vmware PRESS

DevOps for VMware Administrators

Trevor A. Roberts, Jr.
Josh Atwell
Egle Sigler
Yvo van Doorn

DevOps for VMware® Administrators

VMware Press is the official publisher of VMware books and training materials, which provide guidance on the critical topics facing today's technology professionals and students. Enterprises, as well as small- and medium-sized organizations, adopt virtualization as a more agile way of scaling IT to meet business needs. VMware Press provides proven, technically accurate information that will help them meet their goals for customizing, building, and maintaining their virtual environment.

With books, certification and study guides, video training, and learning tools produced by world-class architects and IT experts, VMware Press helps IT professionals master a diverse range of topics on virtualization and cloud computing and is the official source of reference materials for preparing for the VMware Certified Professional certifications.

VMware Press is also pleased to have localization partners that can publish its products into more than 42 languages, including, but not limited to, Chinese (Simplified), Chinese (Traditional), French, German, Greek, Hindi, Japanese, Korean, Polish, Russian, and Spanish.

For more information about VMware Press, please visit **vmwarepress.com.**

This page intentionally left blank

DevOps for VMware® Administrators

Trevor Roberts, Jr.
Josh Atwell
Egle Sigler
Yvo van Doorn

vmware® PRESS

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

DevOps for VMware® Administrators

Published by Pearson Education, Inc.

Publishing as VMware Press

ISBN-10: 0-13-384647-4

ISBN-13: 978-0-13-384647-8

Library of Congress Control Number: 2015900457

Printed in the United States of America

First Printing: April 2015

Warning and Disclaimer

Special Sales

VMWARE PRESS PROGRAM MANAGERS
Erik Ullanderson
Anand Sundaram

ASSOCIATE PUBLISHER
David Dusthimer

ACQUISITIONS EDITOR
Mary Beth Ray

VMWARE PRESS PROGRAM MANAGER
David Nelson

DEVELOPMENT EDITOR
Jeff Riley

MANAGING EDITOR
Sandra Schroeder

PROJECT EDITOR
Mandie Frank

COPY EDITOR
Keith Cline

PROOFREADER
Chuck Hutchinson

INDEXER
Cheryl Lenser

EDITORIAL ASSISTANT
Vanessa Evans

DESIGNER
Chuti Prasertsith

COMPOSITOR
Mary Sudul

This book would not have been possible without significant support. So, I dedicate this book to the following influences in my life: To God, for giving me the capability and the opportunities to work in a field that I love. To my wife, for putting up with the many hours of lab time spent in making this book. To my grandfather James, who left his home country of Trinidad and Tobago to make a better life for his children in America. To my parents, Trevor Sr. and Marva, for encouraging me to succeed and do my best, even when I didn't want to. To my brothers, Michael and Alfonso, for supporting any initiative that I set my mind to. To two special girls who inspire me to do what I do every day: Isla and Yazmine.
—Trevor Roberts, Jr.

This book is dedicated to my family, who continue to support me in everything I pursue. I especially want to thank my wife, Stephanie, who continues to be my greatest supporter and friend. I can always count on her to keep me grounded, focused, and inspired when I get frustrated.
—Josh Atwell

To my husband and love, Roy, for always providing moral, mental, and technical support.
—Egle Sigler

This is to you, Jackie. I wrote everything in Vim, you converted everything to a Word document.
—Yvo Van Doorn

This page intentionally left blank

Contents

About the Authors

Trevor Roberts, Jr. is a Senior Technical Marketing Manager for VMware. Trevor has the CCIE Data Center certification, and he is a VMware Certified Advanced Professional in the Data Center Design and Administration concentrations. In his spare time, Trevor shares his insights on data center technologies at http://www.VMTrooper.com, via the vBrownBag Professional OpenStack and Professional VMware podcasts, and on Twitter (@VMTrooper). His contributions to the IT community have garnered recognition by his designation as a VMware vExpert, Cisco Data Center Champion, and EMC Elect.

Josh Atwell is a Cloud Architect for SolidFire, focusing on VMware and automation solutions. Over the past 10+ years, he has worked very hard to allow little pieces of code to do his work for him through various automation tools. Josh is a father of two boys with wife Stephanie, and a daughter is on the way in early 2015. Based in the Raleigh, North Carolina, area, he enjoys time with his family, golf, audiobooks, and trying new bourbons. Josh has been highly active in the virtualization community, where he's been a leader of technology-based user groups such as CIPTUG, VMUG, and UCS Users Group. Josh has worked with others on preparing for their professional development pursuits through the vBrownBag podcast and the Virtual Design Master competition. Josh is also a regular public speaker and contributing author to the Mastering vSphere series. Never known for lacking an opinion, he blogs at vtesseract.com and talks shop on Twitter as @Josh_Atwell.

Egle Sigler (@eglute, anystacker.com) is currently a Principal Architect at Rackspace. She started her career as a software developer, and still has a soft spot for all the people that write, test, and deploy code, because she had a chance to do all of those tasks. Egle dreams about a day when writing, testing, and deploying code will be a seamless and easy process, bug and frustration free for all. Egle believes that knowledge should be shared, and tries to do so by writing this book, giving talks and workshops at conferences, and blogging.

Yvo van Doorn has more than a decade of system administration experience. The first part of his career, he manually built out and configured bare-metal servers. At Classmates, Yvo became a champion of configuration management and virtualization. Before joining Chef, he learned firsthand the power of VMware's products when he moved a small Seattle technology company's complete production stack over to its virtualization platform. He's a strong believer in the culture change that comes with DevOps. When he isn't busy spreading the gospel of Chef, he's probably enjoying a hoppy IPA, exploring the great outdoors, or celebrating his Dutch heritage while eating a wheel of gouda and watching Oranje lose the World Cup. Yvo lives with his wife and black lab in Seattle, Washington.

This page intentionally left blank

About the Reviewers

Scott Lowe, VCDX 39, has been in the IT industry for more than 20 years. He currently works as an engineering architect for VMware focusing on the intersection of network virtualization, open source, and cloud computing. He also spends time working with a number of DevOps-related products and projects.

Randall "Nick" F. Silkey, Jr. is a Senior Systems Engineer at TheRackspace Cloud. His passions revolve around both infrastructure automation and release engineering. He enjoys organizing several professional technical organizations in the Austin, Texas area. Nick has also spoken at local and national conferences about continuous integration and operations engineering. Apart from work, Nick enjoys spending time with his wife Wendy and raising their three children.

Matt Oswalt is an all-around technology nerd, currently focusing on bringing automation tools and methodologies to networking. He started in IT as an application developer for a large retail chain. After that, he spent four years as a consultant in the area of network infrastructure. He is now using both of these skillsets together in order to create more flexible, resilient, network infrastructure. He is heavily involved with automation and DevOps communities to help drive the conversation around network automation and SDN. He publishes his work in this space, as well as with traditional infrastructure, on his personal blog at keepingitclassless.net, and on Twitter as @Mierdin.

This page intentionally left blank

About the Contributing Author

Chris Sexsmith is a contributing author for the *DevOps for VMware Administrators* book. Chris has been a Staff Solutions Architect at VMware in the Global Center of Excellence for the past four years, focused primarily on automation, DevOps, and cloud management technologies. Chris lives in Vancouver, British Columbia, where he is working toward his MBA while watching as much hockey as humanly possible. Chris and his team lead the LiVefire program, focusing on specialist and partner solution enablement across the software-defined data center (SDDC).

This page intentionally left blank

Acknowledgments

Many people helped make this book possible, and I would like to thank them for their direct and indirect influence in getting the job done:

Gene Kim, for taking time out of his busy schedule with his own book project (*The DevOps Handbook*) and planning the DevOps Enterprise Summit to provide guidance on content for this book as well as on various aspects of the book production process.

Nick Weaver, for getting me started on my own DevOps journey by introducing the VMware community to Puppet through his work on Razor.

Joan Murray, for her strong support at VMware Press to get this book project off the ground.

Kelsey Hightower, for providing expert knowledge on Linux containers and how to orchestrate them at scale.

Aaron Sweemer, for providing contacts within VMware to share the company's DevOps vision with the readers of this book.

My co-authors, for their patience and continued support with my leadership of the book project.

Scott Lowe, Nick Silkey, and Matt Oswalt for providing invaluable feedback on my content in the book.

—Trevor Roberts, Jr.

I would like to acknowledge and thank a few individuals for their assistance in the writing of my portion of this book. Don Jones, Steven Murawski, and Alan Renouf provided me important guidance and feedback as I worked through various ways VMware administrators might benefit from PowerShell DSC. Without their insight and perspective I would likely still be playing in the lab and scratching my head. I would also like to thank Trevor Roberts, Jr. for inviting me to participate in this project. Finally, I'd like to thank the VMware community at large for their considered support and interest in this book. I hope you enjoy it as much as I do.

—Josh Atwell

The open source community, without you, we would not have these wonderful and amazing tools.

—Egle Sigler

First and foremost, I want to thank Trevor Roberts, Jr. for giving me the opportunity to participate in creating this book. To Mark Burgess, co-author of *Promise Theory: Principles and Applications*. Mark wrote the science behind today's configuration management that many of us use every day. Finally, I am grateful for everyone at Chef and in the Chef community that I was able to bounce ideas off of.

—Yvo van Doorn

We Want to Hear from You!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

We welcome your comments. You can email or write us directly to let us know what you did or didn't like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book's title and author as well as your name, email address, and phone number. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: VMwarePress@vmware.com

Mail: VMware Press
ATTN: Reader Feedback
800 East 96th Street
Indianapolis, IN 46240 USA

Reader Services

Visit our website at www.informit.com/title/9780133846478 and register this book for convenient access to any updates, downloads, or errata that might be available for this book.

This page intentionally left blank

Introduction

What is DevOps? Is it a product you can buy from a vendor that will cure all your IT woes? Is it an industry buzzword that analysts and marketers spawned to captivate the attention of CIOs everywhere? Although the IT community's coverage of DevOps may border on the edge of sensationalism, that is more a function of the benefits that DevOps can provide rather than mere industry hype.

DevOps is a term given to a set of practices, ideals, and tools that are helping organizations of various sizes deliver value on their IT investments at quicker rate. What exactly does this mean?

Think of the amount of time and the number of processes in your organization that are required to take a software project from the concept phase through software development and on to production deployment. The longer this process takes, the longer your IT organization takes to demonstrate value to the overall company. Thanks to the ubiquitous availability of technology, customers are expecting IT services to be delivered with the ease of a mobile application store. They will not wait years for a feature to be implemented, and a company that responds slowly to customer demands may find it hard to be successful long term.

How can DevOps solve the customer delivery speed issue? Configuration management technology, for example, can prevent server configuration drift and increase the speed at which new servers can be brought online to handle rapid growth in customer requests. Continuous integration can ensure that automated testing is performed on your software's source code as developers make their commits. These are just a couple examples of technologies and techniques that we discuss in this book.

Web-scale IT organizations like Etsy, Netflix, and Amazon Web Services are seen as the poster children for DevOps. However, the number of attendees at Gene Kim's DevOps Enterprise Summit attests to the value that DevOps can bring to traditional IT organizations as well.

So, brace yourself, DevOps is coming. The good news is that you can be empowered to contribute to the success of DevOps initiatives in your IT organization. This book aims to cover not just the high-level ideas of DevOps, but it also provides hands-on examples of DevOps tools and techniques in action.

About This Book

In our experience, DevOps concepts and tools can provide significant improvements in IT operations. While large IT organizations like Amazon and Rackspace are reaping the benefits of implementing DevOps in their environments, Enterprise IT organizations are still getting acquainted with DevOps practices.

This book aims to give the reader hands-on examples of the DevOps tools that IT organizations are using to achieve success.

You the Reader

This book is intended for system administrators with experience using VMware's vSphere hypervisor as well as Linux operating systems. We will provide step-by-step introductions to the use of software solutions used by DevOps practitioners with additional resources indicated in each chapter for follow-up study.

What This Book Covers

The topics this book covers begin with a high-level overview of what you, the virtualization specialist, can do to become knowledgeable on DevOps practices. Then, the book proceeds to discuss various tools that are used by DevOps practitioners.

Chapter 1 covers a discussion of DevOps concepts, including defining what this term means and why practices associated with DevOps can help your IT organization be more successful.

Chapter 2 introduces some of the popular tools for DevOps practitioners. Chapter 3 prepares us for setting up test environments to use with the sample code in this book.

Chapters 4–6 cover the Puppet configuration management solution, including a basic introduction, a multitier application deployment, and coverage of Puppet's integrations with managing VMware vSphere servers and virtual machines.

Chapters 7–9 cover the Chef configuration management solution, including a basic introduction, common system management tasks, and coverage of Chef's integrations with managing VMware vSphere environments.

Chapters 10 and 11 introduce you to the Ansible configuration management and orchestration solution. These chapters cover the foundational knowledge of this technology and various application deployments.

Chapter 12 covers the foundations of PowerShell Desired State Configuration (DSC), including architecture and primary use cases for this new feature of Microsoft Windows PowerShell. Sample code is provided to demonstrate the basic functionality of DSC as well as explanations of the various components that make up the feature.

Chapter 13 explores ways that VMware administrators may look to implement PowerShell DSC in their environment. This chapter targets specifically use cases where a VMware administrator, who may not be the Windows system administrator as well, can provide additional value and capabilities using DSC. Various methods are discussed in this chapter, with recommendations and limitations of each method highlighted and discussed where appropriate.

Chapter 14 discusses an application deployment paradigm that is relatively new to enterprise IT organizations: the use of Linux containers. The chapter discusses the basics of the Docker container management system, with hands-on examples.

Chapter 15 takes the Linux container discussion further with coverage of Google Kubernetes, an open source tool for managing containers at scale in your data center.

Chapter 16 describes how to set up, configure, and use Razor, a full lifecycle automated provisioning tool that combines install and server management with configuration tools. Chapter 16 walks you through all the key concepts and components of Razor. It starts out by describing how Razor works and how to best get started using it. Once you know what Razor is and how to use it for automated provisioning combined with DevOps tools, you will learn what the different functional components of Razor are. Finally, the chapter covers how best to install and configure Razor.

Chapter 17 gives an introduction to the Elasticsearch, Logstash, and Kibana, also known as ELK, stack. Each of these tools can be used standalone, but the use of all of them together makes a perfect combination for managing logs. This chapter covers each tool individually and how best to combine them and to harness their power for increased productivity while managing the logs.

Chapter 18 features Jenkins for continuous integration. It discusses how to automate delivery of code once it is committed to the source code repository.

Chapter 19 discusses VMware's own DevOps initiatives, including integrations with VMware vRealize Automation with DevOps tools and the new VMware vRealize Code Stream solution.

This page intentionally left blank

Chapter 3

Setting Up a DevOps Configuration Management Test Environment

Before we dive headfirst into implementing DevOps tools, let's examine how we can set up our test environments to adequately prepare ourselves. We'll also take a look at how we can start treating our infrastructure instructions just like a developer's source code.

Topics covered in this chapter include the following:

- Environment provisioning with AutoLab
- Environment provisioning with Vagrant
- Creating images with Packer
- Managing source code
- Git source code control

Environment Provisioning with AutoLab

AutoLab is a vSphere test lab provisioning system developed by Alastair Cooke and Nick Marshall, with contributions from others in the VMware community. AutoLab has grown so popular that at least one cloud provider (Bare Metal Cloud) allows you to provision AutoLab installations on its servers. As of this writing, AutoLab supports the latest release of vSphere, which will enable you to run a virtual lab that includes VSAN support. If you're unfamiliar with the tool, head over to http://www.labguides.com/ to check out the AutoLab link on the home page. After filling out the registration form, you will have access to the template virtual machines (VMs) to get your own AutoLab setup started. The team has a helpful selection of how-to videos on the AutoLab YouTube channel for anyone needing extra assistance.

Environment Provisioning with Vagrant

Vagrant is an environment provisioning system created by Mitchell Hashimoto and supported by his company, HashiCorp. Vagrant can help you quickly bring up VMs according to a pattern defined in a template file known as a Vagrantfile. Vagrant can run on Windows, Linux, and OS X (Mac) operating systems and supports popular desktop hypervisors such as VMware Workstation Professional, VMware Fusion Professional, and VirtualBox. Cloud providers such as Rackspace and Amazon Web Services can be used as well for your test environment. The Vagrant examples in this book are based on the VMware Fusion plugin, but the examples we provide can, with a few modifications, be used for other hypervisors and for cloud platforms. If you will be following along with the examples in this chapter, make sure to create a new directory for each Vagrantfile that you create.

NOTE

The VMware Fusion and Workstation providers require the purchase of a license for the plugin. See http://www.vagrantup.com/vmware for more details.

After installing Vagrant and the VMware Fusion or Workstation plugin, you need to find a Vagrant box to use with the system. A Vagrant box can be thought of as a VM template: a preinstalled operating system instance that can be modified according to the settings that you specify in your Vagrantfile. Vagrant boxes are minimal installations of your desired operating system with some boxes not being more than 300 MB. The idea is to present a bare-bones operating system that is completely configured by your automation tool of choice.

Some box creators opt to include the binaries for popular Vagrant-supported provisioners like Puppet, Chef, and so on, but other box creators do not, and users need to use the shell provisioner to deploy their favorite configuration management solution before it can be used. The box creation process is beyond the scope of this book. However, if you would like to develop your own boxes, there are tools like veewee and Packer (discussed later in this chapter) available that you can try out.

In previous versions of Vagrant, you had to specify the URL of the box file that you want to use when you initialized your vagrant environment:

```
vagrant init http://files.vagrantup.com/precise64_vmware.box
```

If the box file is located on your computer, you can specify the full path to the file instead of a URL.

Starting with Vagrant version 1.5, HashiCorp introduced the Atlas system (formerly known as Vagrant Cloud), an online repository that Vagrant will search for boxes if you use the account and box name of an image stored on its site:

```
vagrant init hashicorp/precise64
```

It is good to know both types of syntax because it will be necessary to use the old method for any boxes not hosted on Atlas. The online site is a great place to search for boxes for various operating systems instead of building your own.

The `vagrant init` command will automatically create a simple Vagrantfile that will reference the box that you specify. Listing 3-1 shows the default Vagrantfile that is generated with the command listed above.

Listing 3-1 Default Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Vagrantfile API/syntax version. Don't touch unless you know what you're
doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "hashicorp/precise64"
end
```

You'll notice that to save space I remove the comments that automatically get generated when you initialize your Vagrantfile. However, if you look in the comments, you'll see some helpful tips for using configuration management technology to make automated changes to your VM when it boots. As of today, the available options for provisioning your VM include basic shell scripts and configuration management tools such as Puppet, Chef, and Ansible. This is an immense value because your development and test environment can be stood up with the exact same settings that are used in your production deployments. This should cut down on the "well, it worked on my laptop" discussions that may go back and forth during a deployment mishap. Docker support was also added so that the provisioner could install the Docker daemon automatically and download the containers that you specify for use.

With your Vagrantfile in place, you can now boot your first test environment by using the following command:

```
vagrant up --provider=vmware_fusion
```

NOTE

If you're using VMware Workstation on the Windows or Linux platform, you would use a different provider: vmware_workstation.

You can now log in to the VM using the following command:

```
vagrant ssh
```

Take a look at a folder in your VM called /vagrant. You'll see that it contains your Vagrantfile! It's a shared folder that is automatically created for the VM so that you can easily transfer files to and from your desktop without having to use SCP, FTP, and so on.

If you examine the operating system resources, you'll notice that you have one vCPU and 512 MB of RAM. This may not be sufficient for the application that you want to run. So, we will take a look at how to modify the resources allocated to your Vagrant VM.

First, let's destroy this VM so that we can move on with the other configuration options. You can do this exiting the VM and then using the following command:

```
vagrant destroy
```

Vagrant will ask you to confirm that you really want to destroy this VM. Alternatively, you can use the `-f` option to skip that confirmation.

Listing 3-2 shows that Vagrant can modify the VM's VMX file to make the changes that we need. We use the `config.vm.provider` block of code to achieve this. By the way, the `memsize` attribute's units are megabytes. Notice that we are creating an object named `v` enclosed in vertical lines to change settings just for this VM. This object name has local scope only to this `config.vm.provider` statement, and it can be used again when defining other VMs, as you'll see in later examples. After executing `vagrant up`, the VM will be created with the desired attributes. At the time of this writing, the size and number of virtual disks cannot be controlled, but your Vagrant VMs will start with 40 GB of thin-provisioned storage.

Listing 3-2 Changing Default Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Vagrantfile API/syntax version. Don't touch unless you know what you're
doing!
VAGRANTFILE_API_VERSION = "2"
```

```
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "hashicorp/precise64"
  config.vm.provider :vmware_fusion do |v|
    v.vmx["memsize"] = 1024
    v.vmx["numvcpus"] = 2
  end
end
```

It is great that we can modify the VM's resources. What about a more complex setup, like multiple VMs? Vagrant supports such a topology as well. Of course, make sure that you have sufficient CPU cores and RAM to support the topology that you want to use! Multi-VM setups would be useful for testing realistic deployments with a separate database server and front-end server, for example. Listing 3-3 shows an example of a multi-VM Vagrantfile setup.

Listing 3-3 Multimachine Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Vagrantfile API/syntax version. Don't touch unless you know what you're
doing!
VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.define :first do |vm1|
    vm1.vm.box = "hashicorp/precise64"
    vm1.vm.hostname = "devops"
    vm1.vm.provider :vmware_fusion do |v|
      v.vmx["memsize"] = 1024
      v.vmx["numvcpus"] = 2
    end
  end

  config.vm.define :second do |vm2|
    vm2.vm.box = "hashicorp/precise64"
```

```
    vm2.vm.hostname = "vmware"
    vm2.vm.provider :vmware_fusion do |v|
      v.vmx["memsize"] = 1024
      v.vmx["numvcpus"] = 2
    end
  end
end
```

The deployment utilizes multiple `config.vm.define` blocks of code: one for each VM that we are creating. `:first` and `:second` are labels that Vagrant will use to identify the two VMs when you run commands like `vagrant status`. These labels will also be used to connect to the VMs via Secure Shell (SSH)—for example, `vagrant ssh first`. If you're familiar with Ruby, you'll notice that these labels are Ruby symbols. The names in the enclosed pipe symbols (for example, `|vm1|`) denote the object whose information that vagrant is using to build and define your VM. The object name can be the same as the symbol (for example, `first.vm.box...`), but it doesn't have to be.

Using this syntax can be a bit tedious when you want to deploy more than two VMs. Thankfully, because Vagrant is written in Ruby, you can use the language's features such as lists, loops, and variables to optimize your Vagrantfile code. Listing 3-4 shows some optimization tips that I learned from Cody Bunch and Kevin Jackson in their *OpenStack Cloud Computing* book

Listing 3-4 Optimized Multimachine Vagrantfile

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
servers = ['first','second']

Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/precise64"
  servers.each do |hostname|
    config.vm.define "#{hostname}" do |box|
      box.vm.hostname = "#{hostname}.devops.vmware.com"
      box.vm.provider :vmware_fusion do |v|
        v.vmx["memsize"] = 1024
        v.vmx["numvcpus"] = 2
      end
    end
  end
end
```

At the top of the file, I create a Ruby list called `servers` whose elements are the names of the VMs that I want to create. Then I use the Ruby list iterator called `each` to loop the execution of the VM definition for each element in the servers list. If we ever want to increase the number of VMs that are deployed, we just add more entries to the list. Not every VM needs to have the same set of resources, and we can use `if` statements within the `box.vm.provider` code block to be selective:

```
if hostname == "first"
  v.vmx["memsize"] = 3128
  v.vmx["numvcpus"] = 4
elsif hostname == "second"
  v.vmx["memsize"] = 1024
end
```

There are many more features in Vagrant that we will not be covering in this book, but with just these simple commands, you can build the test environment setups that we will be using in this book. If you'd like to learn more about Vagrant, be sure to check out the Vagrant website (http://www.vagrantup.com) and Mitchell's book, *Vagrant: Up and Running*.

Creating Images with Packer

Packer is another HashiCorp product that helps you to develop your own boxes for multiple platforms. Let's say you wanted to develop a VM image for Workstation/Fusion and ESXi from the same base box. Packer makes that possible.

Packer uses a JavaScript Object Notation (JSON) file format for you to specify how your Vagrant box will be configured (disk size, memory, and so on), and it will perform the initial OS deployment on your behalf once you specify the relevant automation parameters (for example, Ubuntu preseed files).

Packer is not just useful for creating Vagrant boxes; its main purpose is to produce image files that are compatible with popular cloud provider formats (OpenStack, AWS, and so forth). However, Packer includes a builder capability to automatically output Vagrant boxes that are compatible with VMware Fusion/Workstation and VirtualBox. Just like with Vagrant, popular configuration management technologies like Puppet and Chef can be used to customize the image that is produced.

Although we will not be discussing Packer in depth, we wanted you to be aware of it if you would like to experiment on your own with building custom Vagrant boxes. You can find more information about Packer at http://www.packer.io. If you would like to see examples of Packer definition files that you can use to develop your own VMs, the Chef team maintains a repository called *bento* on their Github account. https://github.com/chef/bento

Managing Source Code

Source code management (SCM) is an essential element in a DevOps environment. Think about it: If you will be turning your infrastructure into code, it is important that there is a way to review any changes and go back to different versions of a file in case new changes introduce problems (for instance, periodic instabilities in the best case, outages in the worst case). Some might think the "easy" way would be to make multiple copies of a file each with a unique name (Vagrantfile1, Vagrantfile2, Vagrantfile01012015, and so on), but then you have to deal with the hassle of renaming the file when you want to use it and trying to remember what was different about all the files.

The various teams in the development organization are most likely using some SCM system already to manage their work (for example, software developers storing their source code, QA teams managing their test scripts). As you begin using SCM technologies, it would be worthwhile discussing best practices with these other groups.

There are many SCM solutions, including SVN, Mercurial, and so on. Git happens to be one of the more popular SCM systems in the DevOps community. So, for this book, we use Git.

Using Git

Git is a distributed version control system, which means that you make a local copy of the central repository instead of just checking out individual files. Local commits can be synchronized back to the central server so that there is consistency in the environment, and users can always pull the latest version of the source from the central repository. This architecture differs from traditional source code management systems, in which only the central server ever has a complete copy of the repository.

As you work through the examples in this book, we recommend using one of the freely available online Git repositories, such as Bitbucket, GitHub, and Gitorious, as your central location for storing your code. Each site has its own unique features. For example, BitBucket allows unlimited free private repositories. GitHub is the online repository system that this book's authors used for their code. However, feel free to use whichever system meets your needs, as the methods by which you obtain code (clone/pull) and store code (push) are universal across any Git system.

NOTE

For projects in your production environment, consult with your legal department before considering using a public repository site. Although many offer private repository capabilities, your company leadership may prefer to use an internal central Git server.

Creating Your First Git Repository

First, install Git using your favorite package manager (for example, homebrew or macports on Mac OS X, apt-get on Ubuntu/Debian, yum on Red Hat/CentOS/Fedora). For Windows users, the popular online repositories like GitHub and BitBucket offer software clients that make it easy to interact with their online repository system. Alternatively, the http://git-scm.com site maintains a standalone install of the Git binary for Windows.

If you are using Linux or Mac OS X, you can open a terminal window to work with the following examples. Windows users must use the special shell that gets installed with whichever Git client that you use. The example syntax will be Linux/Mac-based, but the commands should be equivalent for the Windows platform.

Before we start writing any code, we need to set a couple global variables so that Git knows who we are. It's not as critical for local copies of the repository, but when we start pushing our code to a remote server, it will be very critical. The two global variables are your email address and username:

```
git config --global user.email "you@example.com"
git config --global user.name "Your Name"
```

As a matter of fact, if you try using Git and making your first commit without setting these variables, Git will prompt you to set them before you can continue.

If you followed along with the earlier Vagrant examples, you already have a directory of content that we can work with. Otherwise, create a new directory and create a text file in it. From here on out, make sure that you are in that directory on your command-line prompt.

First, let's initialize this directory to have a Git repository:

```
git init
```

If you do a listing of your directory with the option to show hidden files (`ls -a` on Linux/Mac or `dir /A:H` on Windows), you'll see that there is a hidden directory called .git. This directory contains your repository's files and settings specific to this repository. These local settings are combined with the global settings that we set earlier, and we can confirm this by using the following command:

```
git config -l
```

If you want to see the state of the files in your directory (has the file been added to the repository? Are there any changes since the last command? and so on), you can type **`git status`** and see output similar to what is shown in Listing 3-5.

Listing 3-5 Git Repository Status

```
git-test $ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .vagrant/
        Vagrantfile

nothing added to commit but untracked files present (use "git add" to track)
```

The last line is most important; it tells us that our files need to be tracked for the repository to manage it. This is important to take note of as putting files into the directory does not automatically get it tracked by the SCM tool. This feature prevents us from tracking junk files in the repository and wasting space.

Let's tell Git to track our Vagrantfile:

```
git add Vagrantfile
```

However, the .vagrant/ directory is not essential to be tracked because it only contains temporary files that Vagrant uses to set up your VM. We can explicitly tell Git to ignore this directory by creating a .gitignore file. Use your favorite text editor and create your .gitignore file with a single entry:

```
.vagrant/
```

Alternatively, you could use a simple `echo` command to accomplish the same thing. (Windows users will need to use the special Git Shell binary that is included with their Git install for this to work properly.)

```
echo '.vagrant/' > .gitignore
```

If you run the `git status` command again, you'll see that Git informs us about the .gitignore file as well. What gives? Remember that Git needs to be told what to do with any files or directories that the repository can see including the .gitignore file. Well, there are two ways to deal with your .gitignore file:

- Add the .gitignore file itself to the list of files and directories to ignore.
- Tell Git to track the .gitignore file as well.

I will use the second option so that anyone else who may use my repository will be able to ignore the appropriate files as well:

```
git add .gitignore
```

Now, if we check the status of the repository again, we should see output similar to what is shown in Listing 3-6.

Listing 3-6 Updated Repository Status

```
git-test $ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   .gitignore
        new file:   Vagrantfile
```

All the files in our directory are either ready to be committed or added to the .gitignore list of nonessential files and directories. So, all that's left to do is to commit the repository changes:

```
git commit
```

A file editor will automatically be opened so that you can enter details about the files that you are committing. (By default, vi is used.) See Listing 3-7.

Listing 3-7 Your Commit Message

```
git-test $ git commit
  1 This is my first commit.
  2 # Please enter the commit message for your changes. Lines starting
  3 # with '#' will be ignored, and an empty message aborts the commit.
  4 # On branch master
  5 #
  6 # Initial commit
  7 #
  8 # Changes to be committed:
  9 #       new file:   .gitignore
 10 #       new file:   Vagrantfile
 11 #
```

You must enter a message; otherwise, the commit will be canceled. If you are not familiar with vi, press I, type some text, press the Escape key, and then type **:wq** and press Enter. If you don't want to deal with the text editor, you can use the short form of the `git commit` command with the `-m` option to enter your commit message on the same line:

```
git commit -m "This is my first commit."
```

If the commit is successful, you should see the following output:

```
[master (root-commit) d962cd6] This is my first commit.
 2 files changed, 119 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Vagrantfile
```

Working with a Central Git Server (a.k.a. A Remote)

If you have opened an account on either GitHub, BitBucket, or whatever public Git repository site that you prefer, you will need to provide your computer's SSH public key to the site so that it can verify who you are. Each site may have a different way of doing this. So, consult the documentation for the appropriate steps. For Windows users, this is typically handled for you automatically by installing the client software for the site. Mac and Linux users must generate an SSH public key by using the `ssh-keygen` command.

Once your SSH public key is properly configured on your remote, it's time to create the repository on the remote site that you will be storing your files into: a process known as *pushing*. When you create your remote repository on the website, you should skip the automatic generation of the README file. After the repository is created, the site will provide you with a link that you can use to tell your local repository what is the location of your remote server, often labeled as origin.

In my setup, I gave the same name to my repository as I did to my local repository. This is optional, and the names can differ. I can use the `git remote` command with the link that GitHub gave me to update my local repository settings:

```
git remote add origin git@github.com:DevOpsForVMwareAdministrators/git
```

If I use the `git config -l` command, I will see new data about the location of my remote server:

```
remote.origin.url=git@github.com:DevOpsForVMwareAdministrators/git-test.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
```

I can now push the files from my local repository to my remote repository:

```
git push origin master
```

I should see output similar to what is shown in Listing 3-8.

Listing 3-8 Git Remote Push Results

```
git-test $ git push origin master
Warning: Permanently added the RSA host key for IP address '196.30.252.129'
to the list of known hosts.
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 2.13 KiB | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To git@github.com:DevOpsForVMwareAdministrators/git-test.git
 * [new branch]      master -> master
```

Figure 3-1 shows what the repository looks like on GitHub after I push the first commit to the remote server.

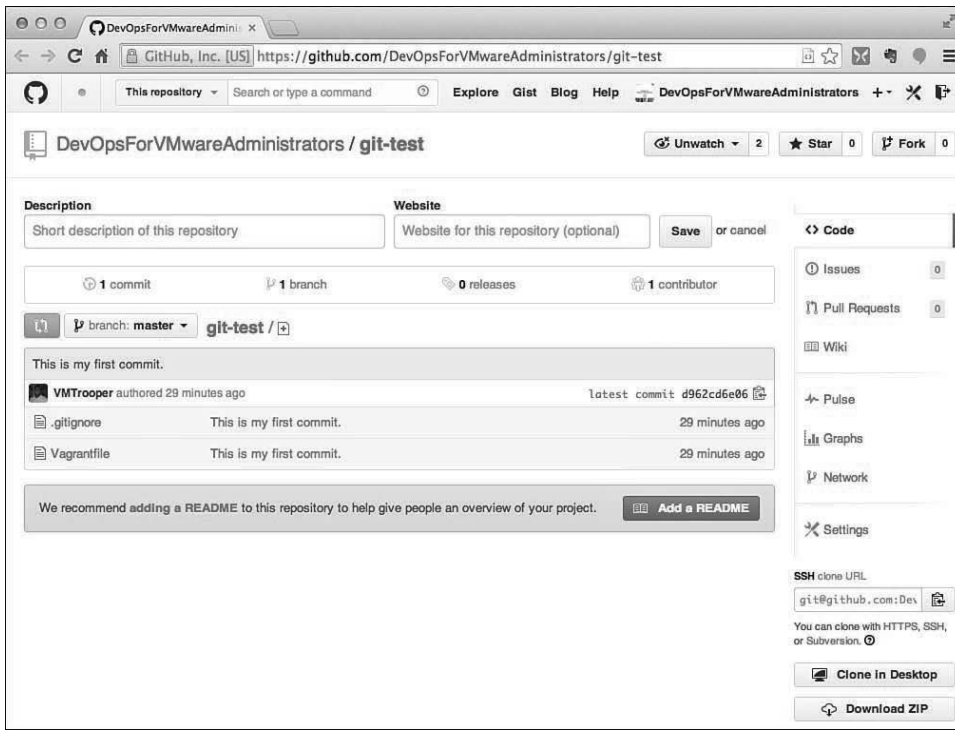

Figure 3-1 Remote Git repository

If there is a remote repository that you want to use, like the samples we're providing for this book, you can either use CLI-based methods or GUI-based methods to clone the remote repository to your local machine. Your remote repository system should have options similar to what is pictured on the lower-right corner of Figure 3-1.

The GUI-based methods will differ according to the site you are using. However, on GitHub, you can either use the Clone in Desktop button if you are using its Mac or Windows client, or you can use the Download Zip button, which will be a simple zip file that contains all the source code of the repository. If you are using the Windows or Mac platform, the Clone in Desktop option is recommended because it will create your Git remote link to the remote repository automatically.

The CLI-based method involves taking the SSH or HTTP clone URL and using the `git clone` command, similar to the following:

```
git clone https://github.com/DevOpsForVMwareAdministrators/git-test.git
```

After executing this command, Git will create a directory with the name of the repository (in this case, git-test) and copy the contents of the repository into that directory. You can begin using and updating the code, and the Git remote link will be created for you automatically just like with the Clone in Desktop button mentioned earlier. If you have permission to make changes to the remote repository, you can perform a Git push to forward any local changes to the remote repository. Requesting to make changes to others' repositories is not within the scope of this book; but if you are interested in the topic, you can research repository forks and pull requests. The implementation of these features may differ between the various public Git repository sites. So, check the appropriate documentation for the site that you are using.

If you are having issues working with the remote repository setup, an alternate workflow consists of the following:

1. Before you begin your work, create an empty repository on your remote Git site of choice (for instance, GitHub).
2. Clone the empty repository to your local machine using the GUI or CLI methods discussed earlier.
3. Begin working in the cloned directory and perform your commits there. You can then use the same `git push` command introduced earlier to forward your source code commits to the remote repository.

Summary

We discussed the foundational tools that we will use throughout the book to build our test environments, namely Vagrant and Git. Vagrant is useful for building your test environments quickly, and Git will help you keep track of changes that you make to your environment definitions in case you need to undo a change. In the next chapter, we begin discussing the Puppet configuration management technology with hands-on examples.

References

[1] http://docs.vagrantup.com/v2/

[2] https://help.github.com/

This page intentionally left blank

This page intentionally left blank

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q-R

S

T

U

V

W-Z