VMware vSphere PowerCLI User's Guide

vSphere PowerCLI 5.1 Release 1

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see http://www.vmware.com/support/pubs.

EN-000896-00

vmware®

You can find the most up-to-date technical documentation on the VMware Web site at:

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to:

docfeedback@vmware.com

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

VMware vSphere PowerCLI User's Guide

The *VMware vSphere PowerCLI User's Guide* provides information about installing and using the VMware vSphere PowerCLI cmdlets (pronounced "commandlets") for managing, monitoring, automating, and handling lifecycle operations for VMware® vSphere and vCloud Director components.

To help you start with vSphere PowerCLI, this documentation includes descriptions of specific vSphere PowerCLI concepts and features. In addition, this documentation provides a set of usage examples and sample scripts.

Intended Audience

This book is intended for anyone who needs to install and use vSphere PowerCLI. This documentation is written for administrators and developers who are familiar with virtual machine technology and Windows PowerShell:

- Basic administrators can use cmdlets included in vSphere PowerCLI to manage their vSphere and vCloud Director infrastructure from the command line.
- Advanced administrators can develop PowerShell scripts that can be reused by other administrators or integrated into other applications.
- Developers can use vSphere PowerCLI views to create .NET applications for managing vSphere objects.

Introduction to VMware vSphere PowerCLI 1

VMware vSphere PowerCLI contains snapins of cmdlets based on Microsoft PowerShell for automating vSphere and vCloud Director administration. It provides C# and PowerShell interfaces to VMware vSphere and vCloud APIs.

- Microsoft PowerShell Basics on page 9

  vSphere PowerCLI is based on Microsoft PowerShell and uses the PowerShell basic syntax and concepts.

- vSphere PowerCLI Concepts on page 10

  vSphere PowerCLI cmdlets are created to automate VMware environments administration and to introduce some specific features in addition to the PowerShell concepts.

Microsoft PowerShell Basics

vSphere PowerCLI is based on Microsoft PowerShell and uses the PowerShell basic syntax and concepts.

Microsoft PowerShell is both a command-line and scripting environment, designed for Windows. It uses the .NET object model and provides administrators with system administration and automation capabilities. To work with PowerShell, you run commands, named cmdlets.

- PowerShell Command-Line Syntax on page 9

  PowerShell cmdlets use a consistent verb-noun structure, where the verb represents the action and the noun represents the object to operate on.

- PowerShell Pipelines on page 10

  A pipeline is a series of commands separated by the pipe operator `|`.

- PowerShell Wildcards on page 10

  PowerShell has a number of pattern-matching operators named wildcards that you can use to substitute one or more characters in a string, or substitute the complete string.

- PowerShell Common Parameters on page 10

  The Windows PowerShell engine retains a set of parameter names, referred to as common parameters. All PowerShell cmdlets, including the vSphere PowerCLI cmdlets, support them.

PowerShell Command-Line Syntax

PowerShell cmdlets use a consistent verb-noun structure, where the verb represents the action and the noun represents the object to operate on.

PowerShell cmdlets follow consistent naming patterns, ensuring that construction of a command is easy if you know the object that you want to work with.

All command categories take parameters and arguments. A parameter starts with a hyphen and is used to control the behavior of the command. An argument is a data value consumed by the command.

A simple PowerShell command has the following syntax:

```
command –parameter1 –parameter2 argument1, argument2
```

PowerShell Pipelines

A pipeline is a series of commands separated by the pipe operator `|`.

Each command in the pipeline receives an object from the previous command, performs some operation on it, and then passes it to the next command in the pipeline. Objects are output from the pipeline as soon as they become available.

PowerShell Wildcards

PowerShell has a number of pattern-matching operators named wildcards that you can use to substitute one or more characters in a string, or substitute the complete string.

All wildcard expressions can be used with the vSphere PowerCLI cmdlets. For example, you can view a list of all files with a `.txt` extension by running `dir *.txt`. In this case, the asterisk * operator matches any combination of characters.

With wildcard patterns you can indicate character ranges as well. For example, to view all files that start with the letter S or T and have a `.txt` extension, you can run `dir [st]*.txt`.

You can use the question mark ? wildcard to match any single character within a sequence of characters. For example, to view all `.txt` files with names that consist of `string` and one more character at the end, run `dir string?.txt`.

PowerShell Common Parameters

The Windows PowerShell engine retains a set of parameter names, referred to as common parameters. All PowerShell cmdlets, including the vSphere PowerCLI cmdlets, support them.

Some of the PowerShell common parameters are `Verbose`, `Debug`, `ErrorAction`, `ErrorVariable`, `OutVariable`, and `OutBuffer`. For a full list of the common parameters and more details on their usage, run `Get-Help about_CommonParameters`.

PowerShell offers two risk mitigation parameters: `WhatIf` and `Confirm`.

`WhatIf`	Displays the effects of a command without running it.
`Confirm`	Prompts for confirmation before running a command that stops a program or service, or deletes data.

vSphere PowerCLI Concepts

vSphere PowerCLI cmdlets are created to automate VMware environments administration and to introduce some specific features in addition to the PowerShell concepts.

- vSphere PowerCLI Components and Versioning on page 11

  VMware vSphere PowerCLI 5.1 Release 1 consists of two components that users can install and use according to their needs and environments.

- Interoperability Between the vSphere PowerCLI and vCloud Director PowerCLI Snapins on page 12

  With the `RelatedObject` parameter of vSphere PowerCLI cmdlets, you can retrieve vSphere inventory objects and vSphere PowerCLI view objects from cloud resources. This interoperability between the vSphere PowerCLI and vCloud Director PowerCLI snapins expands cloud administration, automation, reporting, and troubleshooting options for provider administrators.

- Selecting Objects in vSphere PowerCLI on page 14

  In vSphere PowerCLI, you can pass strings and wildcards to all parameters that take inventory objects, datastores, `OSCustomizationSpec` objects, and `VIServer` objects as arguments. This vSphere PowerCLI approach is named Object-by-Name (OBN) selection.

- Running vSphere PowerCLI Cmdlets Asynchronously on page 15

  By default, vSphere PowerCLI cmdlets return an output only after completion of the requested tasks. If you want a cmdlet to return to the command line immediately, without waiting for the tasks to complete, you can use the `RunAsync` parameter.

- Managing Default Server Connections on page 15

  By default, vSphere PowerCLI and vSphere PowerCLI cmdlets run on the vCenter Server systems or vCloud Director servers you are connected to, if no target servers can be determined from the provided parameters.

- Customization Specification Objects in vSphere PowerCLI on page 15

  vSphere PowerCLI provides two types of objects for customization specification: persistent and nonpersistent.

- vSphere PowerCLI Views Cmdlets on page 16

  The vSphere PowerCLI list of cmdlets includes the `Get-View` and `Get-VIObjectByVIView` cmdlets, which enable access to vSphere PowerCLI views from .NET.

- Using ESXCLI with vSphere PowerCLI on page 16

  vSphere PowerCLI provides you the capability to use ESXCLI through its console.

- vSphere PowerCLI Inventory Provider on page 16

  The Inventory Provider is designed to expose an unfiltered inventory view of the inventory items from a server.

- vSphere PowerCLI Datastore Provider on page 16

  The Datastore Provider is designed to provide access to the contents of one or more datastores.

- vSphere PowerCLI About Articles on page 17

  You can learn more about some vSphere PowerCLI concepts and features from the built-in help articles named about articles. You can access them through a running vSphere PowerCLI process.

vSphere PowerCLI Components and Versioning

VMware vSphere PowerCLI 5.1 Release 1 consists of two components that users can install and use according to their needs and environments.

- VMware vSphere PowerCLI 5.1 Release 1 is the core component of the vSphere PowerCLI package. It contains four snapins with cmdlets for managing vSphere 5.1 features:

VMware.VimAutomation.Core	VMware vSphere PowerCLI 5.1 Release 1 provides cmdlets for automated administration of the vSphere environment.
VMware.VimAutomation.License	VMware vSphere PowerCLI 5.1 Release 1 provides the `Get-LicenseDataManager` cmdlet for managing VMware License components.

VMware.ImageBuilder	VMware vSphere PowerCLI 5.1 Release 1 provides cmdlets for managing depots, image profiles, and VIBs.
VMware.DeployAutomation	VMware vSphere PowerCLI 5.1 Release 1 provides cmdlets that provide an interface to VMware Auto Deploy for provisioning physical hosts with ESXi software.

- VMware vCloud Director PowerCLI 1.5 is an optional component that you can install during the vSphere PowerCLI installation. It provides the following snapin:

VMware.VimAutomation.Cloud	VMware vCloud Director PowerCLI 1.5 provides cmdlets for automating vCloud Director 1.5.1 features.

Interoperability Between the vSphere PowerCLI and vCloud Director PowerCLI Snapins

With the `RelatedObject` parameter of vSphere PowerCLI cmdlets, you can retrieve vSphere inventory objects and vSphere PowerCLI view objects from cloud resources. This interoperability between the vSphere PowerCLI and vCloud Director PowerCLI snapins expands cloud administration, automation, reporting, and troubleshooting options for provider administrators.

Note To use the interoperability feature, you must install the vSphere PowerCLI and vCloud Director PowerCLI snapins, and connect both to a vCloud Director server and a vCenter Server system.

- Retrieving vSphere Inventory Objects from Cloud Resources on page 12

  Provider administrators can use the `RelatedObject` parameter of vSphere PowerCLI cmdlets to retrieve vSphere inventory objects from vCloud Director objects. Passing the retrieved objects to the cmdlets of the vSphere PowerCLI snapin extends administration options.

- Retrieving vSphere PowerCLI Views from vCloud Director PowerCLI Views on page 13

  Provider administrators with advanced knowledge and understanding of .NET Framework, vSphere PowerCLI, PowerShell scripting, and vSphere and vCloud APIs can retrieve vSphere PowerCLI views from vCloud Director PowerCLI views with the `Get-CIView` and `Get-View` cmdlets.

Retrieving vSphere Inventory Objects from Cloud Resources

Provider administrators can use the `RelatedObject` parameter of vSphere PowerCLI cmdlets to retrieve vSphere inventory objects from vCloud Director objects. Passing the retrieved objects to the cmdlets of the vSphere PowerCLI snapin extends administration options.

Important Use of the vSphere PowerCLI snapin to modify the configuration of objects that are managed by vCloud Director might result in unpredictable behavior of the cloud environment.

Table 1-1. List of Supported vSphere Inventory Objects You Can Retrieve from Cloud Objects

Cloud Object	Retrieved vSphere Inventory Object	Sample Script for Retrieving the vSphere Inventory Object
`ProviderVdc`	`Datastore`	`Get-ProviderVdc -Name 'MyProviderVdc' | Get-Datastore`
`ProviderVdc`	`ResourcePool`	`Get-ProviderVdc -Name 'MyProviderVdc' | Get-ResourcePool`
`CIVM`	`VirtualMachine`	`Get-CIVM -Name 'MyCloudVM' | Get-VM`

Table 1-1. List of Supported vSphere Inventory Objects You Can Retrieve from Cloud Objects (Continued)

Cloud Object	Retrieved vSphere Inventory Object	Sample Script for Retrieving the vSphere Inventory Object
NetworkPool	VirtualSwitchBase	Get-NetworkPool -Name 'MyNetworkPool' | Get-VirtualSwitch
NetworkPool	VirtualPortGroupBase	Get-NetworkPool -Name 'MyNetworkPool' | Get-VirtualPortGroup
ExternalNetwork	VirtualPortGroupBase	Get-ExternalNetwork -Name 'MyExternalNetwork' | Get-VirtualPortGroup
OrgNetwork	VirtualPortGroupBase	Get-OrgNetwork -Name 'MyOrgNetwork' | Get-VirtualPortGroup

Retrieving vSphere PowerCLI Views from vCloud Director PowerCLI Views

Provider administrators with advanced knowledge and understanding of .NET Framework, vSphere PowerCLI, PowerShell scripting, and vSphere and vCloud APIs can retrieve vSphere PowerCLI views from vCloud Director PowerCLI views with the `Get-CIView` and `Get-View` cmdlets.

With vSphere PowerCLI views, you can develop .NET applications for creating, customizing, or managing vSphere inventory objects.

IMPORTANT Use of the vSphere PowerCLI snapin to modify the configuration of objects that are managed by vCloud Director might result in unpredictable behavior of the cloud environment.

Table 1-2. List of Supported vSphere PowerCLI Views That You Can Retrieve from vCloud Director PowerCLI Views

vCloud Director PowerCLI View Object	Retrieved vSphere PowerCLI View Object	Sample Script for Retrieving vSphere PowerCLI View Objects from Cloud Resources
VMWExternalNetwork	DistributedVirtualPortGroup	Get-ExternalNetwork -Name 'MyExternalNetwork' | Get-CIView | Get-View
VLanPool	DistributedVirtualSwitch	Get-NetworkPool -Name 'MyVlanPool' | Get-CIView | Get-View
FencePool	DistributedVirtualSwitch	Get-NetworkPool -Name 'MyFencePool' | Get-CIView | Get-View
VimServer	ServiceInstance	$providerVdcView = Get-ProviderVdc -Name 'MyProviderVdc' | Get-CIView Get-CIView -Id $providerVdcView.VimServer[0].Href | Get-View
VMWProviderVdcResourcePool	ResourcePool	$providerVdcView = Get-ProviderVdc -Name 'MyProviderVdc' | Get-CIView $resourcePoolSet = $providerVdcView.GetResourcePools() $resourcePoolSet.VMWProviderVdcResourcePool | Get-View

Table 1-2. List of Supported vSphere PowerCLI Views That You Can Retrieve from vCloud Director PowerCLI Views (Continued)

vCloud Director PowerCLI View Object	Retrieved vSphere PowerCLI View Object	Sample Script for Retrieving vSphere PowerCLI View Objects from Cloud Resources
`Datastore`	`Datastore`	`Get-CIDatastore -Name 'MyDatastore' -ProviderVdc 'MyProviderVdc' | Get-CIView | Get-View`
`CIVM`	`VirtualMachine`	`Get-CIVM -Name 'MyVM' | Get-CIView | Get-View`

Selecting Objects in vSphere PowerCLI

In vSphere PowerCLI, you can pass strings and wildcards to all parameters that take inventory objects, datastores, `OSCustomizationSpec` objects, and `VIServer` objects as arguments. This vSphere PowerCLI approach is named Object-by-Name (OBN) selection.

Instead of assigning an object name to a cmdlet parameter, users can pass the object through a pipeline or a variable. For example, the following three commands are interchangeable:

- `Remove-VM -VM "Win XP SP2"`
- `Get-VM -Name "Win XP SP2" | Remove-VM`
- `Remove-VM -VM (Get-VM -Name "Win XP SP2")`

NOTE In vSphere PowerCLI, passing strings as pipeline input is not supported.

If you provide a non-existing object name, an OBN failure occurs. In such cases, vSphere PowerCLI generates a non-terminating error and runs the cmdlet ignoring the invalid name.

For more details about OBN, run `help about_OBN`.

Example: An OBN failure

This example illustrates the occurrence of an OBN failure.

```
Set-VM -VM “VM1”, “VM2”, “VM3” -Server $server1, $server2 -MemoryGB 4
```

If the *VM2* virtual machine does not exist on either of the selected servers, vSphere PowerCLI generates a non-terminating error and applies the command only on the *VM1* and *VM3* virtual machines.

Running vSphere PowerCLI Cmdlets Asynchronously

By default, vSphere PowerCLI cmdlets return an output only after completion of the requested tasks. If you want a cmdlet to return to the command line immediately, without waiting for the tasks to complete, you can use the `RunAsync` parameter.

When you use the `RunAsync` parameter, the cmdlet returns `Task` objects instead of its usual output. The `Status` property of a returned `Task` object contains a snapshot of the initial state of the task. This state is not updated automatically and has the values `Error`, `Queued`, `Running`, or `Success`. You can refresh a task state by retrieving the task object with the `Get-Task` cmdlet. If you want to observe the progress of a running task and wait for its completion before running other commands, use the `Wait-Task` cmdlet.

NOTE In vSphere PowerCLI, the `RunAsync` parameter affects only the invocation of a cmdlet and does not control whether the initiated tasks run consecutively or in parallel. For example, the `Remove-VM` cmdlet might remove the selected virtual machines simultaneously or consecutively depending on the internal design of vSphere PowerCLI. To make sure that tasks initiated by a cmdlet run consecutively, run the cmdlet in a loop, each time applying it to a single object.

Example: Running Remove-VM with and without the RunAsync parameter

```
Remove-VM $vmList
```

The command returns no output when all virtual machines stored in the *$vmList* variable are removed, irrespective of whether they are removed simultaneously.

```
Remove-VM $vmList -RunAsync
```

The command returns an output that consists of one or more `Task` objects immediately.

Managing Default Server Connections

By default, vSphere PowerCLI and vSphere PowerCLI cmdlets run on the vCenter Server systems or vCloud Director servers you are connected to, if no target servers can be determined from the provided parameters.

When you connect to a vCenter Server system by using `Connect-VIServer`, the server connection is stored in the *$DefaultVIServers* array variable. This variable contains all connected servers for the current vSphere PowerCLI session. To remove a server from the *$DefaultVIServers* variable, you can either use `Disconnect-VIServer` to close all active connections to this server, or modify the value of *$DefaultVIServers* manually.

When you connect to a vCloud Director system by using `Connect-CIServer`, the server connection is stored in the *$DefaultCIServers* array variable. This variable contains all connected servers for the current session. To remove a server from the *$DefaultCIServers* variable, you can either use `Disconnect-CIServer` to close all active connections to this server, or modify the value of *$DefaultCIServers* manually.

Customization Specification Objects in vSphere PowerCLI

vSphere PowerCLI provides two types of objects for customization specification: persistent and nonpersistent.

Persistent Customization

Persistent customization specification objects are stored on the vSphere server. All persistent customization specifications created by using vSphere Client or VMware vSphere PowerCLI 4.1 or later are encrypted. Encrypted customization specifications can be applied only by the server that has encrypted them.

Nonpersistent Customization

Nonpersistent customization specification objects exist only inside the current PowerShell process. Nonpersistent customization specification objects are not encrypted, but cloning them to a vSphere server encrypts them.

vSphere PowerCLI Views Cmdlets

The vSphere PowerCLI list of cmdlets includes the `Get-View` and `Get-VIObjectByVIView` cmdlets, which enable access to vSphere PowerCLI views from .NET.

To find more information about vSphere PowerCLI views, see "vSphere PowerCLI Views," on page 27.

Using the vSphere PowerCLI views cmdlets for low-level VMware vSphere management requires some knowledge of both PowerShell scripting and the VMware vSphere APIs.

Using ESXCLI with vSphere PowerCLI

vSphere PowerCLI provides you the capability to use ESXCLI through its console.

vSphere PowerCLI provides two approaches for working with ESXCLI:

- Through the `Get-ESXCli` cmdlet, which provides direct access to the ESXCLI namespaces, applications, and commands.
- Through .NET methods, which you use to create managed objects that correspond to specific ESXCLI applications. To access the ESXCLI, you can call methods on these managed objects.

> **NOTE** To call a method of an ESXCLI object, you must provide values for all parameters. If you want to omit a given parameter, pass *$null* as its argument.

vSphere PowerCLI Inventory Provider

The Inventory Provider is designed to expose an unfiltered inventory view of the inventory items from a server.

It enables navigation and file-style management of the VMware vSphere inventory. By creating a PowerShell drive based on a managed object (such as a datacenter), you can obtain a view of its contents and the relationships between the items. In addition, you can move, rename, or delete objects by running commands from the vSphere PowerCLI console.

When you connect to a server with `Connect-VIServer`, the cmdlet builds two default inventory drives: `vi` and `vis`. The `vi` inventory drive shows the inventory on the last connected server. The `vis` drive contains the inventory of all vSphere servers connected within the current vSphere PowerCLI session.

You can use the default inventory drives or create custom drives based on the default ones.

vSphere PowerCLI Datastore Provider

The Datastore Provider is designed to provide access to the contents of one or more datastores.

The items in a datastore are files that contain configuration, virtual disk, and the other data associated with a virtual machine.

When you connect to a server with `Connect-VIServer`, the cmdlet builds two default datastore drives: `vmstore` and `vmstores`. The `vmstore` drive provides a list of the datastores available on the vSphere server that you last connected to. The `vmstores` drive contains all datastores available on all vSphere servers that you connected to within the current vSphere PowerCLI session.

You can use the default inventory drives or create custom drives based on the default ones.

vSphere PowerCLI About Articles

You can learn more about some vSphere PowerCLI concepts and features from the built-in help articles named about articles. You can access them through a running vSphere PowerCLI process.

Running `Help About_*` lists all built-in Windows PowerShell and vSphere PowerCLI about articles.

Table 1-3. Accessing Built-In Help Articles for vSphere PowerCLI

Article Title	Command	Article Description
Handling Invalid Certificates	`Help About_Invalid_Certificates`	When you try to connect to a vCenter Server system or a vCloud Director server and the server cannot recognize any valid certificates, the Invalid Certificate prompt appears.
LicenseDataManager	`Help About_LicenseDataManager`	The LicenseDataManager component lets you to extend the vCenter Server inventory with license data.
Object-by-Name (OBN)	`Help About_OBN`	To help you save time and effort, vSphere PowerCLI lets you select objects by their names.
VMware vSphere PowerCLI Objects	`Help About_PowerCLI_Objects`	For their input and output, the vSphere PowerCLI cmdlets use a set of .NET types that reside in the `VMware.VimAutomation.ViCore.Types` namespace.
Using the RunAsync Parameter	`Help About_RunAsync`	When you set the RunAsync parameter, you indicate that you want to run the cmdlet asynchronously.
Authenticating with a vCenter Server System or a vCloud Server	`Help About_Server_Authentication`	To authenticate with vCenter Server and vCloud Director servers, you can provide a user name and password through the User and Password parameters, or a PSCredential object through the Credential parameter.
Unique Identifiers for PowerCLI Objects (UID)	`Help About_UID`	You can uniquely identify a PowerCLI object on a server or across multiple servers by providing its UID.
Datastore Provider (VimDatastore)	`Help About_VimDatastore`	The Datastore Provider (VimDatastore) provides filesystem-style view and access to the contents of datastores.

Installing VMware vSphere PowerCLI 2

VMware vSphere PowerCLI lets you perform managing, monitoring, automating, and handling lifecycle operations on vCenter Server and vCloud Director systems from the command line. You can install VMware vSphere PowerCLI components on all supported Windows operating systems.

After installing the package on your machine, you can connect to your vCenter Server or vCloud Director system by providing valid authentication credentials.

Supported Operating Systems

You can install vSphere PowerCLI only on supported Windows operating systems.

VMware vSphere PowerCLI 5.1 Release 1 is supported on the following operating systems:

- Windows 7 Service Pack 1 (32-bit and 64-bit)
- Windows Server 2008 R2 Service Pack 1
- Windows Server 2008 Service Pack 1 (32-bit)
- Windows XP Service Pack 3 (32-bit)

- Windows XP Service Pack 2 (32-bit and 64-bit)
- Windows Server 2003 R2 (32-bit and 64-bit)

NOTE The list of supported operating systems might not apply to some vSphere PowerCLI cmdlets for managing guest operating systems.

Supported VMware Environments

You can use the vSphere PowerCLI components to manage all supported vSphere and vCloud environments.

VMware vSphere PowerCLI 5.1 Release 1 is compatible with the following vSphere environments:

- vCenter Server 5.1
- VMware ESXi 5.1
- vCenter Server 5.0 Update 1
- VMware ESXi 5.0 Update 1
- vCenter Server 4.1 Update 2
- VMware ESX 4.1 Update 2
- VMware ESXi 4.1 Update 2
- vCenter Server 4.0 Update 4
- VMware ESX 4.0 Update 4
- VMware ESX 4.0i Update 4

VMware vCloud Director PowerCLI 1.5 is compatible with VMware vCloud Director 1.5.1.

Prerequisites for Installing and Running vSphere PowerCLI

Before installing and running vSphere PowerCLI, verify that you have installed the required software on the same machine.

If you want to work with VMware vSphere PowerCLI 5.1 Release 1, make sure that the following software is present on your system:

- Windows PowerShell 2.0
- A supported version of .NET Framework
  - .NET Framework 2.0 with Service Pack 2
  - .NET Framework 3.0 or .NET Framework 3.0 with Service Pack 1, or Service Pack 2
  - .NET Framework 3.5 or .NET Framework 3.5 with Service Pack 1

Install vSphere PowerCLI

vSphere PowerCLI lets you choose which components to install during the installation process. By selecting to install all available components, you perform a complete installation which requires the most disk space.

Prerequisites

- Before installing vSphere PowerCLI, see "Prerequisites for Installing and Running vSphere PowerCLI," on page 20.
- Verify that you have uninstalled VMware vSphere PowerCLI for Tenants from your system, if installed.

Procedure

1 Download the latest version of vSphere PowerCLI from the VMware Web site.

2 Navigate to the folder that contains the vSphere PowerCLI installer file you downloaded and double-click the executable file.

If the installation wizard detects an earlier version of vSphere PowerCLI on your system, it will attempt to upgrade your existing installation.

3 On the Welcome page, click **Next**.

4 On the VMware Patents page, click **Next**.

5 Accept the license agreement terms and click **Next**.

6 On the Custom Setup page, select the components that you want to install.

Option	Description
vSphere PowerCLI	Installs a set of cmdlets for managing vSphere features. This vSphere PowerCLI component is mandatory and selected by default.
vCloud Director PowerCLI	Installs a set of cmdlets for managing vCloud Director features.

7 (Optional) On the Custom Setup page, click **Change** to select a different location to install vSphere PowerCLI.

8 Click **Next**.

9 On the Ready to Install the Program page, click **Install** to proceed with the installation.

10 Click **Finish** to complete the installation process.

What to do next

"Set the Properties to Support Remote Signing," on page 21.

Set the Properties to Support Remote Signing

If you want to run scripts and load configuration files with vSphere PowerCLI, you must set the execution policy of Windows PowerShell to `RemoteSigned`.

For security reasons, Windows PowerShell supports an execution policy feature. It determines whether scripts are allowed to run and whether they must be digitally signed. By default, the execution policy is set to `Restricted`, which is the most secure policy. For more information about the execution policy and script digital signing in Windows PowerShell, run `Get-Help About_Signing`.

You can change the execution policy by using the `Set-ExecutionPolicy` cmdlet.

Procedure

1 From your Windows taskbar, select **Start > Programs > VMware > VMware vSphere PowerCLI**.

The vSphere PowerCLI console window opens.

2 In the vSphere PowerCLI console window, run `Set-ExecutionPolicy RemoteSigned`.

Uninstall vSphere PowerCLI

You can uninstall vSphere PowerCLI components from your Windows system by using the default uninstall tool of your operating system.

Prerequisites

Close the vSphere PowerCLI application.

Procedure

1 Select the default uninstall tool for your Windows system from Control Panel.

2 Select VMware vSphere PowerCLI from the list and click **Change**.

3 On the Program Maintenance page, select **Remove** and click **Next**.

4 Click **Remove**.

Configuring VMware vSphere PowerCLI

3

To extend and customize the features of VMware vSphere PowerCLI, you can configure the application settings for different users and user groups, modify the script configuration file of VMware vSphere PowerCLI, and add custom scripts.

This chapter includes the following topics:

Scoped Settings of vSphere PowerCLI

In vSphere PowerCLI you can set the scope of the settings to enhance security and personalize the configuration.

Configuring the Scope of the vSphere PowerCLI Settings

Scoped configuration enhances system security and prevents nonadministrator users from introducing global changes to the configuration of vSphere PowerCLI.

For greater control over the vSphere PowerCLI configuration, the `Set-PowerCLIConfiguration` cmdlet provides the `Scope` parameter.

Table 3-1. Valid Values for the Scope Parameter

Parameter Value	Description
`Session`	Configures settings for the current vSphere PowerCLI session and does not modify any vSphere PowerCLI configuration files on your system.
`User`	Configures settings for the current Windows user and modifies some vSphere PowerCLI configuration files on your system.
`AllUsers`	Configures settings for all users and modifies some vSphere PowerCLI configuration files on your system.

Priority of Settings Scopes in vSphere PowerCLI

vSphere PowerCLI loads the program configuration based on the scope that you select for each setting.

Table 3-2. Scope Impact on the Behavior of vSphere PowerCLI

Scope	Priority	Impact
`Session`	High	■ When started, vSphere PowerCLI tries to load settings with the `Session` scope first. ■ `Session` settings override `User` and `AllUsers` settings. ■ `Session` settings are valid for the current vSphere PowerCLI session only.
`User`	Medium	■ When vSphere PowerCLI cannot detect `Session` settngs, the program tries to load `User` settings from the vSphere PowerCLI configuration files. ■ `User` settings override `AllUsers` settings. ■ `User` settings are automatically detected from the vSphere PowerCLI configuration files.
`AllUsers`	Low	■ When vSphere PowerCLI cannot detect `Session` and `User` settings, the program loads `AllUsers` settings. ■ `AllUsers` settings do not override `Session` and `User` settings. ■ `AllUsers` settings are automatically detected from the vSphere PowerCLI configuration files.

vSphere PowerCLI Configuration Files

The copies of the `PowerCLI_settings.xml` file on your system contain `User` and `AllUsers` settings for vSphere PowerCLI.

Installing vSphere PowerCLI creates two copies of `PowerCLI_settings.xml` on your system. The version of your Windows operating system determines the location of the copies of `PowerCLI_settings.xml`.

Table 3-3. Location of the Copies of `PowerCLI_settings.xml`

Windows OS Version	Location	Description
Windows Vista and later	`%APPDATA%\VMWare\PowerCLI`	Contains settings for the current Windows user only.
	`%SYSTEMDRIVE %\ProgramData\VMware\PowerCLI`	Contains settings for all users.

Table 3-3. Location of the Copies of `PowerCLI_settings.xml` (Continued)

Windows OS Version	Location	Description
Earlier Windows versions	`%SYSTEMDRIVE%\Documents and Settings\[Username]\Application Data\VMware\PowerCLI`	Contains settings for the current Windows user only.
	`%SYSTEMDRIVE%\Documents and Settings\All Users\Application Data\VMware\PowerCLI`	Contains settings for all users.

Users with advanced knowledge and understanding of Windows PowerShell and VMware vSphere PowerCLI can manually modify the contents of `PowerCLI_settings.xml` to change vSphere PowerCLI settings. Modifying `PowerCLI_settings.xml` might require administrator privileges.

NOTE If you modify the contents of `PowerCLI_settings.xml` manually while vSphere PowerCLI is running, you need to restart vSphere PowerCLI for the changes to take effect.

Loading the Script Configuration File of vSphere PowerCLI

Starting vSphere PowerCLI automatically loads the script configuration file located in the `Scripts` folder in the vSphere PowerCLI installation directory.

Default Script Configuration File

The default script configuration file of vSphere PowerCLI is `Initialize-PowerCLIEnvironment.ps1`. Loading the file provides access to vSphere PowerCLI cmdlets aliases, like `Get-VC`, `Get-ESX`, and to other configuration settings.

`Initialize-PowerCLIEnvironment.ps1` is included in the installation package of vSphere PowerCLI.

Custom Script Configuration File

If you want to load custom vSphere PowerCLI settings automatically, you can create a script configuration file named `Initialize-PowerCLIEnvironment_Custom.ps1` in the `Scripts` folder. The application recognizes and loads the custom file after loading the default script configuration file.

Load the Script Configuration File in Other PowerShell Tools

If you want to work with vSphere PowerCLI from another PowerShell-based tool, such as PowerShell Plus or PowerGUI, you must load the default script configuration file manually.

Procedure

1 Run the PowerShell-based tool you have installed on your system.

2 At the command line, change the active directory to the folder where you have installed vSphere PowerCLI.

3 In the command line, type **`.\Scripts\Initialize-PowerCLIEnvironment.ps1`** and press Enter.

After the tool loads the default script configuration file, custom script configuration files, if any, load automatically.

Customizing vSphere PowerCLI with Script Configuration Files

You can edit and extend the configuration script of vSphere PowerCLI to set up the environment, set vSphere PowerCLI startup actions, or define cmdlets aliases.

Creating a Custom Script Configuration File

If you want to load custom vSphere PowerCLI settings automatically, you can create a script configuration file named `Initialize-PowerCLIEnvironment_Custom.ps1` in the `Scripts` folder. The application recognizes and loads the custom file after loading the default script configuration file.

NOTE Changing the contents of the default configuration file `Initialize-PowerCLIEnvironment.ps1` might cause vSphere PowerCLI to stop running properly.

Signing the Script Configuration File

When the execution policy of your system is set to `Remote Signed`, you do not need to sign the script configuration file after editing.

When the execution policy of your system is set to `All Signed`, you need to sign the script configuration file after editing. If you do not sign the file, vSphere PowerCLI will not load your modified configuration.

To learn more about setting the execution policy, see "Set the Properties to Support Remote Signing," on page 21.

Using Custom Scripts to Extend the Operating System Support for vSphere PowerCLI Cmdlets

Some vSphere PowerCLI features support only Windows 7, Windows Server 2008, Windows XP, Windows Server 2003, and Red Hat Enterprise Linux 5. To add support for other guest operating systems, you can use the scripts that are located in the `Scripts` folder of the vSphere PowerCLI installation directory or you can add your own custom scripts.

When adding new scripts, use the following file naming guidelines:

- Scripts that extend the operating system support for `Get-VMGuestNetworkInterface`, `Set-VMGuestNetworkInterface`, `Get-VMGuestRoute`, `New-VMGuestRoute`, and `Remove-VMGuestRoute` must follow the file-naming convention *`CmdletName_OSIdentifier`*, where *OSIdentifier* is the guest family or the guest ID as returned by `Get-VMGuest`, and *CmdletName* is the cmdlet name written without a hyphen, for example **`GetVMGuestRoute`**.
- Scripts that extend the operating system support for resizing the hard disk by using `Set-HardDisk` must follow the file naming convention *`GuestDiskExpansion_OSIdentifier`*, where *OSIdentifier* is the guest family or the guest ID (as returned by `Get-VMGuest`).

Using VMware vSphere PowerCLI Views from .NET

4

You can use .NET to access and use VMware vSphere PowerCLI views. Views are .NET objects that provide C# and PowerShell interface to vSphere APIs.

With vSphere PowerCLI views, you can develop .NET applications for creating, customizing, or managing vSphere inventory objects.

vSphere PowerCLI Views

vSphere PowerCLI views are .NET objects that correspond to server-side managed objects. Each operation defined on a server managed object has a corresponding view method.

A vSphere PowerCLI view has the following characteristics:

- It includes properties and methods that correspond to the properties and operations of the server-side managed objects.

- It is a static copy of a server-side managed object and is not automatically updated when the object on the server changes.
- It includes additional methods other than the operations offered in the server-side managed object.

Set Up the Environment to Develop vSphere PowerCLI .NET Applications

Before creating and running .NET applications for vSphere PowerCLI, you must set up your developmental environment.

Procedure

1 In Visual Studio 2005 .NET or later, create a new project or open an existing project.

2 Add a reference to the vSphere API .NET Library (`VMware.Vim.dll`) from the vSphere PowerCLI installation folder.

Now you can use `VimClient` and other `VMware.Vim` namespace classes to manage your vSphere inventory.

Updating the Properties of vSphere PowerCLI Views

The properties of a vSphere PowerCLI view contain information about the state of the server-side object at the time the view was created.

In a production environment, the state of managed objects on the server changes constantly. However, the property values of the objects are not updated automatically. You can synchronize the values of client-side views with the corresponding server-side objects by using the `UpdateViewData()` method.

Example: Using the UpdateViewData() method to refresh a view object data

The following code example refreshes the power state information of a virtual machine by using `UpdateViewData()` method.

```
using VMware.Vim;
using System.Collections.Specialized;
namespace Samples {
    public class Example2_2 {
        public void PowerOffVM() {
            VimClient client = new VimClient();

        ...

            IList<EntityViewBase> vmList =
    client.FindEntityViews(typeof(VirtualMachine), null, filter, null);
    // Power off the virtual machines.
    foreach (VirtualMachine vm in vmList) {
        // Refresh the state of each view.
        vm.UpdateViewData();
        if (vm.Runtime.PowerState == VirtualMachinePowerState.poweredOn) {
            vm.PowerOffVM();
            Console.WriteLine("Stopped virtual machine: {0}", vm.Name);
        } else {
            Console.WriteLine("Virtual machine {0} power state is: {1}", vm.Name,
vm.Runtime.PowerState);
     }
    }
...
```

Creating and Using Filters with VimClient.FindEntityView() or VimClient.FindEntityViews()

You can use filters to reduce large sets of output data by retrieving only the objects that correspond to the filter criteria that you provide. You can use vSphere PowerCLI views to define and use filters to select specific objects based on property values.

To apply a filter to the results of `VimClient.FindEntityView()` or `VimClient.FindEntityViews()`, you can supply an optional filter parameter. The value of the parameter is a `NameValueCollection` object containing one or more pairs of filter criteria. Each of the criteria consists of a property path and a match value. The match value can be either a string, or a regular expression object. If the match value is a string, the property value of the target objects must be exactly the same as the string.

Example: Filtering virtual machines by power state

The following commands retrieve all powered-off virtual machines.

```
NameValueCollection filter = new NameValueCollection();
filter.Add("Runtime.PowerState", "PoweredOff")
```

Example: Filtering objects by name

The following commands retrieve all virtual machines with names that start with `Test`.

```
NameValueCollection filter = new NameValueCollection();
filter.Add("name", "^Test");
```

Example: Filter for creating views of Windows virtual machines only

The following example uses `VimClient.FindEntityViews()` in combination with a filter. It retrieves a list of all Windows virtual machines in the virtual environment.

```
NameValueCollection filter = new NameValueCollection();
filter.Add("Config.GuestFullName", "Windows");

IList EntityViewBase vmList =
    client1.FindEntityViews(typeof(VirtualMachine), null, filter, null);

// Print VM names
foreach (VirtualMachine vm in vmList) {
    Console.WriteLine(vm.Name);
```

Example: Multiple criteria filter

This example uses a filter with multiple criteria. It retrieves all powered-on Windows virtual machines.

```
NameValueCollection filter = new NameValueCollection();
filter.Add("Runtime.PowerState", "PoweredOn");
filter.Add("Config.GuestFullName", "Windows");

IList EntityViewBase vmList =
    client1.FindEntityViews(typeof(VirtualMachine), null, filter, null);

// Print VM names
foreach (VirtualMachine vm in vmList) {
    Console.WriteLine(vm.Name);
```

Saving and Using Server Sessions with vSphere PowerCLI Views

With vSphere PowerCLI you can save your server session and restore it later. The `VimClient` class includes several methods for saving and restoring server sessions. This enables you to maintain sessions across applications.

Instead of storing passwords in applications, you can call the `LoadSession()` method with the name of the session file. The session file does not expose password information, and this enhances security.

Example: Saving a session to a file

This example illustrates how to save a server session to a file by calling `SaveSession()` with the file name.

```
VimClient client1 = new VimClient();
client1.Connect("https://hostname/sdk");
client1.Login("user", "pass");
client1.SaveSession("VimSession.txt");
```

Example: Loading a session from a file

This example illustrates how to load a server session in another application by calling `LoadSession()` with the name of the session file.

```
VimClient client2 = new VimClient();
client2.Connect("https://hostname/sdk");
client2.LoadSession("VimSession.txt");
client2.FindEntityView(typeof(VirtualMachine), null, null, null);
```

Handling Server Errors with vSphere PowerCLI Views

Error reporting helps you track and handle server errors. vCenter Server Web Services API server errors are reported as SOAP exceptions that contain a `SoapFault` object.

Using vSphere PowerCLI views provides additional error handling by translating the `SoapFault` object from the `SoapException.Detail` property into a `MethodFault` descendant object and throwing a `VimException` exception.

Example: Simple pattern for error handling

The following example illustrates a basic pattern implementation of error handling with vSphere PowerCLI views.

```
try {
            // call operations
        } catch (VimException ex) {
            if (ex.MethodFault is InvalidLogin) {
                 // Handle Invalid Login error
            } else {
                    // Handle other server errors
            }
        } catch (Exception e) {
            // Handle user code errors
        }
```

Sample Scripts for Managing vSphere with VMware vSphere PowerCLI

5

To help you get started with VMware vSphere PowerCLI, this documentation provides a set of sample scripts that illustrate basic and advanced tasks in vSphere administration.

- Manage Inventory Objects Through Inventory Drives on page 54

  You can use the vSphere PowerCLI Inventory Provider to browse, modify, and remove inventory objects from inventory drives.

- Browse the Default Datastore Drives on page 55

  You can use the vSphere PowerCLI Datastore Provider to browse the default datastore drives: `vmstore` and `vmstores`.

- Create a New Custom Datastore Drive on page 55

  You can use the vSphere PowerCLI Datastore Provider to create custom datastore drives.

- Manage Datastores Through Datastore Drives on page 56

  You can use the vSphere PowerCLI Datastore Provider to browse datastores from datastore drives.

- Modify the Timeout Setting for Web Tasks on page 56

  To avoid unexpected timeout, you can use `Set-PowerCLIConfiguration` to modify the vSphere PowerCLI settings for long-running Web tasks.

Connect to a vCenter Server system

To run vSphere PowerCLI cmdlets on vSphere and perform administration or monitoring tasks, you must establish a connection to an ESX/ESXi instance or a vCenter Server system.

You can have more than one connections to the same server. For more information, see "Managing Default Server Connections," on page 15.

Prerequisites

If you use a proxy server for the connection, verify that it is configured properly, so that the connection is kept alive long enough for long vSphere PowerCLI tasks to complete running.

NOTE If you do not want to use a proxy server for the connection, run `Set-PowerCLIConfiguration -ProxyPolicy NoProxy`.

Procedure

- Run `Connect-VIServer` with the server name and valid credentials.

```
Connect-VIServer -Server esx3.example.com -Protocol http -User admin -Password pass
```

Manage Virtual Machines on vSphere

With vSphere PowerCLI, you can automate various administration tasks on virtual machines, for example retrieving information, shutting down and powering off virtual machines.

Procedure

1. View all virtual machines on the target system.

```
Get-VM
```

2. Save the name and the power state properties of the virtual machines in the *ResourcePool* resource pool into a file named `myVMProperties.txt`.

```
$respool = Get-ResourcePool ResourcePool
Get-VM -Location $respool | Select-Object Name, PowerState > myVMProperties.txt
```

3. Start the *VM* virtual machine.

```
Get-VM VM | Start-VM
```

4 Get information of the guest OS of the *VM* virtual machine.

```
Get-VMGuest VM | fc
```

5 Shut down the OS of the *VM* virtual machine.

```
Shutdown-VMGuest VM
```

6 Power off the *VM* virtual machine.

```
Stop-VM VM
```

7 Move the virtual machine *VM* from the *Host01* host to the *Host02* host.

```
Get-VM -Name VM -Location Host01 | Move-VM -Destination Host02
```

NOTE If the virtual machine you want to move across hosts is powered on, it must be located on a shared storage registered as a datastore on both the original and the new host.

Add a Standalone Host to a vCenter Server System

You can add standalone hosts to a vCenter Server system by using the `Add-VMHost` cmdlet. After adding the hosts, you will be able to manage them through the vCenter Server system.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 View all hosts on the vCenter Server system that you have established a connection with.

```
Get-VMHost
```

2 Add the *Host* standalone host.

```
Add-VMHost -Name Host -Location (Get-Datacenter DC) -User root -Password pass
```

Activate Maintenance Mode for a Host on vCenter Server

To complete some specific administration tasks, you might need to activate maintenance mode for a host. On vCenter Server, you can activate maintenance mode by using the `Set-VMHost` cmdlet.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Save the *Host* host object as a variable.

```
$vmhost = Get-VMHost -Name Host
```

2 Get the cluster to which *Host* belongs and save the cluster object as a variable.

```
$vmhostCluster = Get-Cluster -VMHost $vmhost
```

3 Start a task that activates maintenance mode for the *Host* host and save the task object as a variable.

```
$updateHostTask = Set-VMHost -VMHost $vmhost -State "Maintenance" -RunAsync
```

NOTE If the host is not automated or is partially automated and has powered-on virtual machines running on it, you must use the `RunAsync` parameter and wait until all powered-on virtual machines are relocated or powered off before applying DRS recommendations.

4 Get and apply the recommendations generated by DRS.

```
Get-DrsRecommendation -Cluster $vmhostCluster | where {$_.Reason -eq "Host is entering
maintenance mode"} | Apply-DrsRecommendation
```

5 Get the task output object and save it as a variable.

```
$myUpdatedHost = Wait-Task $updateHostTask
```

Create vSphere Inventory Objects

By using vSphere PowerCLI cmdlets, you can automate creating different inventory objects on vSphere.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get the inventory root folder and create a new folder named `Folder` in it.

```
$folder = Get-Folder -NoRecursion | New-Folder -Name Folder
```

2 Create a new datacenter named `DC` in the `Folder` folder.

```
New-Datacenter -Location $folder -Name DC
```

3 Create a folder named `Folder1` under `DC`.

```
Get-Datacenter DC | New-Folder -Name Folder1
$folder1 = Get-Folder -Name Folder1
```

4 Create a new cluster Cluster1 in the `Folder1` folder.

```
New-Cluster -Location $folder1 -Name Cluster1 -DrsEnabled -DrsAutomationLevel FullyAutomated
```

Distributed Resource Scheduler (DRS) is a feature that provides automatic allocation of cluster resources.

5 Add a host in the cluster by using the `Add-VMHost` command, and provide credentials when prompted.

```
$vmhost1 = Add-VMHost -Name 10.23.112.345 -Location (Get-Cluster Cluster1)
```

6 Create a resource pool in the root resource pool of the cluster.

```
$myClusterRootRP = Get-Cluster Cluster1 | Get-ResourcePool -Name Resources
New-ResourcePool -Location $myClusterRootRP -Name MyRP1 -CpuExpandableReservation $true -
CpuReservationMhz 500 -CpuSharesLevel high -MemExpandableReservation $true -MemReservationGB
1 -MemSharesLevel high
```

7 Create a virtual machine asynchronously.

```
$vmCreationTask = New-VM -Name VM2 -VMHost $vmhost1 -ResourcePool MyRP01 -DiskGB 100 -
MemoryGB 2 -RunAsync
```

The `RunAsync` parameter indicates that the command runs asynchronously. This means that in contrast to a synchronous operation, you do not have to wait for the process to complete before supplying the next command at the command line.

Create Virtual Machines on vCenter Server Using an XML Specification File

You can use a specification provided in an XML file to automate the creation of virtual machines on vCenter Server.

Prerequisites

Verify that you are connected to a vCenter Server system.

The `myVM.xml` file must be present with the following content:

```
<CreateVM>
<VM>
<Name>MyVM1</Name>
<HDDCapacity>100</HDDCapacity>
</VM>
<VM>
<Name>MyVM2</Name>
<HDDCapacity>100</HDDCapacity>
</VM>
</CreateVM>
```

Procedure

1 Read the content of the `myVM.xml` file.

```
[xml]$s = Get-Content myVM.xml
```

2 Create the virtual machines.

```
$s.CreateVM.VM | foreach {New-VM -VMHost $vmHost1 -Name $_.Name -DiskGB $_.HDDCapacity}
```

Manage Virtual Machine Templates on vCenter Server

You can use vSphere PowerCLI to create virtual machines templates and convert them to virtual machines on vCenter Server.

NOTE A virtual machine template is a reusable image created from a virtual machine. The template, as a derivative of the source virtual machine, includes virtual hardware components, an installed guest operating system, and software applications.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Create a template from the *VM1* virtual machine.

```
New-Template -VM VM1 -Name VM1Template -Location (Get-Datacenter DC )
```

2 Convert the *VM1Template* template for use by a virtual machine named *VM3*.

```
Get-Template VM1Template | Set-Template -ToVM -Name VM3
```

3 Create a template from the *VM2* virtual machine.

```
New-Template -VM VM2 -Name VM2Template -Location (Get-Datacenter DC )
```

4 Convert the *VM2Template* template to a virtual machine named *VM4*.

```
Get-Template VM2Template | Set-Template -ToVM -Name VM4
```

5 Convert the *VM4* virtual machine to a template.

```
Set-VM -VM VM4 -ToTemplate -Name "VM4Template"
```

6 Create a template called *VM3Template* by cloning *VM2Template*.

```
Get-Template VM2Template | New-Template -Name VM3Template -VMHost $targetVMHost
```

Create and Use Snapshots on vCenter Server

You can use the `Snapshot` parameter of `Get-VM` to take a snapshot of virtual machines and then revert the states of the virtual machines back to the snapshot.

NOTE A snapshot captures the memory, disk, and settings state of a virtual machine at a particular moment. When you revert to a snapshot, you return all these items to the state they were in at the time you took that snapshot.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Take a snapshot of all virtual machines in the *MyRP01* resource pool.

```
Get-ResourcePool MyRP01 | Get-VM | New-Snapshot -Name InitialSnapshot
```

The `Location` parameter takes arguments of the `VIContainer` type, on which `Cluster`, `Datacenter`, `Folder`, `ResourcePool`, and `VMHost` object types are based. Therefore, the `Location` parameter can use arguments of all these types.

2 Revert all virtual machines in the *MyRP01* resource pool to the *InitialSnapshot* snapshot.

```
$VMs = Get-ResourcePool MyRP01 | Get-VM
foreach( $vm in $VMs ) { Set-VM -VM $vm -Snapshot InitialSnapshot }
```

Update the Resource Configuration Settings of a Virtual Machine on vCenter Server

You can use the `Set-VMResourceConfiguration` cmdlet to modify the resource configuration properties of a virtual machine, including memory, CPU shares, and other settings.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 View the resource configuration for the *VM1* virtual machine.

```
Get-VMResourceConfiguration -VM VM1
```

2 View the disk share of the *VM1* virtual machine.

```
Get-VMResourceConfiguration -VM VM1 | Format-Custom -Property DiskResourceConfiguration
```

3 Change the memory share of the *VM1* virtual machine to low.

```
Get-VM VM1 | Get-VMResourceConfiguration | Set-VMResourceConfiguration -MemSharesLevel low
```

4 Change the CPU shares of the *VM1* virtual machine to high.

```
Get-VM VM1 | Get-VMResourceConfiguration | Set-VMResourceConfiguration -CpuSharesLevel high
```

5 Change the disk share of the *VM1* virtual machine to 100.

```
$vm1 = Get-VM VM1
$vm1disk = Get-HardDisk $vm1
Get-VMResourceConfiguration $vm1 | Set-VMResourceConfiguration -Disk $vm1disk -
DiskSharesLevel custom -NumDiskShares 100
```

Get a List of Hosts on a vCenter Server System and View Their Properties

With vSphere PowerCLI, you can get information about all available hosts in a datacenter and view their properties.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get a list of all hosts that are part of a datacenter.

```
Get-Datacenter DC | Get-VMHost | Format-Custom
```

2 View the properties of the first host in the datacenter.

```
Get-Datacenter DC | Get-VMHost | Select-Object -First 1 | Format-Custom
```

3 View the `Name` and the `OverallStatus` properties of the hosts in the *DC* datacenter.

```
Get-Datacenter DC | Get-VMHost | Get-View | Format-Table -Property Name, OverallStatus -
AutoSize
```

4 View all hosts and their properties, and save the results to a file.

```
Get-Datacenter DC | Get-VMHost | Format-Custom | Out-File -FilePath hosts.txt
```

5 View a list of the hosts that are in maintenance mode and can be configured for vMotion operations.

```
Get-VMHost -State maintenance | Get-View | Where-Object -FilterScript { $_.capability -ne
$null -and $_.capability.vmotionSupported }
```

Change the Host Advanced Configuration Settings on vCenter Server

You can modify host configuration, including advanced settings related to virtual machine migration, and apply them to another host.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Change the migration timeout for the *ESXHost1* host.

```
Get-VMHost ESXHost1 | Set-VmHostAdvancedConfiguration -Name Migrate.NetTimeout -Value
( [system.int32] 10 )
```

2 Enable creation of a checksum of the virtual machines memory during the migration.

```
Get-VMHost ESXHost1 | Set-VmHostAdvancedConfiguration -Name Migrate.MemChksum -Value
( [system.int32] 1 )
```

3 Get the *ESXHost1* host migration settings.

```
$migrationSettings = Get-VMHost ESXHost1 | Get-VmHostAdvancedConfiguration -Name Migrate.*
```

4 Apply the migration settings to *ESXHost2*.

```
Set-VmHostAdvancedConfiguration -VMHost ESXHost2 -Hashtable $migrationSettings
```

Move a Virtual Machine to a Different Host Using VMware vSphere vMotion

You can migrate a virtual machine between vCenter Server hosts by using vSphere vMotion.

NOTE You can use vSphere vMotion to move a powered-on virtual machine from one host to another.

Prerequisites

Verify that you are connected to a vCenter Server system.

The virtual machine must be stored on a datastore shared by the current and the destination host, and the vMotion interfaces on the two hosts must be configured.

Procedure

- Get the *VM1* virtual machine and move it to a host named *ESXHost2*.

```
Get-VM VM1 | Move-VM -Destination (Get-VMHost ESXHost2)
```

Move a Virtual Machine to a Different Datastore Using VMware vSphere Storage vMotion

You can migrate a virtual machine between datastores using the VMware Storage vMotion feature of vCenter Server.

NOTE You can use Storage vMotion to move a powered-on virtual machine from one datastore to another.

Prerequisites

Verify that you are connected to a vCenter Server system.

The host on which the virtual machine is running must have access both to the datastore where the virtual machine is located and to the destination datastore.

Procedure

- Get the *VM1* virtual machine and move it to a datastore named *DS2*:

```
Get-VM VM1 | Move-VM -Datastore DS2
```

Create a Host Profile on a vCenter Server System

The VMware Host Profiles feature enables you to create standard configurations for ESX/ESXi hosts. With vSphere PowerCLI, you can automate creation and modifying of host profiles.

Prerequisites

Verify that you are connected to a host that runs vCenter Server 4.0 or later.

Procedure

1 Get the host named *Host1* and store it in the *$vmhost* variable.

```
$vmhost = Get-VMHost Host1
```

2 Create a profile based on the *Host1* host.

```
New-VMHostProfile -Name MyHostProfile01 -Description "This is my test profile based on Host1."
-ReferenceHost $vmhost
```

3 Get the newly created host profile.

```
$hp1 = Get-VMHostProfile -Name MyHostProfile01
```

4 Change the description of the *HostProfile1* host profile.

```
Set-VMHostProfile -Profile $hp1 -Description "This is my old test host profile based on Host1."
```

Apply a Host Profile to a Host on vCenter Server

To simplify operational management of large-scale environments, you can apply standard configurations called host profiles to hosts on vCenter Server. If you want to set up a host to use the same host profile as a reference host, you can attach the host to a profile.

Prerequisites

Verify that you are connected to a host that runs vCenter Server 4.0 or later.

Procedure

1 Get the *Host2* host.

```
$vmhost2 = Get-VMHost Host2
```

2 Attach the *Host2* host to the *HostProfile1* host profile.

```
Set-VMHost -VMHost $vmhost2 -Profile HostProfile1
```

3 Verify that the *Host2* host is compliant with the *HostProfile1* profile.

```
Test-VMHostProfileCompliance -VMHost $vmhost2
```

The output of this command contains the incompliant settings of the host, if any.

4 Apply the profile to the *Host2* host.

```
$neededVariables = Apply-VMHostProfile -Entity $vmhost2 -Profile $hp1 -Confirm:$false
```

The *$neededVariables* variable contains the names of all required variables and their default or current values, as returned by the server. Otherwise, the *$neededVariables* variable contains the name of the host on which the profile has been applied.

Manage Statistics and Statistics Intervals on vCenter Server

You can use the vSphere PowerCLI cmdlets to automate tasks for viewing and managing statistics for vCenter Server inventory objects.

You can modify the properties of a statistics interval and view statistics for a selected cluster.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Increase the amount of time for which statistics of the previous day are stored.

```
Set-StatInterval -Interval "past day" -StorageTimeSecs 700000
```

2 View the available memory metric types for the *Cluster1* cluster.

```
$cluster = Get-Cluster Cluster1
$statTypes = Get-StatType -Entity $cluster -Interval "past day" -Name mem.*
```

3 View the cluster statistics collected for the day.

```
Get-Stat -Entity $cluster -Start ([System.DateTime]::Now.AddDays(-1)) -Finish
([System.DateTime]::Now) -Stat $statTypes
```

Modify the Settings of the NIC Teaming Policy for a Virtual Switch

You can set the NIC teaming policy on a vSwitch. The NIC teaming policy determines the load balancing and failover settings of a virtual switch and lets you mark NICs as unused.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get a list of the physical NIC objects on the host network and store them in a variable.

```
$pn = Get-VMHost 10.23.123.128 | Get-VMHostNetwork | Select -Property physicalnic
```

2 Store the physical NIC objects you want to mark as unused in separate variables.

```
$pn5 = $pn.PhysicalNic[2]
$pn6 = $pn.PhysicalNic[3]
$pn7 = $pn.PhysicalNic[0]
```

3 View the NIC teaming policy of the *VSwitch01* virtual switch.

```
$policy = Get-VirtualSwitch -VMHost 10.23.123.128 -Name VSwitch01 | Get-NicTeamingPolicy
```

4 Change the policy of the switch to indicate that the *$pn5*, *$pn6*, and *$pn7* network adapters are unused.

```
$policy | Set-NicTeamingPolicy -MakeNicUnused $pn5, $pn6, $pn7
```

5 Modify the load balancing and failover settings of the virtual switch NIC teaming policy.

```
$policy | Set-NicTeamingPolicy -BeaconInterval 3 -LoadBalancingPolicy 3 -
NetworkFailoverDetectionPolicy 1 -NotifySwitches $false -FailbackEnabled $false
```

Create a vApp on vCenter Server

With vSphere PowerCLI, you can create and manage vApps.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Create a new vApp named *VApp* on a host.

```
New-VApp -Name VApp -CpuLimitMhz 4000 -CpuReservationMhz 1000 -Location (Get-VMHost Host1)
```

2 Start the new virtual appliance.

```
Start-VApp VApp
```

Modify the Properties of a vApp

With vSphere PowerCLI, you can start and stop vApps, and modify their properties.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get the vApp named *VApp* and stop it.

```
Get-VApp VApp | Stop-VApp -Confirm:$false
```

2 Change the name and memory reservation for the vApp.

```
Get-VApp VApp | Set-VApp -Name OldVApp -MemReservationGB 2
```

Export or Import vApps

You can import and export vApps to OVA and OVF files.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get the vApp you want to export.

```
$oldVApp = Get-VApp OldVApp
```

2 Export the *OldVApp* vApp to a local directory and name the exported appliance *WebApp*.

```
Export-VApp -VApp $oldVApp -Name WebApp -Destination D:\vapps\ -CreateSeparateFolder
```

3 Import the *WebApp* vApp from a local directory to the *Storage2* datastore.

```
Import-VApp -Source D:\vapps\WebApp\WebApp.ovf -VMHost (Get-VMHost Host1) -Datastore (Get-
Datastore -VMHost MyHost01 -Name Storage2)
```

Configure a Network Interface

You can modify the IP and routing configuration settings of a guest network interface.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get the network interface of a guest operating system.

```
$vm1 = Get-VM -Name VM1
$guest = Get-VMGuest $vm1
$interface = Get-VMGuestNetworkInterface -VMGuest $guest -GuestUser user -GuestPassword pass2
-ToolsWaitSecs 100
```

2 Configure the network interface.

```
Set-VMGuestNetworkInterface -VMGuestNetworkInterface $interface -GuestUser user -
GuestPassword pass2 -IPPolicy static -IP 10.23.112.69 -Gateway 10.23.115.253 -DnsPolicy static
-Dns (10.23.108.1, 10.23.108.2) -WinsPolicy dhcp
```

Add a Guest Route

You can add new guest routes for virtual machines.

Prerequisites

Verify that you are connected to an ESX/ESXi host.

Procedure

1 View the existing routes of the virtual machine stored in the *$myVM1* variable.

```
Get-VMGuestRoute -VM $vm1 -GuestUser user -GuestPassword pass2 -ToolsWaitSecs 50
```

2 View the existing routes of the guest OS stored in the *$guest* variable.

```
Get-VMGuestRoute -VMGuest $guest -GuestUser user -GuestPassword pass2
```

3 Add a new guest route to the virtual machine.

```
$route = New-VMGuestRoute -VM $vmWin -GuestUser user -GuestPassword pass2 -Destination
192.168.100.10 -Netmask 255.255.255.255 -Gateway 10.23.112.58 -Interface
$interface.RouteInterfaceId -ToolsWaitSecs 50
```

Create an iSCSI Host Storage

For a host, you can enable iSCSI, add iSCSI targets, and create new host storages.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Enable software iSCSI on a host.

```
$vmhost = Get-VMHost ESXHost1
Get-VMHostStorage $myHost | Set-VMHostStorage -SoftwareIScsiEnabled $true
```

2 Get the iSCSI HBA that is on the host.

```
$iscsiHba = Get-VMHostHba -Type iScsi
```

3 Add a new iSCSI target for dynamic discovery.

```
$iscsiHba | New-IScsiHbaTarget -Address 192.168.0.1 -Type Send
```

4 Rescan the HBAs on the host.

```
Get-VMHostStorage $vmhost -RescanAllHba
```

5 Get the path to the SCSI LUN.

```
$lunPath = Get-ScsiLun -VMHost $vmhost -CanonicalName ($iscsiHba.Device + "*") | Get-
ScsiLunPath
```

You can provide the LUN path by using its canonical name beginning with the device name of the iSCSI HBA.

6 Create a new host storage.

```
New-Datastore -Vmfs -VMHost $vmhost -Path $lunpath.LunPath -Name iSCSI
```

Add Passthrough Devices to a Host and Virtual Machine

You can get information about existing passthrough devices and add new SCSI and PCI devices to virtual machines and hosts.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get a list of the PCI passthrough devices of the *VMHost* host

```
$vmhost = Get-VMHost ESXHost
Get-PassthroughDevice -VMHost $vmhost -Type Pci
```

2 Get a list of the SCSI passthrough devices of the *VM* virtual machine

```
$vm = Get-VM VM
Get-PassthroughDevice -VM $vm -Type Scsi
```

3 Add a SCSI passthrough device to the *VM* virtual machine

```
$scsiDeviceList = Get-PassthroughDevice -VMHost ESXHost -Type Scsi
Add-PassthroughDevice -VM $vm -PassthroughDevice $scsiDeviceList[0]
```

Create a Custom Property Based on an Extension Data Property

You can create custom properties to add more information to vSphere objects. Custom properties based on extension data properties correspond directly to the property of the corresponding .NET view object.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Create a new custom property based on the `Guest.ToolsVersion` property.

```
New-VIProperty -ObjectType VirtualMachine -Name ToolsVersion -ValueFromExtensionProperty
'Guest.ToolsVersion'
```

2 View the `ToolsVersion` properties of the available virtual machines.

```
Get-VM | Select Name, ToolsVersion
```

You have created a custom property named `ToolsVersion` for `VirtualMachine` objects.

Create a Script-Based Custom Property for a vSphere Object

You can create a custom property by writing a script and providing a name for the property. The script evaluates when the custom property is called for the first time.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Create a new custom property named *NameOfHost* that stores the name of the host on which a virtual machine resides.

```
New-VIProperty -Name NameOfHost -ObjectType VirtualMachine -Value { return
$args[0].VMHost.Name }
```

2 View the *NameOfHost* properties of the available virtual machines.

```
Get-VM | select Name, NameOfHost | Format-Table -AutoSize
```

You created a custom script property named *NameOfHost* for `VirtualMachine` objects.

Apply a Customization Object to a Cloned Virtual Machine

You can apply a custom configuration to a cloned virtual machine by using a customization object.

NOTE This feature runs only on a 32-bit vSphere PowerCLI process.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get the *Spec* customization specification and clone it for temporary use.

```
Get-OSCustomizationSpec Spec | New-OSCustomizationSpec -Type NonPersistent -Name ClientSpec
```

2 Change the `NamingPrefix` property of the customization object to the name of the virtual machine you want to create.

```
Set-OSCustomizationSpec -Spec ClientSpec -NamingPrefix VM1
```

3 Create a virtual machine named *VM1* by cloning the existing *VM* virtual machine and applying the customization specification.

```
Get-VM VM | New-VM -VMHost Host -Datastore Storage1 -OSCustomizationSpec ClientSpec -Name VM1
```

Modify the Default NIC Mapping Object of a Customization Specification

You can modify the default NIC mapping object of a customization specification and apply the specification on a newly created virtual machine.

Procedure

1 Create a nonpersistent customization specification for Windows operating systems.

```
New-OSCustomizationSpec -Type NonPersistent -Name Spec -OSType Windows -Workgroup Workgroup -
OrgName Company -Fullname User -ProductKey “valid_key” -ChangeSid -TimeZone "Central European"
-NamingScheme VM
```

2 View the default NIC mapping objects of the *Spec* specification.

```
Get-OSCustomizationNicMapping -Spec Spec | Set-OSCustomizationNicMapping -IpMode UseStaticIP
-IpAddress 172.16.1.30 -SubnetMask 255.255.255.0 -DefaultGateway 172.16.1.1 -Dns 172.16.1
```

Each customization specification object has one default NIC mapping object.

3 Modify the default NIC mapping object of the *Spec* customization specification to use static IP.

```
Get-OSCustomizationNicMapping -Spec Spec | Set-OSCustomizationNicMapping  -IpMode UseStaticIP
-IpAddress 172.16.1.30 -SubnetMask 255.255.255.0 -DefaultGateway 172.16.1.1 -Dns 172.16.1.1
```

4 Create a new virtual machine named *VM1* from a template, and apply the static IP settings.

```
New-VM -Name VM1 -VMHost Host -Datastore Storage1 -OSCustomizationSpec Spec -Template Template
```

Modify Multiple NIC Mapping Objects of a Customization Specification

You can modify multiple NIC mapping objects of a customization specification and apply the specification to an existing virtual machine.

Procedure

1 Get the network adapters of a virtual machine named *VM*.

```
Get-NetworkAdapter VM
```

When you apply a customization specification, each network adapter of the customized virtual machine must have a corresponding NIC mapping object. You can correlate network adapters and NIC mapping objects either by their position numbers, or by MAC address.

2 Create a customization specification named *Spec*.

```
New-OSCustomizationSpec -Type NonPersistent -Name Spec -OSType Windows -Workgroup Workgroup -
OrgName Company -Fullname User -ProductKey "valid_key" -ChangeSid -TimeZone "Central European"
-NamingScheme VM
```

3 Add a new NIC mapping object that uses a static IP address.

```
New-OSCustomizationNicMapping -Spec Spec -IpMode UseStaticIP -IpAddress 172.16.1.30 -
SubnetMask 255.255.255.0 -DefaultGateway 172.16.1.1 -Dns 172.16.1.1
```

4 View the NIC mapping objects and verify that two NIC mapping objects are available.

```
Get-OSCustomizationNicMapping -Spec Spec
```

The default NIC mapping object is DHCP enabled, and the newly added one uses a static IP address.

5 Apply the Spec customization specification to the *VM* virtual machine.

```
Get-VM VM | Set-VM -OSCustomizationSpec -Spec Spec
```

6 Associate a network adapter from the *VMNetwork* network with the NIC mapping object that uses DHCP mode.

```
$netAdapter = Get-NetworkAdapter VM | where { $_.NetworkName -eq 'VMNetwork' }
Get-OSCustomizationNicMapping -Spec Spec | where { $_.IPMode -eq 'UseDHCP' } | Set-
OSCustomizationNicMapping -NetworkAdapterMac $netAdapter.MacAddress
```

Create a vSphere Role and Assign Permissions to a User

With vSphere PowerCLI, you can automate management of vSphere permissions, roles, and privileges.

NOTE vSphere permissions determine your level of access to vCenter Server, and ESX/ESXi hosts. Privileges define individual rights to perform actions and access object properties. Roles are predefined sets of privileges.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get the privileges of the **Readonly** role.

```
$readOnlyPrivileges = Get-VIPrivilege -Role Readonly
```

2 Create a new role with custom privileges.

```
$role1 = New-VIRole -Privilege $readOnlyPrivileges -Name Role1
```

3 Add the **PowerOn** privileges to the new role.

```
$powerOnPrivileges = Get-VIPrivilege -Name "PowerOn"
$role1 = Set-VIRole -Role $role1 -AddPrivilege $powerOnPrivileges
```

4 Create a permission and apply it to a vSphere root object.

```
$rootFolder = Get-Folder -NoRecursion
$permission1 = New-VIPermission -Entity $rootFolder -Principal "user" -Role readonly -
Propagate
```

The `Principal` parameter accepts both local and domain users and groups if the vCenter Server system is joined in AD.

5 Update the new permission with the custom role.

```
$permission1 = Set-VIPermission -Permission $permission1 -Role $role1
```

You created a new role and assigned permissions to a user.

View the Action Triggers for an Alarm on vCenter Server

You can see which action triggers are configured for an alarm.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get all vSphere PowerCLI supported alarm actions for the *Host Processor Status* alarm.

```
Get-AlarmDefinition -Name "Host Processor Status" | Get-AlarmAction -ActionType
"ExecuteScript", "SendSNMP", "SendEmail"
```

2 Get all the triggers for the first alarm definition found.

```
Get-AlarmAction -AlarmDefinition (Get-AlarmDefinition | select -First 1) | Get-
AlarmActionTrigger
```

Create and Modify Alarm Actions and Alarm Triggers on vCenter Server

With vSphere PowerCLI, you can create and modify vCenter Server alarm actions and alarm triggers.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 For all host alarms, modify the interval after the action repeats.

```
Get-AlarmDefinition -Entity (Get-VMHost) | foreach { $_ | Set-AlarmDefinition -
ActionRepeatMinutes ($_.ActionRepeatMinutes + 1)}
```

2 Modify the name and the description of a selected alarm definition, and enable the alarm.

```
Get-AlarmDefinition -Name AlarmDefinition | Set-AlarmDefinition -Name AlarmDefinitionNew -
Description 'Alarm Definition Description' -Enabled:$true
```

3 Create an alarm action `email` for the renamed alarm definition.

```
Get-AlarmDefinition -Name AlarmDefinitionNew | New-AlarmAction -Email -To 'test@vmware.com' -
CC @('test1@vmware.com', 'test2@vmware.com') -Body 'Email text' -Subject 'Email subject'
```

4 Create an `snmp` alarm action.

```
Get-AlarmDefinition -Name AlarmDefinitionNew | New-AlarmAction -Snmp
```

5 Create a `script` alarm action.

```
Get-AlarmDefinition -Name AlarmDefinitionNew | New-AlarmAction -Script -ScriptPath
'c:\test.ps1'
```

6 Create an action trigger on all actions for the selected alarm.

```
Get-AlarmDefinition -Name AlarmDefinitionNew | Get-AlarmAction | New-AlarmActionTrigger -
StartStatus 'Red' -EndStatus 'Yellow' -Repeat
```

Remove Alarm Actions and Triggers

In some cases, you might want to remove obsolete alarm actions and triggers.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Remove the first one from the action triggers found for an alarm definition.

```
Get-AlarmDefinition -Name AlarmDefinition | Get-AlarmAction | Get-AlarmActionTrigger | select
-First 1 | Remove-AlarmActionTrigger -Confirm:$false
```

2 Remove all the actions for an alarm definition.

```
Get-AlarmDefinition -Name AlarmDefinition | Get-AlarmAction | Remove-AlarmAction -Confirm:
$false
```

Create and Modify Advanced Settings for a Cluster

You can customize the behavior of a cluster on a vCenter Server system by creating and modifying custom advanced settings for it.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Create a new cluster named *Cluster*.

```
$cluster = New-Cluster -Name Cluster -Location (Get-Datacenter Datacenter)
```

2 Create two advanced settings for the new cluster.

```
$setting1 = New-AdvancedSetting -Type "ClusterHA" -Entity $cluster -Name
'das.defaultfailoverhost' -Value '192.168.10.1'
$setting2 = New-AdvancedSetting -Type "ClusterHA" -Entity $cluster -Name
'das.isolationaddress' -Value '192.168.10.2'
```

3 Modify the value of the advanced setting stored in the *$setting2* variable.

```
Get-AdvancedSetting -Entity $cluster -Name 'das.isolationaddress' | Set-AdvancedSetting -
Value '192.168.10.3' -Confirm:$false
```

4 Create another advanced setting.

```
New-AdvancedSetting -Entity $cluster -Name 'das.allowNetwork[Service Console]' -Value $true -
Type 'ClusterHA'
```

5 Get the `Service Console` setting and store it in a variable.

```
$setting3 = Get-AdvancedSetting -Entity $entity -Name 'das.allowNetwork`[Service Console`]'
```

The ` character is used to to escape the wildcard characters [and] in the advanced setting name.

Modify the vCenter Server Email Configuration

You can modify the email configuration settings of a vCenter Server.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 View the current email configuration settings of the vCenter Server from the *$srv* variable.

```
Get-AdvancedSetting -Entity $srv -Name mail.*
```

2 Update the SMTP server name and port.

```
Get-AdvancedSetting -Entity $srv  -Name mail.smtp.server | Set-AdvancedSetting -Value
smtp.vmware.com
Get-AdvancedSetting -Entity $srv  -Name mail.smtp.port | Set-AdvancedSetting -Value 25
```

Modify the vCenter Server SNMP Configuration

To use SNMP, you must first configure the SNMP settings of the vCenter Server.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 View the current SNMP configuration settings of the vCenter Server from the *$srv* variable.

```
Get-AdvancedSetting -Entity $srv  -Name snmp.*
```

2 Modify the SNMP receiver data.

```
Get-AdvancedSetting -Entity $srv  -Name snmp.receiver.2.community | Set-AdvancedSetting -
Value public
Get-AdvancedSetting -Entity $srv  -Name snmp.receiver.2.enabled | Set-AdvancedSetting -Value
$true
Get-AdvancedSetting -Entity $srv  -Name snmp.receiver.2.name | Set-AdvancedSetting -Value
192.168.1.10
```

Now you can use SNMP with vCenter Server.

Use Esxtop to Get Information on the Virtual CPUs of a Virtual Machine

You can use the `Get-EsxTop` cmdlet to retrieve real-time data for troubleshooting performance problems.

Prerequisites

Verify that you are connected to a server that runs ESX 4.0, vCenter Server 5.0 or later.

Procedure

1 Get the group to which the virtual machine belongs and save it as a variable.

```
$group = Get-EsxTop -CounterName SchedGroup | where {$_.VMName -eq $vm.Name}
```

2 Get the IDs of all virtual CPUs of the virtual machine and store them in an array.

```
$gr = Get-EsxTop -TopologyInfo -Topology SchedGroup | %{$_.Entries} | where {$_.GroupId -eq
$group.GroupID}
$cpuIds = @()
$gr.CpuClient | %{$cpuIds += $_.CPUClientID}
```

3 Get the CPU statistics for the virtual machine.

```
$cpuStats = Get-EsxTop -CounterName VCPU | where {$cpuIds -contains $_.VCPUID}
```

4 Calculate the used and ready for use percentage by using the `UsedTimeInUsec` and `ReadyTimeInUsec` stats.

```
$result = @()
$cpuStats | %{ `
$row = "" | select VCPUID, Used, Ready; `
$row.VCPUID = $_.VCPUID; `
$row.Used = [math]::Round(([double]$_.UsedTimeInUsec/[double]$_.UpTimeInUsec)*100, 2); `
$row.Ready = [math]::Round(([double]$_.ReadyTimeInUsec/[double]$_.UpTimeInUsec)*100, 2);`
$result += $row
}
```

5 View the used and ready for use percentage for each virtual CPU of the virtual machine.

```
$result | Format-Table -AutoSize
```

Filter vSphere Objects with Get-View

You can use the `Get-View` cmdlet to filter vSphere objects before performing various actions on them.

The filter parameter is a `HashTable` object containing one or more pairs of filter criteria. Each of the criteria consists of a property path and a value that represents a regular expression pattern used to match the property.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Create a filter by the power state and the guest operating system name of the virtual machines.

```
$filter = @{"Runtime.PowerState" ="poweredOn"; "Config.GuestFullName" = "Windows XP"}
```

2 Get a list of the virtual machines by using the created filter and call the `ShutdownGuest` method for each virtual machine in the list.

```
Get-View -ViewType "VirtualMachine" -Filter $filter | foreach{$_.ShutdownGuest()}
```

The filter gets a list of the powered-on virtual machines whose guest OS names contain the string `Windows XP`. The `Get-View` cmdlet then initiates shutdown for each guest operating system in the list.

Populate a View Object with Get-View

To save time and efforts, you can use `Get-View` to retrieve vSphere PowerCLI views from previously retrieved view objects.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get a view of the *VM2* virtual machine by name.

```
$vm2 = Get-View -ViewType VirtualMachine -Filter @{"Name" = "VM2"}
```

2 Populate the *$vmhostView* object.

```
$vmhostView = Get-View -Id $vm2.Runtime.Host
```

3 Retrieve the runtime information for the *$vmhostView* object.

```
$vmhostView.Summary.Runtime
```

Update the State of a Server-Side Object

You can use the `Get-View` cmdlet to update server-side objects.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get the *VM2* virtual machine by name.

```
$vm2 = Get-View -ViewType VirtualMachine -Filter @{"Name" = "VM2"}
$vmhostView = Get-View -Id $vm2.Runtime.Host
```

2 View the current power state.

```
$vm2.Runtime.PowerState
```

3 Power off the virtual machine.

```
If ($vm2.Runtime.PowerState -ne “PoweredOn”) {
        $vm.PowerOnVM($vm2.Runtime.Host)
} else {
$vm2.PowerOffVM()
}
```

4 View the value of the *$vm2* power state.

```
$vm2.Runtime.PowerState
```

The power state is not updated yet because the virtual machine property values are not updated automatically.

5 Update the view object.

```
$vm2.UpdateViewData()
```

6 Obtain the actual power state of the virtual machine.

```
$vm2.Runtime.PowerState
```

Reboot a Host with Get-View

You can reboot a host by using its corresponding view object.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Use the `Get-VMHost` cmdlet to get a host by its name, and pass the result to the `Get-View` cmdlet to get the corresponding view object.

```
$vmhostView = Get-VMHost -Name Host | Get-View
```

2 Call the reboot method of the host view object to reboot the host.

```
$vmhostView.RebootHost()
```

Modify the CPU Levels of a Virtual Machine with Get-View and Get-VIObjectByVIView

You can modify the CPU levels of a virtual machine using a combination of the `Get-View` and `Get-VIObjectByVIView` cmdlets.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get the *VM2* virtual machine, shut it down, and pass it to the `Get-View` cmdlet to view the virtual machine view object.

```
$vmView = Get-VM VM2 | Stop-VM | Get-View
```

2 Create a *VirtualMachineConfigSpec* object to modify the virtual machine CPU levels and call the `ReconfigVM` method of the virtual machine view managed object.

```
$spec = New-Object VMware.Vim.VirtualMachineConfigSpec;
$spec.CPUAllocation = New-Object VMware.Vim.ResourceAllocationInfo;
$spec.CpuAllocation.Shares = New-Object VMware.Vim.SharesInfo;
$spec.CpuAllocation.Shares.Level = "normal";
$spec.CpuAllocation.Limit = -1;
$vmView .ReconfigVM_Task($spec)
```

3 Get the virtual machine object by using the `Get-VIObjectByVIView` cmdlet and start the virtual machine.

```
$vm =  Get-VIObjectByVIView $vmView  | Start-VM
```

Browse the Default Inventory Drive

You can browse the default inventory drive and view its contents.

NOTE For more information about the Inventory Provider and the default inventory drive, see "vSphere PowerCLI Inventory Provider," on page 16.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Navigate to the `vi` inventory drive.

```
cd vi:
```

2 View the drive content.

```
dir
```

`dir` is an alias of the `Get-ChildItem` cmdlet.

Create a New Custom Inventory Drive

In addition to the default drive, you can create new custom inventory drives by using the `New-PSDrive` cmdlet.

NOTE An alternative to creating an inventory drive is to map an existing inventory path. For example, run: `New-PSDrive -Name myVi -PSProvider VimInventory -Root "vi:\Folder01\Datacenter01"`.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get the root folder of the server.

```
$root = Get-Folder -NoRecursion
```

2 Create a PowerShell drive named `myVi` in the server root folder.

```
New-PSDrive -Location $root -Name myVi -PSProvider VimInventory -Root '\'
```

NOTE You can use the `New-InventoryDrive` cmdlet, which is an alias of `New-PSDrive`. This cmdlet creates a new inventory drive using the `Name` and `Datastore` parameters. For example: `Get-Folder -NoRecursion | New-VIInventoryDrive -Name myVi`.

Manage Inventory Objects Through Inventory Drives

You can use the vSphere PowerCLI Inventory Provider to browse, modify, and remove inventory objects from inventory drives.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Navigate to a host in your server inventory by running the `cd` command with the full path to the host.

```
cd Folder01\DataCenter01\host\Web\Host01
```

2 View the content of the host using the `ls` command.

```
ls
```

`ls` is the UNIX style alias of the `Get-ChildItem` cmdlet.

This command returns the virtual machines and the root resource pool of the host.

3 View only the virtual machines on the host.

```
Get-VM
```

When called within the inventory drive, `Get-VM` gets a list only of the virtual machines on the current drive location.

4 Delete a virtual machine named *VM1*.

```
del VM1
```

5 Rename a virtual machine, for example, from *VM1New* to *VM1*.

```
ren VM1New VM1
```

6 Start all virtual machines with names that start with *VM*.

```
dir VM* | Start-VM
```

Browse the Default Datastore Drives

You can use the vSphere PowerCLI Datastore Provider to browse the default datastore drives: `vmstore` and `vmstores`.

NOTE For more information about default datastore drives, see "vSphere PowerCLI Datastore Provider," on page 16.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Navigate to the `vmstore` drive.

```
cd vmstore:
```

2 View the drive content.

```
dir
```

Create a New Custom Datastore Drive

You can use the vSphere PowerCLI Datastore Provider to create custom datastore drives.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Get a datastore by its name and assign it to the *$datastore* variable.

```
$datastore = Get-Datastore Storage1
```

2 Create a new PowerShell drive `ds:` in *$datastore*.

```
New-PSDrive -Location $datastore -Name ds -PSProvider VimDatastore -Root '\'
```

NOTE You can use the `New-PSDrive` cmdlet, which is an alias of `New-DatastoreDrive`. It creates a new datastore drive using the `Name` and `Datastore` parameters. For example: `Get-Datastore Storage1 | New-DatastoreDrive -Name ds`.

Manage Datastores Through Datastore Drives

You can use the vSphere PowerCLI Datastore Provider to browse datastores from datastore drives.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 Navigate to a folder on the `ds:` drive.

```
cd VirtualMachines\XPVirtualMachine
```

2 View the files of the folder by running the `ls` command.

```
ls
```

`ls` is the UNIX style alias of the `Get-ChildItem` cmdlet.

3 Rename a file by running the `Rename-Item` cmdlet or its alias `ren`.

For example, to change the name of the `vmware-3.log` file to `vmware-3old.log`, run:

```
ren vmware-3.log vmware-3old.log
```

All file operations apply only on files in the current folder.

4 Delete a file by running the `Remove-Item` cmdlet or its alias `del`.

For example, to remove the `vmware-3old.log` file from the `XPVirtualMachine` folder, run:

```
del ds:\VirtualMachines\XPVirtualMachine\vmware-2.log
```

5 Copy a file by running the `Copy-Item` cmdlet or its alias `copy`.

```
copy ds:\VirtualMachines\XPVirtualMachine\vmware-3old.log ds:\VirtualMachines\vmware-3.log
```

6 Copy a file to another datastore by running the `Copy-Item` cmdlet or its alias `copy`.

```
copy ds:\Datacenter01\Datastore01\XPVirtualMachine\vmware-1.log
ds:\Datacenter01\Datastore02\XPVirtualMachine02\vmware.log
```

7 Create a new folder by running the `New-Item` cmdlet or its alias `mkdir`.

```
mkdir -Path ds:\VirtualMachines -Name Folder01 -Type Folder
```

8 Download a file from the datastore drive to the local machine by running the `Copy-DatastoreItem` cmdlet.

```
Copy-DatastoreItem ds:\VirtualMachines\XPVirtualMachine\vmware-3.log C:\Temp\vmware-3.log
```

9 Upload a file from the local machine by running the `Copy-DatastoreItem` cmdlet.

```
Copy-DatastoreItem C:\Temp\vmware-3.log  ds:\VirtualMachines\XPVirtualMachine\vmware-3new.log
```

Modify the Timeout Setting for Web Tasks

To avoid unexpected timeout, you can use `Set-PowerCLIConfiguration` to modify the vSphere PowerCLI settings for long-running Web tasks.

Prerequisites

Verify that you are connected to a vCenter Server system.

Procedure

1 (Optional) Learn more about what settings you can configure with `Set-PowerCLIConfiguration`.

```
Get-Help Set-PowerCLIConfiguration
```

2 Store the value of the timeout setting for the current session in the *$initialTimeout* variable.

```
$initialTimeout = (Get-PowerCLIConfiguration -Scope Session).WebOperationTimeoutSeconds
```

3 Set the timeout setting for the current session to 30 minutes.

```
Set-PowerCLIConfiguration -Scope Session -WebOperationTimeoutSeconds 1800000
```

4 Run your Web task.

For example, run an `esxcli` command to install a software profile.

```
$esxcli.software.profile.install("http://mysite.com/publish/proj/index.xml",$null,$null,
$null,$null,$null,$true,"proj-version",$null)
```

5 Revert the timeout setting for the current session to the initial value.

```
Set-PowerCLIConfiguration -Scope Session -WebOperationTimeoutSeconds $initialTimeout
```

Sample Scripts for Managing vCloud Director with VMware vCloud Director PowerCLI

6

To help you get started with VMware vCloud Director PowerCLI, this documentation provides a set of scripts that illustrate basic and advanced tasks in cloud administration.

- Manage Virtual Machines with vApps on page 65

  For a large-scale approach to administration, you can start, stop, or restart virtual machines or their guest operating systems by running cmdlets on the associated vApps.

- Manage Virtual Machines and Their Guest Operating Systems on page 65

  For a targeted approach to administration, you can use the `CIVM` and `CIVMGuest` cmdlets to handle lifecycle operations for one or more virtual machines.

- Retrieve a List of the Internal and External IP Addresses of Virtual Machines in vApps on page 66

  When managing vApps in the cloud, you might need to obtain information about the NIC settings of the associated virtual machines.

- Create and Manage Access Control Rules on page 67

  By defining access control rules you can assign levels of access to separate users, user groups, or everyone in the organization. You can define access control rules for catalogs and vApps.

- Filter and Retrieve vApp Networks on page 67

  To generate reports about vApp networks, you need to retrieve the respective vApp networks. Use search criteria to filter the results returned by `Get-CIVAppNetwork`.

- Create vApp Networks for a Selected vApp on page 68

  To define how the virtual machines in a vApp connect to each other and access other networks, you need to create a vApp network. When creating the vApp network, you can select the settings for the network, or adopt them from an organization policy.

- Modify or Remove vApp Networks on page 69

  When a networking scenario for a vApp changes, you can modify the corresponding vApp network instead of creating a new one. If the networking scenario is no longer relevant, you can remove the related vApp network.

Connect to a vCloud Director Server

To run cmdlets on a vCloud Director server and perform administration or monitoring tasks, you must establish a connection to the server.

You can have more than one connection to the same server. For more information, see "Managing Default Server Connections," on page 15.

Prerequisites

If you use a proxy server for the connection, verify that it is configured properly, so that the connection is kept alive long enough for vCloud Director PowerCLI tasks to complete running.

NOTE If you do not want to use a proxy server for the connection, run `Set-PowerCLIConfiguration -ProxyPolicy NoProxy`.

Procedure

- Run `Connect-CIServer` with the server name and valid credentials.

```
Connect-CIServer -Server cloud.example.com -User 'MyAdministratorUser' -Password 'MyPassword'
```

Create and Manage Organizations

Organizations provide resources to a group of users and set policies that determine how users can consume those resources. Create and manage organizations for each group of users that requires its own resources, policies, or both.

Prerequisites

Verify that you are connected to a vCloud Director server as a provider administrator.

Procedure

1 Generate a customized report for all organizations on the server.

```
Get-Org | Select Name, Enabled, StoredVMQuota, DeployedVMQuota
```

2 Add a new organization on the server and provide a name and a full name for it.

```
New-Org -Name 'MyOrg1' -FullName 'My Organization 1'
```

By default, the new organization is enabled. Enabling the organization lets users log in.

3 Add a description for the new organization.

```
Get-Org -Name 'MyOrg1' | Set-Org -Description "This organization provides resources to John Doe."
```

4 Disable and remove the new organization.

```
Get-Org -Name 'MyOrg1' | Set-Org -Enabled $false | Remove-Org
```

Create and Manage Organization Virtual Datacenters

To allocate resources to an organization, you need to create an organization virtual datacenter (vDC). When the demands of the organization change, you can modify or remove the organization vDC.

Prerequisites

- Verify that you are connected to a vCloud Director server as a provider administrator.
- Verify that at least one enabled provider vDC is available on the server.

Procedure

1 Create a new organization vDC using the Pay As You Go model for resource allocation.

```
$myOrg = Get-Org -Name 'MyOrg1'
$myPVdc = Get-ProviderVdc -Name 'MyProvidervDC'
New-OrgVdc -Name 'MyOrgvDC' -AllocationModelPayAsYouGo -Org $myOrg -ProviderVdc $myPVdc -VMCPUCoreMHz 1000
```

To create the organization vDC, vCloud Director PowerCLI uses a default configuration based on the selected resource allocation model.

- VMMaxCount is set to 100
- NetworkMaxCount is set to 1024
- The vDC is automatically enabled
- Thin provisioning is disabled
- Fast provisioning is disabled
- NicMaxCount is set to $null (unlimited)

- MemoryGuaranteedPercent is set to 100
- CpuGuaranteedPercent is set to 0

2 Modify the vCPU speed setting for the virtual machines in the organization vDC.

```
Get-OrgVdc -Name 'MyOrgVdc' | Set-OrgVdc -VMCpuCoreMhz 2000
```

3 Enable fast provisioning for the virtual machines in the organization vDC.

```
Get-OrgVdc -Name 'MyOrgVdc' | Set-OrgVdc -UseFastProvisioning $true
```

4 Disable and remove the new organization vDC.

```
Get-OrgVdc -Name 'MyOrgVdc' | Set-OrgVdc -Enabled $false | Remove-OrgVdc
```

Create and Manage Organization Networks

To define how the virtual machines in an organization connect to each other and access other networks, you must create an organization network. To address multiple networking scenarios for an organization, you can create multiple organization networks.

Prerequisites

Verify that you are connected to a vCloud Director server as a provider administrator.

Procedure

1 Retrieve the organization for which you want to create and manage organization networks.

```
$myOrg = Get-Org -Name 'MyOrg'
```

2 Retrieve the external network to which you want to connect your organization networks.

```
$myExternalNetwork = Get-ExternalNetwork -Name 'MyExternalNetwork'
```

3 Retrieve the network pool that will allocate network resources for your organization networks.

```
$myNetworkPool = Get-NetworkPool -Name 'MyNetworkPool'
```

4 Create an organization network that connects directly to the selected external network.

```
$myDirectOrgNet = New-OrgNetwork -Direct -Name 'MyDirectOrgNetwork' -Org $myOrg -
ExternalNetwork $myExternalNetwork -Description "This network is directly connected to the
internet."
```

5 Create an NAT routed organization network.

```
$myNatOrgNet = New-OrgNetwork -Routed -Name 'MyRoutedOrgNetwork' -Org $myOrg -NetworkPool
$myNetworkPool -ExternalNetwork $myExternalNetwork -Gateway 192.168.0.1 -Netmask
255.255.255.0 -Description "This network is secured by NAT and firewall services."
```

6 Create an internal organization network.

```
$myIsolatedOrgNet = New-OrgNetwork -Internal -Name 'MyInternalOrgNetwork' -Org $myOrg -
NetworkPool $myNetworkPool -Gateway 192.168.1.1 -Netmask 255.255.255.0 -PrimaryDns
192.168.1.10 -SecondaryDns 192.168.1.11 -DnsSuffix 'company.example.com' -Description "This
network is not connected to the Internet."
```

7 Modify the DNS and static IP pool settings for *MyInternalOrgNetwork*.

```
Set-OrgNetwork ‘internalOrgNetwork’ -PrimaryDns 192.168.1.253 -SecondaryDns 192.168.1.254 -
DnsSuffix 'example.com' -StaticIPPool "192.168.1.10-192.168.1.99"
```

8 Remove all organization networks in the selected organization.

```
Get-OrgNetwork -Org $myOrg | Remove-OrgNetwork
```

Filter and Retrieve Organization Networks

To generate reports about organization networks, you need to retrieve the respective organization networks. Use search criteria to filter the results returned by `Get-OrgNetwork`.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

- Retrieve all organization networks for the organization named *MyOrg*.

  ```
  Get-Org -Name 'MyOrg' | Get-OrgNetwork
  ```

- Retrieve all organization networks that are connected to the external network named *MyExternalNetwork*.

  ```
  Get-OrgNetwork -ExternalNetwork 'MyExternalNetwork'
  ```

- Retrieve the organization network that is named *MyInternalOrgNetwork* and is connected to the network pool named *MyNetworkPool*.

  ```
  Get-NetworkPool -Name 'MyNetworkPool' | Get-OrgNetwork -Name 'MyInternalOrgNetwork'
  ```

Import a vApp Template from the Local Storage

To make an OVF package from your local storage available to other cloud users, you can import the package and save it as a vApp template in a catalog.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Retrieve the catalog to which you want to add the imported vApp template.

```
$myCatalog = Get-Catalog -Name 'MyCatalog'
```

2 Retrieve the organization virtual datacenter (vDC) to which you want to add the imported vApp template.

```
$myOrgVdc = Get-OrgVdc -Name 'MyOrgVdc'
```

3 Import a virtual machine from your local storage and save it as a vApp template in the cloud.

```
Import-CIVAppTemplate -SourcePath 'C:\OVFs\WindowsXP\WindowsXP.ovf' -Name
'MyWindowsXPVAppTemplate' -OrgVdc $myOrgVdc -Catalog $myCatalog
```

Create a vApp Template from a vApp

Creating vApp templates from vApps in the cloud might minimize future efforts for cloning vApps. You can use the templates later to create new vApps that are based on the source vApp.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Retrieve the source vApp for the vApp template that you want to create.

```
$myVApp = Get-CIVApp -Name 'MyVApp'
```

2 If the source vApp is running, stop it.

```
$myVApp = Stop-CIVApp -VApp $myVApp
```

3 Retrieve the catalog to which you want to add the new vApp template.

```
$myCatalog = Get-Catalog -Name 'MyCatalog'
```

4 Retrieve the organization vDC to which you want to add the new vApp template.

```
$myOrgVdc = Get-OrgVdc -Name 'MyOrgVdc'
```

5 Create the new vApp template.

```
New-CIVAppTemplate -Name 'MyVAppTemplate' -VApp $myVApp -OrgVdc $myOrgVDC -Catalog $myCatalog
```

6 Start the source vApp.

```
$myVApp = Start-CIVApp -VApp $myVApp
```

What to do next

"Create and Modify a vApp," on page 64

Import a vApp from vSphere

To make a virtual machine from the underlying vSphere infrastructure available to your vCloud Director server, you can import it and save it as a vApp.

Prerequisites

- Verify that you are connected to a vCloud Director server as a provider administrator.
- Verify that you are connected to a vCenter Server system.

Procedure

1 Retrieve the vSphere virtual machine that you want to import.

```
$myVm = Get-VM -Name 'MyVMToImport'
```

2 Retrieve the organization vDC to which you want to import the virtual machine.

```
$myOrgVdc = Get-OrgVdc -Name 'MyOrgVdc'
```

3 Import the virtual machine and store it as a vApp.

```
Import-CIVapp -VM $myVm -OrgVdc $myOrgVdc
```

Create and Modify a vApp

You can use vApp templates to instantiate vApps. After creating the vApp, you can modify its settings to minimize the consumption of computing and storage resources.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Retrieve the organization vDC to which you want to add the new vApp.

```
$myOrgVdc = Get-OrgVdc -Name 'MyOrgVdc'
```

2 Retrieve the source vApp template for your new vApp.

```
$myVAppTemplate = Get-CIVAppTemplate -Name 'MyVAppTemplate'
```

3 Create your new vApp.

```
$myVApp = New-CIVApp -Name 'MyVApp' -VAppTemplate $myVAppTemplate -OrgVdc $myOrgVDC
```

By default, the vApp is powered off.

4 Renew the runtime lease for the new vApp and set it to 12 hours.

```
Set-CIVApp -VApp $myVApp -RuntimeLease "12:0:0" -RenewLease
```

To set leases, you can use the *days.hours:minutes:seconds* syntax.

5 Start the new vApp.

```
Start-VApp -VApp $myVApp
```

Manage Virtual Machines with vApps

For a large-scale approach to administration, you can start, stop, or restart virtual machines or their guest operating systems by running cmdlets on the associated vApps.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Power on all virtual machines in all vApps with names starting with *MyVApp*.

```
Get-CIVApp -Name 'MyVApp*' | Start-CIVApp
```

2 Suspend all virtual machines in all vApps with names starting with *YourVApp*.

```
Get-CIVapp -Name 'YourVApp*' | Suspend-CIVApp
```

3 Power off all virtual machines in the vApp named *MyVApp1*.

```
Get-CIVapp -Name 'MyVApp1' | Stop-CIVApp
```

4 Shut down the guest operating systems of all virtual machines in the vApp named *MyVApp2*.

```
Get-CIVapp -Name 'MyVApp2' | Stop-CIVAppGuest
```

5 Restart the guest operating systems of all virtual machines in the vApp named *MyVApp3*.

```
Get-CIVapp -Name 'MyVApp3' | Restart-CIVAppGuest
```

6 Reset all virtual machines in the vApp.

```
Get-CIVapp -Name 'MyVApp4' | Restart-CIVApp
```

Manage Virtual Machines and Their Guest Operating Systems

For a targeted approach to administration, you can use the `CIVM` and `CIVMGuest` cmdlets to handle lifecycle operations for one or more virtual machines.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Retrieve all virtual machines with names starting with *MyVM* and power them on.

```
Get-CIVM -Name 'MyVM*' | Start-CIVM
```

2 Suspend all virtual machines with names starting with *YourVM*.

```
Get-CIVM -Name 'YourVM*' | Suspend-CIVM
```

3 Power off the virtual machine named *MyVM1*.

```
Get-CIVM -Name 'MyVM1' | Stop-CIVM
```

4 Shut down the guest operating system of the virtual machine named *MyVM2*.

```
Get-CIVM -Name 'MyVM2' | Stop-CIVMGuest
```

5 Restart the guest operating system of the virtual machine named *MyVM3*.

```
Get-CIVM -Name 'MyVM3' | Restart-CIVMGuest
```

6 Reset the nonresponsive virtual machine named *MyVM4*.

```
Get-CIVM -Name 'MyVM4' | Restart-CIVM
```

Retrieve a List of the Internal and External IP Addresses of Virtual Machines in vApps

When managing vApps in the cloud, you might need to obtain information about the NIC settings of the associated virtual machines.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Retrieve the organization for which you want to generate the report.

```
$myOrg = Get-Org -Name 'MyOrg'
```

2 Retrieve all vApps in the organization.

```
$vApps = Get-CIVApp -Org $myOrg
```

3 Populate an array with the information that you want to report.

```
$vAppNetworkAdapters = @()
foreach ($vApp in $vApps) {
        $vms = Get-CIVM -VApp $vApp
        foreach ($vm in $vms) {
                $networkAdapters = Get-CINetworkAdapter -VM $vm
                foreach ($networkAdapter in $networkAdapters) {
                        $vAppNicInfo = New-Object "PSCustomObject"
                        $vAppNicInfo | Add-Member -MemberType NoteProperty -Name VAppName -
Value $vApp.Name
                        $vAppNicInfo | Add-Member -MemberType NoteProperty -Name VMName   -
Value $vm.Name
                        $vAppNicInfo | Add-Member -MemberType NoteProperty -Name NIC      -
Value ("NIC" + $networkAdapter.Index)
                        $vAppNicInfo | Add-Member -MemberType NoteProperty -Name ExternalIP -
Value $networkAdapter.IpAddress
                        $vAppNicInfo | Add-Member -MemberType NoteProperty -Name InternalIP -
Value $networkAdapter.ExternalIpAddress
```

```
                        $vAppNetworkAdapters += $vAppNicInfo
                    }
                }
            }
```

Running this script retrieves the names of the virtual machines and their associated vApp, the IDs of the NICs of the virtual machines, and external, and internal IP addresses of the NICs.

4 Display the report on the screen.

```
$vAppNetworkAdapters
```

Create and Manage Access Control Rules

By defining access control rules you can assign levels of access to separate users, user groups, or everyone in the organization. You can define access control rules for catalogs and vApps.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Create a new rule for accessing the vApp named *MyVApp*.

```
New-CIAccessControlRule -Entity 'MyVApp' -EveryoneInOrg -AccessLevel "Read"
```

All users in the organization have read-only access to the vApp.

2 Modify the access rule for a trusted user who needs full control over *MyVApp*.

```
New-CIAccessControlRule -Entity 'MyVApp' -User "MyAdvancedUser" -AccessLevel "FullControl"
```

3 Restrict the full control access of *MyAdvancedUser* to read/write access.

```
$accessRule = Get-CIAccessControlRule -Entity 'MyVApp' -User 'MyAdvancedUser'
$accessRule | Set-CIAccessControlRule -AccessLevel "ReadWrite"
```

4 Remove the custom rule that you created earlier for *MyAdvancedUser*.

```
$accessRule | Remove-CIAccessControlRule
```

Filter and Retrieve vApp Networks

To generate reports about vApp networks, you need to retrieve the respective vApp networks. Use search criteria to filter the results returned by `Get-CIVAppNetwork`.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

- Retrieve the vApp network named *MyVAppNetwork*.

```
Get-CIVAppNetwork -Name 'VAppNetwork'
```

- Retrieve all vApp networks for the vApp named *MyVApp*.

```
Get-CIvapp -Name 'MyVApp'| Get-CIVAppNetwork
```

- Retrieve all vApp networks of connection type direct and direct fenced.

```
Get-CIVAppNetwork -ConnectionType Direct, DirectFenced
```

- Retrieve all direct vApp networks that connect to the organization network named *MyOrgNetwork*.

  ```
  Get-OrgNetwork -Name 'MyOrgNetwork' | Get-CIVAppNetwork -ConnectionType Direct
  ```

Create vApp Networks for a Selected vApp

To define how the virtual machines in a vApp connect to each other and access other networks, you need to create a vApp network. When creating the vApp network, you can select the settings for the network, or adopt them from an organization policy.

To address multiple networking scenarios for a vApp, you can create multiple vApp networks.

- Create an Isolated vApp Network on page 68

  When you do not want the virtual machines in a vApp to connect to objects outside the vApp, you must create an isolated vApp network.

- Create an NAT-Routed vApp Network on page 68

  To provide a vApp network with DHCP, firewall, NAT and VPN services, you must create it as an NAT-routed vApp network.

- Create a Direct vApp Network on page 69

  To establish a network connection between the virtual machines in a vApp and an organization network, you need to create a direct vApp network.

Create an Isolated vApp Network

When you do not want the virtual machines in a vApp to connect to objects outside the vApp, you must create an isolated vApp network.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Retrieve the vApp for which you want to create a vApp network.

```
$myVApp = Get-CIVapp -Name 'MyVApp'
```

2 Create the new vApp network with a selected gateway and network mask.

```
New-CIVAppNetwork -VApp $myVApp -Name 'MyVAppInternalNetwork' -Routed -Gateway '192.168.2.1'
-Netmask '255.255.255.0' -ParentOrgNetwork $null
```

By default, the vApp network has an enabled firewall.

Create an NAT-Routed vApp Network

To provide a vApp network with DHCP, firewall, NAT and VPN services, you must create it as an NAT-routed vApp network.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Retrieve the vApp for which you want to create a vApp network.

```
$myVApp = Get-CIVapp -Name 'MyVApp'
```

2 Retrieve the organization network to which you want to connect the vApp network.

```
$myOrgNetwork = Get-OrgNetwork -Name 'MyOrgNetwork'
```

3 Create the new vApp network with a selected gateway and network mask, defined pool of static IP addresses, and a disabled firewall.

```
New-CIVAppNetwork -VApp $myVApp -ParentOrgNetwork $myOrgNetwork -Name 'MyVAppInternalNetwork'
-Routed -Gateway '192.168.2.1' -Netmask '255.255.255.0' -DisableFirewall -StaticIPPool
"192.168.2.100 - 192.168.2.199"
```

If you do not run `New-CIVAppNetwork` with the `DisableFirewall` parameter, by default, the new vApp network has an enabled firewall.

Create a Direct vApp Network

To establish a network connection between the virtual machines in a vApp and an organization network, you need to create a direct vApp network.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Retrieve the vApp for which you want to create a vApp network.

```
$myVApp = Get-CIVapp -Name 'MyVApp'
```

2 Retrieve the organization network that you want to connect to.

```
$myOrgNetwork = Get-OrgNetwork -Name 'MyOrgNetwork'
```

3 Create a direct vApp network that connects to the selected organization network.

```
New-CIVAppNetwork -VApp $myVapp -Direct -ParentOrgNetwork $myOrgNetwork
```

By default, the new vApp network has an enabled firewall.

Modify or Remove vApp Networks

When a networking scenario for a vApp changes, you can modify the corresponding vApp network instead of creating a new one. If the networking scenario is no longer relevant, you can remove the related vApp network.

Prerequisites

Verify that you are connected to a vCloud Director server.

Procedure

1 Retrieve the vApp for which you want to modify vApp networks.

```
$myVApp = Get-CIVApp -Name 'MyVApp'
```

2 Modify the DNS and Static IP Pool settings for the vApp network named *MyVAppNetwork*.

```
Get-CIVAppNetwork -VApp $myVApp -Name 'MyVAppNetwork' | Set-CIVAppNetwork -PrimaryDns
10.17.0.94 -SecondaryDns 10.17.0.95 -DnsSuffix 'my.domain.com' -StaticIPPool "10.151.168.1 -
10.151.169.240"
```

3 (Optional) Remove *MyVAppNetwork*.

```
$myVApp | Get-CIVAppNetwork -Name 'MyVAppNetwork' | Remove-CIVAppNetwork
```

4 (Optional) Remove all isolated vApp networks for the vApp named *MyVapp*.

```
$myVApp | Get-CIVAppNetwork -ConnectionType Isolated | Remove-CIVAppNetwork
```

5 Retrieve the organization network named *MyOrgNetwork1*.

```
$myOrgNetwork1 = Get-OrgNetwork -Name 'MyOrgNetwork1'
```

6 Retrieve the organization network named *MyOrgNetwork2*.

```
$myOrgNetwork2 = Get-OrgNetwork -Name 'MyOrgNetwork2'
```

7 Redirect all vApp networks that connect to *MyOrgNetwork1* to connect to *MyOrgNetwork2*.

```
Get-CIVAppNetwork -ParentOrgNetwork $myOrgNetwork1 | Set-CIVAppNetwork -ParentOrgNetwork
$myOrgNetwork2 -NatEnabled $false -FirewallEnabled $false
```

The operation disables the firewall and NAT routing for all vApp networks that are connected to *MyOrgNetwork1*.

Index