUN 8 which is represented in the `RuntimeName` as L8, the PowerCLI to scope and return this would be:

```
$LUN = $esxihost | Get-ScsiLun | Where {$_.RuntimeName -
like "*L8"}
```

13. By storing the LUN in a variable, I can verify the returned value to ensure that you have the correct object and number of objects expected before passing it into the `New-Datastore` cmdlet.

```
$esxihost | New-Datastore -Name iSCSIDatastore1 -Path
$LUN.CanonicalName -VMFS
```

14. This provisions the disk as a VMFS filestore and allows it to be used for VM storage. At this point, you can initiate a rescan on all the ESXi hosts in the cluster and they will all see the same shared storage.

How it works…

Provisioning datastores in vSphere works differently for each type of SAN storage. NFS is simpler than iSCSI or Fibre Channel and just requires that you connect (or mount) the datastore for use on the host. Software-based iSCSI requires that you do some additional configuration on the host so that it can connect to the target array, but then iSCSI and Fibre Channel both work the same with backend storage LUNs being presented to the host for consumption.

See also

- Creating and managing a Datastore Cluster, Chapter 4
- Performing Storage vMotion, Chapter 4

Configuring syslog settings on a host

Booting your ESXi from SD or USB flash storage is a common scenario. When booting from SD and USB, however, ESXi does not use that storage for logging and instead keeps the logs in memory, which is non-persistent.   Now that you have established shared, persistent storage, you can point the ESXi hosts syslog functions to store the logs onto the shared disk so that it can survive a reboot or help to troubleshoot. Even hosts booting from local spinning disk may want to redirect their syslog onto a shared SAN drive so that it is accessible from other hosts if a host fails.

Another common use in enterprises is a centralized syslog server or to a third party log collection and analytics service, like Splunk. Third party services offer filters, alarms, search and other advanced features to adding context and value to the logs collected from systems.

This section will cover setting this configuration on an ESXi host.

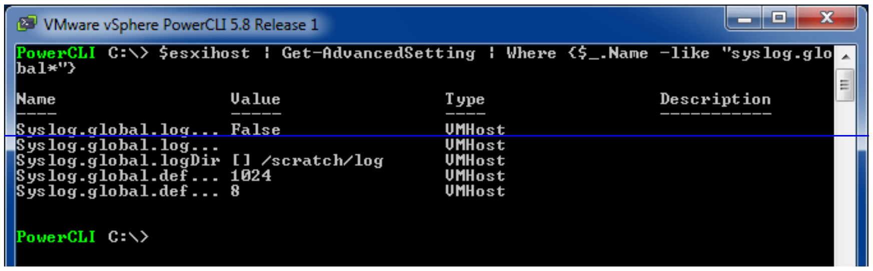
Getting ready

To work in this section, you will need to open a PowerCLI window, connect to an ESXi host, and populate the `$esxihost` variable with a `VMHost` object.

How to do it…

1. PowerCLI provides the `Get-AdvancedConfig` cmdlet which lets us peer into the advanced settings of the ESXi host. Even in the GUI, the syslog settings for an ESXi host are set within the Advanced Configuration setting. If you enumerate all of the advanced settings and then scope for items with `syslog.global`, you will see the settings you want to adjust to set centralize logging.

```
$esxihost | Get-AdvancedSetting | Where {$_.Name -like
"syslog.global*"}
```



```
PowerCLI C:\> $esxihost | Get-AdvancedSetting | Where {$_.Name -like "syslog.glo
bal*"}

Name                 Value                Type                 Description
----                 -----                ----                 -----------
Syslog.global.log... False                VMHost
Syslog.global.log...                      VMHost
Syslog.global.logDir [] /scratch/log      VMHost
Syslog.global.def... 1024                 VMHost
Syslog.global.def... 8                    VMHost


PowerCLI C:\>
```

~~**Insert Image 3724EN_01_12.png**~~

The two settings you want to adjust are `logDirUnique`, which sets a subdirectory for each host in the cluster, and `logDir` which sets the centralized location.

2. The `logDirUnique` setting is an easy one. First, you will need to scope down to retrieve just that setting and then pipe it into the `Set-AdvancedSetting` cmdlet.

```
$esxihost | Get-AdvancedSetting | Where {$_.Name -like
"Syslog.global.logDirUnique"} | Set-AdvancedSetting -value
$true -Confirm:$false
```

3. The second directory takes a bit more configuration. The `logDir` setting is a string that defines a storage path, so in our case, you need to figure out what datastore we're going to locate the syslog files onto. The VMFS datastore is identified as a bracketed name with a path following. In the earlier example, you created a datastore called iSCSIDatastore1 and you will now use it as our syslog global directory.

   ```
   $esxihost | Get-AdvancedSetting | Where {$_.Name -like
   "Syslog.global.logDirUnique"} | Set-AdvancedSetting -value
   "[iSCSIDatastore1] syslog" -Confirm:$false
   ```

Alternatively, if you want to direct all log files to a centralized syslog server, you may set this setting the `Syslog.global.logHost` value.

4. To set the syslog host value, you will use the same cmdlet used to set the previous values for syslog, except you will alter the advanced setting used in the `Where` statement. The value should be `Syslog.global.logHost` to locate the correct value to be set.

   ```
   $esxihost | Get-AdvancedSetting | Where {$_.Name -like
   "Syslog.global.logHost"} | Set-AdvancedSetting -value "
   tcp://syslogserver:514 " -Confirm:$false
   ```

How it works…

The vSphere Advanced Settings control the syslog functions. There are properties in the advanced settings that control how often and at what frequency to roll the log files and in this example, where to store the global syslog directory, and whether to make a unique subdirectory for this host's log files.

The `Get-AdvancedSetting` and `Set-AdvancedSetting` cmdlets expose and allow us to set these Advanced Settings from PowerCLI.

Setting the global log directory requires the administrator to choose a datastore and subdirectory on which to create these log files. The format of the path is set by using the bracketed datastore name and then a relative path inside of the datastore. This is a path definition that vSphere understands, but it is also specific to vSphere. It uses a Linux-like path definition but begins inside of the datastore location.

There's more…

In general, it's best to leave vSphere advanced settings with their default values unless instructed to make changes by VMware support. The vSphere advanced settings can alter the behavior of ESXi significantly and should be done with caution.

Joining an ESXi host to vCenter

Joining an ESXi host to vCenter is done from vCenter. The cmdlets for adding a host to a vCenter installation all require communication with vCenter to add the host. In this section, we'll connect to vCenter and add the host into inventory. All additional configuration to vCenter from PowerCLI will be covered in the next chapter.

Getting ready

Open a new PowerCLI window. This will ensure that no variables are populated and no open connection to an ESXi may be lingering.

How to do it…

In this example, you will connect to a vCenter Server instead of directly connecting to an ESXi host. Our vCenter server has a hostname of vcentersrv.domain.local.

1. To connect to vCenter, use the same cmdlet that you used in the "Connecting to an ESXi host or vCenter instance" recipe.

```
$vcenter = connect-viserver vcentersrv.domain.local
```

> The same certificate warning may be displayed and you may be prompted to login if your computer cannot single sign-in to the vCenter instance.

2. Once connected to vCenter, you can use the `Add-VMHost` to add the host into inventory.

```
Add-VMHost -Server $vcenter -Name esxsrv1.domain.local
-Location "Primary"
```

> For the purposes of this section, the Location is assumed to be a datacenter object already created in vCenter. In the next chapter, you'll see code how to create this datacenter object.

3. When prompted, enter the administrative account credentials for the ESXi to perform the join operation.
4. The host is now added to vCenter Server and may be administered by the server.

How it works…

Joining an ESXi to vCenter is a simple cmdlet to configure and complete. It simply links the ESXi into vCenter so that all additional configuration and control will be directed from the vCenter host.

At this point, connecting to the ESXi host will display a message in the GUI clients that shows its being managed by vCenter and that all changes should be made through vCenter. That is mostly the case from PowerCLI, too, but there may be additional times when configuration needs to be made directly against a host. One example would be changing multipathing settings for storage.

See also

- Creating a virtual datacenter in vCenter
- Creating a cluster and adding ESXi hosts into the cluster
- Setting cluster settings, including HA, DRS, FT and EVC

Creating a configuration script to set all properties uniformly

In this section, you are going to cover bringing all the cmdlets you have covered in this chapter together into a single script. This script will allow us to take an array of ESXi hosts identified by either their hostname or IP address and to run the full scripted configuration against them.

In many ways, this PowerCLI script will function much like a Host Profile in vSphere. Host Profiles are a configuration definition that can be created from an existing, configured host and applied on hosts to establish a desired configuration state. If hosts deviate from the configuration, the profile may be reapplied to remediate any undesired changes.

Unfortunately, Host Profiles are only available to customers with Enterprise Plus licensing. But this PowerCLI solution would work for any vSphere customer with Essentials, Essentials Plus, Standard, or Enterprise licensing.

Getting ready

For this last recipe of the chapter, you'll most likely want to open something like PowerShell ISE. PowerShell ISE provides you with additional tools to edit larger scripts, color code the cmdlets, and ensure no syntax errors. Alternatively, you may want a text editing tool such as NotePad, NoteTab Light, Sublime Text, or NotePad++.

How to do it…

1. First things first, let's begin with pseduocode/documentation of what you want to accomplish. In between each of these sections, you will insert the

code you have previously developed individually and put them into a full file.

```
# Script to mass configure ESXi hosts
# Step 1 - Store credentials for ESXi hosts
# Step 2 – Set a list of target ESXi hosts and IP settings
# Step 3 – Write a ForEach loop to iterate through hosts
# Step 4 – Connect to ESXi host
# Step 5 – Obtain a VMHost object to use for configuration
# Step 6 – Join the ESXi system to Active Directory
# Step 7 – Enable services on the ESXi host & set firewall
# Step 8 – Configure the network settings
# Step 9 – Configure NFS & iSCSI settings
# Step 10 - Join hosts to vCenter
# Step 11 - Rescan for storage changes
# Step 12 – Configure persistent syslog storage
```

2. Since you want this script to do as much without any manual intervention, you want to try and eliminate as many prompts as possible. Since you will be connecting to and executing commands on multiple ESXi hosts, you would normally get prompted to login each time you connect to a host. To avoid this, you can store the credentials in a variable and pass them to each `connect-viserver` cmdlet.

```
# Step 1 - Store credentials for ESXi hosts
$esxiCreds = Get-Credential
```

When you first covered connecting to ESXi servers from PowerCLI, you experienced the login box for the host. The `Get-Credentials` cmdlet causes the same action but returns a credentials object that can be reused. For now, you'll proceed with the stored credentials and you will use them at a later step.

3. You're going to use an array of hostnames to connect to individual ESXi hosts for configuration. To create the array, you set a variable and store a comma separated list of addresses to connect to. The addresses can either be hostnames or IP addresses. For this example, you will use IP addresses, but they could easily be fully qualified domain names.

```
# Step 2 - Set a list of target ESXi hosts and IP settings
$esxiTargets = "192.168.0.241","192.168.0.242",
"192.168.0.243", "192.168.0.244"
```

4. For the network configuration settings, you will need to setup some additional settings. Since each host has 3 additional vmkernel ports configured, you need to build a different address for each of these to be used in Step 8. To allow for this, you will create 3 additional variables containing the first 3 octets of the network for each vmkernel port.

```
$vMotionNetwork = "192.168.50."
$storageNetwork = "192.168.100."
$ftlogNetwork = "192.168.200."
```

5. The next step with this is to go back and pull in all of the code you have previously written in one form or another. For this, you will reuse the `ForEach` loop to execute the cmdlets on multiple ESXi hosts.

```
# Step 3 - Write a ForEach loop to iterate through hosts
ForEach ($hostname in $esxiTargets) {
```

6. The curly brace marks the beginning of the `ForEach` loop. You will close the loop with a right curly brace later in the script. Inside of the loop, you're going to include Steps 4 – 9 from the outline.
7. For the next step, you're going to use our stored credentials to connect to an ESXi host. Immediately after, you will store our VMHost object for use throughout the rest of the loop.

```
# Step 4 - Connect to ESXi host
$connectedHost = connect-viserver $hostname -Credential
$esxiCreds

# Step 5 – Obtain a VMHost object to use for configuration
$esxihost = Get-VMHost $hostname
```

8. For the next several steps, you're just going to pull code you have already developed. Since each step was covered in depth, you will just bring over the code.

```
# Step 6 – Join the ESXi system to Active Directory
$esxihost | Get-VMHostAuthentication |
Set-VMHostAuthentication -JoinDomain -Domain domain.local -
user username -password ***** -Confirm:$false

# Step 7 – Enable services on the ESXi host & set firewall
$esxihost | Get-VMHostService | where { $_.key -eq "TSM-
SSH" } | Set-VMHostService -Policy "On" -Confirm:$false

$esxihost | Get-VMHostService | where { $_.key -eq "TSM-
SSH" } | Start-VMHostService -Confirm:$false

# Step 8 – Configure the network settings
$esxihost | Get-VirtualPortGroup -Name "VM Network" |
Remove-VirtualPortGroup –Confirm:$false
```

9. For the network settings, you will need 3 additional IP addresses for the vMotion, Storage and FT Logging vmkernel ports. You will compute these addresses using the last octet of the service console IP. To do this, you will first retrieve the IP address of the host.

```
$esxihost_ipaddress = $esxihost | Get-VMHostNetworkAdapter
-name vmk0
```

10. Next, you will split the string based on the period between octets, then take the last octet of the IP address and store it as a variable. The IP is in a property called IP. To split that IP into an array, you use the `Split()` method, which is a built-in PowerShell method that transforms a string into an array by separating characters with the character passed into the method.

    For instance, you want to separate the string at the periods of the IP address, so you pass `"."` into the `Split()` method. Since the `Split()` turns it into an array, you can then reference the element you want to return - the fourth element, but remember arrays begin count at 0, so you return element 3 using square brackets.

    ```
    $lastOctet = $esxihost_ipaddress.IP.Split(".")[3]
    ```

> Because data is stored in Objects, objects have both properties and methods. Methods perform operations on the data of the object and properties contain the data of the object. In subsequent recipes throughout the book, you will look at and use other methods to gain more experience using built-in PowerShell functionality to manipulate data stored in objects.

11. Last step in building the address for this host in the ForEach loop, you need to concatenate the final octet with the network strings to build a full IP address.

    ```
    $vmotionIP = $vMotionNetwork + $lastOctet
    $storageIP = $storageNetwork + $lastOctet
    $ftlogIP = $ftlogNetwork + $lastOctet
    ```

12. Now that your unique IP addresses are created on the 3 additional networks, you may use them with the cmdlets you wrote in the "Setting network configuration" recipe.

    ```
    $esxihost | New-VMHostNetworkAdapter -PortGroup "VMotion
    Network" -VirtualSwitch vSwitch0 -IP $vmotionIP -SubnetMask
    255.255.255.0 -VMotionEnabled $true

    $esxihost | Get-VirtualPortGroup -Name "VMotion Network" |
    Set-VirtualPortGroup –VlanID 50

    # Create new virtual switch for Storage and FT Logging
    $esxihost | New-VirtualSwitch -Name vSwitch1 -Nic
    vmnic2,vmnic3

    # Create vmkernel ports for Storage and FT Logging
    $esxihost | New-VMHostNetworkAdapter -PortGroup "Storage
    Network" -VirtualSwitch vSwitch1 -IP $storageIP -SubnetMask
    255.255.255.0 -VsanTrafficEnabled $true
    ```

```
$esxihost | Get-VirtualPortGroup -Name "Storage Network" |
Set-VirtualPortGroup –VlanID 100

$esxihost | New-VMHostNetworkAdapter -PortGroup "FT Logging
Network" -VirtualSwitch vSwitch1 -IP $ftlogIP -SubnetMask
255.255.255.0 -FaultToleranceLoggingEnabled $true

$esxihost | Get-VirtualPortGroup -Name "FT Logging Network"
| Set-VirtualPortGroup –VlanID 200

# Create new Virtual Switch for Virtual Machines
$esxihost | New-VirtualSwitch -Name vSwitch2 -Nic
vmnic4,vmnic5

# Create Port Groups for Virtual Machines
New-VirtualPortGroup -Name "Infrastructure Network"
-VirtualSwitch vSwitch2 -VLanId 1 -Server $esxihost

New-VirtualPortGroup -Name "Application Network"
-VirtualSwitch vSwitch2 -VLanId 2 -Server $esxihost

# Step 9 – Configure NFS & iSCSI settings
# Connect NFS datastore
$esxihost | New-Datastore -Nfs -Name DataStoreName -Path
/data1/export -NfsHost nfsserver.domain.local

# Configure iSCSI Settings
$esxihost | Get-VMHostStorage | Set-VMHostStorage -
SoftwareIScsiEnabled $true$iSCSIhba = $esxihost | Get-
VMHostHba -Type iScsi

New-IScsiHbaTarget -IScsiHba $iSCSIhba -Address $target
-ChapType Required -ChapName vSphere -ChapPassword
Password1

$esxcli = Get-ESXCLI -VMHost $esxihost

$esxcli.iscsi.networkportal.add($iscsihba, $true,"vmk2")
```

13. The final part of the ESXi host configuration is closing the `ForEach` loop and disconnecting from this host so that you can connect to the next host.

```
Disconnect-VIServer -Server $connectedHost -Confirm:$false
}
```

At this point in the initial configuration, you would want to format your datastores on iSCSI or Fibre Channel arrays, but this is not really a repeatable set of steps. I would suggest one of the two things - either configure the datastore manually from PowerCLI or from the GUI and then come back and run the remainder of the script. Since the focus of this example to making a repeatable configuration script, the datastore formatting doesn't fit since it is a one-time command.

14. The next step is to take our hosts and connect them to vCenter. The easiest way to do this is to connect to vCenter and then use `Add-VMHost` to add them into inventory. While in the same `ForEach` loop to accomplish this, you can set central syslog and rescan the hosts for all storage changes.

    ```
    $vcenter = connect-viserver vcentersrv.domain.local
    $datacenter = Get-Datacenter "Primary"
    ```

> For the purpose of this script, you are going to assume that vCenter already has a datacenter created and named "Primary." You will use this location to place the ESXi host into vCenter.

15. Next, you will run through an additional `ForEach` loop to add the hosts and set their settings in vCenter.

    ```
    ForEach ($hostname in $esxTarget) {
    ```

16. Now, you are ready to add the host into vCenter, from the "Joining an ESXi host to vCenter" recipe.

    ```
    # Step 10 - Join hosts to vCenter
    Add-VMHost -Server $vcenter -Name $hostname l -Location
    $datacenter -Credential $esxiCreds
    ```

17. After adding the host to vCenter, you want to store a `VMHost` object pointing to the host to use with later cmdlets in this loop.

    ```
    $esxihost = Get-VMHost $hostname
    ```

18. For the next few steps, you will pull the host settings related to rescanning for datastores and setting the syslog settings.

    ```
    # Step 11 - Rescan for storage changes
    $esxihost | Get-VMHostStorage -RescanAllHBA -RescanVmfs

    # Step 12 – Configure persistent syslog storage
    $esxihost | Get-AdvancedSetting | Where {$_.Name -like
    "Syslog.global.logDirUnique"} | Set-AdvancedSetting -value
    $true -Confirm:$false

    $esxhost | Get-AdvancedSetting | Where {$_.Name -like
    "logDir"} | Set-AdvancedSetting -value "[iSCSIDatastore1]
    syslog" -Confirm:$false
    ```

19. Finally, you close the loop with a right curly brace.

    ```
    }
    ```

With `connect-viserver`, you may have to login a second time in the script with different credentials to vCenter versus to individual hosts. Afterwards, the hosts should be populated into vCenter.

Finally, your settings and desired state should be fully transferred to the ESXi host by the script.

How it works…

In this example, you wrap up all the code you have developed throughout the chapter. You bring together the pieces of code that achieve specific tasks into a full scripted configuration that you can apply towards a number of ESXi hosts. The script gives us repeatability, so when you need to extend the cluster with a new ESXi or when you rebuild the host because you've replace or upgraded the hardware, you can run this script against it and be back to the same working condition as before replacement.

The basis of the script is a `ForEach` loop. Because you define the ESXi hosts in an array, you can connect to each of them and run all the commands and then move the next entry in the array. The script also suppresses confirmation dialogs so that it can continue to issue cmdlets against the host. You also stored the login credentials, meaning that you only have to login once and the script will use the same credentials to connect and configure all the hosts in the defined array.

See also...

- VMware vSphere Host Profiles
  http://www.vmware.com/products/vsphere/features/host-profiles