PowerCLI Cookbook

PowerCLI allows faster administration by executing tasks on groups of objects in the virtual environment and is flexible enough to allow complex, scripted routines to solve complex problems.

PowerCLI Cookbook illustrates the ease of performing repetitive tasks using native PowerCLI commands to speed up administration. This book teaches you how to create custom functions and modules to solve specific problems and deploy these solutions to operators. It covers all vSphere administration areas including host, cluster, and virtual machine management utilizing PowerCLI.

Finally, this book will enable administrators to execute scripts that will open new possibilities for automation and also enable them to manage VM workloads effectively.

Who this book is written for
If you are a virtualization professional who wants to unleash the power of automation and combat the complexity of sprawling virtual environments, this book is ideal for you. This book will enhance your skills of administering VMware vSphere and vCloud Director with PowerCLI.

What you will learn from this book
- Configure and administer VMware ESXi hosts, clusters, and vCenter
- Manage virtual machine configurations and operations
- Administer datastores, datastore clusters, and virtual machine snapshots
- Create and configure virtual networking in a vSphere environment
- Manage cluster features and affinity groups
- Make custom reports and notifications from vSphere
- Direct operations in vCloud Director

Over 75 step-by-step recipes to put PowerCLI into action for efficient administration of your virtual environment

Philip Sellers

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Configuring the Basic Settings of an ESXi Host with PowerCLI'
- A synopsis of the book’s content
- More information on **PowerCLI Cookbook**

About the Author

Philip Sellers is an IT enthusiast residing in Myrtle Beach, South Carolina. Philip has more than 16 years of industry experience in consulting and systems administration. Currently, he is a senior-level systems administrator with Horry Telephone Cooperative, America’s largest telecommunications cooperative. Philip focuses on Microsoft and VMware software solutions along with server and SAN infrastructure. He spends a lot of time wrangling unwieldy systems and tries to tame as much as he can with automation using tools such as PowerCLI.

Philip has a bachelor of science degree in interdisciplinary studies with a minor in computer science from Coastal Carolina University and holds certifications as a VMware VCAP5-DCA and VCP5-DCV and is a Microsoft Certified IT Professional.

He maintains a technology blog at [http://www.techazine.com](http://www.techazine.com) that provides explanations and reviews of enterprise IT solutions, and he is a leader with the Myrtle Beach VMware Users Group. You may also follow him on Twitter [@pbsellers](https://twitter.com/pbsellers).

Philip is married to his college sweetheart, Jennifer, and has two kids who keep him busy when he's not working.
PowerCLI Cookbook

VMware PowerCLI offers a compelling command-line alternative to the point-and-click administration of vSphere and vCloud Director. As virtualization has become mainstream and deployments begin to sprawl, the simple commands of PowerCLI allows faster administration by executing tasks on groups of objects in the virtual environment.

Since PowerCLI follows a very logical pattern, it can be quickly adopted, making it the first choice for many vSphere administrators. However, with simplicity, it also combines extensibility to allow users to build their own functions and modules to solve specific problems not addressed by out-of-box functionalities.

What This Book Covers

Chapter 1, Configuring the Basic Settings of an ESXi Host with PowerCLI, covers the configuration of a fresh installation of VMware ESXi on a host system.

Chapter 2, Configuring vCenter and Computing Clusters, teaches you how to perform a basic vCenter configuration and add multiple ESXi hosts into a cluster with vSphere features, such as Dynamic Resource Scheduler (DRS) and High Availability (HA).

Chapter 3, Managing Virtual Machines, provides you with many of the common tasks needed to manage virtual machines from PowerCLI, including deploying and cloning virtual machines, changing hardware settings on virtual machines, and reloading inaccessible virtual machines in vCenter.

Chapter 4, Working with Datastores and Datastore Clusters, introduces the PowerCLI cmdlets needed to create and manage datastores and datastore clusters for individual ESXi hosts or vSphere clusters.

Chapter 5, Creating and Managing Snapshots, covers cmdlets and routines to work with snapshots on virtual machines, how to manage and report on snapshots before they become problems, and uses the topic to teach you how to write your own function in PowerCLI that can be reused easily. This chapter also covers how to take your code and schedule it to run with defined triggers using native PowerShell commands.

Chapter 6, Managing Resource Pools, Reservations, and Limits for Virtual Machines, covers the topic of creating and managing resource pools and their associated settings that include reservations and limits both at a pool and virtual machine level.

Chapter 7, Creating Custom Reports and Notifications for vSphere, teaches you how to use many of the native PowerShell features for reporting and leveraging those with PowerCLI cmdlets to create custom reports and notifications.
Chapter 8, Performing ESXCLI and in-guest Commands from PowerCLI, works with the advanced topics of using ESXCLI, an alternative command-line administration tool, from within PowerCLI to access and manage settings that are not natively accessible from PowerCLI. This chapter also covers some of the basics of performing in-guest commands invoked from PowerCLI.

Chapter 9, Managing DRS and Affinity Groups PowerCLI, is built on everything covered in the previous chapters to discuss managing the vSphere DRS features from PowerCLI by building your own functions and modules to alter the group memberships of DRS groups and keep the membership updated per defined rules.

Chapter 10, Working with vCloud Director from PowerCLI, changes gears and covers managing vCloud Director and vCloud deployments in multi-tenanted environments.

Appendix, Setting up and Configuring vCloud Director, covers certain installation tips and techniques.
In this chapter, you will cover the following recipes:

- Connecting to an ESXi host or a 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 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 with an IP address configured on it, and you will configure the server continually from there. This chapter will take an administrator through the basic configuration tasks needed to perform the initial configuration, join them to vCenter, and get it into an operational state. At the end of this chapter, all of these steps will build into a scripted configuration that can be executed against new hosts in the future.
Connecting to an ESXi host or a vCenter instance

To begin working with PowerCLI, you must first have PowerShell installed and available on the system on which you will run PowerCLI. PowerShell is a 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 http://www.vmware.com. The specific link is listed in the See also section of this recipe.

Once you have installed PowerCLI, you will need to build an ESXi host 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 administer VMware environments in PowerCLI is to connect to the ESXi host or to a vCenter server. In this chapter, you will focus on how to configure a single ESXi host. In the next chapter, you will focus on how to configure a vCenter Server and a 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 cmdlet references throughout the book are available to you. Each version of PowerCLI builds additional native cmdlets and functionalities. 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 5.5 Release 1 and have also been tested with VMware PowerCLI 5.8 Release 1, which accompanies vCloud Suite 5.8. The recipes also work on vSphere and PowerCLI 6.0 with any differences noted.

How to do it...

1. At the PowerCLI prompt, you will execute the Connect-VIServer cmdlet as follows:  
   Connect-viserver <hostname or IP>

2. When executed, the code will attempt to perform a 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, which is displayed as follows, and you should log in with the root username and the password you specified during your ESXi installation:
3. Once you successfully log in to 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 to, as shown in the following example:

<table>
<thead>
<tr>
<th>Name</th>
<th>Port</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.0.241</td>
<td>443</td>
<td>root</td>
</tr>
</tbody>
</table>

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. This cmdlet initiates a connection to the vCenter or ESXi web services to allow additional commands to be passed to the server and be executed.

The `Connect-VIServer` cmdlet requires only the name of the host to which you want to connect. There are additional parameters that you might 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 the `PSCredentials` objects, type `get-help about_server_authentication`. 
Configuring the Basic Settings of an ESXi Host with PowerCLI

Once you execute the cmdlet, a warning will be displayed in yellow, similar to the following one:

```
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 an SSL certificate on ESXi hosts will be covered later in the book, but the warning can be ignored this time. The cmdlet will continue to execute even though the warning is displayed.
```

You can also prevent the invalid certificate errors by running the following PowerCLI cmdlet that 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 log in against the ESXi host or vCenter server if your account has access to the server. If your account does not have access to the server that it is attempting to connect to, a login box will be presented like our example, shown in the preceding screenshot, in this recipe.

See also

- The Joining an ESXi host into Active Directory recipe
- The Setting permissions on vCenter objects recipe in Chapter 2, Configuring vCenter and Computing Clusters
- VMware PowerCLI Documentation Center and Installation Download https://www.vmware.com/support/developer/PowerCLI/
Getting the VMware host object

Cmdlets become available to manage a host after we connect to that host to manage it. The first concept that you will need to become aware of are PowerShell objects. Objects are defined as data obtained from commands that run in PowerShell and PowerCLI. To perform configuration on an ESXi host, the commands that you run will need a host object, which is 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 a vCenter instance.

How to do it...

1. PowerCLI is straightforward. To retrieve an ESXi host object, just run the following command line:
   ```
   Get-VMHost
   ```

2. After running the `Get-VMHost` cmdlet, an object that contains one or more ESXi hosts is returned. You are connecting to a single ESXi host in this example and running `Get-VMHost` that returns the host object with a single host. If you were connecting against a vCenter instance, `Get-VMHost` (with no other arguments) would return an object that contains all of the hosts managed by vCenter. When running against vCenter, you can 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 calling 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 using `$` followed by a name. The following is an example of our ESXi host:
   ```
   $esxihost = Get-VMHost
   ```

How it works...

To learn 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 as follows:

```
$esxihost | Get-Member
```
PowerCLI is an extension of PowerShell that is used specifically for VMware product management. PowerShell is an object-based language that 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 can 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. These methods are used to manipulate the data in the object. The output of the preceding command is shown in the following screenshot:

![PowerCLI Output Screenshot](image)

You can call a method by using a dot notation (.) and by calling the method name followed by parenthesis, such as in the following example:

```
$esxihost.ConnectionState.ToString()
```
In the preceding example, the State property is an object inside 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 the configuration.

There's more...

Get-VMHost has other applications other than 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 might want to find hosts that have been assigned a particular tag in vSphere using the -Tag parameter or you might 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 of the hosts that have a particular datastore connected.

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

See also

- The Setting up folders to organize objects in vCenter recipe in Chapter 2, Configuring vCenter and Computing Clusters
- The Creating basic reports of VM properties using VMware Tools and PowerCLI recipe in Chapter 3, Managing Virtual Machines

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 to the host without entering the credentials for administrators. Active Directory is a Windows implementation of Lightweight Directory Access Protocol (LDAP). It contains accounts for users, computers, and groups. It 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. However, 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 can skip this recipe and move on to other topics of host configuration without any impact to future recipes.
Configuring the Basic Settings of an ESXi Host with PowerCLI

Getting ready

PowerCLI has `Get-VMHostAuthentication` and `Set-VMHostAuthentication`, two cmdlets to deal with host authentication. To get ready to set up 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 `Get-VMHost` to either populate a variable or to pipe an object to the next object. The first step is to obtain a VMHost object for our target ESXi host. This can be done using the following command line:
   ```powershell
   $esxihost = Get-VMHost 192.168.0.241
   ```
2. Once you have your VMHost object, you can look at setting up the authentication. The `Set-VMHostAuthentication` cmdlet needs to be executed. The cmdlet requires several parameters to join an ESXi host to the domain. The syntax needed is displayed as follows:
   ```powershell
   $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 your answer is Y, the cmdlet will continue and execute the operation as follows:
   ```powershell
   Performing operation?
   Joining VMHost '192.168.0.241' to Windows Domain 'domain.local'.
   (default is "Y"):
   ```
   ```powershell
   Domain         DomainMembershipStatus    TrustedDomains
   ------         ----------------------    --------------
   DOMAIN.LOCAL  Ok
   ```

How it works...

One of the first things you will 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 `Set`? It would seem that you can simply pipe the VMHost object into cmdlet to specify your target host and the cmdlet will execute the function. But as you try that, using the following command line, PowerCLI displays an error, as shown in the following screenshot:
Chapter 1

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

In this case, the cmdlet looks 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 as follows, it will prompt you for a `VMHostAuthentication` object and wait for an input:

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

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

The `Get-Help` cmdlet for `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 piped input, you also learned that you can debug and learn what input the cmdlet is expecting and 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 this need.
In addition to setting up an ESXi host to accept Active Directory authentication, PowerCLI also provides a number of cmdlets to add local users, groups, and permissions inside 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 a group.

See also

- The Setting permissions on vCenter objects recipe in Chapter 2, Configuring vCenter and Computing Clusters

**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 might want to enable SSH on the host. However, since VMware does not recommend enabling SSH and will display a warning. You can set an advanced setting to disable this warning.

**Getting ready**

To begin with, 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 named `$esxihost`.

**How to do it...**

1. The first step 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 as follows:

   `$esxihost | Get-VMHostService`
2. The output of the preceding cmdlet will display a list of the available services on the ESXi host along with its policy (whether it is set on or off by default) if it’s 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 will use a PowerShell `where` clause as follows:

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

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** as follows:

   ```powershell
   $esxihost | Get-VMHostService | where { $_.key -eq "TSM-SSH" } | Set-VMHostService -Policy "On"
   ```
Configuring the Basic Settings of an ESXi Host with PowerCLI

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).

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

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:

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

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

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

How it works...

For configuring host services, the native cmdlets follow the expected pattern of `Get` and `Set` functionality in PowerCLI. `Get-VMHostService` expects a `VMHost` object as the input which 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. The `Start` and `Stop` cmdlets 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 the `HostService` objects as inputs, just like the `Set-VMHostService` cmdlet.
In the specific use case of the SSH Server service, it causes a warning to be displayed to the client. To disable this warning from being displayed, you can use an advanced setting named UserVars.SuppressShellWarning. While this is not recommended for production systems, there are plenty of use cases where SSH is needed and is helpful in lab environments, where you might 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 might need to enable SSH in order to tail a log file or to install a custom module. In these cases, starting SSH in bulk might be handy. To do this, you take the preceding code and wrap it in the loop. An example of a connection to a vCenter host, a variable with multiple VMHost objects returned, and a loop to step through and start SSH on each is shown as follows:

```powershell
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 will no longer be running and you can easily change the preceding code 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 to configure 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 that is connected to a VMware Standard-vSwitch. The vSwitch has two port groups created: one labeled Management Network for management traffic and the other is labeled VM Network. The Management Network is a vmkernel port with the IP defined on the console attached to it.
In this example, our host contains six 10 Gigabit NICs that will connect the host to 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.

Best practices of vSphere networking are far beyond the scope of this book. The network layout shown in the preceding diagram 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 with, 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 you the port groups associated with that vSwitch. To do this, review the output of the 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’s not the best practice to have Virtual Machine traffic on the management ports, and this default port group is not a part of the design you outlined for this configuration:

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

3. The preceding command combines the Get-VirtualPortGroup and Remove-VirtualPortGroup cmdlets to change the configuration. When executed, you will receive either a confirmation or an 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 this is outside the scope of this book, there are many different ideas for the 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's 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 will 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 the IP information. You will also pass in a parameter to specify that this vmkernel port should be used for VMotion as follows:

   $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, the traffic will be separated using active and standby links on each port group. This is done by changing the **NIC Teaming Policy** with the `Set-NicTeamingPolicy` cmdlet. You can see in the following two commands that the active and standby NIC assignments are opposite between the two port groups:

```powershell
$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's on the wrong VLAN. Our vMotion traffic is on a different VLAN, so you need to set this on the port group as follows:

```powershell
$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 shown in our diagram. To confirm that the physical NICs exist, you can run the following cmdlet:

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

The `Get-VMHostNetworkAdapter` cmdlet displays all of the vmkernel ports along with all of 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's currently an island with no connectivity for Management or virtual servers:

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

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

```powershell
$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
```

```powershell
$esxihost | Get-VirtualPortGroup -Name "FT Logging Network" | Set-VirtualPortGroup –VlanID 200
```
10. Again, you want to make sure that our Storage Traffic and Fault Tolerance traffic don't end up competing for bandwidth. Therefore, 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 of 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 as follows:

```
$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. However, in this case, 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 will 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 you 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. As 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 big time savings and less potential for manual error when compared to manually clicking on each through the GUIs to configure the new port group on each host in the cluster.
Configuring the Basic Settings of an ESXi Host with PowerCLI

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 and the Storage and Fault Tolerance traffic separated so they will not cause the performance of one another to be degraded.

**There's more...**

In this recipe, you focused 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**


**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 that alleviate some of the management of datastores. However, 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 might need to provision: NFS, iSCSI, and Fibre Channel. For this example, you will focus on iSCSI and NFS, and you 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 that you have the VMHost object stored in a variable named `$esxihost`, covered in the Getting the VMware host object section.
Chapter 1

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 of 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, as follows:

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

2. With this, 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 Fibre Channel. We will focus on iSCSI in this example and I will make a note of 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, as follows:

   $esxihost | Get-VMHostStorage

5. By default, there isn't a software iSCSI target that is created. To create this, you need to expand on the previous cmdlet and set this value to `true`, as follows:

   $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 can be omitted. However, it’s a bad practice to deploy a production storage array without authentication.
7. The final step of the initial iSCSI 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. As you will see, there are other vmkernel ports listed. In my case, vmk0; you can remove them with a simple `remove()` method, as follows:

```
$esxcli.iscsi.networkportal.remove($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 in the same way after initial configuration. 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 idea so that your system recognizes all of the 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 might 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 purpose of this example, we will assume that you have a disk identified by the iSCSIDisk model 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:

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

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

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

13. By storing the LUN in a variable, we 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:

```powershell
$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 of 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 will work in the same way with backend storage LUNs being presented to the host for consumption.

**See also**

- The Creating and managing datastore clusters and the Performing Storage vMotion recipes in Chapter 4, Working with Datastores and Datastore Clusters
Configuring syslog settings on an ESXi host

Booting your ESXi from SD or USB flash storage is a common scenario. However, when booting from SD and USB, ESXi does not use that storage for logging. Instead, it keeps the logs in memory, which is nonpersistent. Now that you have established a 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 you to troubleshoot. Even hosts booting from a local spinning disk might want to redirect their syslog onto a shared SAN drive so that it's accessible from another hosts if one of the hosts fails.

Another common use in enterprises is a centralized syslog server or a third-party log collection and analytics service, such as Splunk. Third-party services offer filters, alarms, search, and other advanced features to add 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 that 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 output](image)

   The two settings you want to adjust are: logDirUnique that sets a subdirectory for each host in the cluster, and logDir that 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 which datastore we're going to locate the syslog files onto. The VMFS datastore is identified as a bracketed name, which is followed by a path name. 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 can 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 that 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:

```
```

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 a 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 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 it begins inside the datastore location.
Configuring the Basic Settings of an ESXi Host with PowerCLI

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. 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 is 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 the hostname: vcentersrv.domain.local.

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

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

   The same certificate warning might be displayed and you might be prompted to log in 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:

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

   For the purpose of this section, the value of -Location is assumed to be a datacenter object already created in vCenter. In the next chapter, you'll see code on 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 can 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 of the 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 it's being managed by vCenter and that all changes should be made through vCenter. That is mostly the case from PowerCLI too, but there might be additional times when configuration needs to be made directly against a host. One example would be to change multipathing settings for storage.

**See also**

- The *Creating a virtual datacenter in vCenter, Creating a cluster and adding ESXi hosts, and Setting cluster advanced features, including HA, DRS, and EVC recipes* in *Chapter 2, Configuring vCenter and Computing Clusters*

**Creating a configuration script to set all properties uniformly**

In this section, you are going to cover all of the cmdlets that have been covered in this chapter by bringing them 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 might be reapplied to remediate any undesired changes.

Unfortunately, Host Profiles are only available to customers with Enterprise Plus licensing. However, this PowerCLI solution will 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 such as PowerShell ISE. PowerShell ISE provides you with additional tools to edit larger scripts, color code the cmdlets, and ensure that there are no syntax errors. Alternatively, you might 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 pseudocode/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:

```powershell
# 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 will 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.

```powershell
# 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 in 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 can easily be fully qualified domain names:

```bash
# 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 set up some additional settings. Since each host has three additional vmkernel ports configured, you need to build a different address for each of these to be used in Step 8. To allow this, you will create three additional variables that contain the first three octets of the network for each vmkernel port:

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

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

```bash
# 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 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 this, you will store our VMHost object for use throughout the rest of the loop:

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

```bash
# 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:

```bash
# Step 6 - Join the ESXi system to Active Directory
$esxihost | Get-VMHostAuthentication |
```
Configuring the Basic Settings of an ESXi Host with PowerCLI

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 three 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 will 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() method turns it into an array, you can then reference the element you want to return – the fourth element. However, remember arrays begin count at 0, so you will 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 this 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. The last step to build the address for this host in the ForEach loop is 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 three additional networks, you can use them with the cmdlets you wrote in the Setting network configuration recipe.

```powershell
$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 $connectedhost

New-VirtualPortGroup -Name "Application Network" -VirtualSwitch vSwitch2 -VlanID 2 -Server $connectedhost

# Step 9 – Configure NFS & iSCSI settings
# Connect NFS datastore
Configuring the Basic Settings of an ESXi Host with PowerCLI

```powershell
$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 then disconnecting from this host so that you can connect to the next host:

```powershell
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 configure it from the GUI and then come back and run the remainder of the script. Since the focus of this example is to make a repeatable configuration script, the datastore formatting doesn't fit since it is a command used just one time.

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:

```powershell
$vccenter = 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:

```powershell
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 -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

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

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

```
}
```

With `connect-viserver`, you might have to log in a second time in the script with different credentials to vCenter versus 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 of the code you have developed throughout the chapter. You bring together the pieces of code that achieve specific tasks into a fully scripted configuration that you can apply toward 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 replaced or upgraded the hardware, you can run this script against it and be back to the same working condition as before replacement.
Configuring the Basic Settings of an ESXi Host with PowerCLI

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 of 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, which means that you only have to log in once and the script will use the same credentials to connect and configure all of the hosts in the defined array.

See also

- VMware vSphere Host Profiles at http://www.vmware.com/products/vsphere/features/host-profiles
Where to buy this book

You can buy PowerCLI Cookbook from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.