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.


VMware® Press is a publishing alliance between Pearson and VMware, and is the official publisher of VMware books and training materials that provide guidance for the critical topics facing today’s technology professionals and students.

With books, eBooks, certification 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 certification exams.

vmwarepress.com
Complete list of products • User Group Info • Articles • Newsletters

Make sure to connect with us!
vmwarepress.com
This page intentionally left blank
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
# Contents

About the Authors xvi
About the Reviewers xviii
Acknowledgments xix
About the Contributing Author xx
Introduction xxii
About This Book xxiii
You the Reader xxiii
What This Book Covers xxiii

## Part 1  Introduction to DevOps

### 1 Introduction to DevOps  1

- An Overview of DevOps Principles  1
- Implement Systems Thinking  3
  - Change the Approach to Team Interactions  3
  - Change the Approach to Infrastructure Deployment  5
  - Change the Approach to Software Development and Deployment  6
- Collect and Respond to Useful Systems Feedback Often and Adjust Accordingly  7
- Furthering Your DevOps Knowledge and Skills  7

- Summary  8
- References  8

### 2 DevOps Tools  9

- Organizing for Success: Kanban  9
- Server Deployment  13
- Configuration Management  14
- Continuous Integration  14
- Log Analysis  15

- Summary  15
- References  15

### 3 Setting Up a DevOps Configuration Management Test Environment  17

- Environment Provisioning with AutoLab  17
- Environment Provisioning with Vagrant  18
- Creating Images with Packer  23
Part 5 PowerShell 4.0

12 Introduction to PowerShell Desired State Configuration (DSC) 173

What Is PowerShell DSC? 174
PowerShell DSC Requirements 175
PowerShell DSC Components 175
  Native Cmdlets 175
  Managed Object Format File 176
  Local Configuration Manager 176
PowerShell DSC Configurations 178
PowerShell DSC Modes 180
  Local Push Mode 181
  Remote Push Mode 181
  Pull Mode 182
PowerShell DSC Resources 184
Summary 186
References 187

13 Implementation Strategies with PowerShell DSC 189

Use Cases for PowerShell DSC in VMware Environments 189
Scripted Deployments of VMs with PowerCLI 190
Incorporating PowerShell DSC in VM Templates 192
Challenges Implementing PowerShell DSC Configurations to New VMs 193
  PowerCLI Invoke-VMscript 193
  PowerCLI Copy-VMGuestFile 195
Kubernetes Cluster Nodes 223
Kubernetes Service Deployment 225
Kubernetes Workload Deployment 226
Platform-as-a-Service with Docker 230
Summary 231
References 231

Part 7  DevOps Tool Chain

16  Server Provisioning Using Razor 233
   How Razor Works 233
   Using Razor 236
      Razor Collections and Actions 238
      Building Razor Collections 245
   Using Razor APIs 257
   Razor Components 258
      Razor Server 258
      Razor Microkernel 258
      Razor Client 259
   Setting Up Razor 259
      PE Razor 259
      Puppet Install 259
      Install from Source 260
      Manual Release Install 260
      Other Services 260
   Summary 263
   References 263

17  Intro to the ELK: Elasticsearch, Logstash, Kibana 265
   Elasticsearch Overview 265
      Getting Started 266
      Understanding the Index 267
      Working with Data 267
      Installing Plugins 271
      Using Clients 274
   Logstash Overview 275
      Getting Started 276
      Configuring Input to Logstash 276
      Applying Filters 278
      Understanding Output 280
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.
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.
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
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 provider 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

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

```ruby
# -*- 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
end

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

```ruby
# -*- 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:

```ruby
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 --list
```

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
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](image-url)
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

Symbols

--link environment variables in Docker (listing 14-2), 207
l (pipeline), in PowerShell, 185
&& symbol, combining commands (Docker), 210

A

acceptance testing, 306
actions (Razor), 238
ActiveState Stackato, 231
activity log for nodes, viewing (listing 16-7), 243
ad hoc command execution (Ansible), 142-143
agility, 310-311
Anderson, David J., 11
another_tag.json file (listing 16-24), 255
another_tag.json file, alternate version (listing 16-25), 256
Ansible, 14, 139
architecture, 139-141. See also LAMP deployment (Ansible)
groups, 142
modules. See modules (Ansible)
playbooks, 144-151
roles. See roles (Ansible)
anible-doc command, 142
Ansible Galaxy, 156-157
ansible-galaxy init web utility, 153
ansible-galaxy install command, 157
ansible-playbook command, 146, 167
Ansible Tower, 140
ansible-vault encrypt command, 164
Apache Mesos, 231
apache.yml (listing 10-2), 145
apache.yml file, updated (listing 10-7), 153
APIs
  Chef servers, 87
  Razor, 257-258
Application Services, 313-315
  Puppet integration, 315-321
application tier
Ansible, 160-162
Puppet, 68-69

applications
defined, 199
dependencies, 200
isolation, 200. See also Linux containers, 200
multitier deployment, 226-230
architecture. See also workflow
Ansible, 139-141. See also LAMP deployment (Ansible)
Chef provisioning, 134-135
Jenkins, 292-293
PowerShell DSC, 175-178
Puppet, 33-37. See also LAMP deployment (Puppet)
assigning servers to environments (Chef), 118-119
Atlas, 19
attributes
Chef, 88
Puppet resources, 38
Auto Deploy, 13
AutoLab, 17
automation, 2
vRealize Application Services, 313-321
vRealize Automation, 312-313
vRealize Code Stream, 321-327

B
base image roles (Ansible), 164-166
before metaparameter, 39-41
bento repository, 23
Berkshelf, 105
Bitbucket, 24
boot_install.erb file for ESXi 5.5 (listing 16-4), 240
bootstrapping
nodes (Chef), 109-111
virtual machines with Knife, 130-134
brokers, adding to collections (Razor), 246-251
build artifacts (Jenkins), 304-306
build job definition (Jenkins), 298-302
Bunch, Cody, 22

C
CAMS (culture, automation, measurement, and sharing), 2
case conditional statement, 44-45
CD (continuous delivery), defined, 322
CD (continuous deployment)
CI (continuous integration) systems versus, 290
defined, 322
central Git server access, 28-30
chaining arrows, 41
checksum files, creating in PowerShell DSC, 183
Chef, 13-14, 85-86
Ansible versus, 140
attributes, 88
ChefDK, installing, 90-91
clients. See clients (Chef)
cookbooks. See cookbooks (Chef)
drivers, 135
environments. See environments (Chef)
Knife, 92-94
Knife plug-ins, 126-134
nodes. See nodes (Chef)
philosophies behind, 86-87
recipes. See recipes (Chef)
resources, 88, 134
roles, 88
run list, 88

servers. See servers (Chef)
terminology, 87-89

Chef brokers, 246
configuration file (listing 16-15), 249
sample file (listing 16-14), 248
chef-client --why-run command, 115
chef-client -z vsphere-metal.rb machine.rb
command, 138
chef command, 92
ChefDK, 90
installing, 90-91
chef gem install chef-provisioning
chef-provisioning-vsphere command, 135
chef gem install knife-vsphere
command, 128
chef gem install knife-windows
command, 93
chef gem install plugin command, 126
chef gem list knife-windows command, 92

Chef provisioning, 125, 134
architecture, 134-135
installing, 135
node configuration, 136-138
chef-provisioning-vsphere driver, 135
Chef Server, 89

CI (continuous integration), 5-6, 289-290.
See also Jenkins
CD (continuous deployment) systems
versus, 290
defined, 321
QA engineers and, 307
test automation, 290-292
classes (Puppet), 46-48
defined types versus, 78
clients, searching index (Elasticsearch)
with, 274-275
clients (Chef)
endpoint processing, 87

installing, 109-111
running, 113-115
clients (Docker), 204
clients (Razor), 259
cloning virtual machines with Knife,
130-134
cloud computing, stages of, 309
Cloud Foundry, 230
cloud provisioner (Puppet), 73
Puppet master preparation, 74-76
VM template preparation, 73
cloud provisioning in Puppet, 37
CloudForms, 13
cluster deployment (CoreOS), 217-222
cluster management utilities
(Kubernetes), 216
cluster nodes (Kubernetes), 223-225
cmdlets (PowerShell DSC), native, 175
Cobbler, 13
Code Stream, 321-327
collections (Razor), 238
building, 245-257
nodes, 241-245
tasks, 238-240
command module (Ansible), 144
commands
Ansible, ad hoc execution, 142-143
Razor, 236-238
comments
Ansible, 148
Dockerfile, 209
commit message (listing 3-7), 27
committing version control changes, 54
community cookbooks (Chef), 105
complexity of applications, 200
conditional expressions
Ansible, 146-151
Puppet manifests, 43-46
configuration files
etcd server (listing 15-1), 217
hiera.yaml (listing 5-6), 65
Kubernetes server (listing 15-2), 218
configuration management tools, 14
containers (Docker) versus, 211
Puppet. See Puppet
configuration resources (PowerShell DSC). See resources (PowerShell DSC)
configuration.yaml (listing 16-15), 249
configurations (PowerShell DSC), 178-180
configuring
etcd server, 222
input data (Logstash), 276-278
Kibana, 281-285
knife.rb file, 128-130
nodes with Chef provisioning, 136-138
servers (Jenkins), 296-298
Vagrant files, 20
conflicts of application dependencies, 200
connecting to running containers (Docker), 206-208
Console (Puppet Master component), 36
container-optimized operating systems, 213
containers (Docker)
building and distributing, 208
configuration management tools versus, 211
connecting to running, 206-208
deployment, 205-206
Dockerfile, 209-210
Docker Hub, 210-211
multitier application deployment, 226-230
orchestration, 213-214
viewing running, 206
virtual machines versus, 211
containers (Kubernetes)
deployment. See deployment (Kubernetes)
etcd, 214
pods, 214
workflow, 214-215
containers (Linux), 200-201
control groups, 201
management, 203
namespaces, 201-202
content attribute, 54
continuous delivery (CD), defined, 322
continuous deployment (CD)
continuous integration (CI) systems versus, 290
defined, 322
continuous integration (CI), 5-6, 289-290. See also Jenkins
continuous deployment (CD) systems versus, 290
defined, 321
QA engineers and, 307
test automation, 290-292
tools, 14
control groups (Linux), 201
controllers (Ansible), 139
Conway’s Law, 311-312
cookbooks (Chef), 86-88
community cookbooks, 105
root password management, 108, 115-116
time management, 105-107, 111-113
Cooke, Alastair, 17
Copy-VMGuestFile cmdlet, 195-196
CoreOS
as container-optimized, 214
cluster deployment, 217-222
cron scheduler, 300
Crowbar, 13
culture, 2
curl command, 266-267
custom data views (Kibana), 286
custom metadata, adding to nodes (Razor), 243

d garnet

Data

daemons (Docker), 204
dashboards. See Kibana
data
  adding to index (Elasticsearch), 268
custom viewing (Kibana), 286
input configuration (Logstash), 276-278
output (Logstash), 280
retrieving from index (Elasticsearch), 268
searching in index (Elasticsearch), 269-271
transforming with filters (Logstash), 279-280
data separation (Puppet), 58-59
  Hiera, 63-67
  node classification, 67
  params class, 59-63
database tier
  Ansible, 162-164
  Puppet, 70
Debian.yml variable file (listing 10-6), 150
default Vagrantfile (listing 3-1), 19
default Vagrantfile, changing (listing 3-2), 20
defaults folder (Ansible roles), 152
defined types (Puppet), 78
DeHaan, Michael, 13, 139
Deis, 231
deleting
  existing node (listing 16-8), 244
  virtual machines (listing 11-12), 170
dependencies
  application complexity, 200

community cookbooks (Chef), 105
conflicts, 200
deployment
  Ansible. See LAMP deployment (Ansible)
  containers (Docker), 205-206
  Jenkins, 293-296
  multitier applications, 226-230
  profiles, 314
  Puppet, 34-37. See also LAMP deployment (Puppet)
  scripted with PowerCLI, 190-192
  server. See Razor
deployment (Kubernetes)
  cluster management utilities, 216
  CoreOS cluster, 217-222
  etcd server configuration, 222
  Kubernetes cluster, 223-225
  Kubernetes service deployment, 225-226
  Kubernetes workload deployment, 226-230
  network overlays with flannel, 223
  software requirements, 215
Desired State Configuration (DSC). See PowerShell DSC
development team, QA engineers on, 306
DevOps
  Conway’s Law, 311-312
  defined, 309
  emergence of, 309-310
  knowledge development, 7
  principles, 1-3
DHCP (Dynamic Host Configuration Protocol), 217
  for Razor, 260
distribution of containers (Docker), 208
distributions of Puppet, 33
dnsmasq.conf file (listing 16-29), 261
DNS services for Razor, 260
Docker, 203
  client, 204
  containers. See containers (Docker)
  daemon, 204
  index, 205
  installing, 203-204
  Dockerfile, 209-210
  docker -h command, 204
  Docker Hub, 208-211
  docker ps command, 206
  docker push command, 211
  docker run command, 205-207
  docker version command, 204
  documents (Elasticsearch), 267
  downloading MySQL module, 70
  drivers (Chef), 135
  DSC (Desired State Configuration). See
  PowerShell DSC
  Dynamic Host Configuration Protocol
  (DHCP), 217
  for Razor, 260

E

  editing run list (Chef), 111-113, 119-120
  Elasticsearch, 15, 265
    documents, 267
    index. See index (Elasticsearch)
    for output (Logstash), 280
    plugins, installing, 271-274
    rivers, 273-274
    starting, 266
    elasticsearch-head plugin, 271-273
  ELK stack, 265. See also Elasticsearch;
    Kibana; Logstash
  Enable-PSRemoting cmdlet, 182
  encryption in Ansible, 164
  endif statement, 156
  endpoint processing (Chef), 87
  Enterprise Chef, 89
  environment provisioning
    AutoLab, 17
    Vagrant, 18-23
  environments (Chef), 88, 115
    assigning servers to, 118-119
    creating files, 116-117
    uploading files, 117-118
  environments (Puppet), test lab preparation,
  37-38
  ETCDCTL_PEERS environment
  variable, 222
  etcd distributed key-value store, 214
  etcd server configuration, 217, 222
  extending Knife, 125

F

  Fabric, 14
  facter binary (Puppet), 43
  facter command, 43
  facts (Puppet), 43
  files folder
    Ansible roles, 152
    Puppet, 49
  filters (Logstash), 278-280
  flannel, network overlays, 223
  FLEETCTL_ENDPOINT environment
  variable, 222
  fleetctl list-units command, 225-226
  Flynn, 231
  .fog file sample (listing 6-1), 74
  force parameter, 171
  Foreman, 13
  fragile boxes, 6
FROM command, 209
future use cases (PowerShell DSC), 197-198

G
Geppetto, 44
Gerrit, 14
Get-Command -Module
  PSDesiredStateConfiguration
cmdlet, 175
Get-Credential cmdlet, 194
Get-DscResource cmdlet, 182
  -Syntax parameter, 185
Get-DSCResource cmdlet, 184
Get-Help New-VM – Full command, 192
Get-LocalConfigurationManager
cmdlet, 177
Ghost, 13
Git, 14, 24
  central server access, 28-30
  installing, 25
  repository creation, 25-28
  repository status (listing 3-5), 26
  servers, hook feature, 300-304
  tracking files, 26
git clone command, 30
git config -l command, 28
git push origin master command, 28
git remote command, 28
git status command, 25
GitHub, 24
  .gitignore file, creating, 26
Gitorious, 24
gold images, 5
Graphite, 15
Grok, 278
Grok Debugger, 278
Grok filters (listing 17-14), 279

H
handlers folder (Ansible roles), 152
handlers: keyword, 146
Hashimoto, Mitchell, 18, 23
hello-world recipe (Chef), creating, 94-98
Hiera, 63-67
hiera command, 64
hiera_include command, 67
hiera.yaml (listing 5-6), 65
Hightower, Kelsey, 226
hook feature (Git servers), 300-304
Hosted Chef, 89
  assigning servers to environments,
  118-119
  root password management, 108, 115-116
  signing up, 100-102
  time management, 105-107, 111-113
  uploading environment files, 117-118
  workstation setup, 102-105
hostname namespace (Linux), 202
hosts: keyword, 145
HTML content creation, adding tasks
  (listing 10-10), 156
HTTP access for Razor, 260

I
idempotence (Chef), 86
if conditional statement, 44-45
ignore_errors parameter, 163
image creation (Packer), 23
include apache command, 54
include_vars: keyword, 154
index (Elasticsearch), 267
   adding data, 268
   retrieving data, 268
   searching data, 269-271
   searching with clients, 274-275
indexes (Docker), 205
   Docker Hub, 210-211
index.html.j2 web content template (listing 10-9), 155
infrastructure deployment, 5-6
inherits keyword, 61
init.pp file (Puppet initialization manifest), 49-51
input configuration (Logstash), 276-278
installing
   ChefDK, 90-91
   Chef provisioning, 135
   clients (Chef), 109-111
   Docker, 203-204
   Git, 25
   knife-vsphere plug-in, 128
   plugins (Elasticsearch), 271-274
   Razor, 259-261
Integrated Scripting Environment (ISE), 178
interpolation, 43
inventory file (Ansible), 142
Invoke-VMscript cmdlet, 193-195
IP addresses
   DHCP, 217
   etcd and Kubernetes servers, 221
   network overlays with flannel, 223
   for pods (Kubernetes), 215
ipmi.json file (listing 16-10), 245
IPMI support (Razor), 244-245
ISE (Integrated Scripting Environment), 178
isolation between applications, 200. See also Linux containers

J

Jackson, Kevin, 22
Jacob, Adam, 85
Java processes, checking running state (listing 16-30), 262
JavaScript Object Notation (JSON) format, 227
Jenkins, 14
   architecture, 292-293
deployment, 293-296
   plug-ins, 295
   workflow. See workflow (Jenkins)
Jinja2 templates, 154-156
JRuby, Razor clients, 259
JSON (JavaScript Object Notation) format, 227

K

Kanban system, 9-13
Kibana, 15, 265
   configuring, 281-285
custom data views, 286
   sharing and saving, 285-286
   starting, 280
Kim, Gene, 1
Knife, 92-94
   bootstrapping nodes (Chef), 109-111
   extending, 125
   plug-ins, 126-134
   running Chef client, 113-115
   searching with, 114
   knife bootstrap command, 110
knife cookbook list managedroot command, 108
knife cookbook list ntp command, 107
knife cookbook site download ntp command, 106
knife cookbook upload managedroot command, 108
knife cookbook upload ntp command, 106
knife environment from file command, 117
knife environment show command, 117
knife-esxi plug-in, 126
knife node list command, 111
knife.rb file, configuring, 128-130
knife ssh command, 113, 120-121
knife user list command, 104
knife vsphere datastore list command, 130
knife-vsphere plug-in, 126
  cloning and bootstrapping virtual machines, 130-134
  configuring knife.rb file, 128-130
  features, 127
  installing, 128
  validating configuration, 130
knife vsphere vm clone command, 131-133
knife-windows plug-in, 92
knowledge development, 7
kubectlf list minions command, 226
kubectlf list pods command, 230
kubectlf list services command, 230
kubectlf utility, 226
kube-kubelet.service (listing 15-4), 224
Kubelet, 215
kube-register service, 226
Kubernetes
  deployment. See deployment (Kubernetes)
etcd, 214
pods, 214-215
workflow, 214-215
KUBERNETES_MASTER environment variable, 226
Kubernetes Proxy, 215
Kubernetes server configuration file (listing 15-2), 218
kube-scheduler.service (listing 15-3), 224

L

LAMP (Linux-Apache-MySQL-PHP)
deployment (Ansible), 159
  application tier, 160-162
database tier, 162-164
  role structure optimization, 164-166
  web server deployment, 159-160
LAMP (Linux-Apache-MySQL-PHP)
deployment (Puppet), 57
  application tier, 68-69
database tier, 70
data separation, 58-67
  NTP servers, 70-71
lamp.yml (listing 11-10), 165
LCM (Local Configuration Manager), 176-178
  configuration example (listing 12-3), 180
lib folder (Puppet), 49
  --link environment variables in Docker (listing 14-2), 207
Linux containers, 200-201
  control groups, 201
  management, 203
  namespaces, 201-202
listings
  adding data to index, 268
  adding NTP deployment tasks to /etc/ansible/roles/web/tasks/main.yml, 160
  adding tasks to create HTML content, 156
  alternative Puppet manifest for Apache web server, 41
  another_tag.json file, 255
  another_tag.json file alternate version, 256
  Ansible ad hoc command execution, 143
listings

Ansible group_by function for module execution based on facts, 148
Ansible inventory file, 142
Ansible vars_files for module execution based on facts, 149
Ansible when: conditional statement for module execution based on facts, 147
Apache deployment tasks stored in web roles tasks and handlers subdirectories, 153
Apache module init.pp with Hiera, 64
API call to view tag details for ubuntu_small, 258
base role for essential system packages, 165
base role variable files, 165
basic required components (PowerShell DSC configurations), 179
boot_install.erb file for ESXi 5.5, 240
built-in Razor tasks, 239
changing default Vagrantfile, 20
check whether Java process is running, 262
commit message, 27
configuration file for Chef broker configuration.yaml, 249
console log output, 278
creating a VM, 190
creating MOF file, 194
creating new index, 267
creating policy, 252
creating Puppet class from existing manifest, 46
creating Razor broker, 248
creating Razor Chef broker sample_chef_broker.json file, 248
creating tag, 255
db role default value file, 164
db role variable files, 163
default Vagrantfile, 19
deleting existing node, 244
details for individual node, 241
Docker container deployment, 205
Docker --link environment variables, 207
Dockerfile for Nginx container, 209
etcd server configuration file, 217
example of user prompt, 192
filter for adding fields, 279
formatted search results, 269
formatted search results, searched for jenkins, 270
Git remote push results, 29
Git repository status, 26
Grok filters, 279
hiera.yaml, 65
implementing DSC via Invoke-VMscript, 196
index.html.j2 web content template, 155
init.pp file for Apache web server module, 50
ipmi.json file for setting IPMI credentials, 245
Jenkins deployment instructions, 294
kube-kubelet.service, 224
Kubernetes server configuration file, 218
kube-scheduler.service, 224
lamp.yml, 165
LCM configuration example, 180
list Elasticsearch indices, 267
listing Elasticsearch indices, 268
listing existing policies, 253
list of current nodes registered with Razor, 241
macs_tag.json file, 254
making initial change to LCM configuration, 178
multimachine Vagrantfile, 21
multiple outputs, 280
Logstash

MySQL deployment tasks in /etc/ansible/roles/db/tasks/main.yml, 162
MySQL pod definition JSON file (mysql-pod.json), 227
MySQL service definition JSON file (mysql-service.json), 228
operating system-specific YAML files, 66
optimized multimachine Vagrantfile, 22
original Apache module, 58
parameter class, 59
parameterized role call from database playbook file, 164
PHP deployment tasks in /etc/ansible/roles/php/tasks/main.yml, 161
PHP role variable files, 161
PHP values added to operating system family Hiera files, 68
policy.json, 251
Puppet listing your VMs, 75
Puppet manifest for Apache web server, 39
Puppet manifest using conditional statements to support multiple platforms, 45
Puppet manifest written with chaining arrows, 41
Razor client help and available commands, 236
RedHat.yml and Debian.yml variable files, 150
reinstalling existing node, 244
sample dnsmasq.conf file, 261
sample .fog file, 74
sample init.pp file for PHP module, 69
sample JSON query to search for jenkins, 271
sample puppetbroker.json file for creating Puppet broker, 247
search.rb file, 274
setting IPMI credentials on node, 245
site.pp entries for Puppet agent servers, 62
syslog.conf file, 277
tag details for ubuntu_small using Razor client, 257
tags portion of updated ubuntu_one policy, 256
testing Elasticsearch status, 266
unformatted log file entry, 279
updated Apache module, 60
updated apache.yml, 153
updated repository status, 27
vcenter::host Puppet defined type, 79
vCenterText.pp Puppet manifest, 77
vCenterText.pp Puppet manifest with ESXi shell and SSH enabled, 82
viewing all created brokers, 250
viewing individual details for Puppet broker, 250
viewing node2s activity log, 243
viewing policy details, 253
VMware virtual machine creation, 166
VMware virtual machine deletion, 170
VMware virtual machine modification, 170
vmware_esxi.yaml file for vmware_esxi task, 240
web and application pod definition JSON file (web-pod.json), 228
web and application service definition JSON file (web-service.json), 229
web server playbook: apache.yml, 145
Local Configuration Manager (LCM), 176-178
configuration example (listing 12-3), 180
local push mode (PowerShell DSC), 181
log analysis tools, 15
log management. See Elasticsearch; Kibana; Logstash
Logstash, 15, 265, 275-276
filters, 278-280
MOAR (manifest order analysis of resources), 42
modes (PowerShell DSC), 180
   local push mode, 181
   pull mode, 182-184
   remote push mode, 181-182
modifying virtual machines (listing 11-13), 170
modulepath Puppet setting, 48
modules (Ansible)
   ad hoc command execution, 142-143
   command, 144
   ping, 143
   setup, 144
   user, 144
   vsphere_guest, 166-171
modules (Puppet)
   application tier, 68-69
   creating, 48-50
   database tier, 70
   data separation, 58-67
   defined, 33, 46-48
   deploying, 71
   init.pp file, 50-51
   Puppet Forge repository, 48
   rtyler/jenkins, 293
   templates, 51-54
   usage example, 54
   version control commit, 54
   VMware management, 77-83
MOF (Managed Object Format) files, 176
   creating (listing 13-3), 194
master (Puppet), preparing, 74-76
master-agent deployment (Puppet), 34-37
Maven, 295
MCollective (Puppet Master component), 14, 37
measurement, 2
meta folder (Ansible roles), 152
metadata, adding custom to nodes (Razor), 243
metadata.json file, 50
metaparameters, 39
microkernel (Razor), 258
minions (Kubernetes), 226

M

machine resource (Chef), 134
macs_tag.json file (listing 16-22), 254
MAINTAINER command, 209
managed nodes (Ansible), 139
Managed Object Format (MOF) files, 176
   creating (listing 13-3), 194
manifest order analysis of resources (MOAR), 42
manifests (Puppet)
   conditional expressions and variables, 43-46
   defined, 33, 39-43
   vCenterText.pp (listing 6-3), 77
   vCenterText.pp with ESCi shell and SSH enabled (listing 6-5), 82
manual release, installing Razor, 260
Marshall, Nick, 17
Marvel plugin, 273
master (Puppet), preparing, 74-76
master-agent deployment (Puppet), 34-37
Maven, 295
MCollective (Puppet Master component), 14, 37
measurement, 2
meta folder (Ansible roles), 152
metadata, adding custom to nodes (Razor), 243
metadata.json file, 50
metaparameters, 39
microkernel (Razor), 258
minions (Kubernetes), 226
MOAR (manifest order analysis of resources), 42
modes (PowerShell DSC), 180
   local push mode, 181
   pull mode, 182-184
   remote push mode, 181-182
modifying virtual machines (listing 11-13), 170
modulepath Puppet setting, 48
modules (Ansible)
   ad hoc command execution, 142-143
   command, 144
   ping, 143
   setup, 144
   user, 144
   vsphere_guest, 166-171
modules (Puppet)
   application tier, 68-69
   creating, 48-50
   database tier, 70
   data separation, 58-67
   defined, 33, 46-48
   deploying, 71
   init.pp file, 50-51
   Puppet Forge repository, 48
   rtyler/jenkins, 293
   templates, 51-54
   usage example, 54
   version control commit, 54
   VMware management, 77-83
MOF (Managed Object Format) files, 176
   creating (listing 13-3), 194
mount namespace (Linux), 202
multimachine Vagrantfile
   listing 3-3, 21
   optimizing (listing 3-4), 22
multitier application deployment, 226-230.
   See also LAMP deployment (Ansible);
   LAMP deployment (Puppet)
mvn package command, 302, 305
MySQL module, downloading, 70
mysql-pod.json (listing 15-5), 227
mysql-service.json (listing 15-6), 228

N

name: keyword, 145
namespaces (Linux), 201-202
  mount, 202
  network, 202
  UTS, 202
namevar attribute (Puppet), 39
naming conventions, Puppet types, 40
native cmdlets (PowerShell DSC), 175
network namespace (Linux), 202
network overlays with flannel, 223
Network Time Protocol (NTP), 70-71
  with Chef, 105-107, 111-113
  server deployment (Ansible), 160
New-DscChecksum cmdlet, 183
New-VM cmdlet, 190
node classification (Puppet), 67
nodes (Chef)
  applying changes, 120
  assigning to environments, 118-119
  bootstrapping, 109-111
  configuring with Chef provisioning, 136-138
  editing run list, 111-113, 119-120
  running Chef client, 113-115
  validating policies, 120-122
nodes (Razor collection), 241-245
nodes, scaling, 314
Noop brokers, 246
noop command, 63
notify: keyword, 146
notify metaparameter, 39-41

NTP (Network Time Protocol), 70-71
  with Chef, 105-107, 111-113
  server deployment (Ansible), 160

O

Ohno, Taiichi, 10
Omnibus package (Chef), 90
OpenStack Cloud Computing (Bunch and Jackson), 22
operating system identifiers, 168
operating systems, container-optimized, 213
Opscode. See Chef
optimizing
  multimachine Vagrant files (listing 3-4), 22
  role structure (Ansible), 164-166
orchestration of containers, 213-214
order of recipes (Chef), 86
organizational management, 9-13
output (Logstash), 280

P

PaaS (platform-as-a-service) with Docker, 230
Packer, 23
params class, 59-63
password management (Chef), 108, 115-116
paths for Puppet modules, 51
people versus processes, 311-312
philosophies behind Chef, 86-87
The Phoenix Project (Kim), 1
Pienaar, R. I., 67
ping module (Ansible), 143
pipeline (l), in PowerShell, 185
platform-as-a-service (PaaS) with Docker, 230
playbooks (Ansible), 144-146
  conditional expressions and variables, 146-151
plays (Ansible), 144-145
plug-ins (Elasticsearch), installing, 271-274
plug-ins (Knife), 295
plug-ins (Knife), 126
  knife-exi, 126
  knife-vsphere, 126-134
  knife-windows, 92
pod definition file
  MySQL (listing 15-5), 227
  web and application (listing 15-7), 228
pods (Kubernetes), 214-215
policies (Chef), validating, 120-122
policies (Razor)
  adding tags, 254-257
  adding to collections, 251-254
policy.json (listing 16-18), 251
PostgreSQL, 260
PowerCLI
  Copy-VMGuestFile cmdlet, 195-196
  Invoke-VMscript cmdlet, 193-195
  scripted deployments with, 190-192
PowerShell DSC (Desired State Configuration), 14, 173-175
  architecture, 175-178
  configurations, 178-180
LCM (Local Configuration Manager), 176-178
  modes, 180-184
MOF files, 176
native cmdlets, 175
requirements, 175
resources, 184-186
use cases, 189-198
prefixes in Ruby, 52
principles of DevOps, 1-3
Private Chef, 89
private keys, starter kit (Chef), 102
processes versus people, 311-312
properties of resources (PowerShell DSC), 185
providers (Puppet resources), 38
provisioner (Chef provisioning), 134
provisioning servers. See Razor
pull mode (PowerShell DSC), 182-184
pull servers (PowerShell DSC), 192-193
Puppet, 13-14
  Ansible versus, 140
  architecture, 33-37. See also LAMP deployment (Puppet)
  broker, 247, 250
  classes, 46-48, 78
  cloud provisioner, 73-76
  defined types, 78
  distributions, 33
  installing Razor, 259
  integration with Application Services, 315-321
  manifests. See manifests (Puppet)
  master, preparing, 74-76
  modules. See modules (Puppet)
  resources, 33, 38-39
  test lab preparation, 37-38
puppet agent command, 63, 71
puppet apply command, 40
puppet cert list command, 63
puppet cert sign --all command, 38, 63
Puppet Enterprise
  brokers, 247
  installing Razor, 259
Puppet Forge repository, 48
puppet module generate command, 48
puppet module install command, 70, 293
puppet node install command, 76
roles (Ansible) 343

puppet node_vmware create command, 75
puppet node_vmware list command, 74
puppet node_vmware start command, 76
puppet node_vmware stop command, 76
puppet node_vmware terminate command, 76
puppetbroker.json file (listing 16-12), 247
PuppetDB (Puppet Master component), 36
push modes (PowerShell DSC)
  local push mode, 181
  remote push mode, 181-182
pushing, 28
  Git remote push results (listing 3-8), 29
Python, Ansible requirements, 139-140

Q-R

QA (quality assurance) teams, 306-307
Razor, 13, 233
  actions, 238
  APIs, 257-258
  client, 259
  collections, 238-257
  command syntax, 236-238
  installing, 259-261
  microkernel, 258
  server, 258
  testing, 259-261
  troubleshooting, 261-263
  versions, 236
  workflow, 233-235
razor add-policy-tag command, 256
razor brokers command, 250
razor <collection> command, 238
razor create-broker command, 247
razor create-policy command, 252
razor create-repo command, 246
razor create-tag command, 255
razor delete-node command, 244
razor –h command, 236
razor nodes command, 241
razor policies command, 253
razor reinstall-name command, 244
recipes (Chef), 86-88
  creating hello-world recipe, 94-98
  idempotence, 86
  order of, 86
  test-driven infrastructure, 87
Red Hat images, saving templates as, 131
Red Hat Satellite, 13
RedHat.yml variable file (listing 10-6), 150
redis-cli command, 206
reinstalling existing node (listing 16-9), 244
releases, installing Razor, 260
remote Git server access, 28-30
remote push mode (PowerShell DSC), 181-182
repositories
  adding to collections (Razor), 246
  central server access, 28-30
  creating, 25-28
  legal issues, 24
  Puppet Forge, 48
  updated status (listing 3-6), 27
require metaparameter, 39-40
requirements, PowerShell DSC, 175
resources (Chef), 88
  machine, 134
resources (PowerShell DSC), 184-186
resources (Puppet), defined, 33, 38-39
rivers (Elasticsearch), 273-274
roles (Ansible), 151-154
  Ansible Galaxy, 156-157
  application tier, 160-162
  base image, 164-166
database tier, 162-164
NTP server deployment, 160
path updates, 159
templates, 154-156
roles (Chef), 88
root password management (Chef), 108, 115-116
rylter/jenkins module (Puppet), 293
Ruby
with Chef, 90
prefixes, 52
Razor clients, 259
RUN command, 209
run list (Chef), 88
editing, 111-113, 119-120
running clients (Chef), 113-115

S

sample_chef_broker.json file (listing 16-14), 248
saving
dashboards (Kibana), 285-286
templates as Red Hat images, 131
scaling nodes, 314
SCM (source code management)
committing changes, 54
Git, 24-30
scope, explicitly defining variable, 60
scripted deployments with PowerCLI, 190-192
search.rb file (listing 17-9), 274
searching
data in index (Elasticsearch), 269-271
index (Elasticsearch) with clients, 274-275
with Knife, 114
Secure Shell (SSH), Ansible, 140
security
ekryption in Ansible, 164
Linux containers, 202
Select-Object cmdlet, 185
server configuration (Jenkins), 296-298
server deployment tools, 13
server provisioning. See Razor
servers (Chef)
API-based, 87
assigning to environments, 118-119
Chef Server, 89
Hosted Chef, 89, 100-105
root password management, 108, 115-116
time management, 105-107, 111-113
validating policies, 120-122
servers (Razor), 258
service: keyword, 146
service definition file
MySQL (listing 15-6), 228
web and application (listing 15-8), 229
service deployment (Kubernetes), 225-226
service requirements, installing Razor, 260-261
services
vRealize Application Services, 313-321
vRealize Automation, 312-313
vRealize Code Stream, 321-327
Set-DscLocalConfigurationManager cmdlet, 177
setup module (Ansible), 144
sharing, 2
dashboards (Kibana), 285-286
signing up for Hosted Chef, 100-102
site.pp file, 62
skills development, 7
snowflake servers, 5-6
software development/deployment, 6
source attribute, 54
tools

source code management (SCM)
  committing changes, 54
  Git, 24-30
source code, installing Razor, 260
spec folder (Puppet), 49
Splunk, 15
SSH (Secure Shell), Ansible, 140
stable agility, 310-311
standalone deployment (Puppet), 34
starter kit (Chef), private keys, 102
starting
  Elasticsearch, 266
  Kibana, 280
  Logstash, 276
subscribe metaparameter, 39-40
sudo ntpdate 0.pool.ntp.org command, 109
swim lanes, 13
syslog.conf file (listing 17-10), 277
Sysprep for Windows, 13
system, defined, 3
system facts, viewing, 144
systems feedback, 7
systems thinking
  defined, 3
  infrastructure deployment, 5-6
  software development/deployment, 6
  systems feedback, 7
  team interactions, 3-5

tags, adding to collections (Razor), 254-257
tasks
  adding for HTML content creation (listing 10-10), 156
  Razor collection, 238-240
tasks folder (Ansible roles), 152
tasks: keyword, 145
team interactions, 3-5
technical debt, 11
template keyword, 54
templates
  Jinja2, 154-156
  Puppet, 51-54
  saving as Red Hat images, 131
templates (VM)
  listing, 75
  PowerShell DSC incorporation in, 192-193
  preparing, 73
templates folder
  Ansible roles, 152
  Puppet, 49
terminology (Chef), 87-89
test automation, 290-292
  acceptance testing, 306
test environments
  AutoLab, 17
  Vagrant, 18-23
test lab preparation (Puppet), 37-38
test-driven infrastructure (Chef), 87
testing
  Elasticsearch status (listing 17-1), 266
  Razor, 259-261
tests folder (Puppet), 49
time management (Chef), 105-107, 111-113
  on virtual guests, 109
titles (Puppet resources), 38
tools
  configuration management, 14, 211
  continuous integration, 14
  image creation, Packer, 23
  log analysis, 15
  server deployment, 13
  source code management, Git, 24-30
  test environments, 17-23
work-in-progress management (Kanban system), 9-13
topology. See architecture
TorqueBox, 258
tracking Git files, 26
transforming data with filters (Logstash), 279-280
troubleshooting Razor, 261-263
client, 259
nodes, 243
trouble ticket systems, 9
types (Puppet resources), 38

V

Vagrant, 18-23
installing Razor, 260
Razor setup, 234
vagrant destroy command, 20
vagrant init command, 19
vagrant ssh command, 20
Vagrant: Up and Running (Mitchell), 23
Vagrant files
changing default (listing 3-2), 20
default (listing 3-1), 19
multimachine (listing 3-3), 21
optimized multimachine (listing 3-4), 22
validating
knife-vsphere plug-in configuration, 130
policies (Chef), 120-122
validation_key value (Razor brokers), 250
variable scope, explicitly defining, 60
variables
in Ansible, 146-151
Puppet manifests, 43-46
vars_files keyword, 150
vars folder (Ansible roles), 152
vcenter::host defined type, 78-81
vCenterText.pp Puppet manifest
with ESXi shell and SSH enabled (listing 6-5), 82
listing 6-3, 77
version control. See source code management (SCM)
versions of Razor, 236
viewing
all created brokers (listing 16-16), 250
individual details for Puppet broker (listing 16-17), 250
node2s activity log (listing 16-7), 243
policy details (listing 16-21), 253

U

ubuntu_one policy, tags portion (listing 16-26), 256
ubuntu_small tag details, viewing
with API call (listing 16-28), 258
with Razor client (listing 16-27), 257
unit testing, 290-292
update-alternatives --list java command, 297
uploading environment files (Chef), 117-118
use cases (PowerShell DSC), 189
challenges in implementation, 193
Copy-VMGuestFile cmdlet, 195-196
future use cases, 197-198
incorporating in VM templates, 192-193
Invoke-VMScript cmdlet, 193-195
lessons learned, 196-197
scripted deployments with PowerCLI, 190-192
user module (Ansible), 144
UTS namespace (Linux), 202
running containers (Docker), 206
system facts, 144
ubuntu_small tag details, 257-258
virtual guests
  bootstrapping nodes (Chef), 109-111
  editing run list (Chef), 111-113
  running Chef client, 113-115
  time management, 109
vm_hardware parameter, 168
VM templates
  listing, 75
  preparing, 73
VMs (virtual machines)
  cloning and bootstrapping with Knife, 130-134
  containers (Docker) versus, 211
  scripted deployments with PowerCLI, 190-192
VMware management modules
  Ansible, 166-171
  Puppet, 77-83
VMware vCenter Log Insight, 15
VMware vSphere operating system
  identifiers, 168
vmware_esxi.yaml file for vmware_esxi task
  (listing 16-3), 240
vmware_guest_facts parameter, 169
vmware/vcenter module (Puppet), 77-83
vRealize Application Services, 313-315
  Puppet integration, 315-321
vRealize Automation, 312-313
vRealize Code Stream, 321-327
vsphere_guest module (Ansible), 166-171
vsphere-metal.rb file, 137

W-Z

Weaver, Nick, 13
web server deployment (Ansible). See also
  playbooks (Ansible); roles (Ansible)
  NTP server updates, 160
  role path updates, 159
web server playbook: apache.yml (listing 10-2), 145
web tier (Puppet), data separation, 58-59
  Hiera, 63-67
  node classification, 67
  params class, 59-63
web-pod.json (listing 15-7), 228
web-service.json (listing 15-8), 229
when: conditional statement, 147
Willis, John, 2
with_items loop, 161-163
work-in-progress management, 9-13
workflow. See also architecture
  Kubernetes, 214-215
  Razor, 233-235
workflow (Jenkins), 296
  build artifacts, 304-306
  build job definition, 298-302
  Git hooks, 302-304
  server configuration, 296-298
workload deployment (Kubernetes), 226-230
workstation setup (Hosted Chef), 102-105