



Community Experience Distilled

# Learning Chef

Automate your infrastructure using code and leverage DevOps with Chef

**Rishabh Sharma  
Mitesh Soni**

**[PACKT]** open source\*  
PUBLISHING community experience distilled

# Learning Chef

Automate your infrastructure using code and leverage  
DevOps with Chef

**Rishabh Sharma**

**Mitesh Soni**



**BIRMINGHAM - MUMBAI**

# Learning Chef

Copyright © 2015 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: March 2015

Production reference: 1190315

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78328-521-1

[www.packtpub.com](http://www.packtpub.com)

# Credits

**Authors**

Rishabh Sharma  
Mitesh Soni

**Project Coordinator**

Kinjal Bari

**Reviewers**

Dr. Gaurav Bajpai  
Nitin Goel  
Frank Smieja

**Proofreaders**

Maria Gould  
Linda Morris

**Commissioning Editor**

Saleem Ahmed

**Indexer**

Tejal Soni

**Acquisition Editors**

Saleem Ahmed  
Rebecca Youé

**Graphics**

Valentina D'silva  
Abhinash Sahu

**Content Development Editor**

Adrian Raposo

**Production Coordinator**

Aparna Bhagat

**Technical Editor**

Utkarsha S. Kadam

**Cover Work**

Aparna Bhagat

**Copy Editor**

Neha Vyas

# About the Authors

**Rishabh Sharma** is currently working as a chief technology officer (CTO) at JOB Forward, Singapore (<http://job-fw.sg/>). JOB Forward was the first social recruiting platform in Singapore. Prior to working for JOB Forward, he worked for Wipro Technologies, Bangalore, as a solution delivery analyst. He was involved in the research projects of cloud computing, proof of concepts (PoC), infrastructure automation , big data solutions, and various giant customer projects related to cloud infrastructure and application migration.

He completed his master's thesis from Jaypee Institute of Information Technology, Noida, in cloud computing and has in-depth knowledge of distributed systems and cloud computing research issues of the industry. In a short span of time, he has worked on various technologies and tools, such as Java/J2EE, SAP(ABAP), AWS, OpenStack, DevOps, big data, and Hadoop.

He has authored four technical textbooks until now. He recently launched *Cloud Computing: fundamentals, industry approach and trends*, Wiley India publication, which is a comprehensive book on cloud computing and covers academic syllabi of all Indian universities. Before this, he has authored *Advance Computing Technology* for Gujarat Technical University (GTU) and *Software Quality Engineering and Mobile Computing* for Uttar Pradesh Technical University (UPTU). He has also authored many research papers in international journals and IEEE on a variety of issues related to cloud computing.

He is also an open source enthusiast and writes for the *Open Source For You* (OSFY) magazine. His other interests are mimicry, fun, horoscope reading, traveling, meditation, spirituality, and yoga. You can get in touch with him at [er.rishabh.sharma@gmail.com](mailto:er.rishabh.sharma@gmail.com).

---

I would like to express my special gratitude to my spiritual guru for his guidance and blessings. I am very grateful to my family for their support and encouragement during this project. I would like to give my special gratitude to Dr. Gaurav Bajpai and Nitin Goel for being the reviewers of my book and giving their precious feedback.

I am very thankful to Packt Publishing for providing me this opportunity to present this book and for their valuable support and guidance during this endeavor. Your views, comments, and suggestions are welcome.

---

**Mitesh Soni** is a technical lead who has 7.5 years of experience in the IT industry. He is a SCJP, SCWCD, and VCP. While he has interest in technology, his real passion is to play with kids and with his camera and capture photographs at Indroda Park. He lives in the capital of Mahatma Gandhi's home state. He loves to spend time alone and loves walking at Punit van.

---

I would like to dedicate this book to my professional and philosophical guide-cum-friend Vinay Kher, for believing in me when I lost myself; Simba, for inspiring me that I can do it; and Yohan Wadia, for being a great competitor.

I want to say thanks... and share my gratitude for everything I've been blessed with. I would like to thank mummy-papa, Jigisha-Nitesh, dada-dadi, Priyanka, and all family members who have encouraged me to take up the challenge of writing this book.

Feeling gratitude and not expressing it is like wrapping a present and not giving it, so huge thanks to Nalini, Aakanksha, Hemant-Priyanka, Mihir, Anupama, Ashish, Jamba, Nirali, Munal, Nitesh, Mayur, Chintan, Navrang, Dharmesh, Rohan, Jyoti, Vishwajit, Sree, and Rohini.

---

# About the Reviewers

**Dr. Gaurav Bajpai** received his BTech degree in computer science and engineering from Rohlkhanda University, India, in 2000; MTech degree in software engineering from Motilal Nehru National Institute of Technology, Allahabad, India, in 2005; and PhD in computer engineering from Uttar Pradesh Technical University, Lucknow, India, in 2006. He was an assistant professor in the departments of computer science and business administration at the Academy of Medical Sciences and Technology, Khartoum, Sudan, from April 2006 to March 2007. Since March 2007, he has worked as a senior lecturer in the department of computer engineering and information technology in the faculty of engineering at Kigali Institute of Science and Technology (KIST), which is now referred to as College of Science and Technology - University of Rwanda.

His research interests include software engineering, network routing, network hardware security, and biomedical engineering. He has published over 60 international journals and conference papers. He has convened, reviewed, been an editor, attended and presented in several workshops and seminars during his 14-year career from 2000. He has been on several international projects with income generation to University, as well.

He is a member of several distinguished organizations, such as ISOC and IEEE, Institution of Engineers. He is also a lifetime member of ISTE, CSI, and so on.

Currently, he is the head of the computer and software engineering at college of science and technology, University of Rwanda, Rwanda.

**Nitin Goel** (March 1987) received his MTech degree in computer engineering in 2011 and BTech degree in computer engineering in 2008 with honors from Kurukshetra University, Kurukshetra, Haryana, India. He has 2 years and 6 months of teaching experience and 1 year 2 months of IT experience as a software engineer for J2EE/J2ME(RIM). He has approximately 2 years of experience in the Intellectual Property Research (IPR) industry to date.

He has published 18 research publications and journals and conducted conferences internationally as well as nationally. His areas of interest are mobile ad-hoc network (MANETs) and sensor networks, application/desktop programming, Java, C, C++, and ns-2.

Presently, he is working with USA attorneys for USA patents' litigation; drafting, offensive/defensive review analysis, invalidity, source code review, prior art searches, landscaping, and benchmarking for the largest corporation in the valley.

**Frank Smieja** has many years of experience in running developmental, architectural, and engineering organizations and building applications for both large and small companies. In his career, he has worked in financial, telecommunications, software, and insurance sectors. Before entering the industry, he spent a number of years as a research scientist in Germany, building intelligent robots.

He is passionate about utilizing the value of technology to impact the bottom line of a business. After many years of successful implementation of the Agile methodology (XP, Scrum, Kanban, and Lean) within organizations, it is difficult for him to envisage doing any work in a non-Agile way. This includes the interaction between development and operations teams. Hence, the concept of DevOps has been an easy one that assimilates with his world view. It was therefore an obvious step for him to move from local, hand-crafted environments to cloud-based solutions configured and managed through a tool such as Chef. Given that he became an avid Ruby-on-Rails developer some years ago, it was a natural fit.

Frank is a freelance consultant who contracts through his own limited company, SmartaTech, which provides consultancy services apart from offering cloud-based applications, such as <https://urlpoke.com/> (to monitor your websites), and courses for smart thinking, <http://www.smartathinking.com/>.

[www.PacktPub.com](http://www.PacktPub.com)

## **Support files, eBooks, discount offers, and more**

For support files and downloads related to your book, please visit [www.PacktPub.com](http://www.PacktPub.com).

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.PacktPub.com](http://www.PacktPub.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [service@packtpub.com](mailto:service@packtpub.com) for more details.

At [www.PacktPub.com](http://www.PacktPub.com), you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<https://www2.packtpub.com/books/subscription/packtlib>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can search, access, and read Packt's entire library of books.

## **Why subscribe?**

- Fully searchable across every book published by Packt
- Copy and paste, print, and bookmark content
- On demand and accessible via a web browser

## **Free access for Packt account holders**

If you have an account with Packt at [www.PacktPub.com](http://www.PacktPub.com), you can use this to access PacktLib today and view 9 entirely free books. Simply use your login credentials for immediate access.

# Table of Contents

<b>Preface</b>	<b>xi</b>
<b>Chapter 1: An Overview of Automation and Advent of Chef</b>	<b>1</b>
<b>Automation</b>	<b>2</b>
<b>Why automation is needed</b>	<b>2</b>
<b>Introduction to Chef</b>	<b>8</b>
Why Chef is a preferred tool	9
The salient features of Chef	9
Automation with Chef	11
<b>Existing automation tools and comparison with Chef</b>	<b>12</b>
InstallShield	13
Features of InstallShield	13
Autolt	13
Features of Autolt	13
Windows PowerShell scripting	14
Features of PowerShell	14
CFEngine	14
Features of CFEngine	14
Puppet	15
Bcfg2	15
Cobbler	16
Sprinkle	17
cdist	17
Pallet	18
Rex	18
Glu	19
RunDeck	19
Crowbar	20
Fabric	20

---

*Table of Contents*

---

Ansible	21
SaltStack	21
Mina	22
Juju	22
Comparison with other popular tools	24
Chef versus Puppet	24
Chef versus CFEngine	25
<b>Self-test questions</b>	<b>26</b>
<b>Summary</b>	<b>27</b>
<b>Chapter 2: Different Components of Chef's Anatomy</b>	<b>29</b>
<b>The Chef automation</b>	<b>30</b>
<b>The Chef framework</b>	<b>30</b>
<b>Chef components</b>	<b>32</b>
The Chef server	32
Different types of Chef servers	32
Chef server tools	33
Different types of Chef server tools	34
Workstations	40
Workstation tools	41
Nodes	43
Types of nodes	43
Tools of a node	44
<b>The Chef server API</b>	<b>45</b>
GitHub	45
<b>Chef-solo</b>	<b>46</b>
<b>The Chef community</b>	<b>46</b>
Databases	47
Web servers	47
Process management	47
Programing languages	48
Monitoring	48
Package management	48
Virtualization and cloud	48
<b>Chef-run</b>	<b>49</b>
<b>Integration of Chef with Vagrant</b>	<b>50</b>
<b>A quick hands-on experience of a Hosted Chef server</b>	<b>50</b>
<b>Self-test questions</b>	<b>56</b>
<b>Summary</b>	<b>57</b>
<b>Chapter 3: Workstation Setup and Cookbook Creation</b>	<b>59</b>
<b>The VirtualBox installation</b>	<b>61</b>
<b>The Vagrant installation</b>	<b>66</b>

---

<b>The Git installation</b>	<b>69</b>
<b>Installation and configuration of a workstation</b>	<b>74</b>
<b>Workstation setup - creating a Chef repository</b>	<b>80</b>
Workstation setup using Git on Windows 8	80
Workstation setup without Git on CentOS	81
The Ruby installation and required settings	81
Setting up the Chef repository and downloading cookbooks	87
<b>Launching a virtual machine with Vagrant and a workstation setup</b>	<b>90</b>
<b>Creating and uploading a simple cookbook</b>	<b>92</b>
Uploading cookbooks	101
<b>Troubleshooting</b>	<b>101</b>
Error code – type 1	101
Meaning	101
Troubleshooting steps	102
Error code – type 2	102
Meaning	102
Troubleshooting steps	102
Error code – type 3	103
Meaning	103
Troubleshooting steps	103
Error code – type 4	103
Meaning	103
Troubleshooting steps	103
Error code – type 5	105
Meaning	105
Troubleshooting steps	105
Error code – type 6	105
Meaning	105
Troubleshooting steps	106
Error code – type 7	106
Meaning	106
Troubleshooting steps	106
<b>Self-test questions</b>	<b>106</b>
<b>Summary</b>	<b>107</b>
<b>Chapter 4: Learning about Cookbooks</b>	<b>109</b>
<b>Cookbook types</b>	<b>110</b>
Application cookbooks	110
Library cookbooks	110
Wrapper cookbooks	110
<b>Components of a cookbook</b>	<b>111</b>
<b>Attributes</b>	<b>113</b>

---

*Table of Contents*

<b>Definitions</b>	<b>114</b>
Syntax of a definition	115
Example of a definition	116
<b>Files</b>	<b>116</b>
Syntax of a file	117
Example of a file	117
<b>Libraries</b>	<b>118</b>
Syntax of a library	118
Example of a library	119
<b>Resources and providers</b>	<b>120</b>
<b>Syntax of resources</b>	<b>121</b>
<b>Example of resources</b>	<b>121</b>
<b>Templates</b>	<b>122</b>
Syntax of a template	122
Example of a template	122
An LWRP	123
Components of an LWRP	123
<b>Metadata</b>	<b>124</b>
The metadata.rb file	125
The Error message	125
<b>Self-test questions</b>	<b>126</b>
<b>Summary</b>	<b>126</b>
<b>Chapter 5: Managing the Nodes</b>	<b>127</b>
<b>Adding and deleting a node</b>	<b>127</b>
Adding a new node	128
Deletion of a node	131
Editing a node	131
<b>Bootstrapping target nodes</b>	<b>133</b>
The Knife.bootstrap command	134
The verification process for a node	135
<b>Introducing search</b>	<b>136</b>
Syntax of a search query	136
Search by different options	137
Search by node	137
Search by node and environment	138
Search for nested attributes	138
Search for multiple attributes	139
A partial search	139
<b>Introducing data bags</b>	<b>140</b>

---

<b>Introducing handler</b>	<b>145</b>
Types of handlers	145
Installation and configuration of a handler	145
The manual installation	146
Using chef_handler	146
Writing a simple handler	147
Open source handlers	147
<b>Self-test questions</b>	<b>148</b>
Summary	148
<b>Chapter 6: Working with an Open Source Chef Server</b>	<b>149</b>
<b>System requirements</b>	<b>150</b>
<b>Installing an open source Chef server</b>	<b>151</b>
FQDN and hostnames configuration	152
Restarting the virtual machine	153
Changing the hostname	154
<b>Installing an open source Chef server on a VMware Fusion virtual machine – Ubuntu 12.04</b>	<b>155</b>
VM machine settings	155
Installing an open source Chef server on a VM machine	155
Installing an open source Chef server on a VMware Workstation virtual machine – CentOS 6.x	156
Installing an open source Chef server on Amazon Web Services ( AWS )	167
Setting up the workstation	167
System requirements	168
Bootstrapping a node	176
The Nodes tab	178
The Clients tab	179
<b>Using community cookbooks</b>	<b>179</b>
<b>Upgradation of the open source Chef server</b>	<b>188</b>
Existing requirements	189
Accessing the Chef server 0.10.x	189
Downloading data from the Chef server 0.10.x	190
Accessing the Chef server 11.x	190
Updating Chef-validator settings	191
Verifying the admin public key	192
Verification of user passwords	192
Uploading data to the Chef server 11.x	193
The last steps	193
<b>Self-test questions</b>	<b>193</b>
Summary	193

---

*Table of Contents*

<b>Chapter 7: Working with the On-premises Chef Server Setup</b>	<b>195</b>
The on-premises Chef server	196
Benefits of on-premises Chef	196
Simple to scale	196
Completely automotive solution	196
Fast and easy configuration management	197
Reduced complexity within infrastructure	197
Improved data encryption policies	197
Types of on-premises Chef installations	197
Standalone on-premises Chef	197
Tiered on-premises Chef	198
High-availability on-premises Chef	198
Downloading the installation package	199
Prerequisites for the standalone on-premises Chef installation	202
Firewall requirements	202
Installing standalone on-premises Chef	203
Installing the on-premises Chef package on CentOS and Red Hat	203
Prerequisites for the tiered on-premises Chef installation	205
Load balancer requirements	206
Configuring api_fqdn	206
Firewall requirements	206
Ports for frontend servers	206
Ports for backend servers	206
Configuring the private-chef.rb file	207
The required settings for the backend server	208
The required settings for the frontend server	208
Adding on-premises Chef packages to servers	209
Installing tiered on-premises Chef	209
Configuring Bootstrap and installing on-premises Chef	210
Configuring the frontend server and installing on-premises Chef	210
Installing on-premises Chef packages	210
Prerequisites for the high-availability Chef installation	211
Load balancer requirements	211
Configuring api_fqdn	212
Ports for frontend servers	212
Ports for backend servers	212
Configure the private-chef.rb file	213
The required settings for the backend server with Bootstrapping	213
The required settings for other backend servers	213
The required changes for frontend entries	214
Installing the high-availability Chef server	216
Installing on-premises Chef on the backend server	216
Installing DRBD on the backend servers	217

---

---

*Table of Contents*

The DRBD configuration on the backend Bootstrap server	217
The DRBD configuration on the backend non-Bootstrap server	217
Configuring on-premises Chef on the Bootstrap backend server	220
Configuring on-premises Chef on the non-Bootstrap backend server	220
Configuring and installing on-premises Chef on the frontend servers	221
Installing on-premises Chef packages	221
<b>Managing on-premises Chef</b>	<b>221</b>
Service commands	222
Viewing Chef commands	222
Uninstalling on-premises Chef	222
View configuration	222
Reconfiguring Chef	222
Service subcommands	223
The hup subcommand	223
The int subcommand	223
The kill subcommand	223
The once subcommand	224
The service-list subcommand	224
The start subcommand	224
The restart subcommand	225
The stop subcommand	225
The status subcommand	226
The tail subcommand	226
The term subcommand	227
Log files	227
<b>Self-test questions</b>	<b>227</b>
<b>Summary</b>	<b>228</b>
<b>Chapter 8: Managing Chef on Cloud Infrastructure</b>	<b>229</b>
<b>What is cloud computing?</b>	<b>230</b>
<b>Why Chef with cloud infrastructure?</b>	<b>232</b>
<b>AWS EC2 bootstrapping using Chef</b>	<b>232</b>
Preparing your workstation	232
Installing the knife-ec2 plugin	233
Configuring the AWS settings and the knife.rb file	233
Configuring knife.rb with your AWS Cloud credentials	235
Bootstrapping the EC2 instance	235
Various configuration options	236
The expected output	236
Running the Chef-client on the new client node (cloud instance)	238
Verification of the complete installation	240
Managing recipes on the new client node	241
Running the Chef-client as a daemon	242
<b>Rackspace Cloud server bootstrapping</b>	<b>243</b>
The prerequisite to work with Rackspace Cloud	243

---

*Table of Contents*

Installing plugins for knife-rackspace	243
Preparing the workstation with Rackspace credentials	243
Bootstrapping the Rackspace Cloud server with the Chef-client	244
Deleting Rackspace servers	245
The Knife-cloud plugin	245
<b>VMware and Chef</b>	<b>245</b>
<b>Self-test questions</b>	<b>246</b>
<b>Summary</b>	<b>246</b>
<b>Chapter 9: Best Practices while Using Chef</b>	<b>247</b>
<b>Chef anti-patterns and patterns</b>	<b>247</b>
A wrapper cookbook	249
A default cookbook	250
<b>Testing cookbooks</b>	<b>252</b>
Types of cookbook tests	252
Checking the syntax	252
Integration testing	252
Checking the result	253
Checking the consistency	253
Checking the performance	253
<b>Best practices for effective usage of Chef</b>	<b>254</b>
Planning in advance	254
Designing a cookbook	254
Using a private recipe	255
Avoiding the use of one giant cookbook	255
Avoid overloading of a Chef environment	255
<b>Self-test questions</b>	<b>256</b>
<b>Summary</b>	<b>256</b>
<b>Chapter 10: Case Studies on Different Chef Deployments</b>	<b>257</b>
<b>Case studies of Hosted Chef deployments</b>	<b>258</b>
<b>Admeld</b>	<b>258</b>
Challenges with the infrastructure of Admeld	258
The solution with Hosted Chef	259
The final outcome	259
<b>Fanhattan</b>	<b>260</b>
Challenges with the infrastructure of Fanhattan	260
The solution with Hosted Chef	260
The final outcome	261
<b>Zumba Fitness</b>	<b>261</b>
Challenges with the infrastructure of Zumba Fitness	261
The solution with Hosted Chef	261
The final outcome	262

---

---

*Table of Contents*

The Limelight video platform	263
Challenges with the infrastructure of Limelight	263
The solution with Hosted Chef	263
The final outcome	264
<b>Imagination</b>	<b>264</b>
Challenges with the infrastructure of Imagination	264
The solution with Hosted Chef	264
The final outcome	265
<b>Getaroom</b>	<b>265</b>
Challenges with the infrastructure of Getaroom	266
The solution with Hosted Chef	266
The final outcome	267
<b>Case studies of Private Chef deployment</b>	<b>268</b>
Ancestry.com	268
Challenges with the infrastructure of Ancestry.com	268
The solution with Private Chef	269
The final outcome	269
Facebook	270
Challenges with the infrastructure of Facebook	270
The solution with Private Chef	270
The final outcome	271
DreamHost	271
Challenges with the infrastructure of DreamHost	271
The solution with Private Chef	272
The final outcome	272
<b>Case studies of the open source Chef deployment</b>	<b>273</b>
SolutionSet	274
Challenges with the infrastructure of SolutionSet	274
The solution with open source Chef	274
The final outcome	275
<b>Case studies of the Chef-solo deployment</b>	<b>275</b>
Wharton School - University of Pennsylvania	275
Challenges with the infrastructure of Wharton School	275
The solution with Chef-solo	276
The final outcome	276
<b>Self-test questions</b>	<b>277</b>
<b>Summary</b>	<b>277</b>
<b>Index</b>	<b>279</b>

---



# Preface

Chef is an open source configuration management tool that helps to transform infrastructure into simple code. It implies that the process of building and rebuilding IT infrastructure, configuration management, and scaling is possible in a short span of time in accordance with the customer's needs in different deployment environments. Businesses can gain huge competitive advantage in the increasingly digital economy by enabling business transformations and by automating the pipeline of building, testing, and deploying new applications. It also provides a way to implement innovative features by utilizing less time and saving costs.

Organizations expect quick response in terms of application delivery and industries to gain competitive advantages and hence, DevOps culture is getting popular day by day. The DevOps culture provides effective collaboration, efficient integration, and effectual communication and feedback between software developers and operations teams. In order to fulfill the requirements of DevOps, configuration management tools, such as Chef, are used to fill the gap. DevOps includes a continuous delivery system that is an automated process to accelerate the release of an application. Chef is an automation platform for continuous delivery, as it provides a way to model and manage infrastructure as a code. With Chef, it is possible to achieve faster or lesser time to market and good/high quality application.

Chef was built from the very beginning with the cloud concepts in mind. Chef allows you to dynamically provision and deprovision cloud resources to keep up with the demands when the usage is high and when there is a peak in traffic. With Chef, you can take advantage of all the flexibility and cost savings that the cloud offers. Chef is integrated with all the major cloud service providers, such as Amazon EC2, VMware, IBM SmartCloud, HP Cloud, OpenStack, Windows Azure, Joyent Cloud, Rackspace, Google Compute Engine, and so on.

This book covers all the basic and architectural concepts of Chef with step-by-step explanations, hands-on exercises, and screenshots. Some best practices and customer's case studies are also included. They provide you with practical understanding to automate your infrastructure into code.

## What this book covers

*Chapter 1, An Overview of Automation and Advent of Chef*, introduces automation and discusses its need in the current market scenario. It also describes the DevOps culture in detail with its benefits in the Agile development. This chapter discusses the importance of cloud computing and how DevOps and cloud computing can bring competitive advantages to businesses. It includes an introduction of Chef, its features, the existing configuration management tools and their comparison.

*Chapter 2, Different Components of Chef's Anatomy*, covers the Chef automation, Chef framework, Chef components and the types of Chef servers. It includes information of popular community cookbooks. This chapter provides a quick, hands-on experience of the Hosted Chef server.

*Chapter 3, Workstation Setup and Cookbook Creation*, delves into VirtualBox, Vagrant, Git, and the Ruby installation. It also covers a step-by-step explanation of the workstation setup and how to create a Chef repository. This chapter also includes details on how to launch a virtual machine with Vagrant and a workstation setup; here, you will learn steps to create and upload a simple cookbook.

*Chapter 4, Learning about Cookbooks*, focuses on the types of cookbooks and components of a cookbook, such as definitions, files, libraries, resources and providers, templates, lightweight resources, and metadata in detail.

*Chapter 5, Managing the Nodes*, covers basic operations, such as adding, deleting nodes and bootstrapping the target nodes. This chapter also introduces a search facility, data bags, and types of handlers. You will also learn different phases of a Bootstrap operation here.

*Chapter 6, Working with an Open Source Chef Server*, focuses on in-depth explanation of how to install the open source Chef server, and how to upgrade it. This chapter also covers details of how to use and upload a community cookbook of Tomcat on an open source Chef server and how to install the Tomcat cookbook on a node registered with the open source Chef server.

*Chapter 7, Working with the On-premises Chef Server Setup*, introduces the Private Chef server and its benefits, types of Private Chef installations, prerequisites and installing procedure for the standalone Private Chef, tiered Private Chef, high-availability Chef and also details of how to manage Private Chef.

*Chapter 8, Managing Chef on Cloud Infrastructure*, covers the basic concepts of cloud computing, details of a Cloud's infrastructure, and why Chef can be used with a Cloud infrastructure. This chapter also describes the AWS EC2 bootstrapping and Rackspace Cloud server bootstrapping using Chef in detail.

*Chapter 9, Best Practices while Using Chef*, introduces Chef's anti-patterns and patterns in a predefined framework. This chapter will give you insight of how to test cookbooks and best practices for usage of Chef.

*Chapter 10, Case Studies on Different Chef Deployments*, focuses on case studies of the Hosted Chef deployment, Private Chef deployment, open source Chef deployment, and Chef-solo deployment. This chapter also describes the advantages of Chef in the current IT scenario across different industries.

## What you need for this book

This books assumes that you are aware of the fundamental concepts of object-oriented programming and have knowledge of at least one programming language. Basic understanding of the client-server approach will also be beneficial here. Chef uses Ruby for its components and hence, familiarity with Ruby is an additional value that you may have.

Hands-on experience of the installation of different software on Windows or packages on Linux-based OSes is essential to install Chef and to execute various commands. Approach in this book uses Windows PowerShell and a CentOS terminal to configure and Bootstrap; hence, administrative access to use applications such as PowerShell is necessary. It also covers details of how to manage resources on a cloud as well as how to use Hosted Chef. For this purpose, it is desirable to have the basic understanding of concepts such as cloud service models and cloud deployment models. Though cloud concepts are covered in brief phases, which will be useful to have better understanding of cloud concepts.

Furthermore, you will need Internet access to download software packages that you do not already have. You will also need to install the Ruby programming language version 1.9 or a higher version, Chef client, Chef server, and other open source software.

## Who this book is for

This book is aimed at beginners, developers, system administrators, Linux administrators, cloud developers, or cloud administrators who are on a path to learn and apply Chef automation on existing or new infrastructure. This book includes the open source Chef server and Hosted Chef server to give a basic understanding of both with easy examples of cookbooks. In a nutshell, anybody who wants to know "how it works?" and "how to use it?" will get benefited by the content and representation of it from this book.

## Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "We can include other contexts through the use of the `include` directive."

Any command-line input or output is written as follows:

```
gpg --keyserver hkp://keys.gnupg.net --recv-keys D39DC0E3
```

New terms and important words are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Search **Windows PowerShell** in your system and open it."



Warnings or important notes appear in a box like this.



Tips and tricks appear like this.

## Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail [feedback@packtpub.com](mailto:feedback@packtpub.com), and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at [www.packtpub.com/authors](http://www.packtpub.com/authors).

## Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

## Downloading the example code

You can download the example code files from your account at <http://www.packtpub.com> for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

## Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/submit-errata>, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to <https://www.packtpub.com/books/content/support> and enter the name of the book in the search field. The required information will appear under the **Errata** section.

## Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.



# 1

## An Overview of Automation and Advent of Chef

*"The first rule of any technology used in a business is that automation applied to an efficient operation will magnify the efficiency. The second is that automation applied to an inefficient operation will magnify the inefficiency."*

- Bill Gates

Before moving to the details of different Chef components and other practical things, it is recommended that you know the foundation of automation and some of the existing automation tools. This chapter will provide you with a conceptual understanding of automation, and a comparative analysis of Chef with the existing automation tools.

In this chapter, we will cover the following topics:

- An overview of automation
- The need for automation
- A brief introduction of Chef
- The salient features of Chef
- Automation with Chef
- Existing automation tools and comparison with Chef

## Automation

Automation is the process of automating operations that control, regulate, and administrate machines, disparate systems, or software with little or no human intervention. In simple English, automation means automatic processing with little or no human involvement.

An automated system is expected to perform a function more reliably, efficiently, and accurately than a human operator. The automated machine performs a function at a lower cost with higher efficiency than a human operator, thereby, automation is becoming more and more widespread across various service industries as well as in the IT and software industry.

Automation basically helps a business in the following ways:

- It helps to reduce the complexities of processes and sequential steps
- It helps to reduce the possibilities of human error in repeatable tasks
- It helps to consistently and predictably improve the performance of a system
- It helps customers to focus on business rather than how to manage complexities of their system; hence, it increases the productivity and scope of innovation in a business
- It improves robustness, agility of application deployment in different environments, and reduces the time to market an application

Automation has already helped to solve various engineering problems such as information gathering, preparation of automatic bills, and reports; with the help of automation, we get high-quality products and products that save cost.

IT operations are very much dependent on automation. A high degree of automation in IT operations results in a reduced need for manual work, improved quality of service, and productivity.

## Why automation is needed

Automation has been serving different types of industries such as agriculture, food and drink, and so on for many years, and its usage is well known; here, we will concentrate on automation related to the information technology (IT) service and software industry.

Escalation of innovation in information technology has created tremendous opportunities for unbelievable growth in large organizations and small- and medium-sized businesses. IT automation is the process of automated integration and management of multifaceted compute resources, middleware, enterprise applications, and services based on workflow. Obviously, large organizations with gigantic profits can afford costly IT resources, manpower, and sophisticated management tools, while for small- and medium-scale organizations, it is not feasible. In addition to this, huge investments are at stake in all resources and most of the time, this resource management is a manual process, which is prone to errors. Hence, automation in the IT industry can be proved as a boon considering that it has repeatable and error-prone tasks. Let's drill down the reasons for the need of automation in more detail:

- **Agile methodology:** An agile approach to develop an application results in the frequent deployment of a process. Multiple deployments in a short interval involve a lot of manual effort and repeatable activities.
- **Continuous delivery:** Large number of application releases within a short span of time due to an agile approach of business units or organizations require speedy and frequent deployment in a production environment. Development of a delivery process involves development and operation teams that have different responsibilities for proper delivery of the outcome.
- **Non-effective transition between development and production environment:** In a traditional environment, transition of a latest application build from development to production lasts over weeks. Execution steps taken to do this are manual, and hence, it is likely that they will create problems. The complete process is extremely inefficient. It becomes an exhaustive process with a lot of manual effort involved.
- **Inefficient communication and collaboration between teams:** Priorities of development and IT operations teams are different in different organizations. A development team is focused on the latest development releases and considers new feature development, fixing the existing bugs, or development of innovative concepts; while an operations team cares about the stability of a production environment. Often, the first deployment takes place in a production-like environment when a development team completes its part. An operations team manages the deployment environment for the application independently, and there is hardly any interaction between both the teams. More often than not, ineffective or virtually, no collaboration and communication between the teams causes many problems in the transition of application package from the deployment environment to the production environment because of the different roles and responsibilities of the respective teams.

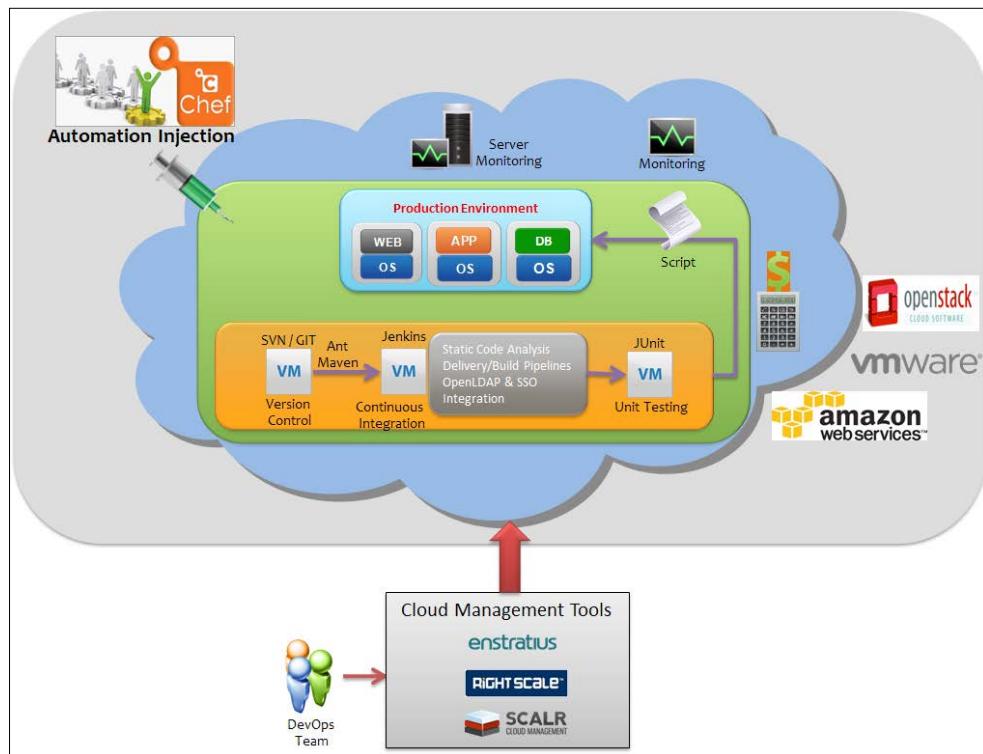
- **Cloud computing:** The surfacing of cloud computing in the last decade has changed the perspective of the business stakeholders. Organizations are attempting to develop and deploy cloud-based applications to keep up their pace with the current market and technology trends. Cloud computing helps to manage a complex IT infrastructure that includes physical, consolidated, virtualized, and cloud resources, as well as it helps to manage the constant pressure to reduce costs. Infrastructure as a code is an innovative concept that models the infrastructure as a code to pool resources in an abstract manner with seamless operations to provision and deprovision for the infrastructure in a flexible environment of the cloud. Hence, we can consider that the infrastructure is redeployable using configuration management tools. Such an unimaginable agility in resources has provided us with the best platform to develop innovative applications with an agile methodology rather than the slow and linear waterfall of the **Software Development Life Cycle (SDLC)** model.

Automation brings the following benefits to the IT industry by addressing preceding concerns:

- **Agility:** It provides promptness and agility to your IT infrastructure. Productivity and flexibility is the significant advantage of automation, which helps us to compete with the current agile economic condition.
- **Scalability:** Using automation, we can manage the complications of the infrastructure and leverage the scalability of resources in order to fulfill our customers demand. It helps to transform infrastructure into a simple code, which means that building, rebuilding, configuring, and scaling of the infrastructure is possible in just a few minutes according to the need of the customers in a real-world environment.
- **Efficiency and consistency:** It can handle all the repeated tasks very easily, so that you can concentrate on innovative business. It increases the agility and efficiency of managing a deployment environment and application deployment itself.
- **Effective management of resources:** It helps to maintain a model of infrastructure which must be consistent. It provides a code-based design framework that leads us to a flexible and manageable way to know all the fundamentals of the complex network.
- **Deployment accuracy:** Application development and delivery is a multifaceted, cumbersome, repetitive, and time-bound endeavor. Using automation, testability of a deployment environment and the enforcing discipline of an accurate scripting of the changes needs to be done to an environment, and the repeatability of those changes can be done very quickly.

We have covered DevOps-related aspects in the previous section, where we discussed the need for automation and its benefits. Let's understand it in a more precise manner. Recently, the DevOps culture has become very popular. A DevOps-based application development can handle quick changes, frequent releases, fix bugs and continuous delivery-related issues in the entire SDLC process. In simple English, we can say that DevOps is a blend of the tasks undertaken by the development and operation teams to make application delivery faster and more effective. DevOps (includes coding, testing, continuous integration of applications, and version releases) and various IT operations (includes change, incident, problem management, escalation, and monitoring,) can work together in a highly collaborative environment. It means that there must be a strong collaboration, integration, and communication between software developers and IT operations team.

The following figure shows you the applied view of DevOps, and how a development and an operations team collaborate with each other with the help of different types of tools. For different kinds of operations such as configuration management and deployment, both Chef and Puppet are being used. DevOps also shows how cloud management tools such as Dell Cloud Manager, formerly known as Enstratus, RightScale, and Scalr can be used to manage cloud resources for development and operations activities:

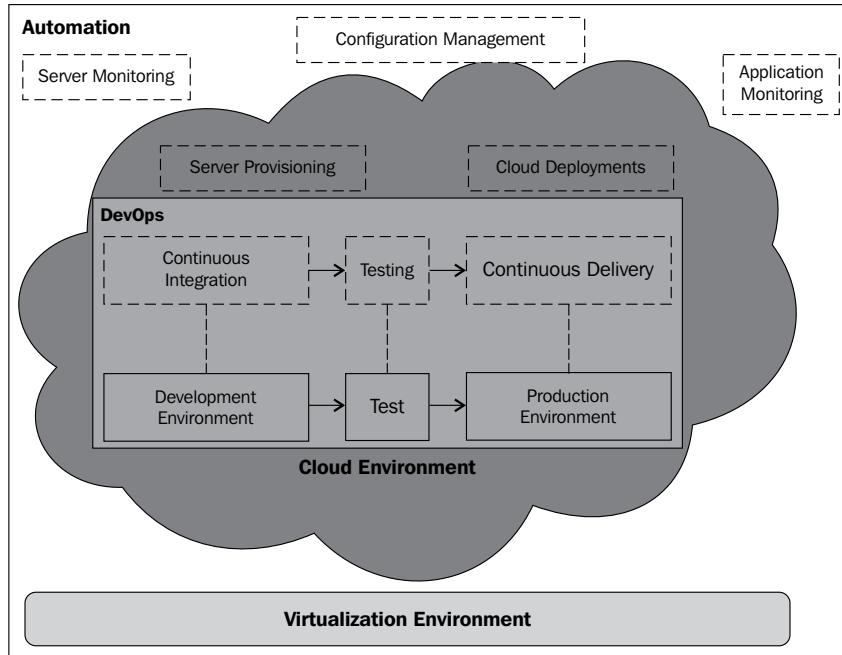


DevOps is not a technology or a product, but it is a combination of culture, people, process, and technology. Everyone who is involved in the software development process, including managers, works together and collaboratively on all the aspects of a project. DevOps represents an important opportunity for organizations to stay ahead of their competition by building better applications and services, thus opening the door for increased revenue and improved customer experiences. DevOps is the solution for the problems that arise from the interdependence of IT operations and software development.

There are various benefits of DevOps:

- DevOps targets application delivery, new feature development, bug fixing, testing, and maintenance of new releases
- It provides stable operating environments similar to an actual deployment environment and hence, results in less errors or unknown scenarios
- It supports an effective application release management process by providing better control over the distributed development efforts, and by regulating development and deployment environments
- It provides continuous delivery of applications and hence provides faster solutions to problems
- It provides faster development and delivery cycles, which help us to increase our response to customer feedback in a timely manner and enhance customer experience and loyalty
- It improves efficiency, security, reliability, predictability of outcome, and faster development and deployment cycles

In the following figure, we can see all the necessities that are based on the development of DevOps. In order to serve most of the necessities of DevOps, we need a tool for configuration management, such as Chef:



In order to support DevOps-based application development and delivery approach, infrastructure automation is mandatory, considering extreme need of agility. The entire infrastructure and platform layer should be configurable in the form of code or a script. These scripts will manage to install operating systems, install and configure servers on different instances or on virtual machines, and these scripts will manage to install and configure the required software and services on particular machines.

Hence, it is an opportunistic time for organizations that need to deliver innovative business value in terms of services or offerings in the form of working outcome – deployment ready applications.

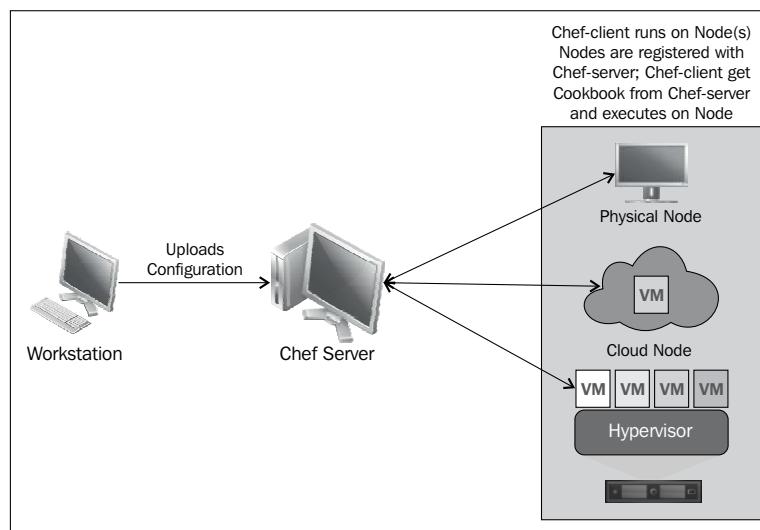
With an automation script, same configuration can be applied to a single server or thousands of identical servers simultaneously. Thereby, it can handle error-prone manual tasks more efficiently without any intervention, and manage horizontal scalability efficiently and easily.

In the past few years, several open-source commercial tools have emerged for infrastructure automation, in which, Bcfg2, Cobbler, CFEngine, Puppet, and Chef are the most popular. These automation tools can be used to manage all types of infrastructure environments such as physical or virtual machines, or clouds. Our objective is to understand Chef in detail, and hence, we will look at the overview of the Chef tool in the next section.

## Introduction to Chef

Chef is an open source configuration management tool developed by the Opscode community in 2008. They launched its first edition in January 2009. Opscode is run by individuals from the data center teams of Amazon and Microsoft. Chef supports a variety of operating systems; it typically runs on Linux, but supports Windows 7 and Windows Server too. Chef is written in Ruby and Erlang, both are real-time programming languages.

The Chef server, workstation, and nodes are the three major components of Chef. The Chef server stores data to configure and manage nodes effectively. A Chef workstation works as a local Chef repository. Knife is installed on a workstation. Knife is used to upload cookbooks to a Chef server. Cookbook is a collection of recipes. Recipes execute actions that are meant to be automated. A node communicates with a Chef server and gets the configuration data related to it and executes it to install packages or to perform any other operations for configuration management .



Most of the outages that impact the core services of business organizations are caused by human errors during configuration changes and release management. Chef helps software developers and engineers to manage server and application configurations, and provides for hardware or virtual resources by writing code rather than running commands manually. Hence, it is possible to apply best practices of coding and design patterns to automate infrastructure. Chef was developed to handle most critical infrastructure challenges in the current scenario; it makes deployment of server and applications to any physical, virtual, or cloud instances easy. Chef transforms infrastructure to code.

Considering virtual machines in a cloud environment, we can easily visualize the possibility of keeping versions of infrastructure and its configurations and creating infrastructure repeatedly and proficiently. Additionally, Chef also supports system administration, network management, and continuous delivery of an application.

## Why Chef is a preferred tool

Currently, IT operations and processes are very much based on virtual systems and cloud deployments, which have increased the complexity and the number of systems managed. In order to manage these types of systems and environments, we need highly consistent, reliable, and secure automated processes. However, many existing configuration management tools are not sufficient in the current environment; they are actually adding complexity to an already complicated problem.

For these kinds of special scenarios, we need a tool that has built-in functionalities and doesn't require a dedicated team of developers to maintain it. We need a complete automated solution that must be easy to learn and can be used by developers easily. Chef is aligned in this direction. Chef is one of the most popular configuration management tools used by DevOps engineers across the world. To support this argument, let's examine the salient features of Chef in the next section.

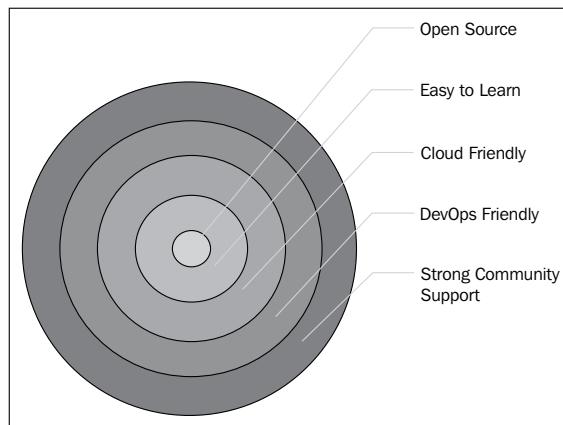
## The salient features of Chef

Based on comparative analysis with Chef's competitors, the following are the salient features of Chef, which make it an outstanding and the most popular choice among developers in the current IT infrastructure automation scenario:

- Chef has different flavors of automated solutions for current IT operations such as Open Source Chef, Hosted Chef, and Private Chef.
- Chef enables the highly scalable, secure, and fault-tolerant automation capability features of your infrastructure.
- Every flavor has a specific solution to handle different kinds of infrastructure needs. For example, the Open Source Chef server is freely available for all, but supports limited features, while the Hosted Chef server is managed by Opscode as a service with subscription fees for standard and premium support. The Private Chef server provides an on-premise automated solution with a subscription price and licensing plans.

- Chef has given us flexibility. According to the current industry use cases, we can choose among Open Source, Hosted, and Private Chef server as per our requirement.
- Chef has the facility to integrate with third-party tools such as Test Kitchen, Vagrant, and Foodcritic. These integrations help developers to test Chef scripts and perform proof of concept (POC) before deploying an actual automation. These tools are very useful to learn and test Chef scripting.
- Chef has a very strong community. The website, <https://www.chef.io/> can help you get started with Chef and publish things. Opscode has hosted numerous webinars, it publishes training material, and makes it very easy for developers to contribute to new patches and releases.
- Chef can quickly handle all types of traditional dependencies and manual processes of the entire network.
- Chef has a strong dependency management approach, which means that only the sequence of order matters, and all dependencies would be met if they are specified in the proper order.
- Chef is well suited for cloud instances, and it is the first choice of developers who are associated with cloud infrastructure automation. Therefore, demand for Chef automation is growing exponentially. Within a short span of time, Chef has acquired a good market reputation and reliability.

In the following figure, we can see the key features of Chef automation, which make it the most popular choice of developers in the current industry scenario:



## Automation with Chef

Chef has been around since 2009. As discussed, it was very much influenced by Puppet and CFEngine. Chef supports multiple platforms including Ubuntu, Debian, RHEL/CentOS, Fedora, Mac OS X, Windows 7, and Windows Server. The working of Chef is based on the master/agent architecture and pulls mechanism.

In a short span of time, Chef has become quite popular in all types of industries, including mid-size and giant industries. There are various success stories of effective utilization of Chef by customers. Chef has proven its capabilities in all types of Chef solutions, including Open Source, Private, and Hosted versions of Chef server.

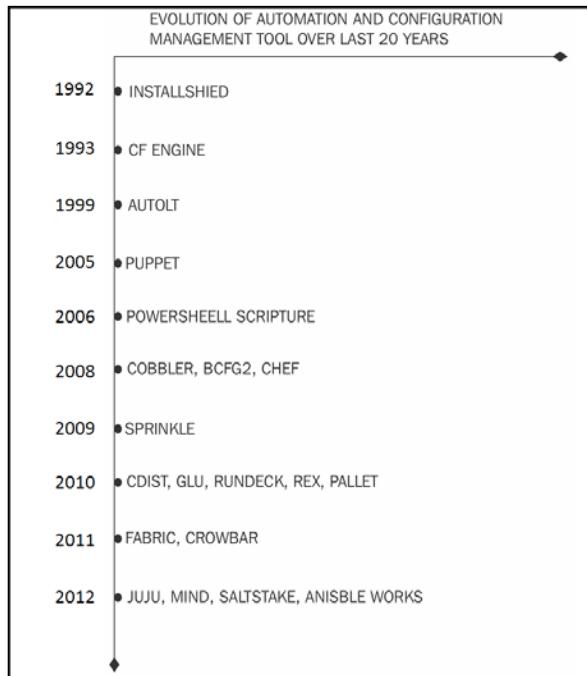
Chef is considered easy to use, and it is very much user- and developer-based. Everything in Chef is based on a Ruby script that follows a particular model, which developers use to work. There is rapid growth in the Chef community, and more open source developers are contributing to the Chef community to enhance its functionalities.

In order to practice, learn, or perform POC (proof of concept) with a Chef script, Chef allows great flexibility for usage of third-party tools and allows experimentation via Test kitchen, Vagrant, and Foodcritic. These third-party tools integrate with Chef and help users to test scripts without actually running them on a Chef server. Thereby, to test the accuracy of Chef, these tools are very useful, and the Chef community provides full support to this integration.

In the next section, we will go through the details of existing automation scripts and tools. It will help us to compare Chef with other automation tools or configuration management tools later, in the chapter.

## Existing automation tools and comparison with Chef

In the following figure, we can see the evolution of various configuration automation tools over the past 20 years. This gives us a clear picture of all the existing automation tools:



The previous figure describes the order of evolution of different automation tools in the IT industry. In the current scenario, most of the tools are not that popular and some are used very less practically.

There are various automation tools for different types of industries. Here, we will concentrate on automation related to IT and software processes, which has existed and been of benefit to customers for a long time.

Various already existing configuration management systems such as CFEngine and Puppet determine the current state of the system, and then compile a list of configurations and services that need to be applied to generate a desired state.

We will consider some of the existing automation frameworks including some traditional and advance methods of automation to get a clear idea of the existing automation techniques in the software industry and IT service processes.

## InstallShield

InstallShield is a software tool to create setups and software packages. InstallShield was developed by Stirling Technologies in 1992.

### Features of InstallShield

- This is basically used to install software in Microsoft Windows and Windows Server. InstallShield is a complete development solution for Windows installation.
- It is designed to make development teams more manageable, accurate, and flexible with regards to collaboration while building optimal Installscript and Windows Installer (MSI) installations for the Web, a server system, desktop, PC, and cellular applications.
- It is a unique software installer that builds Microsoft's App-V virtual packages.
- It simplifies multilayered installations and automates the installation of Windows roles and features.
- It runs PowerShell scripts, while setting up installation packages.

## AutoIt

AutoIt is the automation and scripting language specially designed for Microsoft Windows in 1999.

### Features of AutoIt

- AutoIt v3 is a freeware BASIC-like scripting language designed to automate the Windows GUI and general scripting. AutoIt v3 manipulates window processes and interacts with all the standard window controls. It is compatible with Windows XP, Vista, 2000, 2003, 2007, and Windows 2008 R2.
- It uses a combination of simulated keystrokes, mouse movements, and window/control manipulations in order to automate tasks in a way that is not possible or reliable with other languages (for example, VBScript and SendKeys).
- One of the biggest advantages of an AutoIt automation script is that it can be changed into a compressed and standalone executable form. Therefore, execution of an AutoIt script is possible on those computers where an AutoIt interpreter is not installed.

## **Windows PowerShell scripting**

Windows PowerShell is an automation framework developed by Microsoft in 2006. It is built on the .NET framework.

### **Features of PowerShell**

- Windows PowerShell is one of the most popular tools of automation on Windows Server. It is a command shell which is extendable. It is also a language used for scripting, which can be used to supervise server environments, such as Windows Server, Microsoft Exchange Server, and SharePoint 2010. With the help of PowerShell, we can save a lot of window admin tasks and ensure its effective usage.
- PowerShell is a replacement shell for the Microsoft Windows operating system, which carries advanced scripting to Windows. Initially, Windows PowerShell was bundled as a distinct add-on to Windows, marketed mainly to server administrators.
- Using PowerShell scripting, we can complete repetitive and composite processes in less time and in an easy manner. We can do this by combining several commands together and automating tasks such as deployment, for example, which reduces the risk of human error.

## **CFEngine**

CFEngine was developed by Mark Burgess in 1993. It is one of the oldest automation scripting frameworks.

### **Features of CFEngine**

- CFEngine manages the complete life cycle of an IT infrastructure. It has a powerful language and tools to define the desired state of your infrastructure, irrespective of whether it is a single server or a complex, global network with thousands of servers, and storage and network devices.
- It comprises of a powerful agent technology, which ensures that the state of the processes are continuously maintained. It basically works on three principles: define, automate, and verify.
- CFEngine has automated software distribution, change management, copy configuration, inventory/asset management, job initiation, tracking and execution. It also has automated network provisioning, remote configuration, resource initialization, resource shut down and service activation, fault management, accounting, and allocation management capability of resources.

- The prominent features of CFEngine are security and compliance of mission-critical applications and services. It is built upon well-established theory and high-quality engineering practices. CFEngine has an outstanding security record over the past 19 years. Now, we are going to see some advance automation tools.

## Puppet

Puppet is a well-known automation framework developed by Puppet Labs in 2005.

The features of Puppet are as follows:

- A Puppet framework has different components such as Collective, Puppet Dashboard, Puppet DB, Hiera, and Facter.
- A Puppet framework is used to provide continuous automation and orchestration. It has solved many real-time integration challenges with different types of server deployment.
- With the help of Puppet, we can easily automate repetitive tasks, quickly adapt to changes, and scale up servers on demand. A Puppet framework is also well suited to cloud deployment.
- Puppet uses a declarative model-based approach for IT automation. It has four major stages: define, simulate, enforce, and report. The Puppet community supports reusable configuration modules. It has more than 1,000 prebuilt and freely downloadable configuration modules.
- If we have a specific requirement than using Puppet's configuration language, we can build our own custom module. After defining our own custom module, we can reuse it for any type of requirement such as physical, virtual, or cloud.

## Bcfg2

Bcfg2 is a configuration management tool developed by Narayan Desai at Argonne National Laboratory. He launched its first release in 2008. After some more releases, the latest stable version was launched in July 2013.

The features of Bcfg2 are as follows:

- Bcfg2 has developed in such way that it can provide full support and clear understanding of specification and current states of client.
- It is designed in a way that it appropriately deals with manual system modifications.

- If we talk about generations in configuration management tools, it is of the fifth generation and was developed in the Mathematics and Computer Science division of Argonne National Laboratory.
- Bcfg2 enables system administrators to produce a consistent, reproducible, and representable description of their environment. It also offers visualization and reporting tools to support day-to-day administrative tasks.

## Cobbler

Cobbler is a Linux-based installation server that speeds up the setup of installation of network environments. It was developed in 2008 by some open source community members. Initially, it supported Red Hat Linux, but later on, it was represented as a part of the Fedora project. Since January 2011, Cobbler has been packaged with Ubuntu.

The features of Cobbler are as follows:

- Cobbler is a Linux installation server, which is designed for fast setups of installation environments in network connectivity. It gives you the facility to connect and automate many Linux tasks together, which in return gives you the ease of not jumping along the commands and applications while configuring new systems. Also, sometimes, it is useful to change the existing ones.
- Cobbler's easy and simple methods to write the commands help in system configuration designing. Network installs can be configured for reinstallations, media-based net-installs, PXE, and virtualized installs (It also supports XEN, KVM, and some variants of VMware.)
- As an option, Cobbler can also help to manage DNS, DHCP, and yum package mirroring infrastructure. In this respect, it is a more general automation application rather than an application that just deals with installations.
- After the initial setup, newly registered users can use setup steps as in the command (Cobbler check and Cobbler import). This leads you to a pretty good approach for initialization.
- Cobbler offers many features like reduced memory consumption, built-in configuration management system and it integrates with systems such as Pallet.

- Cobbler has a web interface with a command-line interface and several API access options. New users can start with a web application after performing the initial setup steps on the command line: Cobbler check and Cobbler import. This will give the user a suitable idea of all the available properties. All the features (advanced) need not be understood at the same time, they can be learned over time as the need for them arises.
- Cobbler consumes less memory as it is a very small application, that is, it has 15,000 lines of Python code. It works fine for all level installations. It is functional and valuable in all the enterprises as it has numerous properties and features. It gives you the flexibility to complete the work in a small span of time. Also, it saves time for all the manual tasks that are repeated.

## **Sprinkle**

Sprinkle is a tool that can be run locally and maintained easily. It was developed by Marcus Crafte in 2009. Sprinkle is also one of the open source products.

The features of Sprinkle are as follows:

- Sprinkle is an easy-maintenance tool. Also, it can be managed and is run as a standalone tool.
- Sprinkle is a software provisioning tool you can use to build remote servers, for example, you can use Sprinkle to install Rails directly after it is created.
- It has a nice collection of Installers that lets you install applications from various sources.
- It is best suited for small infrastructures. Sprinkle is based on Capistrano. It follows the same push model without any additional infrastructure.
- Among commands from APT to ad-hoc commands, if you use Sprinkle, it can make your life easier (for those who write scripts).

## **cdist**

cdist is a reusable configuration management system developed by Nico Schottelius and Steven Armstrong in 2010.

The features of cdist are as follows:

- cdist is a reusable management system. It is configured in such a way that it can be used by small enterprises to grade environments.
- The configuration of cdist is done in a shell script. A shell script has been used by Unix system engineers for decades.

- If we consider target system, the requirement of cdist is very less because shell script is used in all cases and all dependencies are usually fulfilled only by shell script.
- cdist does not require an agent or a high-level programming language on the target host; it will run on any host that has a running SSH server and a POSIX compatible shell (/bin/sh). Compared to other configuration management systems, it does not require you to open any additional port.
- From a security point of view, only one machine needs access to the target hosts. No target host will ever need to connect back to the source host, which contains the full configuration.

## Pallet

Pallet is also an open source project developed by Hugo Duncan and Antoni Batchelli in 2010.

The features of Pallet are as follows:

- Pallet is a tool for automation, and was especially designed for the cloud environment. It can also work with traditional servers.
- As per its historic explanation, it was developed with Clojure, a JVM implementation of the Lisp (classic)
- Pallet is not just a tool for system administrators, it is built for developers as well
- It is a library which can be used with other applications, as it is more than a server.
- In a DevOps world, it means that infrastructure as code starts with the development team and trickles down to operations.

## Rex

Rex is a pure open source project developed in 2010 and managed by the [www.rexify.org](http://www.rexify.org) community.

The features of REX are as follows:

- Rex is a small and lightweight framework. It is basically a server orchestration tool that does not need an agent on the hosts you want to manage as it uses SSH. Therefore, no agent installation is required for nodes.
- It provides significant integration without any conflict. It is easy to learn just like Perl scripting.

- Rex fully supports DevOps-based deployment.
- Apart from open source support, REX also provides commercial support for all related services.

## Glu

Glu is a free, open source deployment and monitoring automation platform, which was also developed by open source community members in 2010.

The features of Glu are as follows:

- It deploys and monitors applications efficiently.
- It is secure to use and provides reproducibility of infrastructure.
- The Glu script provides a set of instructions describing how to deploy and run an application. It processes arbitrary large set of nodes efficiently with minimum manual security efforts.

## RunDeck

RunDeck is open source software, which was developed by many members of open source in 2010.

The features of RunDeck are as follows:

- RunDeck processes things according to standard procedures. Technically, it enables operations, tools, and processes into standard operating procedures, which are for public use so anybody can view it in the organization.
- It has an open, pluggable design, which makes RunDeck easy to integrate and extend.
- RunDeck creates workflows that can connect various scripts, tools, and other operations systems easily and safely. It provides built-in standard authorization, logging, and notifications, which make it easy and safe to give others self-service usage of RunDeck's API or WebGUI.
- RunDeck also provides a **Graphical User Interface (GUI)** for monitoring on-demand and scheduled operations tasks.
- It executes actions to nodes over SSH, WinRM, or any other transport.
- It has workflow options that can be pulled from tools such as build servers, package repos, ticketing systems, or anything with an **Application Programming Interface (API)**.

## Crowbar

Crowbar is the open source deployment tool developed by Dell in 2011. It was initially developed to support Dell's OpenStack and Hadoop-powered solution.

The features of Crowbar are as follows:

- Crowbar enables you to provision a server from BIOS, via Chef, up to higher-level server states.
- Crowbar can be extended via plugins called **barclamps**. So far, there are barclamps available for provisioning on various platforms such as Cloud Foundry, Zenoss, Hadoop, and more.
- With Dell Crowbar, we can discover and configure hardware (BIOS and RAID), and deploy as well as configure operating systems and applications. You can do all of this repeatedly in a fraction of the usual time required.

## Fabric

Fabric is also developed and managed by open source community members. It was mainly developed by Christian Vest Hansen and Jeffrey E. Forcier in 2011.

The features of Fabric are as follows:

- It works as a command-line tool to streamline the use of SSH.
- It is an advanced tool that allows you to orchestrate various configuration operations.
- It is used for application deployment or systems administration tasks. It is the appropriate solution for uploading/downloading files, and is used for auxiliary functionalities such as prompting the current user for input, or canceling the execution.
- Fabric is used to write and execute Python's function code or tasks to automate communications with remote servers. It works on clusters of machines and allows you to deploy applications. It starts/stops services on a cluster of machines.
- Fabric is a tool written in the scripting language, Python (2.5 or a higher version) library.
- It provides the best solution and best suite of operations to execute local or remote shell commands normally or through sudo.

## **Ansible**

Ansible is a configuration management tool. It is also a deployment and ad hoc task execution tool. It is an open source software, which was developed by many members of open source in 2012.

The features of Ansible are as follows:

- Ansible is an execution and open source automation tool which is served as a configuration and deployment management and it is also served as ad-hoc task execution.
- It works using SSH, which is very popular among Linux users and administrators. It does not need any daemons or software for remote machine management.
- It is fast and simple to install as no configuration file, daemon, or database is needed.
- The setup is very simple to install as no software needs to be installed on remote machines. This means that Ansible starts arranging the system at once if you have a clear image of your favorite running OS with you.
- Ansible is designed in such a manner that it does not need anything more than a password or SSH key to start managing systems and it does so without installing any software agent. All these features make it quite useful when there are many nodes to be managed.

## **SaltStack**

SaltStack is a systems and configuration management software for any enterprise that follows a cloud or DevOps deployment. It was developed by Tom Hatch and Marc Chenn in 2012.

The features of SaltStack are as follows:

- It is one of the most active and fastest growing open source communities in the world.
- It delivers a completely different approach to legacy alternatives not built for the speed and scale of a cloud.
- The commendable achievement of SaltStack is that for the purpose of orchestrating and controlling any cloud and providing automation for DevOps tool chain, it is used by the largest IT enterprises and DevOps organizations in the world.

- It provides heterogeneous support for cloud configuration and infrastructure or any software platform.
- SaltStack is mainly known for parallel management. Moreover, it provides real-time data automation. It is much less time consuming, so if we check time specification in this case, remote execution takes place in seconds and not in minutes or hours.

## Mina

Mina is a tool for fast server deployment and automation. It was developed by many open source contributors in 2012.

The features of Mina are as follows:

- Mina has been designed to build and run scripts to manage your app deployments on servers via SSH
- It is known for its fast working as it deploys a Bash script generator, and it can be used on just about any type of project deployable via SSH, Ruby or not
- It produces an entire procedure as a Bash script and runs it remotely in the server
- It only creates one SSH session per deploy, minimizing the SSH connection overhead
- It even provides safe deployment and locking

## Juju

Juju is a single-service orchestration tool. It was introduced by Canonical Ltd. in 2012. Earlier, it was named as Ensemble.

The features of Juju are as follows:

- It runs on public clouds, private clouds, and micro clouds.
- Juju requires Ubuntu developer workstations. It was developed to coexist with tools such as Puppet and Chef, and it was developed to provide extra services. Orchestration toolsets such as Juju take the process one step further by gluing all these applications together.
- Juju provides you with a unique, easy, and straight way to extend deployments.
- Unlike old, script-based approaches, it can grow fast and compress on demand by adding various layers or substituting components on the fly.

All advance automation tools work on master agent architecture, which is similar to the traditional client-server architecture and some work on the standalone architecture.

In the following table, we will compare some of the most demanding advance automation tools of the IT industry in the current scenario:

	<b>Chef</b>	<b>Puppet</b>	<b>SaltStack</b>	<b>Ansible</b>
License	Apache	Apache	Apache	GPL
Stable release	Chef-client: 12.0.3   16 December 2014  Chef-server: 12.0.1   17 December 2014	3.7.1   15 September 2014	2014.7.0[1]   3 November 2014	1.8.2   5 December 2014
Language	Ruby (client) and Ruby/Erlang (server)	Ruby	Python	Python
Architecture	Master/Agent	Master/Agent	Master/Agent	Standalone
Push/Pull Mechanism	Pull	Pull	Push	Push
Selling feature	Industry leader	Industry leader	Speed and scale	Simple to install
Quality of Documentation	The quality of Chef's documentation is very good. It has free webinars for beginners and is easy to understand.	The quality of Puppet's documentation is good. It has free online training available.	The quality of SaltStack's documentation is evolving. Comparatively, it is not so good.	The quality of Ansible's documentation is good and well structured.
Cloud Integration	Yes  Amazon EC2, Windows Azure, HP Cloud, Google Compute Engine, Joyent Cloud, Rackspace, VMWare, IBM Smartcloud, OpenStack.	Yes  AWS, VMware, Google Compute Engine	Yes  Amazon AWS, Rackspace, SoftLayer, GoGrid, HP Cloud, Google Compute Engine, VMware, Windows Azure, and Parallels.	Yes  AWS, VMWare, OpenStack, CloudStack, Eucalyptus Cloud, and KVM,

	<b>Chef</b>	<b>Puppet</b>	<b>SaltStack</b>	<b>Ansible</b>
Support Services	Support Tickets, Premium, Standard, and Free Support, IRC, Mailing lists	Customer support portal, Enterprise mailing list, Open source community	Salt IRC Chat, SaltStack Mailing List, and SaltStack User Groups	FAQs, Support Requests, Mail
Industry example	Facebook, Linkedin, Youtube, Splunk, Rackspace, GE Capital, Digital Science, and Bloomberg	Twitter, Verizon, VMware, Sony, Symantec, Redhat, Salesforce, Motorola, and Paypal	Lyft	Apple, Juniper, Grainger, WeightWatchers, SaveMart and NASA

To get more insight in the comparison, refer to the article, *Review: Puppet vs. Chef vs. Ansible vs. Salt* by Paul Venezia at InfoWorld's website, <http://www.infoworld.com/article/2609482/data-center/review--puppet-vs--chef-vs--ansible-vs--salt.html>.

## Comparison with other popular tools

For better understanding, we will compare Chef with some of the most popular infrastructure automation tools, which are the competitors of Chef.

### Chef versus Puppet

The following are the key differences between Puppet and Chef:

- Puppet uses a custom, JSON-like language. Although a Ruby option is available with Puppet, while in Chef, you write in Ruby. Chef is purely a **Domain Specific Language (DSL)**, which extends the Ruby language to the resources that are useful to manage hosts and their applications. The unique quality of Ruby is that if you are writing, then you can get simple stuff done even without actually knowing Ruby. Therefore, Chef is preferred by most of the software developers or web developers.
- Puppet is still struggling with proper documentation and webinar sessions for learners and developers, while Chef is very accurate with its documentation, training materials, recent updates, and releases. It has hosted various webinar sessions. Therefore, from a learning perspective, Chef is the preferred choice over Puppet.

- It is true that Chef is very much influenced by Puppet. Adam Jacobs contributed to the creation of Chef and he was once a Puppet user himself. Therefore, it is obvious that developers of Chef have learned from Puppet itself to make Chef much better and more flexible for users.
- Chef provides many more cloud integration options. We can integrate through APIs to different types of cloud providers such as AWS, Rackspace, MS Azure, OpenStack, Eucalyptus, VMware ESXi, VMware vCenter, VMware vCloud, and vCloud Air. While in Puppet, these kinds of integrations are not properly enabled, although Puppet supports cloud providers to some extent.
- It has been observed that the Chef community provides very good support and gives quick responses to user queries for any problem faced while installing, using, and debugging Chef scripts. While using Puppet, we cannot expect promptness in response, therefore, Chef users are much more satisfied than Puppet users. Hence, Chef has become a more preferable choice of developers. Chef has a variety of choices according to customer use cases. We can use private, open source Chef or hosted Chef according to our use case, but in Puppet, we don't have this type of option. Therefore, Chef is much more flexible in terms of usage.

## **Chef versus CFEngine**

The following are the key differences between Chef and CFEngine:

- CFEngine is very much older than Chef; it was developed by Mark Burgess in 1993. CFEngine is also called the grandfather of configuration automation tools.
- CFEngine runs on C while Chef runs on Ruby. This is the major problem with CFEngine because C is a low-level language and most of the communities don't support C. It takes much effort to learn CFEngine, while Chef can be learned easily by developers. Therefore, Chef is the preferred option nowadays for system administrators with limited coding experience.
- CFEngine has been present in the market for the last 20 years. According to CFEngine's site, currently, they are managing more than 10 million nodes and have a good list of customer support, but as we know, the current IT infrastructure scenario is very much based on virtualization and cloud. Chef serves perfectly for current industry automation needs and is the preferred choice of reputed organizations such as Amazon and Facebook.

- One more problem with CFEngine is its documentation. Being an open source community, they have less focus on the documentation of the latest releases and learning tutorials, while Chef has a strong community support for proper documentation and provides latest training material to all users. Therefore, popularity of Chef is increasing day by day and it has gone on to become the most popular open source automation tool.

Here, we got the conceptual understanding of Chef and its effectiveness over other configuration automation competitors. In Chef, there are different components that interact with one another during the execution and installation of scripts, which we are going to learn in the next chapter.

## **Self-test questions**

1. What is automation and why is it necessary in modern IT world?
2. What is DevOps and what are its benefits?
3. Chef is written using which languages?
4. What does Chef actually provide? How does automation support DevOps?
5. What is the dependency management feature of Chef, which makes it preferable in the current IT scenario?
6. With respect to managing cloud instances, why is Chef more preferred over Puppet?
7. CFEngine is very much older than any other modern automation scripting languages, but why it is not recommend by system administrators?
8. What are the available third-party tools that can be integrated with Chef for testing purposes?

## Summary

Here, we got the fundamental understanding of automation and the various ways in which automation helps the IT industry. DevOps is most popular nowadays, because it brings a highly collaborative environment in the entire software development process.

We got an overview of several traditional and advance automation tools, which have been used over the past 15 years. We got a clear idea why Chef is needed in the current scenario of IT automation process and why it is more preferable.

We saw the evolution of various IT automation tools and the comparison of some advance automation tools. We also discussed the comparison of Chef with other popular tools. We understood the salient features of Chef.

We will take a deep dive in to Chef's architecture, the different Chef components, and learn how to work with each one of these components in the next chapter.



# 2

## Different Components of Chef's Anatomy

*I couldn't tell you in any detail how my computer works. I use it with a layer of automation.*

- Conrad Wolfram

This chapter builds on the ingredients of the previous chapter, where we discussed automation and its need and the overview of Chef.

This chapter will introduce you to the Chef automation, Chef framework, and Chef components. Here, we will also describe the relationships between the server, nodes, and workstations. This trio instructs a chef-client what to do and how to do it in terms of configuration of resources.

We will also cover the following topics:

- The Chef server APIs
- GitHub
- Chef-solo
- The Chef community
- Chef run
- Vagrant and Chef integration

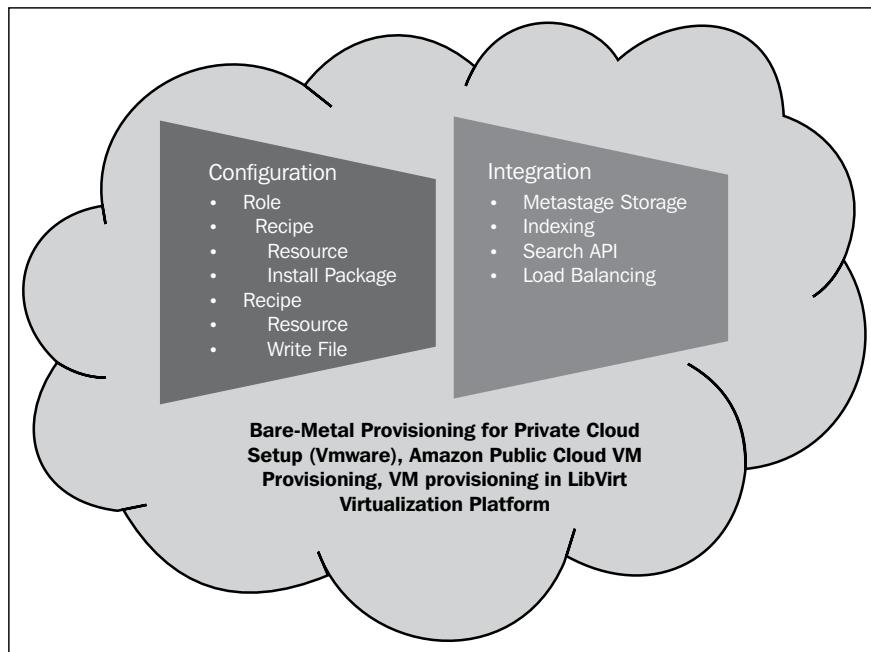
## The Chef automation

As we know from the previous chapter, Chef provides us with the flexibility to design our infrastructure as code, and we can fully automate all the process and functionalities of an infrastructure.

Chef provides us with a fully automated platform, which means that we can provision, configure, and integrate our infrastructure in just a few clicks without much intervention.

A Chef script can be used in various ways in the entire infrastructure automation procedure. We can use a Chef script for bare metal provisioning, virtual machine provisioning, and cloud instance provisioning as well.

The following diagram represents the fully automated infrastructure scenario:



## The Chef framework

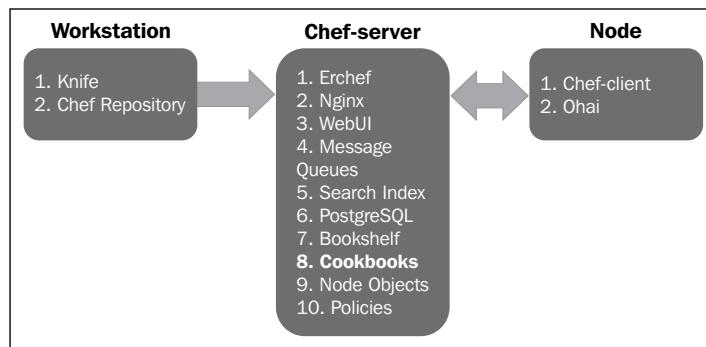
The Chef framework is basically a combination of the Chef server, node, and workstation. Essentially, these are predefined roles for resources, which are the core components of the automated configuration and deployment process. In order to provide instruction and information to Chef to run its job, these three elements facilitate each other simultaneously.

Using Chef elements, we can easily deploy and maintain servers and applications of physical, virtual, and cloud instances.

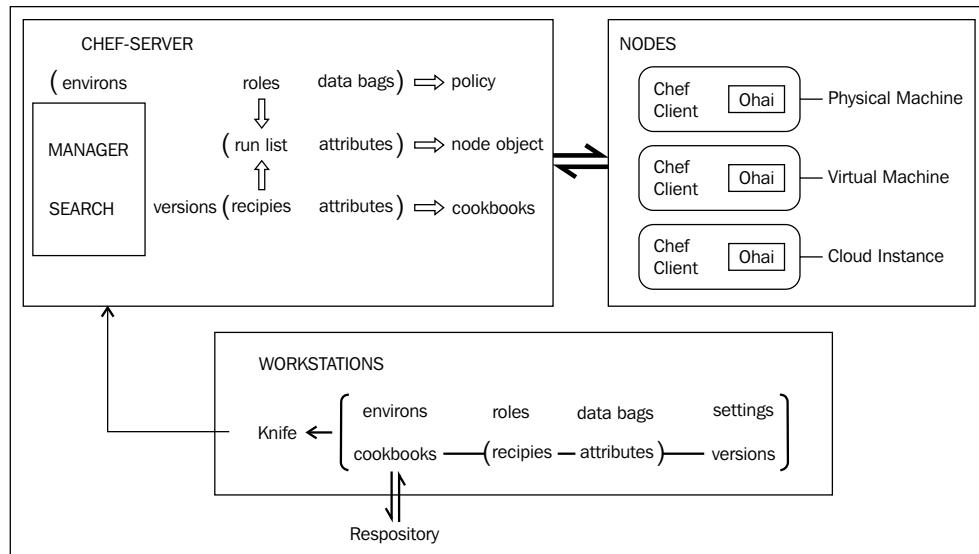
Each element also has different components that interact with each other in order to process and run a Chef script. These components are known as **Chef tools**.

As discussed previously that there are three main elements of Chef (Chef server, node, and workstation), but it contains several other components within them. We are going to take a deep dive into each one of these elements. Cookbook is the prominent component of Chef server, which contains the main executable code.

The following figure represents each and every component of a Chef framework:



The following diagram represents the relationships among all the three main elements of a Chef framework:



## Chef components

Each element in Chef also contains different components that work together. These are also known as Chef tools. Now, we are going to see the working of each tool.

### The Chef server

Chef server is the central element or brain of a Chef framework. Chef server works as a central repository or hub that contains configuration policies for nodes and cookbooks and has detailed information of registered components. Other components of a Chef framework use it for automated configuration and deployment.

All communications in Chef take place using Chef server. Chef server is written in the Erlang programming language, which was developed in 1986 and launched as open source in 1998. This language works well with critical issues of an enterprise in a distributed environment, such as fault tolerance, resiliency, and concurrency. It provides seamless scalability. This functionality is enabled with the new version release of a Chef sever 11.x. Regarding the configuration work on every node, it is the main responsibility of a Chef-client, which is installed on every node.

The main functionalities of Chef server are as follows:

- Chef server stores various policies that are applicable for different cookbooks
- Chef server stores metadata that has the information of each node that is registered with Chef server
- Chef server provides facility to search any type of data with the help of search indexes using indexing
- Chef server uses PostgreSQL as a data storage repository

### Different types of Chef servers

There are three types of Chef server:

#### Hosted Chef server

Hosted Chef is hosted by Opscode. It is a **software as a service (SaaS)** offering.

We do not need to manage this Chef server. The whole management and operation will be taken care by Opscode. Opscode provide us with a hosting service to use Chef server. It provides full scalability, all-time availability of resources, and resource-based access control.

Hosted Chef provides full-automation capability. We can use a free tier with up to five nodes and can manage nodes with Hosted Chef. From a learning perspective, we can utilize this feature of Hosted Chef. There is also a standard and premium service, which is a paid service, and we can manage up to 100 nodes using these paid services. A Chef subscription provides Hosted Chef with no additional charge. Hosted Chef supports many cloud platforms, such as AWS, Rackspace, Windows Azure, and so on.

Additionally, it brings down the total cost of ownership. Hosted Chef's latest features are available to use as upgrades and its updates are managed by a provider. It is also easy to perform proof of concept and pilot execution on Hosted Chef for know-how. An exciting feature of Hosted Chef is that we need not worry about its installation and configuration as it saves a lot of time during the installation and configuration of Chef. We can use this time to innovate new things or to improve the existing feature set of products or services.

## **Private Chef**

Private Chef server is a special kind of Chef server that provides all infrastructure automation capabilities within boundaries of an organization. It is fully owned and managed by individual organizations. It is a kind of local deployment.

Functionality wise, Private Chef is very similar to Hosted Chef. However, in private Chef, you don't have any limitation of nodes. All the nodes can be managed by you according to your organization's needs. This option is favorable in case of privacy and security concerns.

## **Open source Chef-server**

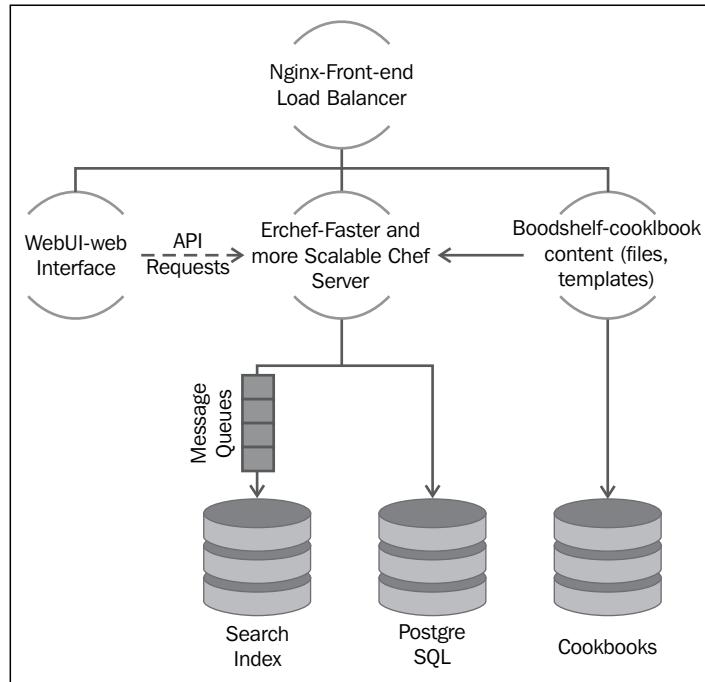
As is indicated by the name, Chef server is an open source and a free version. Its functions and operations work quite similar to Hosted Chef. The Chef community gives support to open source Chef server, but it doesn't include support from Opscode. Each instance is configured; it operates and performs data transfers and migrations, implementing modifications and updates to the open source Chef server. All of this is managed locally. This also ensures that the open source Chef server remains as the local infrastructure and supports upgrades.

## **Chef server tools**

The Chef server itself has different components that work together in a Chef server deployment, such as Nginx, WebUI, Bookshelf, message queues, search index, PostgreSQL, and Cookbooks.

## Different Components of Chef's Anatomy

The following figure represents the workings of different Chef server tools and their relationship with each other:



## **Different types of Chef server tools**

There are various components within Chef server that work together for the successful execution of all the processes of Chef server. Here, we will understand the details of each of the components.

### **erchef**

erchef is a replacement of the core API of Chef server. Cookbooks and recipes were used to work on API-based Chef server. Now, again, Ruby is used for programming and cookbooks and recipes are authored to work with an Erlang-based Chef server. This is a rapid and compatible way to work on Chef server. It is, as a whole, authored to the core API for Chef server. Ruby is always the preferred language for a chef-client. It is more compatible and more scalable than the existing versions. Although, if we talk about API, it's still compatible with a Ruby-based Chef server. erchef is backward compatible for cookbooks written in previous Ruby-based versions.

## Nginx

Nginx is the frontend load balancer of a Chef server. Whenever any request comes to a Chef server API, it first interacts with Nginx and then all requests get routed according to the API's call. Nginx is a high-performance, open source service. It acts as a reverse proxy server for different protocols such as HTTP/HTTPS, POP3, SMTP, and IMAP.

Nginx is popular for its simple configuration settings, various features, and low resource consumption; thereby, it is used by many other sites such as RightScale, Engine Yard, Zynga, WordPress.com, and GitHub.

## WebUI

WebUI is the web interface for the Chef server. It is written in the Ruby on Rails 3.0 platform.

## Message queues

Messages are sent to a search index with the help of the following components:

- **RabbitMQ:** This is an open source, message-oriented middleware, which is based on advanced message queuing protocol. In a Chef server, we use RabbitMQ for messaging queues. All the items that are going to be used for a search index repository are first added to the message queue.
- **Chef-expander:** This is an internal component that is used to pull the message from RabbitMQ in order to process, format, and then forward it to Chef-solr (an internal component for indexing).
- **Chef-solr:** This implements Apache Solr (an open source search platform used for full-text search and real-time indexing) and uses REST APIs to index and searches.

## Search index

A Chef server has a dedicated search index repository to store all messages. A Chef search index is powered by a Solr search platform; a Chef-solr service is used for indexing and conducting searches. A full-text list of all the objects is maintained in the server. According to user queries, various search indexes are built such as environments, nodes, roles, and so on.

**Knife search** is a subcommand, which is used to execute search queries that are indexed on a Chef server.

## **PostgreSQL**

PostgreSQL is an open source object relational database system used by the Chef server to store environments, data bags, roles, configurations, and node attributes. Postgres (Postgress SQL) is used as a data storage repository.

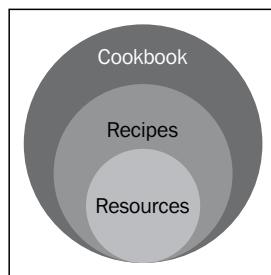
## **Bookshelf**

Bookshelf is used to store contents of cookbooks, such as Chef files, recipes, and templates that are uploaded to the Chef server. Every time the contents of a cookbook are verified by a content checksum. If different versions of the same cookbook contain the same file, Bookshelf stores it only once. It checks the integrity of the file or template then stores it.

Bookshelf has the overall management responsibility for cookbooks. All the contents of cookbooks are stored in separate files that are totally distinct from Chef server and search index repositories. So, there is a separate dedicated repository to store cookbooks that are managed by Bookshelf.

## **Cookbooks**

Cookbook is the most important and reusable component of a Chef server, and it is widely used by developers and infrastructure administrators because cookbook is the basic unit used to define rules and regulations for configuration purposes. It describes the complete picture of deployment and configuration of resources or applications. Sometimes, cookbooks are also dealt with as separate elements because they are the soul components that get updated through nodes, server, and repository. A pattern is defined in every cookbook so that MySQL is configured by installing all the important and necessary things. Moreover, a cookbook contains all the required components that are needed to support the scenario. The following are some of the important features of a cookbook:



- It contains the recipes that have all the specifications of resource management and also, it has the order of application of resources
- It contains the procedure to reuse the collection of resources

- It also has the attribute values that are applied on each node
- It contains the file distribution and libraries to increase Chef and/or provide enhancers to help in Ruby code
- It contains custom resources and providers
- It contains templates and metadata to control version constraints and supported recipes together with dependencies

If we talk about specific resources such as Chef's creation of cookbooks and recipes, Ruby is supported by Chef with an extended DSL. It gives us a collective set of resources that are needed to support basic patterns and scenarios of the most common infrastructure. DSL can also be extended when resources and capabilities are required.

Cookbooks contain the following functionalities:

## Recipes

A recipe is the most basic and fundamental part of the Chef environment that is stored in a cookbook. It is written in Ruby as it's a programming language. Everything in a recipe is designed and configured in such a way that it is easy to read and behaves in a possible, predictive manner. Recipes are deployed on the nodes and they are used to configure the node.

A recipe has the following characteristics:

- It is a gathering of similar types of resources.
- It is written in the Ruby language with the help of some syntax.
- It can use the outcomes of a search query and get through its content matter of a data bag through the encryption of data bag.
- One recipe depend on one or more recipes.
- It can be attached to a given facility. It is more compatible and makes it easier to build or constitute arbitrary groups that are exclusive to the general standards (naming conventions) that one industry standard can contain.
- It should be necessary to get it concatenated to the end of run-list so that it can be utilized by Chef.
- It runs/executes in the same manner as is categorized in the run-list.

## Attributes

Attributes represent the characteristics of a node. If we define the default settings then an attribute comes into the scenario as it is defined in a cookbook.

The Chef-client uses attributes for the following reasons:

- To describe the present specification of the node
- To represent the state of the node in the last Chef-client run
- To provide the state of the node at the end of the current Chef-client run

## Versions

A version is generally originated through continuous development. There are many reasons why we need new versions. The following are the main reasons:

- For the sake of updating a cookbook
- Appropriate usage of a third-party component
- To update a cookbook after providing the solution that bugs (debugging)
- After adding any modifications, corrections, or enhancements in the cookbook

Version can be defined by utilizing some specific syntax and operators. The definition also contains how this version is seen and how it is connected with environments, metadata of a cookbook or run-list. As far as safety is concerned, a version can be saved (frozen) to prevent further unwanted modifications. A version is all about how to use the cookbook during a Chef-run.

## Node objects

Attributes and run-lists are in groups made of necessary parts for nodes and an important aspects in Chef. As well the definition of Chef says that an attribute is a basic set of data information regarding a node (for example, a network interface). It also contains data of a filesystem. It also has the compatible information regarding the number of clients and the concerned service running on a node. Added to this, attribute is arranged, systematic list of recipes and roles that keeps all dependencies in order and also runs in the same order.

The node objects have both run-list and node attributes in a JSON format, which is stored on the Chef server. It is very important to note what really happens during the Chef-runs. The Chef-client receives a copy of the node object from the Chef server and puts a modified/updated new copy on the Chef server after each Chef-run.

There are many types of node objects; let's go through the details of each node object:

## Run-lists

A run-list has an arranged way of keeping roles/recipes in a proper manner and order. It is a collection of roles/recipes for a specific node. A run-list describes the specification of a node. However, it is also quite possible that many nodes contain the same run-list.

A run-list is an important node object that is stored on a Chef server and is uploaded through a Chef user interface.

One more important point is that Chef always configures a node in a proper and strict order mentioned by its run-list and will never run the same recipe for the second time.

## Attributes

An attribute is a common functionality of node objects and cookbooks. We have already discussed attributes in the cookbook section.

## Policies

A Policy is used to set up a map for the capable functionalities of Chef's requirements on a business as well as an operational level (for example, process and workflow). Now, the important aspects under this are roles and environments. Roles define the types in a server, for example, database or web server.

Process is defined by the environment. For example, the process of development, testing, or production. As far as the security aspect is concerned, data bags are placed in a safe sub area of Chef, which can only be reached by nodes that have appropriate SSL certificates. Data bags can be used to keep sensitive data, such as passwords, information of users or accounts.

## Roles

A role is defined for a unit's job function and provides a path to explain several different patterns and workflow processes in an industry. For example, the Tomcat server role or web server role can consist of Tomcat server recipes and any custom attributes.

Every single role has zero or more attributes and definitely a run-list that has a systematic order. It also has a node that can have zero or more roles assigned to it.

A notable process is that whenever a role is made to run against a node, the specification summary of that particular node is matched against the attributes of the role. After this is done, the data or matter of that role's run-list is implemented on the node's detail configuration.

## Data bags

A data bag contains variable data and is very specific to the deployment that is stored on a Chef server. It contains sensitive information such as passwords or account details. It is used like a searching index. It is a global variable that is stored in a JSON format and is reachable from a Chef server.

## An environment

An environment is the method to sense and gather the information of the real workflow of an organization. We can clone existing environments and they can be managed and configured by a Chef server.

Chef organizations also have one default environment that cannot be edited and deleted. Other environments, such as Chef development, testing, and production can be created using one or more versions of a cookbook.

## Workstations

A workstation is the most important element of the Chef framework. It is the starting point where you can define and deploy the Chef code to the Chef server. Basically, a workstation is a computer system or machine that is configured to run `Knife`. Using `Knife`, it interacts with a single Chef server and synchronizes with the Chef repository. A workstation is like a work place where you can begin your work with Chef. Using workstation, we can do the following:

- Maintain the Chef repository and proper synchronization with version control systems, for example, GitHub.
- Develop recipes and cookbooks and provide authorization for a Chef-client and Chef server. We can upload items from Chef repositories to a Chef server using a `Knife` command.
- Interact with one or more nodes (where a Chef-client is installed) during a Bootstrap operation.
- Configure organizational policy and define roles and environmental settings.

Workstations basically have two important tools.

## Workstation tools

The following are the two tools that work for a workstation's configuration:

- Knife
- A Chef repository

### Knife

Knife is a tool which provides a command-line interface between a Chef server and Chef repository. It is used in a workstation in order to execute Chef commands as that is the way your workstation communicates with the Chef server. Knife is used for the data that is to be uploaded from the Chef repository (which is stored in a workstation) to the Chef server after the proper upload of the entire information data is completed. Chef manages all nodes that are registered with a Chef server. Knife is a mandatory tool used to run Chef commands.

Knife acts as an interface between a Chef repository and Chef server. For authentication of Knife with a Chef server, RSA's public key pair is used whenever Knife tries to access the Chef server. It gives confirmation that Knife is properly enabled with the Chef server and now, only trusted users can change the data.

Using Knife, we can manage following:

- Workstations
- Nodes
- Roles
- Cookbooks and recipes
- Environments
- Store encrypted data
- Index data searches on the Chef server
- Manage various cloud resources

All the preceding are different data objects that we will understand later on in this chapter.

### A Chef repository

A Chef repository is the central location where all data objects are stored. All of the data in a Chef repository is treated as code, therefore, it must be coordinated with a control system, such as GitHub.

### Different Components of Chef's Anatomy

In GitHub, we contain all our project development related modules and codes which get regularly uploaded and communicate with the Chef server. All the data object development is stored and controlled in a GitHub repository.

A Chef repository is another logical name that contains a version control system.

These data objects include:

- Environments
- Roles
- Data bags
- Cookbooks which include recipes, versions, attributes, templates, resources, and libraries
- Various configuration files for workstations, servers, and clients

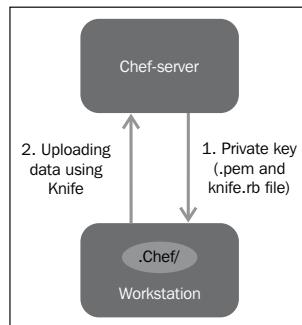
As the preceding data objects are part of a Chef server execution, we will explain all of these in the Chef server element section, later in this chapter.

## Authentication procedure between a workstation and Chef server

Here, you will understand the main authentication procedure between a workstation and Chef server.

1. A private key is required by each workstation. This private key is generated by the Chef server and it must be downloaded from the server, which is then copied to the `.chef` directory in the Chef repository. In case a new private key is required, then it should be simply regenerated from the Chef server. Also, it's required to be recopied in the Chef repository.
2. Knife is used to upload data to the Chef server from the Chef repository. To arrange nodes, it can be used by a Chef-client. These nodes are registered with the Chef server.

In the following diagram, you can understand this authentication concept easily:



## Nodes

After workstation, the second important element of the Chef framework is the node. A node can be a physical server, virtual machine or cloud instance that is managed and configured by a Chef-client. A Chef organization supports all types of nodes. A node is registered with the Chef server. Nowadays, IT infrastructure is being migrated to the cloud computing so most of the organizations use cloud instances as a node.

A Chef-client is the main component of a node. It works on each node, where it gets a choice to perform either configuration-related tasks or automation-related tasks of infrastructure.

Another component of a node is Ohai, which is used to access information of the entire system so that the information is always accessible to the Chef-client on each Chef-run process. We will discuss the node's components in more detail in the next section.

## Types of nodes

There are mainly three types of nodes.

### Physical nodes

A physical node is a server or machine that is configured to run a Chef-client. There are physical networks and devices attached to this node that allow you to send and receive information using communication channels. The simple examples of a physical node are computer systems/laptops that are attached to any communicating network.

### Virtual nodes

A virtual node refers to a virtual machine that runs on traditional hardware. We can run more than one virtual machine on a single system using various virtualization techniques such as hypervisor. Every machine works as a separate node and a Chef-client can run on that particular virtual machine.

### Cloud instances

There are various cloud providers that provide infrastructure as a service, such as Amazon, Rackspace, Linode, Windows Azure, and Google's Compute Engine. We can choose any provider and make provisions for virtual machines that are hosted by cloud providers. A running state of a hosted virtual machine is known as an instance. So, we can use a cloud instance as a node. There is an external plugin provided by Opscode that helps you to use a particular cloud provider's service. Knife uses these plugins to create instances on a cloud. After creation of an instance, Chef manages the configuration, deployment, and maintenance of these instances.

## Tools of a node

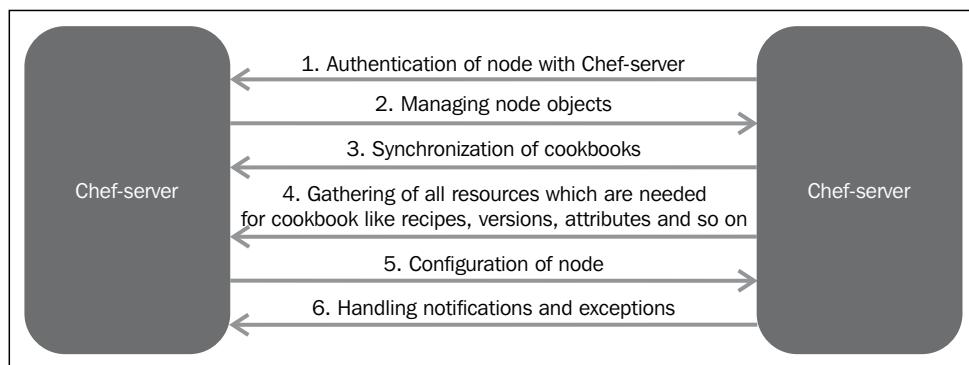
The following are the tools of a node:

### A Chef-client

A Chef-client is a program that works like an agent on every node where it is installed. It is similar to a traditional client-server interaction. Before running a Chef-client on each node, it is mandatory that you complete the registration process with the Chef server. The purpose of a Chef-client execution is to perform all the desired steps that are necessary in order to take the node to the desired (according to the user's requirement) state.

During every Chef-run, a Chef-client requests to access data that is stored in the Chef server. Therefore, the RSA public key pairs are used for authentication processes between the node and server. This authentication process ensures that only registered nodes can access information from the server.

During every Chef-client run, the following defined sequential steps take place, as you can see in the following figure:



### Ohai

Ohai is a special kind of tool that is used for the attribute identification process of every node. Ohai gives all the information related to the environment of a node. This information includes the following:

- Operating system details
- Information of the fully qualified domain name (FQDN)
- A list of host names
- Networking details

- Processor and memory usage
- Kernel data related information
- Other configuration information

All this information of a node is called automatic attributes. At the time of every Chef-run, all these attributes get configured, and after that, the Chef-client uses these attributes to validate that some properties must not be changed.

## The Chef server API

The Chef server API is a REST-based API that allows access to the Chef server resources, such as cookbooks, roles, nodes, and environments to give JSON responses. It also manages the API client list and corresponding RSA public key pairs. There are some prerequisites in order to use Chef server APIs. The following settings are necessary before you start communicating with Chef server APIs:

1. A request must be well defined. It is recommended that you use Chef::REST libraries for requests.
2. There is one authentication named the Mixlib authentication. A request cannot be processed without signing the Mixlib authentication.
3. A Chef server API version must be aligned with Chef's version.
4. The setting of the accept header must be application/JSON.
5. All these API supports are available only in version 0.10 or higher versions.

## GitHub

GitHub is a code repository website for software development projects. It a web-based hosting service for open source projects. It also offers paid services for private repositories. GitHub was launched in 2008 and in a short span of time, GitHub has 3 million users and more than 5 million repositories. GitHub works like a version control system, which means that every change in code is identified by the latest revision number, also called as a version number.

Chef users and developers frequently use GitHub's repository service . We can download a number of cookbooks from Opscode Chef's community. Apart from this, we can also download cookbooks from the repository on GitHub.

We can install the Knife GitHub plugin from <https://github.com/websterclay/knife-github-cookbooks>, after which, we can install cookbooks from <https://github.com/cookbooks/yum> using the Knife command cookbook GitHub install cookbooks/yum.

For example , in order to install yum cookbook which is located at <https://github.com/websterclay/knife-github-cookbooks>, we need to run following command:

```
knife cookbook github install cookbooks/yum
```

## Chef-solo

For beginners, it is sometimes difficult to understand the distributed framework and hence, it becomes a roadblock in the usage of Chef. Here, Chef-solo is the answer to this specific problem. It is far easier to learn and you can gain experience of Chef using Chef-solo. It is a special case where a Chef server, Chef-client, and workstation, all are located onto a single node.

Chef-solo is mostly used for learning purposes; it has a limited functionality where you can gain some hands-on experience of Chef. It does not support search indexes, node data storage, and authentication or authorization that are essential and frequently used operations in a production environment.

However, Chef-solo supports two types of location from where cookbooks can be run:

1. A Local directory.
2. A URL where tar.gz is located

Generally, it manages small-scale resources as expected, and it is not efficient for large-scale infrastructure management.

## The Chef community

The Chef community is a blend of experts and volunteers who work together to make Chef better. Community members have roles assigned based on their contributions. As of now, four types of roles are available in the Chef community:

- A decider decides the community guidelines, punitive actions, and appeals
- A community ombuds person manages activities of the community
- A community advocate assists in enforcing the community's guidelines
- Community members are active and knowledge seeking enthusiasts

The Chef community offers a number of cookbooks freely available for download, such as the Chef cookbook for databases, web server, process management, programming languages, package management, networking, operating system and virtualization, monitoring and trending, and many other utilities. These cookbooks are fully free to download and anyone can use them. If users want to contribute to the Opscode community, they can add a new cookbook and that will be available for other users. As of now there are 1,882 cookbooks and 60,343 Chefs available on the Chef community as per community's stats.

Some of the most popular cookbooks are as follows; you can download these cookbooks from the community.

## Databases

Some of the popular database related cookbooks are:

- `mysql`: This provides `mysql_service`, `mysql_config`, and `mysql_client` resources
- `Elasticsearch`: This installs and configures Elasticsearch
- `postgresql`: This installs and configures PostgreSQL for clients or servers
- `mongodb`: This installs and configures MongoDB

## Web servers

Some of the popular web server related cookbooks are:

- `apache2`: This installs and configures all aspects of Apache2
- `jboss`: This installs/configures JBoss
- `tomcat`: This installs/configures Tomcat
- `nginx`: This installs and configures Nginx
- `glassfish`: This installs/configures GlassFish Application Server

## Process management

Some of the popular process management related cookbooks are:

- `runit`: This installs runit and provides the `runit_service` definition
- `monit_bin`: This installs/configures `monit_bin`
- `Bluepill`: This installs bluepill gem and configures to manage services

## Programming languages

Some of the popular programming related cookbooks are:

- `java`: This installs Java runtime
- `php`: This installs and maintains php and php modules
- `python`: This installs Python, pip, and virtualenv
- `nodejs`: This installs/configures nodejs

## Monitoring

Some of the popular monitoring related cookbooks are:

- `nagios`: This installs and configures the Nagios server
- `newrelic_monitoring`: This installs/configures New Relic monitoring
- `collectd`: This installs and configures the collected monitoring daemon

## Package management

Some of the popular package management related cookbooks are:

- `Apt`: This configures apt and apt services and LWRPs to manage apt repositories and preferences
- `Yumrepo`: This installs and configures EPEL, ELFF, Dell, and VMware yum repositories
- `Yum`: This configures various yum components on Red Hat-like systems

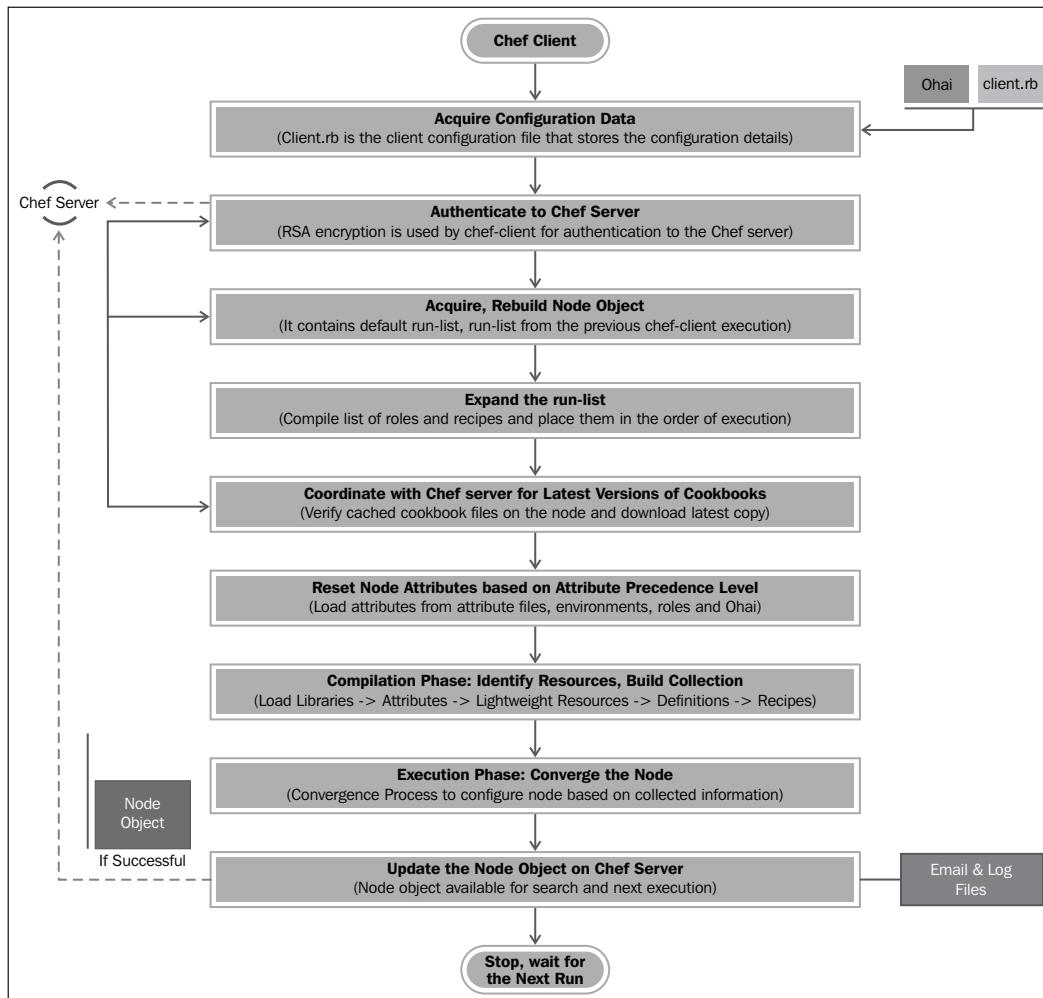
## Virtualization and cloud

Some of the popular virtualization and cloud related cookbooks are:

- `vmware_workstation`: This installs and configures VMware Workstation 10
- `aws`: This is used to manage AWS resources
- `vmware`: This installs VMware tools
- `openstack`: This is a library cookbook to interact with OpenStack Clouds

## Chef-run

What happens when a Chef-client runs? Chef-run is performed when a Chef-client configures a node. Chef-run is the term used to describe the sequence of steps to configure the node. The following figure represents the major sequential steps undertaken during a chef-run:



## Integration of Chef with Vagrant

Vagrant is a software used for open source information. It is also used to build and distribute virtualized development environments. To use Vagrant, we require virtual machines so it can be used with the help of VirtualBox or VMware workstation.

In order to test Chef cookbooks before they go in a production environment, or for learning purposes, Vagrant is a good way to start with Chef scripting.

The following are the benefits of using Vagrant with Chef:

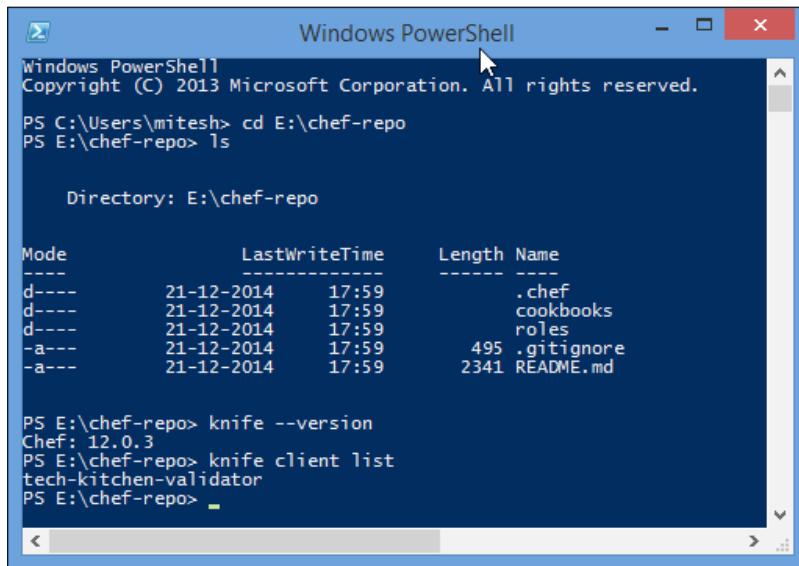
- Vagrant is a lightweight command-line wrapper, which can provision for isolated production and a test environment by just one command
- The environment created by Vagrant is very flexible, which means that it can be destroyed, rebuilt anytime, and anyone in the team can use that
- Vagrant allows us to start, stop, and destroy virtual machines with good abstraction layer
- A Vagrant file is the main file of Vagrant that is written in a Ruby file, which is placed in the project root and allows us to write our environment declaratively
- Vagrant works on the concept of boxes that defines preconfigured base VMs after which we can start work
- It is a very good and recommended way to develop and test cookbooks in your workstation

## A quick hands-on experience of a Hosted Chef server

Let's experience the workings of a Hosted Chef server:

1. We only need to sign up and we can start working with automation.
2. Create an organization named `tech-kitchen`; the concept of organization provides multitenancy as it has a unique name across all organizations that exist on Hosted Chef.
3. Download the **Starter Kit**.
4. It will download `chef-starter.zip`.
5. Unzip/Extract `chef-starter.zip`.
6. It will create the `chef-repo` directory.

7. Download Chef-client from <https://www.chef.io/download-chef-client/> and perform the following steps:
  - Select Operating System: Windows.
  - Select version of Operating System: 8.
  - Select an Architecture: x86\_64.
  - Select Chef version: 12.0.3-1.
  - Download Chef-client by clicking on the link available on the download page.
8. Install the Chef-client and **Restart** the system to allow environment variables to take effect.
9. Search **Windows PowerShell** in your system and open it.
10. Change **Directory** to chef-repo.
11. Run the command, `ls` to know the directory's structure.
12. Verify the Chef-client version with the command: `knife --version`.



```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\mitesh> cd E:\chef-repo
PS E:\chef-repo> ls

Directory: E:\chef-repo

Mode                LastWriteTime     Length Name
----                -              -          -
d----
```

13. Verify the list of clients configured on Chef server the `knife client list` command.

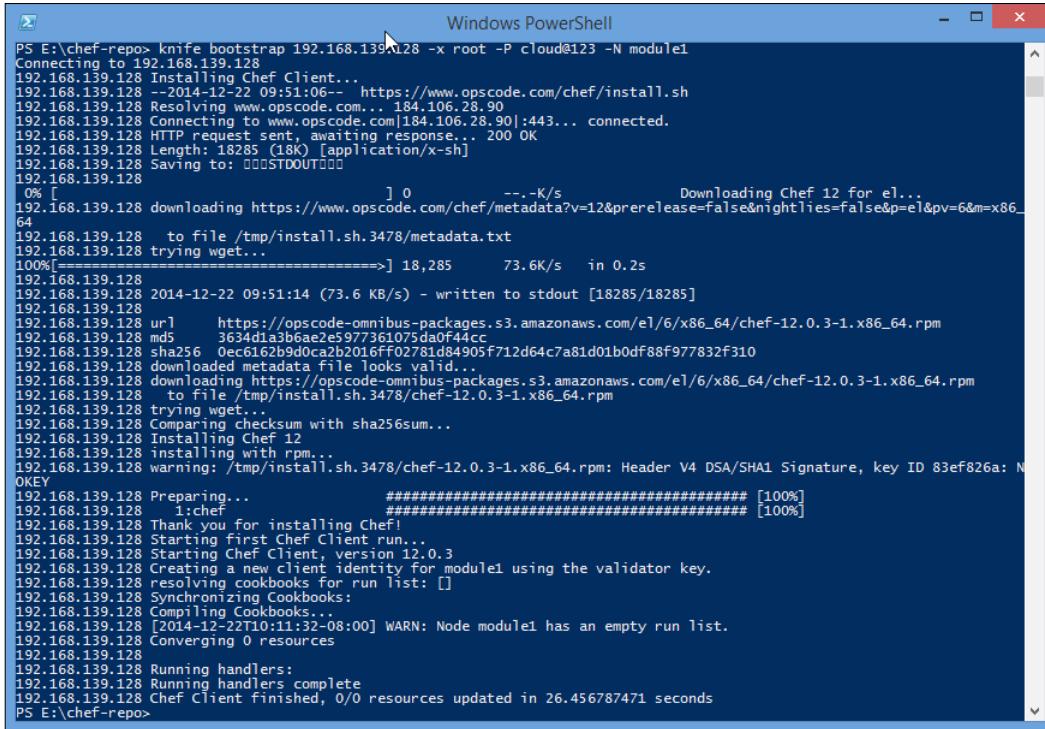
The `/chef-repo/.chef/knife.rb` file contains information of the Chef server and organization, as shown following:

 Refer to [https://docs.chef.io/config\\_rb\\_knife.html](https://docs.chef.io/config_rb_knife.html) for more information on Knife configuration options.

```
current_dir = File.dirname(__FILE__)
log_level           :info
log_location        STDOUT
node_name           "mitesh51"
client_key          "#{current_dir}/mitesh51.pem"
validation_client_name "tech-kitchen-validator"
validation_key       "#{current_dir}/tech-kitchen-
validator.pem"
chef_server_url     "https://api.opscode.com/
organizations/tech-kitchen"
cache_type          'BasicFile'
cache_options( :path => "#{ENV['HOME']}/.chef/checksums"
)
cookbook_path        ["#{current_dir}/../cookbooks"]
```

14. Create a Virtual Machine (VM) with CentOS in VMware Workstation or VirtualBox. We have used VMware workstation here.
15. Start the CentOS VM.
16. Use PuTTY from Windows 8 to verify that we connect to the CentOS VM with SSH.
17. Use the command, `knife bootstrap 192.168.139.128 -x root -P cloud@123 -N module1`; it will download, install, configure, and run Chef-client on CentOS VM.
  - Bootstrap machine having IP address 192.168.139.128 (verify the IP address using the `ifconfig` command on CentOS VM and replace this IP address with it)
  - User name: `root`
  - Password: `secr@t@123`

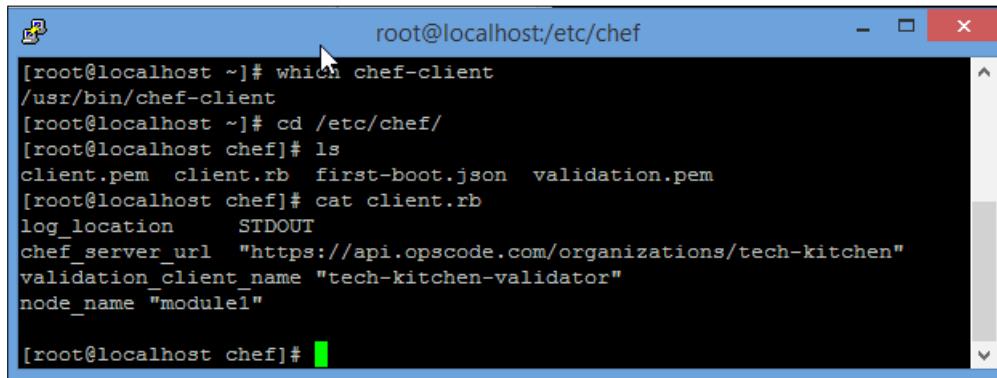
- -N: Module name that the Chef server will use to identify the node:



```

PS E:\chef-repo> knife bootstrap 192.168.139.128 -x root -P cloud@123 -N module1
Connecting to 192.168.139.128
192.168.139.128 Installing Chef Client...
192.168.139.128 --2014-12-22 09:51:06-- https://www.opscode.com/chef/install.sh
192.168.139.128 Resolving www.opscode.com... 184.106.28.90
192.168.139.128 Connecting to www.opscode.com|184.106.28.90|:443... connected.
192.168.139.128 HTTP request sent, awaiting response... 200 OK
192.168.139.128 Length: 18285 [application/x-sh]
192.168.139.128 Saving to: <>STDOUT<>
192.168.139.128
0% [=====] 0          0K/s      Downloading Chef 12 for el...
192.168.139.128 downloading https://www.opscode.com/chef/metadata?v=12&prerelease=false&nightlies=false&p=el&pv=6&m=x86_64
192.168.139.128 to file /tmp/install.sh.3478/metadata.txt
192.168.139.128 trying wget...
100% [=====] 18,285    73.6K/s  in 0.2s
192.168.139.128 2014-12-22 09:51:14 (73.6 KB/s) - written to stdout [18285/18285]
192.168.139.128
192.168.139.128 url   https://opscode-omnibus-packages.s3.amazonaws.com/el/6/x86_64/chef-12.0.3-1.x86_64.rpm
192.168.139.128 md5   3634d1a3b6ae2e5977361075da0f44cc
192.168.139.128 sha256 0ec6162b9d0cazb2016ff02781d84905f712d64c7a81d01b0df88f977832f310
192.168.139.128 downloaded metadata file looks valid...
192.168.139.128 downloading https://opscode-omnibus-packages.s3.amazonaws.com/el/6/x86_64/chef-12.0.3-1.x86_64.rpm
192.168.139.128 to file /tmp/install.sh.3478/chef-12.0.3-1.x86_64.rpm
192.168.139.128 trying wget...
192.168.139.128 Comparing checksum with sha256sum...
192.168.139.128 Installing Chef 12
192.168.139.128 installing with rpm...
192.168.139.128 warning: /tmp/install.sh.3478/chef-12.0.3-1.x86_64.rpm: Header V4 DSA/SHA1 Signature, key ID 83ef826a: NOKEY
192.168.139.128 Preparing... #####
192.168.139.128 1:chef #####
192.168.139.128 Thank you for installing Chef!
192.168.139.128 Starting first Chef Client run...
192.168.139.128 Creating a new client identity for module1 using the validator key.
192.168.139.128 resolving cookbooks for run list: []
192.168.139.128 Synchronizing Cookbooks:
192.168.139.128 Compiling Cookbooks...
192.168.139.128 [2014-12-22T10:11:32-08:00] WARN: Node module1 has an empty run list.
192.168.139.128 Converging 0 resources
192.168.139.128
192.168.139.128 Running handlers:
192.168.139.128 Running handlers complete
192.168.139.128 Chef Client finished, 0/0 resources updated in 26.456787471 seconds
PS E:\chef-repo>
```

### 18. Verify the Chef-client on CentOS VM:



```

root@localhost ~]# which chef-client
/usr/bin/chef-client
[root@localhost ~]# cd /etc/chef/
[root@localhost chef]# ls
client.pem  client.rb  first-boot.json  validation.pem
[root@localhost chef]# cat client.rb
log_location      STDOUT
chef_server_url  "https://api.opscode.com/organizations/tech-kitchen"
validation_client_name "tech-kitchen-validator"
node_name "module1"

[root@localhost chef]#
```

## Different Components of Chef's Anatomy

---

### 19. Verify the Hosted Chef Dashboard:

The screenshot shows the Chef Dashboard interface. At the top, there are tabs for Nodes, Reports, Policy, and Administration. The Nodes tab is selected, showing a table titled "Showing All Nodes". The table has columns for Node Name, Platform, FQDN, IP Address, Uptime, Last Check-in, Environment, and Actions. One row is highlighted for "module1", which is a CentOS VM with IP 192.168.139.128, uptime of an hour, last checked in 24 minutes ago, and environment set to "default". Below the table, a modal window is open for "Node: module1". It has tabs for Details, Attributes, and Permissions. The Details tab is selected, displaying information such as Last Check In (24 Minutes Ago), Uptime (An Hour), and Environment (\_default). The Attributes tab shows the node's platform as centos, FQDN as localhost, and IP Address as 192.168.139.128.

### 20. Let's look at the sample recipe to install Apache server (`httpd`) on CentOS VM and start the service:

- Create a cookbook with the `knife cookbook create apache` command:

```
PS E:\chef-repo> knife cookbook create apache
** Creating cookbook apache in E:/chef-repo/cookbooks
** Creating README for cookbook: apache
** Creating CHANGELOG for cookbook: apache
** Creating metadata for cookbook: apache
PS E:\chef-repo> ls .\cookbooks\apache
```

Directory: E:\chef-repo\cookbooks\apache

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
d----	23-12-2014	00:13	attributes
d----	23-12-2014	00:13	definitions
d----	23-12-2014	00:13	files
d----	23-12-2014	00:13	libraries
d----	23-12-2014	00:13	providers
d----	23-12-2014	00:13	recipes

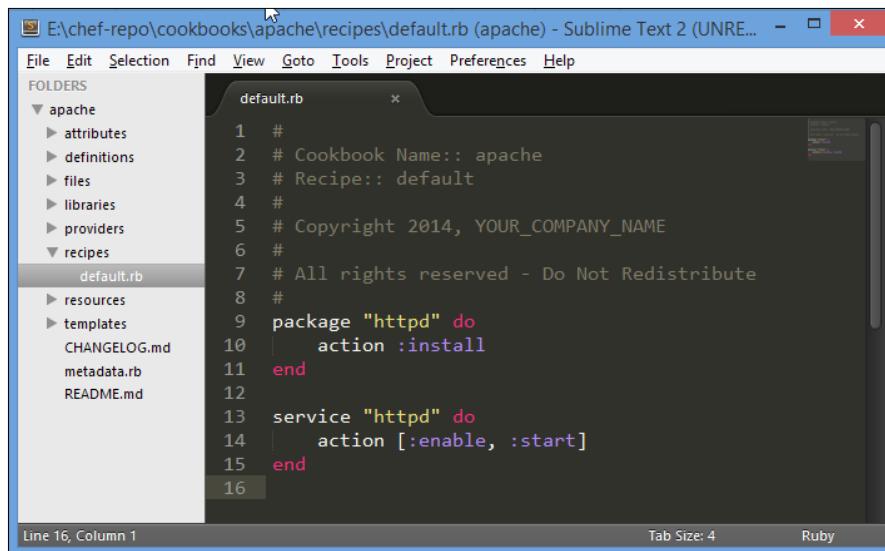
```

d---- 23-12-2014 00:13 resources
d---- 23-12-2014 00:13 templates
-a--- 23-12-2014 00:13 464 CHANGELOG.md
-a--- 23-12-2014 00:13 283 metadata.rb
-a--- 23-12-2014 00:13 1516 README.md

```

- Open the default.rb recipe by navigating to \chef-repo\ cookbooks\apache\recipes\default.rb.

Write "package" and "service" resources as given in the following screenshot:



```

E:\chef-repo\cookbooks\apache\recipes\default.rb (apache) - Sublime Text 2 (UNRE... - □ ×
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
apache
  attributes
  definitions
  files
  libraries
  providers
  recipes
    default.rb
  resources
  templates
CHANGELOG.md
metadata.rb
README.md

default.rb
1  #
2  # Cookbook Name:: apache
3  # Recipe:: default
4  #
5  # Copyright 2014, YOUR_COMPANY_NAME
6  #
7  # All rights reserved - Do Not Redistribute
8  #
9  package "httpd" do
10   action :install
11 end
12
13 service "httpd" do
14   action [:enable, :start]
15 end
16

Line 16, Column 1                                     Tab Size: 4                                     Ruby

```

21. Publish an Apache cookbook to the Chef server with the knife cookbook upload apache command. We will get the following output:

```

Uploading apache      [0.1.0]
Uploaded 1 cookbook.

```

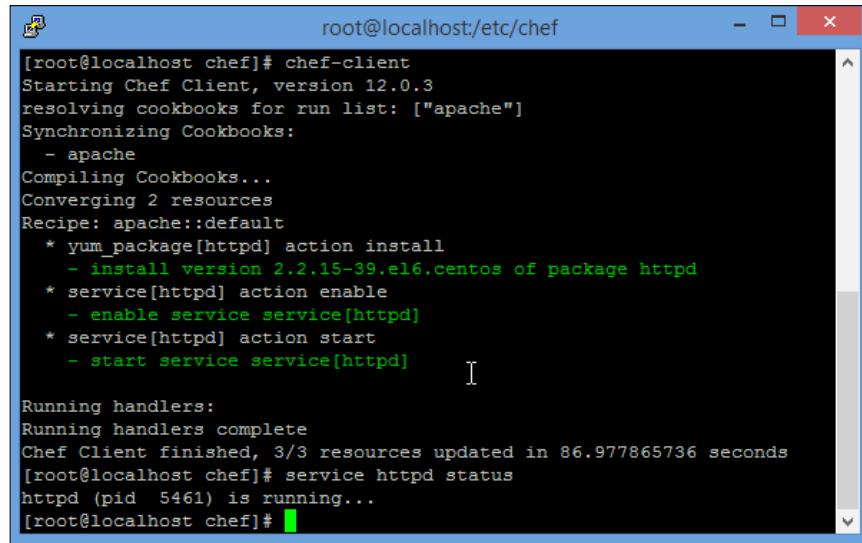
22. Create a **Run-list**, ordered set of recipes, and roles with the knife node run\_list add module1 "recipe[apache]" command. It stores the run-list on the Hosted Chef server on the node object in the Chef server. We get the following output:

```

module1:
run_list: recipe[apache]

```

23. Now, run the Chef-client from CentOS VM and verify the status of httpd:



The screenshot shows a terminal window titled "root@localhost:etc/chef". The output of the command "chef-client" is displayed. It starts by resolving cookbooks for the run list ["apache"], synchronizing cookbooks, and compiling them. It then converges 2 resources under the "apache::default" recipe. The resources include installing the httpd package, enabling and starting the httpd service. Finally, it runs handlers, which complete the process. The total duration is 86.977865736 seconds. A command to check the status of the httpd service is shown at the end.

```
[root@localhost chef]# chef-client
Starting Chef Client, version 12.0.3
resolving cookbooks for run list: ["apache"]
Synchronizing Cookbooks:
  - apache
Compiling Cookbooks...
Converging 2 resources
Recipe: apache::default
  * yum_package[httpd] action install
    - install version 2.2.15-39.el6.centos of package httpd
  * service[httpd] action enable
    - enable service service[httpd]
  * service[httpd] action start
    - start service service[httpd]

Running handlers:
Running handlers complete
Chef Client finished, 3/3 resources updated in 86.977865736 seconds
[root@localhost chef]# service httpd status
httpd (pid 5461) is running...
[root@localhost chef]#
```

## Self-test questions

1. Explain components of Chef Framework.
2. What are the main functionalities of Chef server?
3. Which are the three types of Chef server and explain significance of each of it.
4. What is a cookbook and what are its main functionalities?
5. Explain the role of workstation in Chef Framework.
6. Explain different types of nodes.
7. Explain the procedure when a Chef client runs.

## Summary

So far, we understood the details of working with each and every component of Chef and how they relate to each other during the Chef-run process. It has provided us with a better conceptual view and a clear picture of Chef's automation.

Here, we also got the understanding of GitHub, the Chef community, and Vagrant. In the next chapter, we will perform a hands-on exercise using VirtualBox and will test some cookbooks on VM. We have also experienced how Chef components work together with the use of Hosted Chef.

Now, we are ready to perform hands-on exercises with Chef. From the next chapter onwards, we will perform experimental exercises with every aspect of Chef.



# 3

## Workstation Setup and Cookbook Creation

*"An organized system of machines, to which motion is communicated by the transmitting mechanism from a central automation, is the most developed form of production by machinery."*

- Karl Marx

So far, we have the basic, conceptual understanding of automation and the various components of Chef. We have also completed a hands-on exercise with Hosted Chef to understand how node, workstation, and hosted Chef server interact with one another. Now, it is time to perform a few hands-on exercises with Chef in a detailed manner. In order to write our cookbook, setup of a workstation is essential. In this chapter, we will go through the step-by-step installation procedure of all the required prerequisites, which are important to set up the workstation. We are also focusing on various troubleshooting steps, which is one of the prerequisites to become a good Chef programmer.

In this chapter, you will learn:

- VirtualBox installation
- Vagrant installation
- Git installation
- How to set up workstation on Windows and CentOS to create a Chef repository
- How to launch a virtual machine with Vagrant
- How to create and upload a simple cookbook
- Troubleshooting

All these topics come under two broad categories:

- Workstation preparation:

To start learning Chef, it is necessary that you prepare a workstation. The following are the prerequisites to start with Chef:

- **The VirtualBox installation:** This provides a local virtual machine to manage the use of Chef.
- **The Vagrant installation:** This provides a command-line interface to manage a virtual machine.
- **The Git installation:** This provides the revision control of our Chef code.
- **The Ruby installation and required settings:** Chef runs on Ruby and therefore, installation of a valid Ruby runtime is required. Also, some initial settings are required.
- **Creating a Chef repository:** The chef repository provides a place or structure to categorize cookbooks, roles, environments, configuration files, and so on to represent automation details. It also organizes versions, cookbook attributes, recipes, resources, templates, and libraries. It is essentially done at the time of the cookbook development.
- **Launching a virtual machine:** How to launch virtual machine, required setting for virtual machine, creating directory for your work, and Vagrant file setup with Git repository.
- **Managing authentication files (.Pem files):** How to set a Chef directory with the required authentication files and store them in a Git repository.

- Cookbook creation and troubleshooting:

After setting up a workstation, the next step is to learn about the cookbook.

- **Creating and uploading a simple cookbook:** After the workstation setup is done, it is required that you know how to create and upload a cookbook.
- **Troubleshooting:** While running various chef commands, some outputs/results may differ from expected. In order to overcome this, we must have some knowledge of troubleshooting.

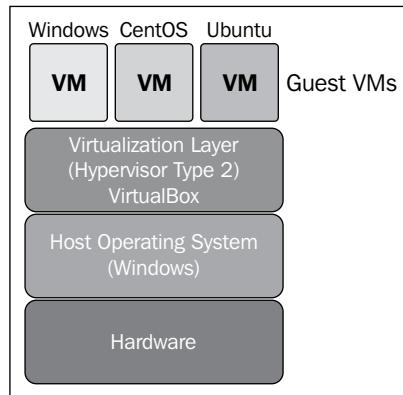
Let's start with the step-by-step installation of all the prerequisites.

Here, we are considering a Windows OS for our hands-on exercise. We can do the same with Linux also. Although, in industries and organizations Linux machines are more preferred for a Chef set up, but here, we are starting with a Windows OS to download the Chef installable files; and centOS virtual machine for command-line operations, so that we can get familiar with the workings of all types of OSes.

We can see that all the components such as VirtualBox, Vagrant, and Git have the multiple operating system download option. We can select the same according to our convenience.

## The VirtualBox installation

VirtualBox is a virtualization platform that can be used to run virtual machines on a local workstation. With the help of VirtualBox, we can run multiple operating systems on one host machine. For example, If we are using a Windows operating system, we can still run some other operating systems, such as Linux, Ubuntu, and so on over Windows OS. VirtualBox is a Type-2 hypervisor, which is also known as hosted hypervisors. Following diagram represents hosted hypervisor behavior:



## Workstation Setup and Cookbook Creation

VirtualBox provides one hypervisor layer over the host operating system, so that all the other operating systems can act as guest operating systems and we can perform the required operations and settings on guest operating systems. Let's start with the step-by-step installation procedure of VirtualBox:

1. Go to <http://www.virtualbox.org>.



2. Go to the **Downloads** section and select the **Downloads** option, **VirtualBox 4.3.20 for Windows host x86/amd64** if you are using a Windows OS.

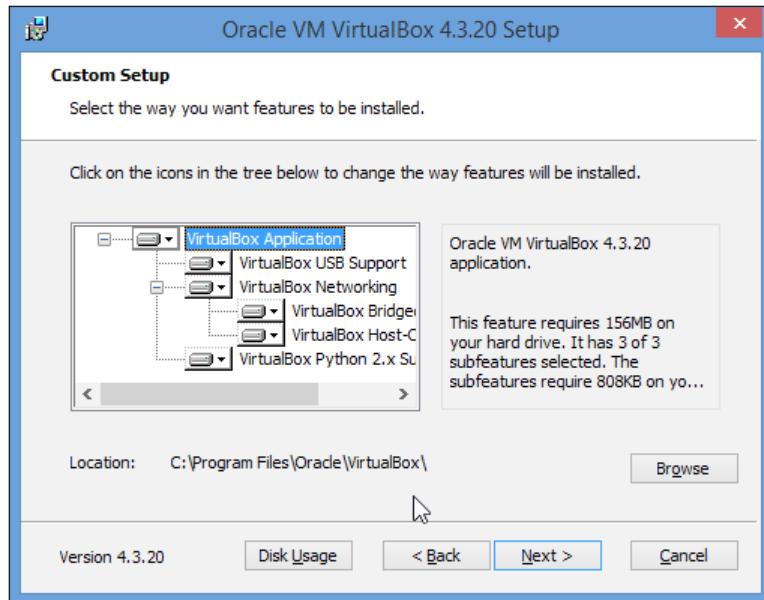


3. After selecting the appropriate version, VirtualBox will start downloading. It will take a few minutes to get downloaded.

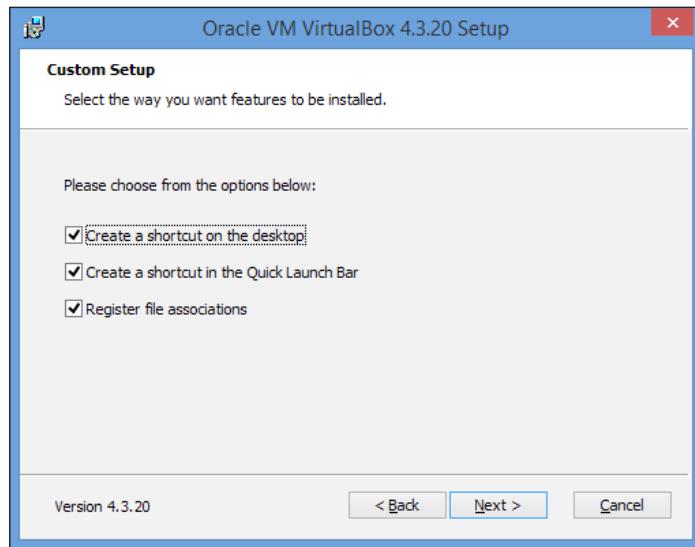
4. Once **VirtualBox 4.3.20** gets downloaded, **Run it**.



5. Now, click on **Next** and select the features you want to install.



6. Select the way you want the features to be installed in **Custom Setup** and click on **Next**.



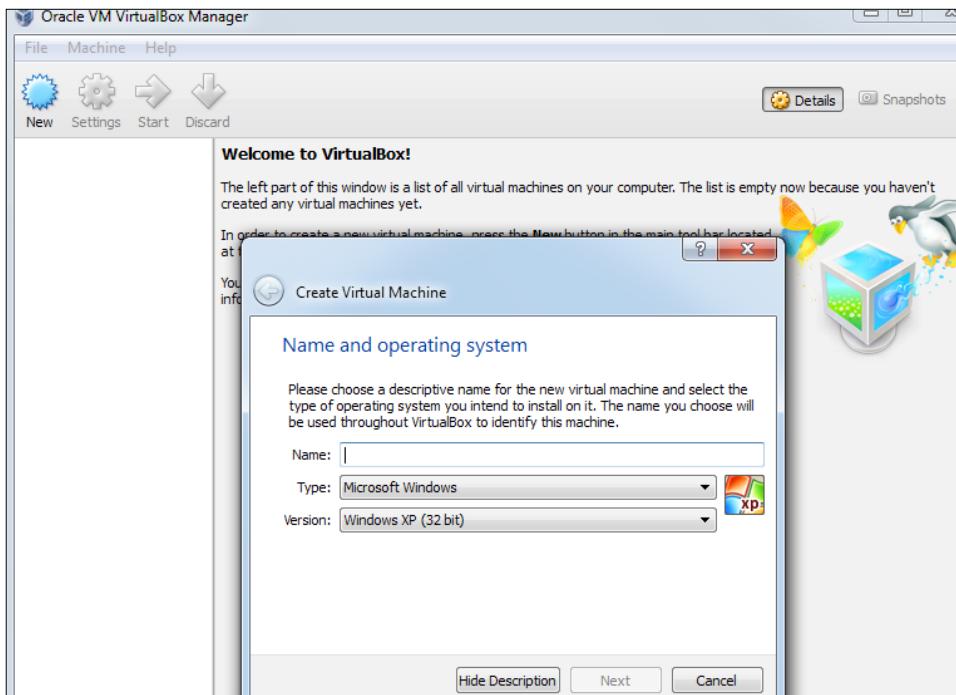
7. After clicking on **Next**, you may get a warning message, because installation procedure may reset your network connection and you may get disconnected temporarily.



8. Accept **Yes** followed by some more **Next** clicks. Click on **Finish** and your VirtualBox is ready for use.



9. After the complete installation. If you click on the **VirtualBox** icon, you will get the following screen:

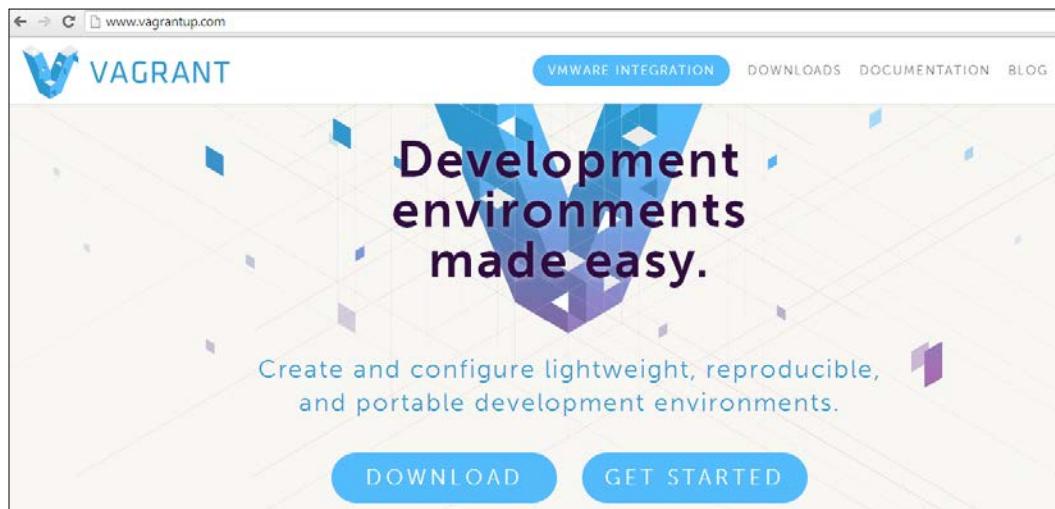


For practice purposes, install VirtualBox on Linux-based operating systems, such as Ubuntu, CentOS, or OpenSUSE.

## The Vagrant installation

Vagrant is an open source and Ruby-based infrastructure provisioning solutions while Chef is an open source configuration management tool. Vagrant works with several virtualization and cloud platforms, such as VirtualBox, VMware Workstation, and other hypervisors, VMware, and AWS to provision resources. Chef can be used to install and configure packages on the resources provisioned with the use of Vagrant. Vagrant works on a 32-bit and 64-bit edition of Windows, 32-bit and 64-bit flavor of Linux Debian, 32-bit and 64-bit flavor of Linux RPM, and 32-bit and 64-bit flavor of Mac OS X environments. Now, it also supports Docker containers.

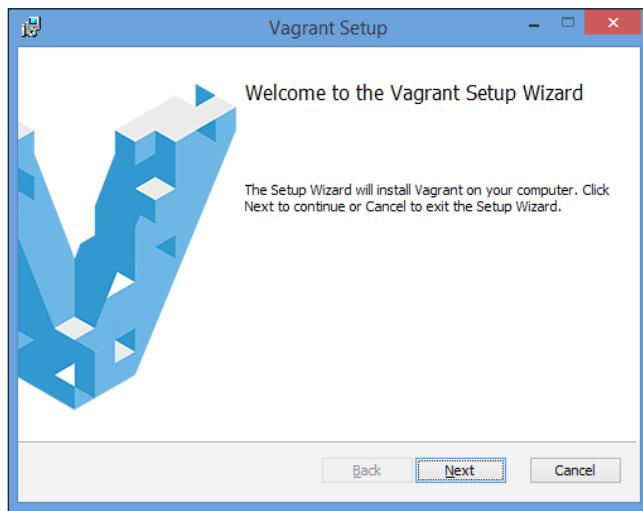
1. Go to <http://vagrantup.com>.



2. Select **Vagrant 1.7.1** in the **DOWNLOAD** option of **WINDOWS Universal (32 and 64-bit)** if you are using a Windows OS. It will take a few minutes to download.

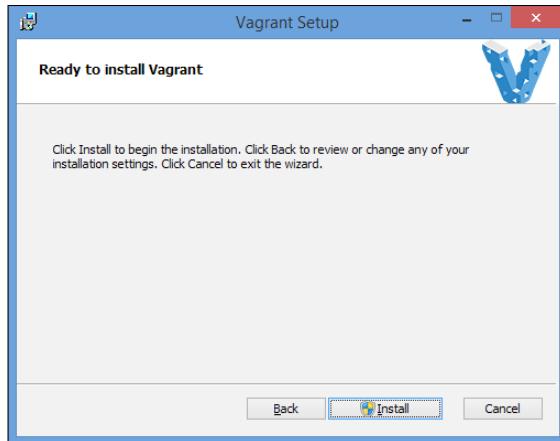


3. Once the **Vagrant 1.7.1** download gets completed, run the installable file.

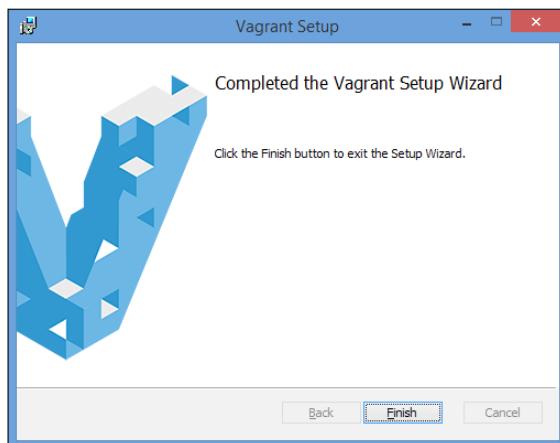


4. Click on **Accept** for the **End-User License Agreement** and click on **Next**; select the **Destination** folder to install Vagrant and click on **Next**.

5. After that, click on the **Install** option.



6. Now, the Vagrant 1.7.1 installation is finished and it is ready to use.



After the installation is completed, verify Vagrant's installation on the Windows 8 machine. Open **Windows PowerShell** and execute the `vagrant -v` command, as shown in the following screenshot:

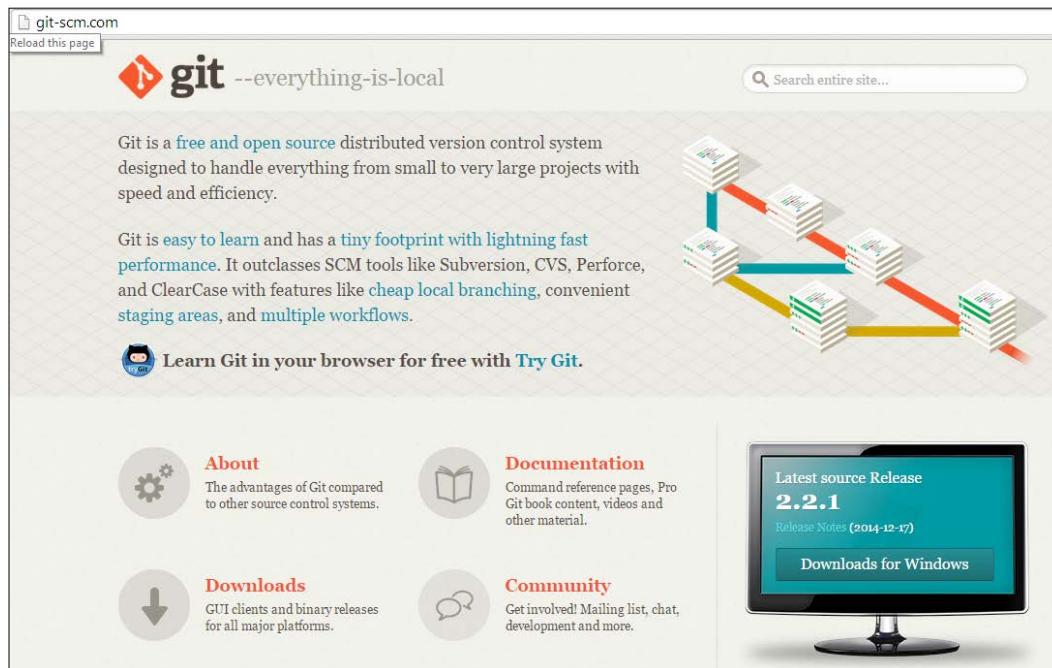
A screenshot of a Windows PowerShell window titled "Windows PowerShell". The title bar includes the Microsoft logo and the text "Windows PowerShell". The window shows the command prompt "PS C:\Users\mitesh>" followed by the output of the `vagrant -v` command: "vagrant 1.7.1". The background of the window is dark blue, and the text is white.

For practice purpose, install Vagrant on Linux-based operating systems, such as Ubuntu, CentOS, or OpenSuse.

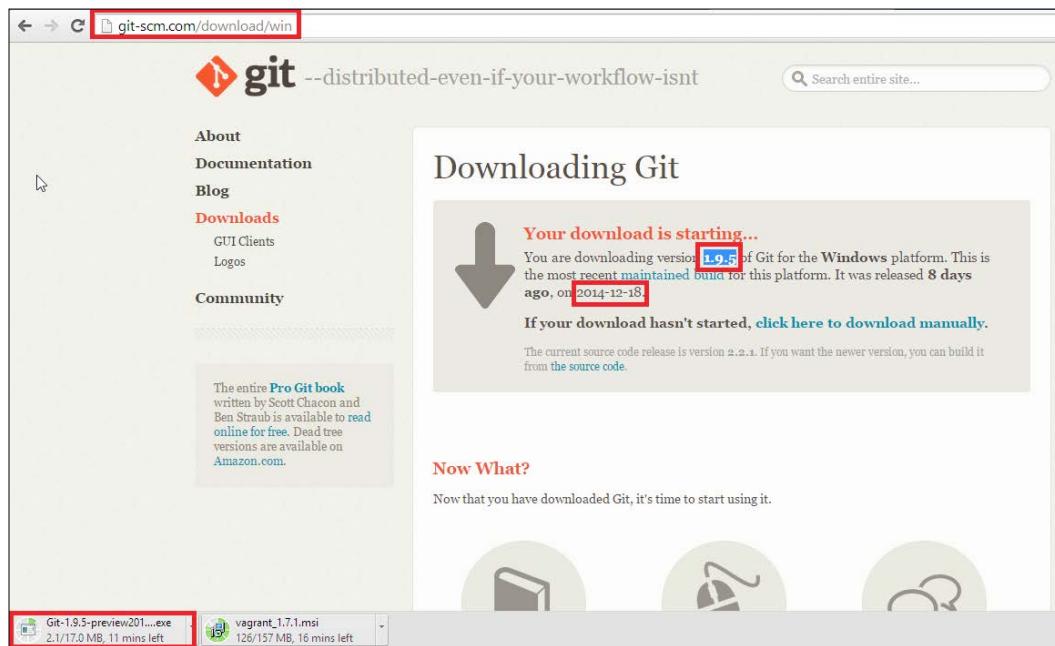
## The Git installation

With Chef, you will be managing your infrastructure as code. All of the code should be stored in a **Version control system (VCS)**. Here, we are using the Git repository to store code revisions.

1. Go to <http://git-scm.com/>.



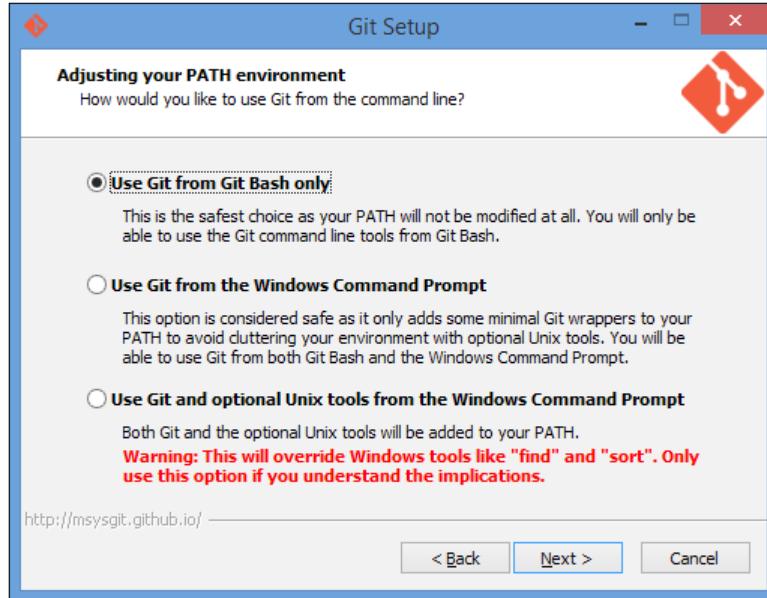
2. Go to the **Downloads** section for the Windows option with the latest 1.9.5 version.



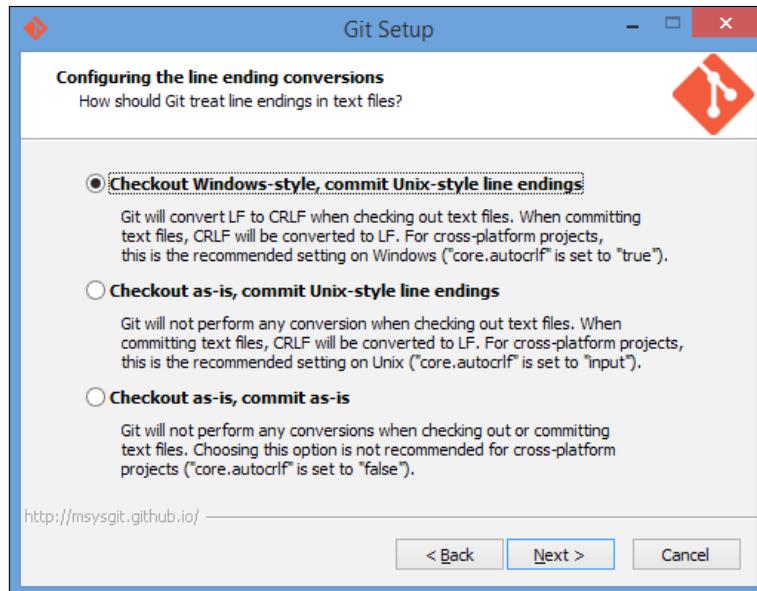
3. It will take a few minutes to get downloaded. After that, run the installable file.
4. The Git setup will start immediately. Click on the **Next** tab.



5. Select the **Use Git from Git Bash only** option, it will give you a command-line interface to run Git commands.



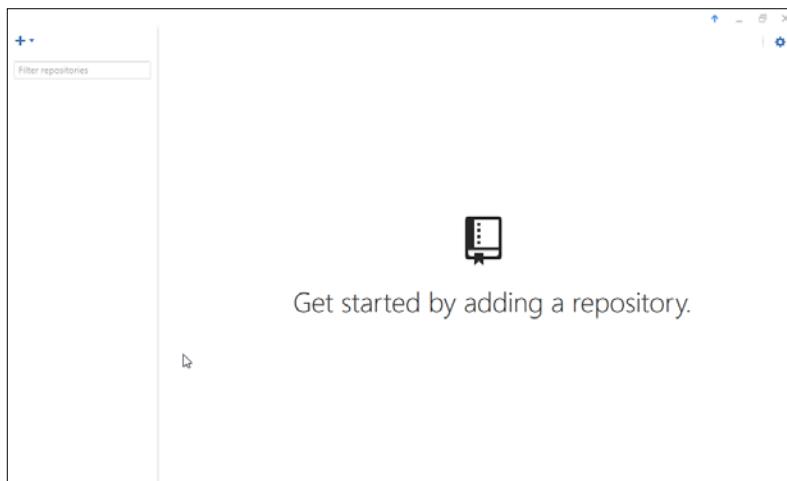
6. After that, select the **Checkout Window-style, commit Unix-style line endings** option.



7. After selecting the preceding option, click on **Finish**. The Git installation procedure is finished and ready for use.



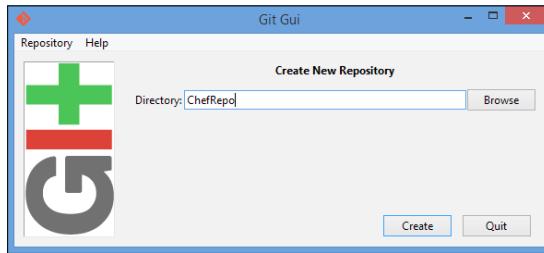
8. Now, we can see the following screen after clicking on the **Git** icon.



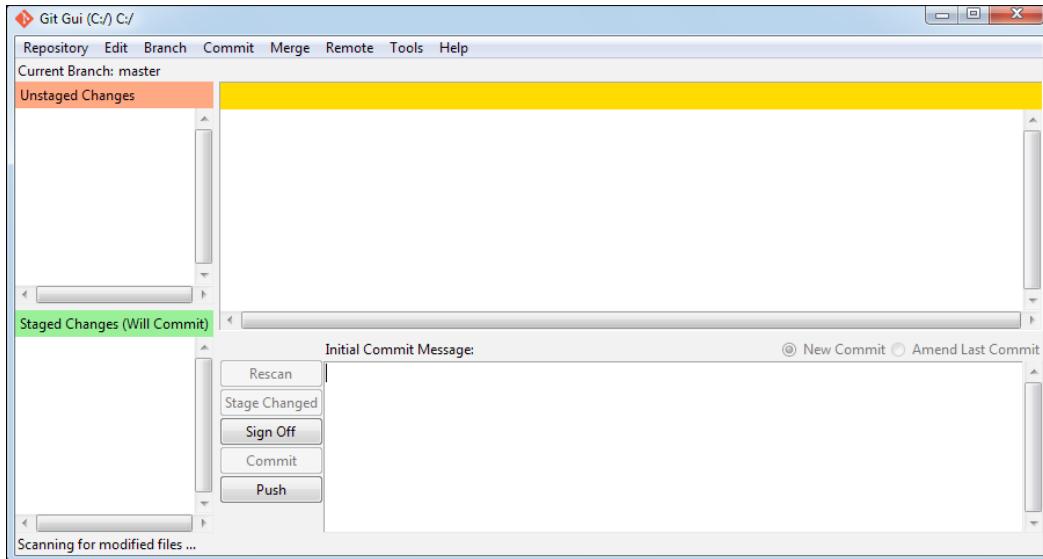
9. Open Git Gui.



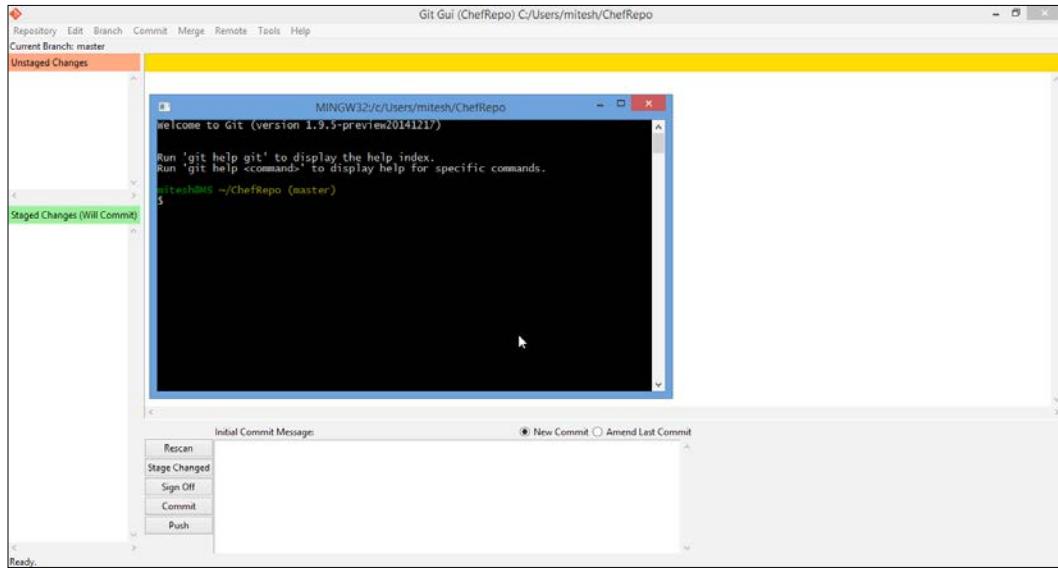
10. In the Directory field, enter ChefRepo.



11. Once this is done, click on **Create**. The **Git Gui** window will appear as follows:



12. In the **Repository** tab, select the **Git Bash** option, and run the Git commands.



13. Once the preceding installation procedure is completed, we need to set the user name and e-mail immediately. Git's commit operations use these settings.
14. Open **Git bash** and update the following commands:

```
gitconfig --global user.name "<enter your name>"  
gitconfig --global user.email "<enter your email>"
```

For practice purpose, install Git on Linux-based operating systems, such as Ubuntu, CentOS, or OpenSuse.

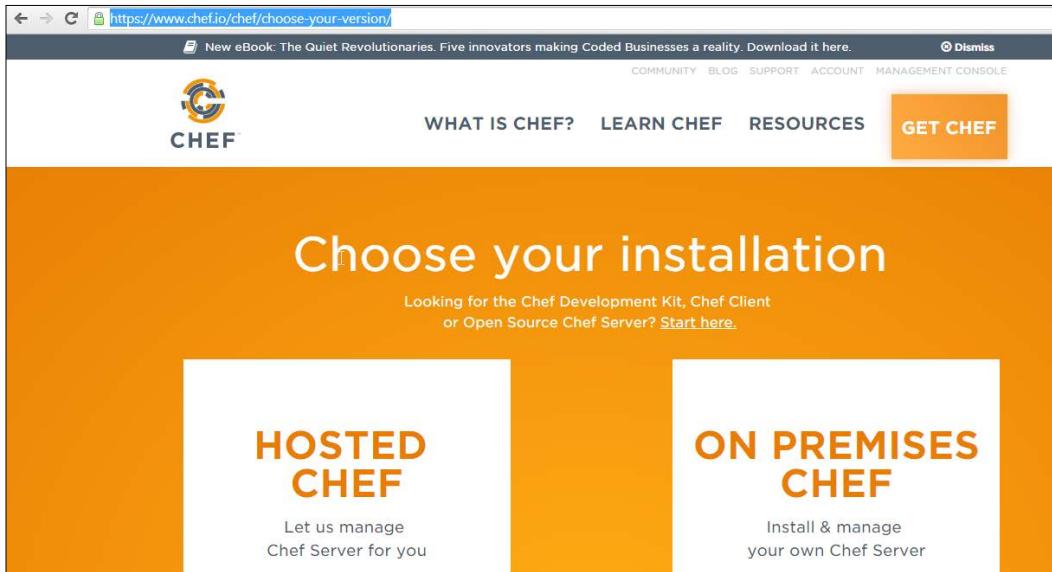
## Installation and configuration of a workstation

To setup a workstation on a Windows-based machine, follow the given steps:

Download a Chef-client version based on your operating system; in our case, it is Windows:

1. Visit <https://www.chef.io/chef/choose-your-version/>.

2. Select HOSTED CHEF.



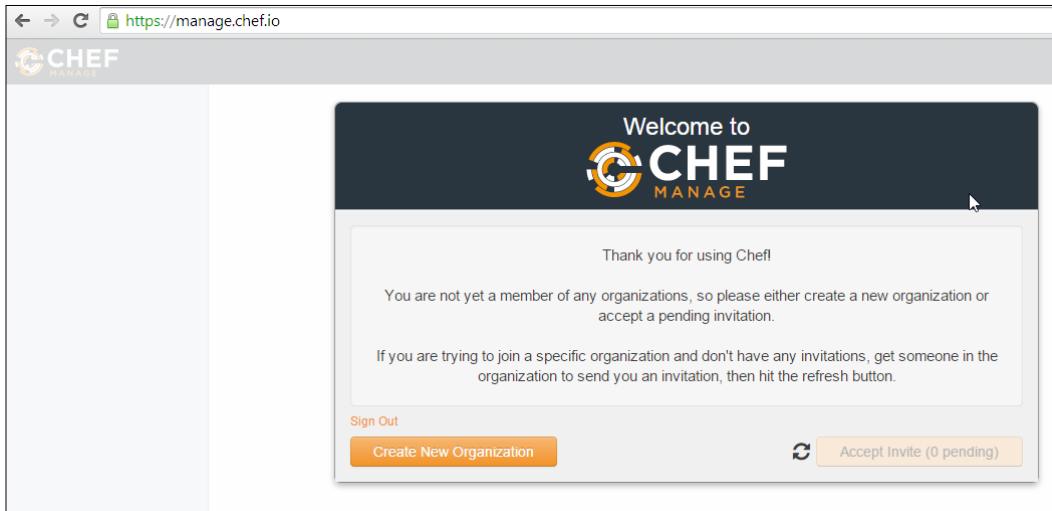
3. After this, you need to fill in the sign up details with Hosted Chef. We only need to sign up and we can start working with automation.

The screenshot shows the sign-up page for Hosted Chef at <https://manage.chef.io/signup>. The page features a 'Start your free trial of hosted Chef' heading and a brief description of the service. It contains five input fields: 'Full Name' (Mitesh Soni), 'Email' (mitesh.soni@outlook.com), 'Username' (mitesh52), 'Password' (redacted), and 'Company' (OOP). A checkbox for agreeing to the 'Terms of Service' and 'Master License and Services Agreement' is checked. To the right of the form is a sidebar with links for existing users ('Already have an account? Click here to sign in'), open-source Chef ('Looking for open-source Chef? Start with the Chef client and server installation and check out our extensive documentation.'), and the Chef community ('Join the Chef Community Join our worldwide developer community!'). At the bottom, there's a copyright notice ('Copyright © 2012 – 2014 Chef Software, Inc.') and a help link ('Need help? If you have questions or you're stuck, we are here to help.').

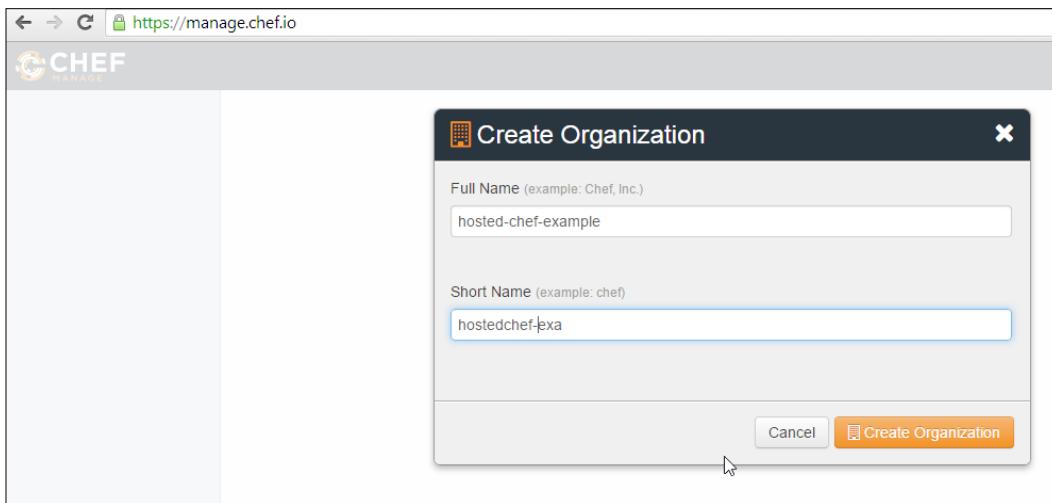
*Workstation Setup and Cookbook Creation*

---

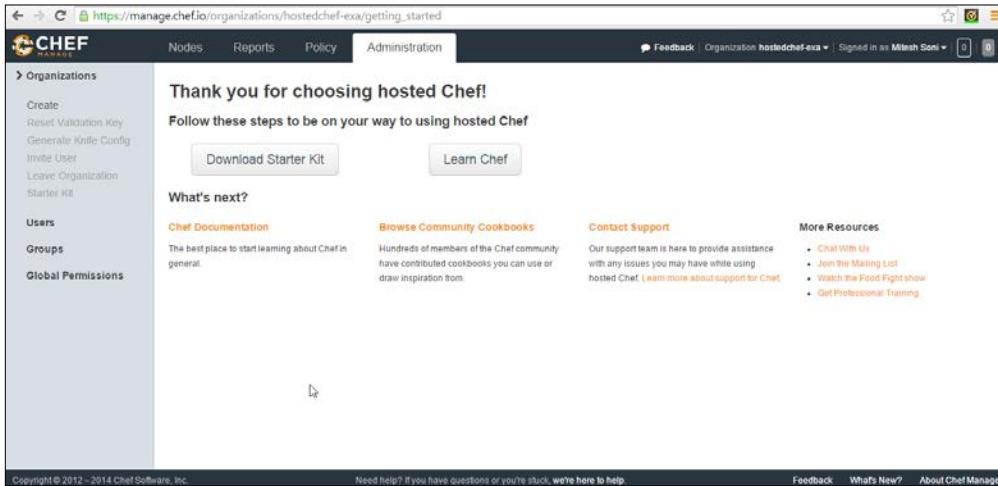
- Once the sign up is completed, click on the **Create New Organization** tab.



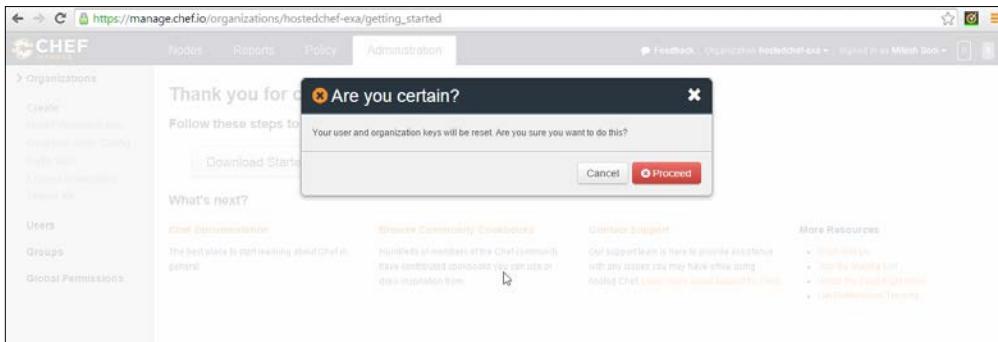
- Now, we need to fill in the details of your organization with the **Full Name** and **Short Name** of your organization.



6. Then, click on **Create Organization**. We will get the following options; we need to select **Download Starter Kit**.



- It will download `chef-starter.zip`.

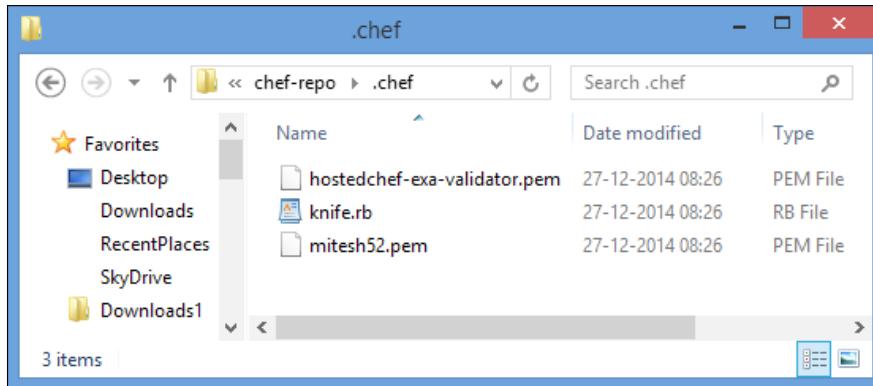


7. Unzip/Extract `chef-starter.zip`; after extracting, we will get a folder named `.chef`.

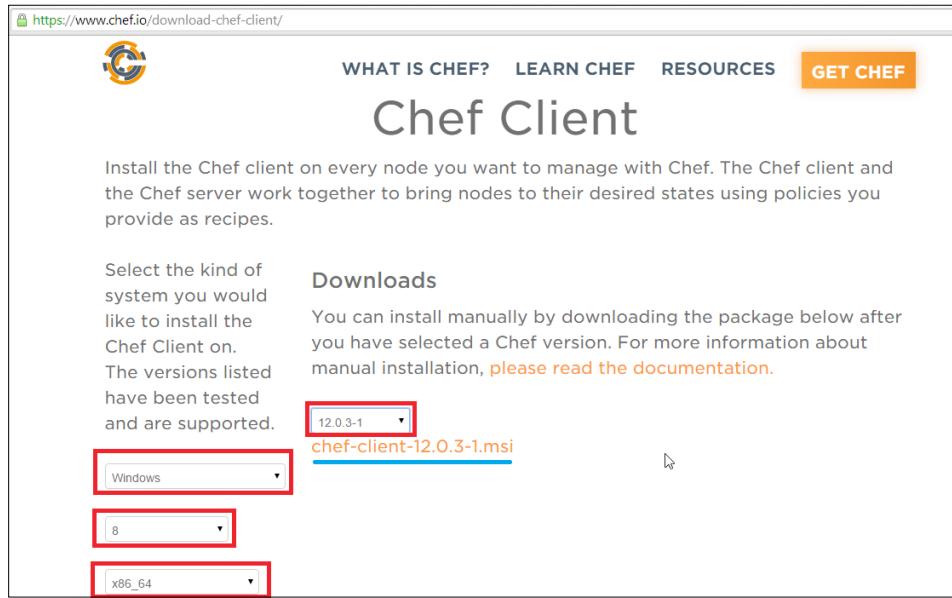
Downloads > chef-starter > chef-repo >						
11 files						
Name	Type	Compressed size	Password ...	Size	Ratio	
.chef	File folder					
cookbooks	File folder					
roles	File folder					
README.md	Text Document	1 KB	No	1 KB	33%	
Vagrantfile	MD File	2 KB	No	3 KB	57%	
	File	2 KB	No	4 KB	58%	

8. The following are the three main authentication files of Chef. You need to verify these files in the .chef folder:

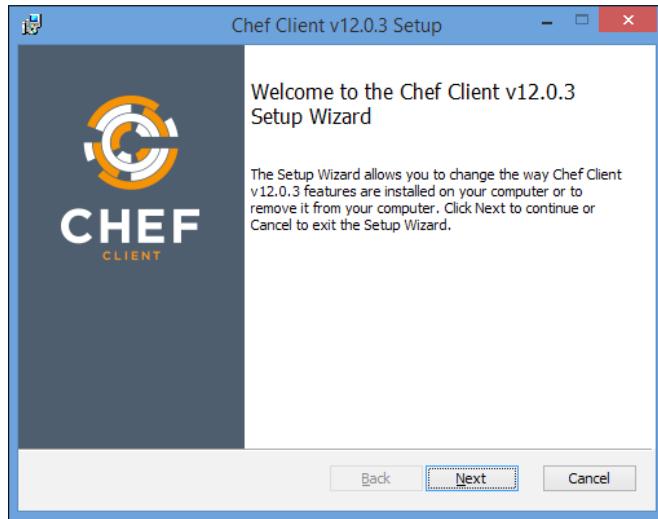
- Knife.rb
- <>Organization\_name>>-validator.chef
- <>Username>>.pem



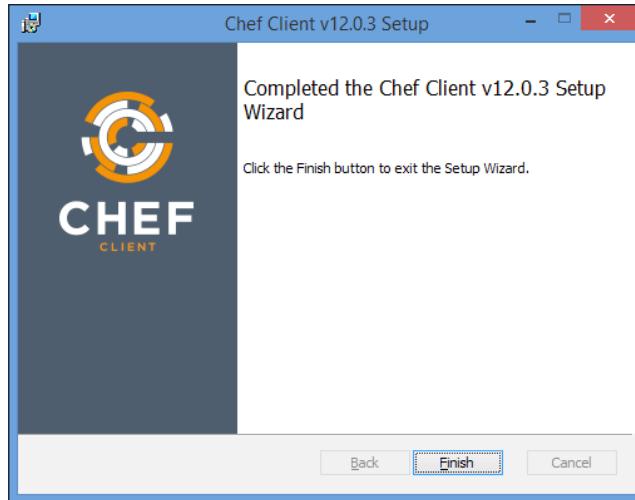
9. Copy the .chef folder with the three authentication files at C:\Users\mitesh\.chef (C:\Users\<>user\_name>>\.chef) for Windows.
10. Download the Chef-client from <https://www.chef.io/download-chef-client/> and perform the following steps:
  - Select the **Windows Operating System** option from the drop-down list on the website
  - Select the operating system version **8**
  - Select the architecture **x86\_64**
  - Select the Chef version **12.0.3-1**
  - Download the Chef-client by clicking on the link



**11. Run `chef-client-12.0.3-1.msi`.**



12. Accept **End User License Agreement** and click on **Next**; select the location to install the Chef-client and click on **Next**. Click on **Install** and wait until the installation completes.



## **Workstation setup - creating a Chef repository**

There are two ways to create a repository on a workstation. One way to do this is by using Git and another is without Git. Now, we will see both of these in detail.

### **Workstation setup using Git on Windows 8**

Opscode provides a skeleton framework for the Chef repository, which can be downloaded by cloning skeleton framework from Github in our local workstation.

The following command makes a new folder named `chef-repo` in the home directory, and after this, we can make the clone of this repository with another directory with the help of this command:

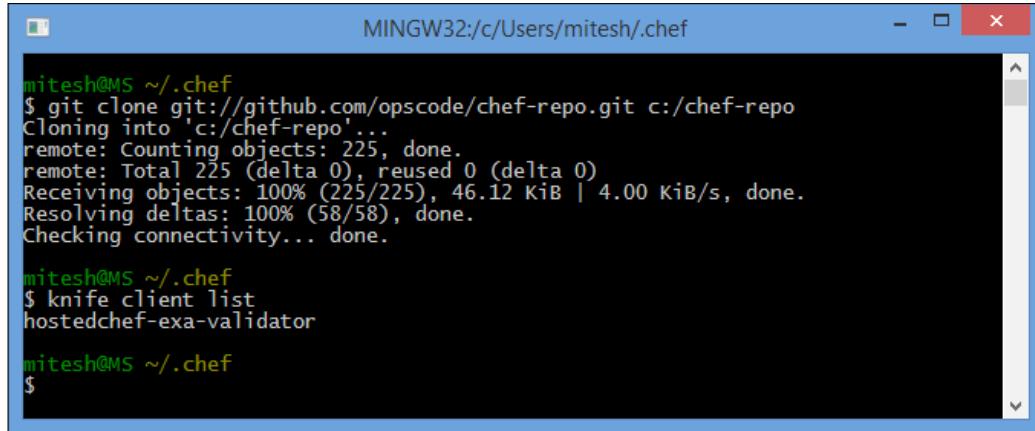
```
git clone git://github.com/opscode/chef-repo.git /path/to/alternate/  
directory
```

Now, change the directory into your `chef-repo` directory. Type the following command:

```
$ cd~/chef-repo
```

Before proceeding further, you need to verify your credentials using this command:

```
$ knife client list
```



The screenshot shows a terminal window with the title 'MINGW32:c/Users/mitesh.chef'. The terminal output is as follows:

```
mitesh@MS ~/.chef
$ git clone git://github.com/opscode/chef-repo.git c:/chef-repo
Cloning into 'c:/chef-repo'...
remote: Counting objects: 225, done.
remote: Total 225 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (225/225), 46.12 KiB | 4.00 KiB/s, done.
Resolving deltas: 100% (58/58), done.
Checking connectivity... done.

mitesh@MS ~/.chef
$ knife client list
hostedchef-exa-validator

mitesh@MS ~/.chef
$
```

## Workstation setup without Git on CentOS

Please note that the `Knife` command will run once the Ruby installation is done. Therefore, Ruby's installation is a must on a Linux machine environment.

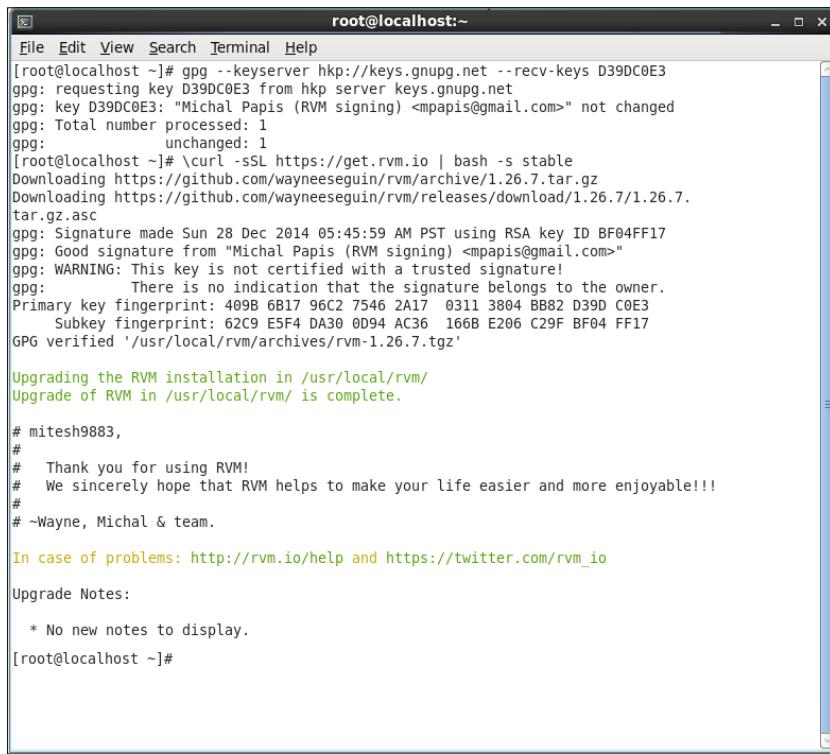
## The Ruby installation and required settings

As we know that Chef runs on Ruby, and therefore, first, we need to install a valid Ruby runtime.

RubyGems (formerly known as Gemcutter) is a package manager for the Ruby programming language. It provides a standard format for packaging, managing, and distributing Ruby programs and libraries. This standard format is called gem. RubyGems provides an easy way to install gems. Its latest stable release is 2.4.2 and it has been publicly available since October 1, 2014. RVM is a command-line tool to manage gems. It provides an easy way to manage a Ruby environment. Following are the steps to install RVM:

1. Install RVM:

```
gpg --keyserver hkp://keys.gnupg.net --recv-keys D39DC0E3
\curl -sSL https://get.rvm.io | bash -s stable
```



The screenshot shows a terminal window with the title 'root@localhost:~'. The terminal displays the command to add the RVM GPG key and download the RVM source code. It includes the GPG verification output, which shows a signature from Michal Papis (RVM signing) and a warning about the key not being certified. The terminal then shows the upgrade notes, which are mostly empty, followed by the command to update RVM.

```
[root@localhost ~]# gpg --keyserver hkp://keys.gnupg.net --recv-keys D39DC0E3
gpg: requesting key D39DC0E3 from hkp server keys.gnupg.net
gpg: key D39DC0E3: "Michal Papis (RVM signing) <mpapis@gmail.com>" not changed
gpg: Total number processed: 1
gpg: unchanged: 1
[root@localhost ~]# \curl -sSL https://get.rvm.io | bash -s stable
Downloading https://github.com/wayneeseguin/rvm/archive/1.26.7.tar.gz
Downloading https://github.com/wayneeseguin/rvm/releases/download/1.26.7/1.26.7.tar.gz.asc
gpg: Signature made Sun 28 Dec 2014 05:45:59 AM PST using RSA key ID BF04FF17
gpg: Good signature from "Michal Papis (RVM signing) <mpapis@gmail.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 4098 6B17 96C2 7546 2A17  0311 3804 BB82 D39D C0E3
Subkey fingerprint: 62C9 E5F4 DA30 0D94 AC36  166B E206 C29F BF04 FF17
GPG verified ' /usr/local/rvm/archives/rvm-1.26.7.tgz'

Upgrading the RVM installation in /usr/local/rvm/
Upgrade of RVM in /usr/local/rvm/ is complete.

# mitesh9883,
#
#   Thank you for using RVM!
#   We sincerely hope that RVM helps to make your life easier and more enjoyable!!!
#
# ~Wayne, Michal & team.

In case of problems: http://rvm.io/help and https://twitter.com/rvm_io

Upgrade Notes:
* No new notes to display.

[root@localhost ~]#
```

2. Update RVM:

```
rvm get stable
```

3. Install the following Ruby dependencies:

```
[root@localhost ~]# rvm requirements
Checking requirements for centos.
Requirements installation successful.
```



```
root@localhost:~#
[root@localhost ~]# rvm list
rvm rubies
* ruby-1.9.3-p551 [ x86_64 ]
  ruby-2.0.0-p598 [ x86_64 ]
=> ruby-2.2.0 [ x86_64 ]

# => - current
# *= - current && default
# * - default

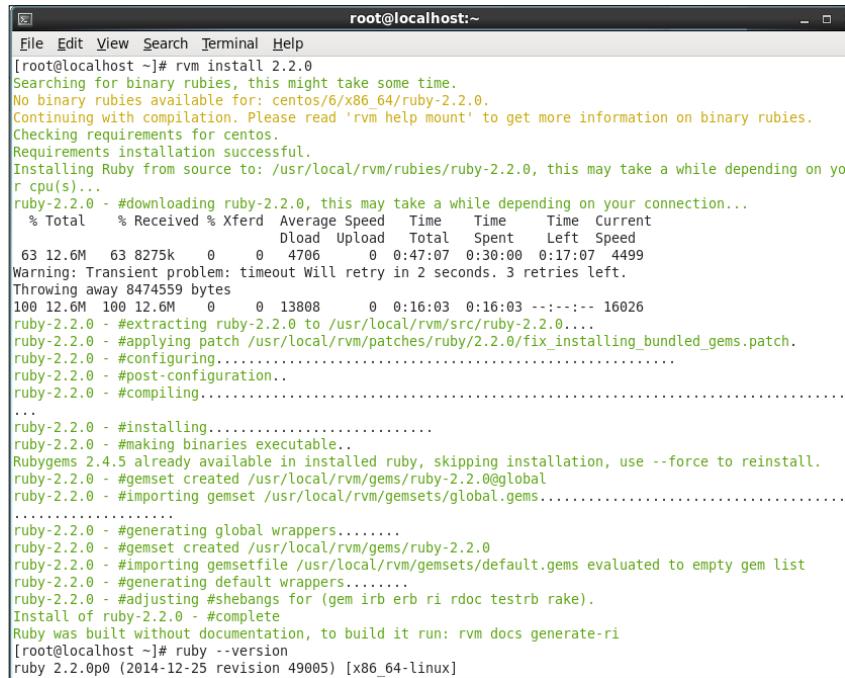
[root@localhost ~]# rvm list gemsets
rvm gemsets

ruby-1.9.3-p551 [ x86_64 ]
ruby-1.9.3-p551@global [ x86_64 ]
ruby-2.0.0-p598 [ x86_64 ]
ruby-2.0.0-p598@global [ x86_64 ]
ruby-2.0.0-p598@rails4 [ x86_64 ]
=> ruby-2.2.0 [ x86_64 ]
ruby-2.2.0@global [ x86_64 ]

[root@localhost ~]# rvm gemset list
gemsets for ruby-2.2.0 (found in /usr/local/rvm/gems/ruby-2.2.0)
=> (default)
  global

[root@localhost ~]#
```

- Once RVM is installed, you can install a recent version of Ruby, for example, Ruby 2.2.0.



```
root@localhost:~#
[root@localhost ~]# rvm install 2.2.0
Searching for binary rubies, this might take some time.
No binary rubies available for: centos/6/x86_64/ruby-2.2.0.
Continuing with compilation. Please read 'rvm help mount' to get more information on binary rubies.
Checking requirements for centos.
Requirements installation successful.
Installing Ruby from source to: /usr/local/rvm/rubies/ruby-2.2.0, this may take a while depending on your cpu(s)...
ruby-2.2.0 - #downloading ruby-2.2.0, this may take a while depending on your connection...
% Total    % Received % Xferd  Average Speed   Time     Time  Current
          Dload  Upload Total Spent   Left Speed
 63 12.6M  63 8275k    0      0  4706      0:47:07  0:30:00  0:17:07 4499
Warning: Transient problem: timeout Will retry in 2 seconds. 3 retries left.
Throwing away 8474559 bytes
100 12.6M 100 12.6M    0      0 13808      0:16:03  0:16:03 ----- 16026
ruby-2.2.0 - #extracting ruby-2.2.0 to /usr/local/rvm/src/ruby-2.2.0...
ruby-2.2.0 - #applying patch /usr/local/rvm/patches/ruby/2.2.0/fix_installing_bundled_gems.patch.
ruby-2.2.0 - #configuring...
ruby-2.2.0 - #post-configuration...
ruby-2.2.0 - #compiling...
...
ruby-2.2.0 - #installing...
ruby-2.2.0 - #making binaries executable...
Rubygems 2.4.5 already available in installed ruby, skipping installation, use --force to reinstall.
ruby-2.2.0 - #gemset created /usr/local/rvm/gems/ruby-2.2.0@global
ruby-2.2.0 - #importing gemset /usr/local/rvm/gemsets/global.gems...
...
ruby-2.2.0 - #generating global wrappers...
ruby-2.2.0 - #gemset created /usr/local/rvm/gems/ruby-2.2.0
ruby-2.2.0 - #importing gemsetfile /usr/local/rvm/gemsets/default.gems evaluated to empty gem list
ruby-2.2.0 - #generating default wrappers...
ruby-2.2.0 - #adjusting #shebangs for (gem irb erb ri rdoc testrb rake).
Install of ruby-2.2.0 - #complete
Ruby was built without documentation, to build it run: rvm docs generate-ri
[root@localhost ~]# ruby --version
ruby 2.2.0p0 (2014-12-25 revision 49005) [x86_64-linux]
```

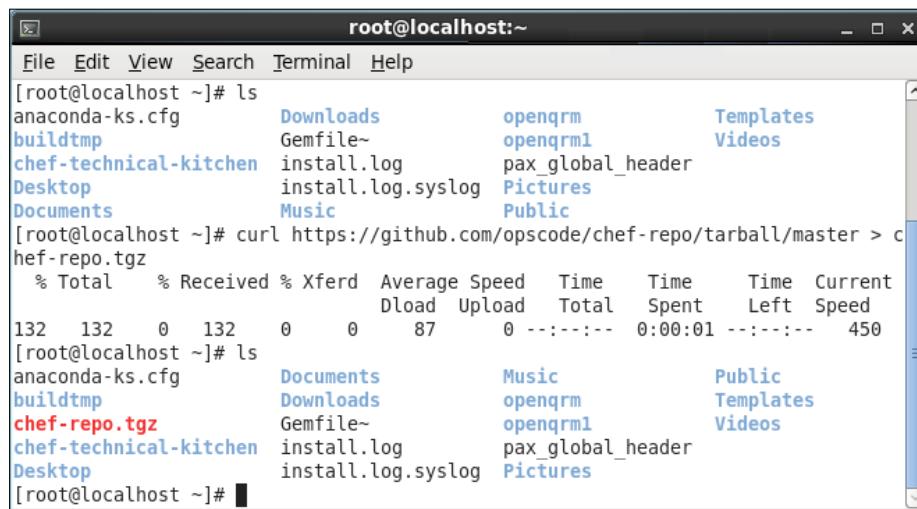
5. Now, Install Chef as gem:

```
$ gem install chef  
$ rbenv rehash
```

We can install the Chef repository without using Git. The following commands are applicable for a Linux machine:

1. Copy the Chef repository from Opscode's GitHub to your workstation with the use of the following commands:

```
$ cd ~  
$ curl https://github.com/opscode/chef-repo/tarball/master > chef-repo.tgz
```



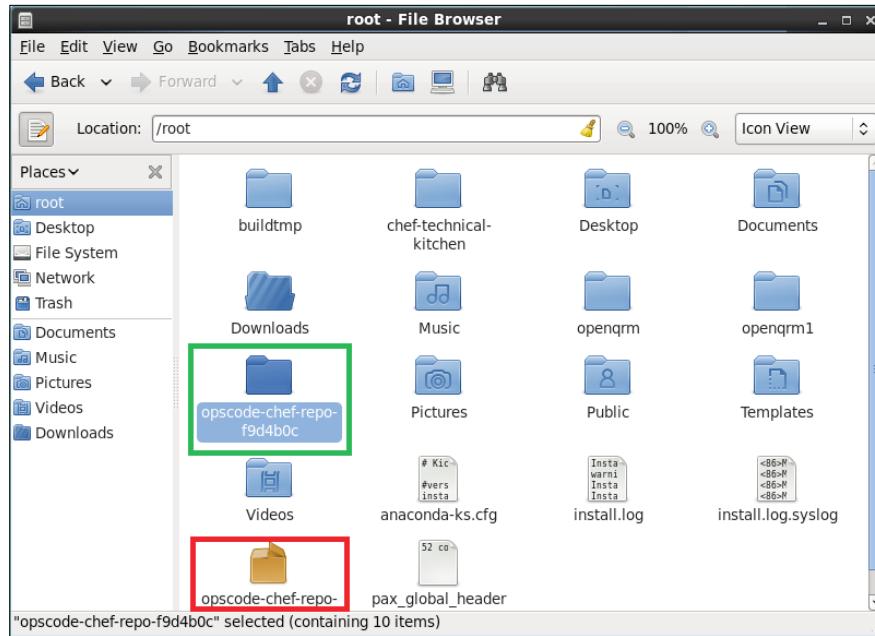
The screenshot shows a terminal window titled "root@localhost:~". The user has run the command "curl https://github.com/opscode/chef-repo/tarball/master > chef-repo.tgz" to download the Chef repository. The terminal also displays the file listing command "ls" and its output, which includes files like anaconda-ks.cfg, buildtmp, chef-technical-kitchen, Desktop, Documents, Downloads, Gemfile~, install.log, install.log.syslog, Music, openqrm, openqrm1, pax\_global\_header, Pictures, Public, Templates, Videos, and .gitignore.

2. You can download Chef repository from <https://github.com/opscode/chef-repo/tarball/master>.



Downloading the example code

You can download the example code files from your account at <http://www.packtpub.com> for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.



3. Extract all files from `chef-repo.tgz`:

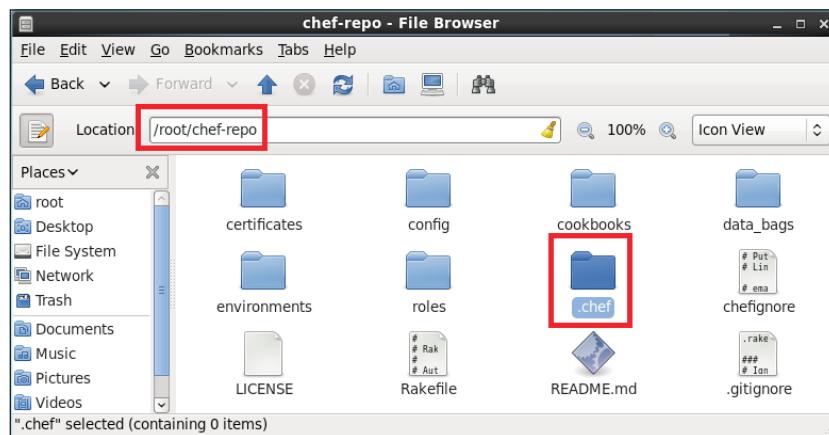
```
$ tar xvf chef-repo.tgz
```

4. Change the name of the directory to `chef-repo`:

```
$ mv -i opscode-chef-repo-a3bec38/ chef-repo
```

5. Now, we need to make the configuration folder and create a directory with the `mkdir` command:

```
$ mkdir -p ~/chef-repo/.chef
```



The required keys and the `knife` configuration must be copied in this directory (the copied files were downloaded from Hosted Chef earlier in this chapter):

```
$ cp USERNAME.pem ~/chef-repo/.chef  
$ cp ORGANIZATION-validator.pem ~/chef-repo/.chef  
$ cp knife.rb ~/chef-repo/.chef
```



However, before repository configuration, you will have to customize repository as follows:

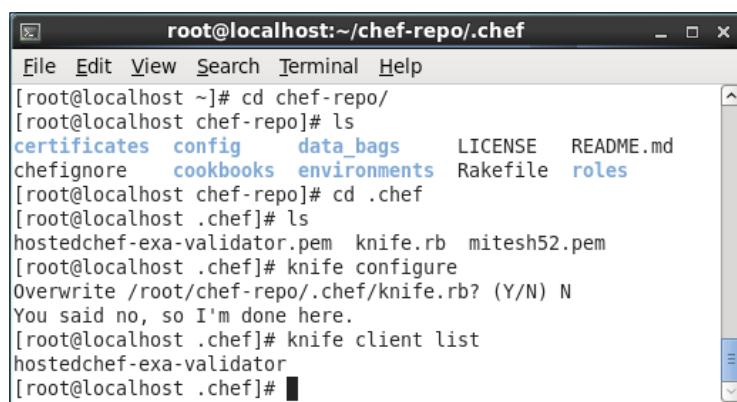
Here, you need to change the `ORGANIZATION` name with your actual organization's name and `USERNAME` with your Hosted Chef username.

Now, you need to create a configuration file with the help of the `knife` command:

```
$ Knife configure
```

To verify that your configuration is working, type the following command:

```
$ Knife client list
```



We have completed our workstation setup on Windows as well as CentOS.

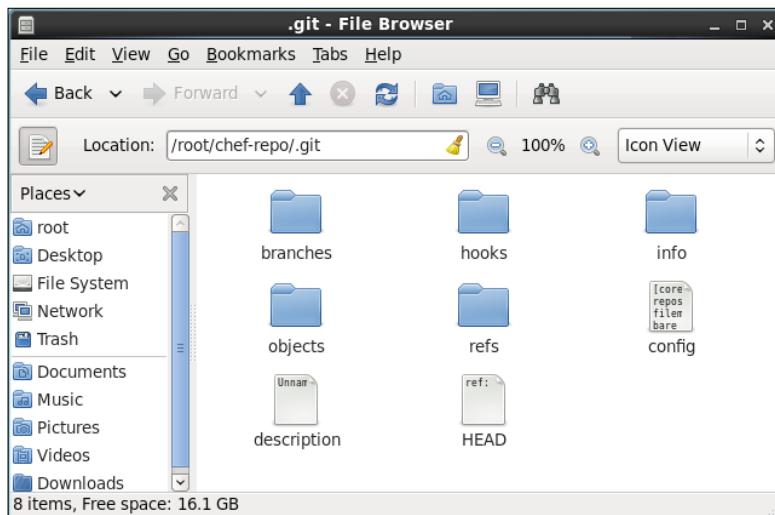
While doing this, you may encounter some errors, such as **Unauthorized 401 error**. We will see the details of all the possible errors and troubleshooting steps in the last section of this chapter.

## Setting up the Chef repository and downloading cookbooks

After creating a local repository, we need to perform the following settings:

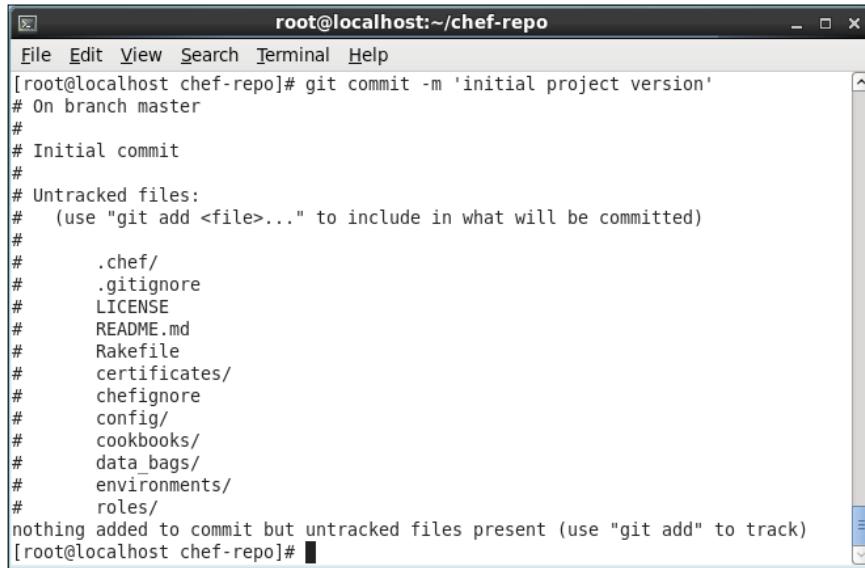
1. There are two approaches to create a Git project: importing the existing project into Git and cloning the existing Git repository by making the Chef-repo as a Git repository. To initialize a repository in an existing directory:

```
$ git init
```



2. Process of initializing Git repository creates a subdirectory called `.git`. It contains repository files, such as a Git repository skeleton. To track or start version controlling the existing files in the `chef-repo` directory, use `add` commands and perform an initial commit:

```
$ git add  
$ git commit -m "Initial commit"
```



The screenshot shows a terminal window with the title 'root@localhost:~/chef-repo'. The window contains the following text:

```
File Edit View Search Terminal Help  
[root@localhost chef-repo]# git commit -m 'initial project version'  
# On branch master  
#  
# Initial commit  
#  
# Untracked files:  
#   (use "git add <file>..." to include in what will be committed)  
#  
#   .chef/  
#   .gitignore  
#   LICENSE  
#   README.md  
#   Rakefile  
#   certificates/  
#   chefignore  
#   config/  
#   cookbooks/  
#   data_bags/  
#   environments/  
#   roles/  
nothing added to commit but untracked files present (use "git add" to track)  
[root@localhost chef-repo]#
```

3. Now, we will download the `apt` and `apache2` cookbooks from the community using `knife`.
4. Linux users are very much familiar with apt services. This `apt` cookbook installs `apt-get update` to ensure that the local `apt` package is up to date. We are also downloading one more cookbook of `apache2`; this will install the `apache2` web server component on a Linux OS.

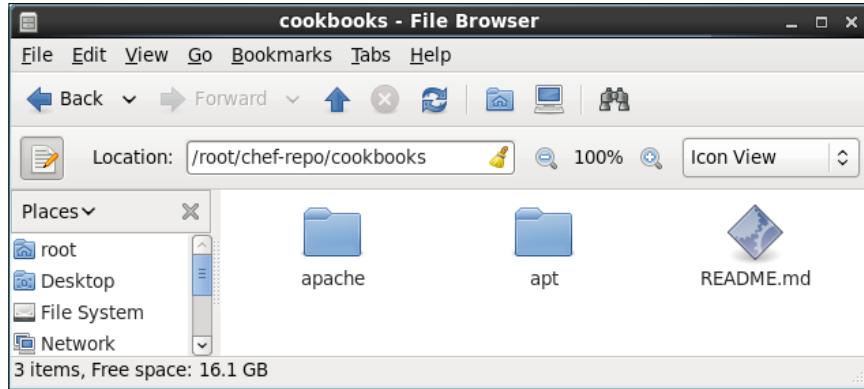
5. Type the following command on a terminal:

```
$ knife cookbook install apt  
$ knife cookbook install apache2
```

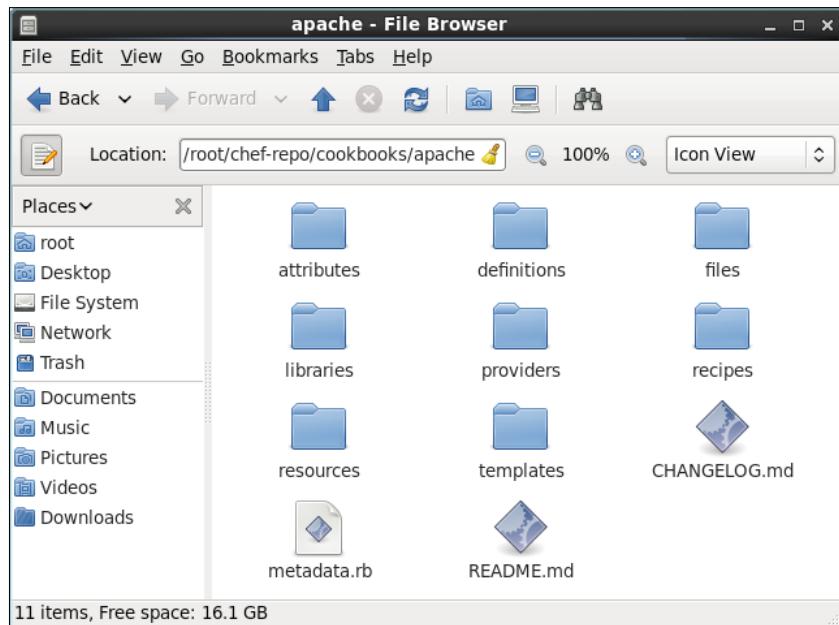
6. To display the list of all your files in your cookbook's directory, type the following command:

```
$ ls cookbooks  
Alternately, browse the cookbook directory
```

7. Now, we can see two folders, apt and apache2.



8. Verify the apache cookbook directory.



## Launching a virtual machine with Vagrant and a workstation setup

Vagrant can manage multiple guest operating systems using a Vagrant file. It provides a multimachine controlling environment. We have already seen that Vagrant is a tool to manage virtual machines via a command-line interface. The Vagrant up command is used to create an environment based on a standard template called base boxes. A box can be utilized on any platform supported by Vagrant. There is a public catalog of Vagrant boxes available at <https://atlas.hashicorp.com/boxes/search>. Another location to download boxes is <http://www.vagrantbox.es/> as shown in the following screenshot:

Name	Provider	URL	Size (MB)
Debian 7.3.0 64-bit Puppet 3.4.1 (Vagrant 1.4.0)	VirtualBox 4.3.6	<a href="https://dl.dropboxusercontent.com/u/29173892/vagrant-boxes/debian7.3.0-vbox4.3.6-puppet3.4.1.box">Copy</a> https://dl.dropboxusercontent.com/u/29173892/vagrant-boxes/debian7.3.0-vbox4.3.6-puppet3.4.1.box	682
OpenBSD 5.5 64-bit + Chef 11.16.0 + Puppet 3.4.2	VirtualBox /openbsd55.box	<a href="https://github.com/jose-lpa/veewee-openbsd/releases/download/v0.5.5/openbsd55.box">Copy</a> https://github.com/jose-lpa/veewee-openbsd/releases/download/v0.5.5/openbsd55.box	315
OpenBSD 5.4 64-bit + Chef 11.8.2 (150GB HDD)	VirtualBox	<a href="http://vagrant.inagile.org/vagrant-obsd54-amd64.box">Copy</a> http://vagrant.inagile.org/vagrant-obsd54-amd64.box	1800
OpenBSD 5.3 64-bit (Vagrant 1.2)	VirtualBox /openbsd53_amd64_vagrant12.box	<a href="https://dl.dropboxusercontent.com/u/12089300/VirtualBox/openbsd53_amd64_vagrant12.box">Copy</a> https://dl.dropboxusercontent.com/u/12089300/VirtualBox/openbsd53_amd64_vagrant12.box	296
OpenBSD 5.3 64-bit	VirtualBox	<a href="https://dl.dropboxusercontent.com/u/12089300/VirtualBox/openbsd53_amd64.box">Copy</a> https://dl.dropboxusercontent.com/u/12089300/VirtualBox/openbsd53_amd64.box	303
Aegir-up Aegir (Debian Squeeze 6.0.4 64-bit)	VirtualBox	<a href="http://ergonlogic.com/files/boxes/aegir-current.box">Copy</a> http://ergonlogic.com/files/boxes/aegir-current.box	297
Aegir-up Debian (Debian Squeeze 6.0.4 64-bit)	VirtualBox	<a href="http://ergonlogic.com/files/boxes/debian-current.box">Copy</a> http://ergonlogic.com/files/boxes/debian-current.box	283
Aegir-up LAMP (Debian Squeeze 6.0.4 64-bit)	VirtualBox	<a href="http://ergonlogic.com/files/boxes/debian-LAMP-current.box">Copy</a> http://ergonlogic.com/files/boxes/debian-LAMP-current.box	388
AppScale 1.12.0 (Ubuntu Precise 12.04 64-bit)	VirtualBox	<a href="http://download.appspot.com/download/AppScale%201.12.0%20VirtualBox%20Image">Copy</a> http://download.appspot.com/download/AppScale%201.12.0%20VirtualBox%20Image	1900
Arch Linux 64 (2014-06-20)	VirtualBox	<a href="http://www.eduardoheredia.com.br/rep/vagrant/archlinux64.box">Copy</a> http://www.eduardoheredia.com.br/rep/vagrant/archlinux64.box	292
Arch Linux 64 (2013-08-01)	VirtualBox	<a href="https://dl.dropboxusercontent.com/u/31112574/arch64-20130801.box">Copy</a> https://dl.dropboxusercontent.com/u/31112574/arch64-20130801.box	578
Arch Linux x86_64 (2013-08)	VirtualBox	<a href="https://googledrive.com/host/0B_BLFE4aPn5zUVpyaHdLanVnMTg/vagrant-archlinux-2013-8.box">Copy</a> https://googledrive.com/host/0B_BLFE4aPn5zUVpyaHdLanVnMTg/vagrant-archlinux-2013-8.box	394

The following is the procedure to manage multiple guest OSs with Vagrant:

1. First open a **Terminal** window.
2. Make a directory for your work and move this to the working directory:

```
mkdir -p ~/myprojects/chef/startup_chef  
$ cd ~/myprojects/chef/startup_chef
```

3. Initialize a Git repository:

```
$ git init
```

4. Initialize Vagrant:

```
$ vagrant init
```

5. Add the initial Vagrant file to the Git repository:

```
git add Vagrantfile  
$ git commit -m "initializing Vagrantfile"
```

Vagrant does not create a virtual machine instance completely from scratch, instead it provides a base image for VM and builds on that; this base image is called a box. Therefore, we need to add the name of that VM box in config.vm.box; in our case, we are using an Apache web server, so we can assign the Apache Web name config.vm.box.

6. Open the **Vagrant** file in a text editor mode and modify the following contents of the file:

```
Vagrant::Config.run do |config|  
  config.vm.box = "Apacheweb"  
  config.vm.box_url = "https://opscode-vm-bento.s3.amazonaws.com/  
vagrant/opscode_ubuntu-12.04_chef-10.18.2.box"  
end
```

7. Launch the virtual machine:

```
$ vagrant up  
The preceding command will launch the virtual machine.
```

8. Now, add the new updated Vagrantfile to the Git repository:

```
git add Vagrantfile  
$ git commit -m "Initializing Vagrantfile with one Virtual  
Machine"
```

9. Finally, add a file named .gitignore, so that Git will automatically ignore the Vagrant directory that is generated while launching a virtual machine:

```
$ echo'.vagrant'>>.gitignore
```

Now, once a node is ready, install the Chef-client. Configure a workstation with the following commands:

1. Make a .chef directory using this command:

```
$ mkdir .chef
```

2. Downloaded files should be moved to the .chef directory with the following command:

```
$ mv ~/Downloads/knife.rb .chef/  
$ mv ~/Downloads/USER.pem .chef/  
$ mv ~/Downloads/ORG-validator.pem .chef/
```

3. Add this `.chef` directory to your `.gitignore` file; you should not store these sensitive files in your **Git repository**:

```
$ echo '.chef' >> .gitignore
```

4. Stage this `.gitignore` file:

```
git add .gitignore
```

5. Now, you need to commit the change:

```
git commit -m "Add .chef to the .gitignore"
```

6. Finally, push the change to `github.com`:

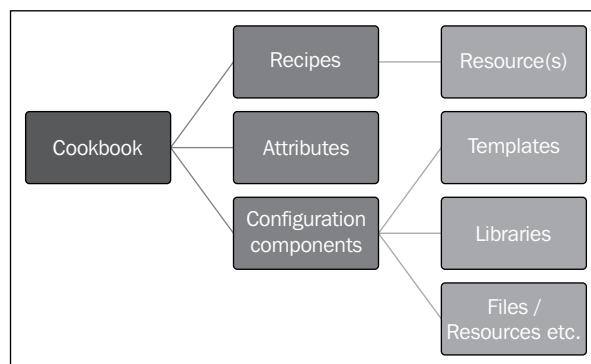
```
git push origin master
```

## Creating and uploading a simple cookbook

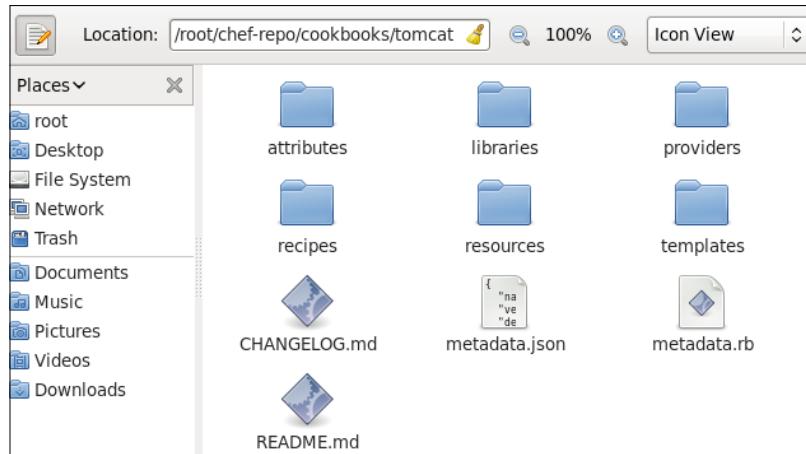
We can also write our own cookbooks. This section will give us a better understanding of cookbooks.

A cookbook works as a container that describes the configuration and policy definition of an application or software in Chef. It has dependencies as well. For example, the development repository for Opscode's Cookbook Tomcat is available at <https://github.com/opscode-cookbooks/tomcat>. It installs and configures Tomcat on various platforms, such as Ubuntu, Debian, Red Hat 6+, CentOS 6+, and so on. Java and OpenSSL are the dependencies for a Tomcat cookbook.

The following is the high-level structure of a cookbook:



The following screenshot shows the directory structure of a Tomcat cookbook:



The following is an expanded list of cookbook structures that shows a Tomcat cookbook. We have already covered the components of a cookbook in *Chapter 2, Different Components of Chef Anatomy*.

Attributes represent the characteristics of a node and a HashMap-like mechanism, as shown in the following screenshot:

```

FOLDERS
└── tomcat
    ├── attributes
    │   ├── default.rb
    ├── libraries
    │   └── chef_tomcat_cookbook.rb
    ├── providers
    │   └── instance.rb
    ├── recipes
    │   ├── default.rb
    │   └── users.rb
    ├── resources
    │   └── instance.rb
    ├── templates
    │   └── default
    │       ├── default_tomcat6.erb
    │       ├── logging.properties.erb
    │       ├── manifestxml.erb
    │       ├── server.xml.erb
    │       ├── setenv.sh.erb
    │       ├── sysconfig_tomcat6.erb
    │       └── tomcat-users.xml.erb
    └── CHANGELOG.md
        README.md
        metadata.json
        metadata.rb
  
```

```

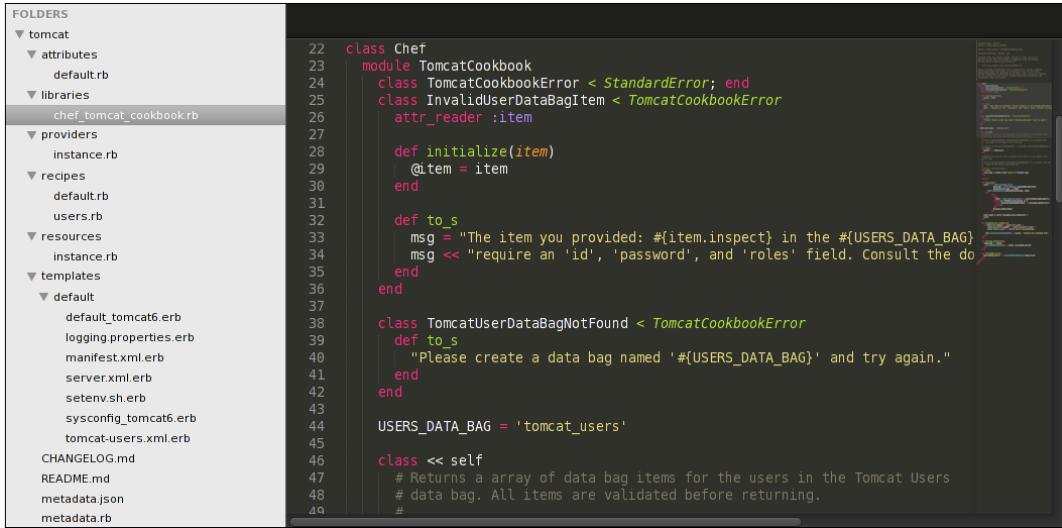
instance.rb

16 # See the License for the specific language governing permissions and
17 # limitations under the License.
18
19 default['tomcat']['base_version'] = 6
20 default['tomcat']['port'] = 8080
21 default['tomcat']['proxy_port'] = nil
22 default['tomcat']['ssl_port'] = 8443
23 default['tomcat']['ssl_proxy_port'] = nil
24 default['tomcat']['ajp_port'] = 8009
25 default['tomcat']['shutdown_port'] = 8005
26 default['tomcat']['catalina_options'] = ''
27 default['tomcat']['java_options'] = '-Xmx128M -Djava.awt.headless=true'
28 default['tomcat']['use_security_manager'] = false
29 default['tomcat']['authbind'] = 'no'
30 default['tomcat']['deploy_manager_apps'] = true
31 default['tomcat']['max_threads'] = nil
32 default['tomcat']['ssl_max_threads'] = 150
33 default['tomcat']['ssl_cert_file'] = nil
34 default['tomcat']['ssl_key_file'] = nil
35 default['tomcat']['ssl_chain_files'] = []
36 default['tomcat']['keystore_file'] = 'keystore.jks'
37 default['tomcat']['keystore_type'] = 'jks'
38 # The keystore and truststore passwords will be generated by the
39 # openssl cookbook's secure_password method in the recipe if they are
40 # not otherwise set. Do not hardcode passwords in the cookbook.
41 # default["tomcat"]["keystore_password"] = nil
42 # default["tomcat"]["truststore_password"] = nil
43 default['tomcat']['truststore_file'] = nil
  
```

## *Workstation Setup and Cookbook Creation*

---

A library provides a way to extend the Chef functionality:



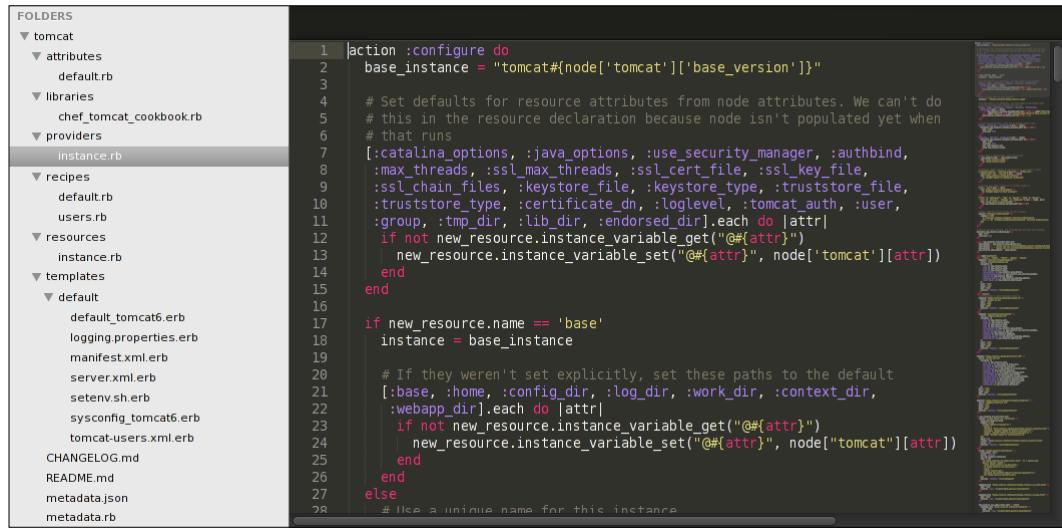
The screenshot shows a terminal window with two panes. The left pane displays the directory structure of the cookbook:

```
FOLDERS
└── tomcat
    ├── attributes
    │   └── default.rb
    ├── libraries
    │   └── chef_tomcat_cookbook.rb
    ├── providers
    │   └── instance.rb
    ├── recipes
    │   └── default.rb
    ├── resources
    │   └── instance.rb
    ├── templates
    │   └── default
    │       ├── default_tomcat6.erb
    │       ├── logging.properties.erb
    │       ├── manifest.xml.erb
    │       ├── server.xml.erb
    │       ├── setenv.sh.erb
    │       ├── sysconfig_tomcat6.erb
    │       └── tomcat-users.xml.erb
    └── CHANGELOG.md
    └── README.md
    └── metadata.json
    └── metadata.rb
```

The right pane shows the content of the `chef_tomcat_cookbook.rb` file:

```
22 class Chef
23   module TomcatCookbook
24     class TomcatCookbookError < StandardError; end
25     class InvalidUserDataBagItem < TomcatCookbookError
26       attr_reader :item
27
28       def initialize(item)
29         @item = item
30       end
31
32       def to_s
33         msg = "The item you provided: #{item.inspect} in the #{USERS_DATA_BAG}"
34         msg << "require an 'id', 'password', and 'roles' field. Consult the do"
35       end
36     end
37
38     class TomcatUserDataBagNotFound < TomcatCookbookError
39       def to_s
40         "Please create a data bag named '#{USERS_DATA_BAG}' and try again."
41       end
42     end
43
44     USERS_DATA_BAG = 'tomcat_users'
45
46     class << self
47       # Returns a array of data bag items for the users in the Tomcat Users
48       # data bag. All items are validated before returning.
49     end
```

A provider describes the steps that are required to keep the resources in the desired state:



The screenshot shows a terminal window with two panes. The left pane displays the directory structure of the cookbook:

```
FOLDERS
└── tomcat
    ├── attributes
    │   └── default.rb
    ├── libraries
    │   └── chef_tomcat_cookbook.rb
    ├── providers
    │   └── instance.rb
    ├── recipes
    │   └── default.rb
    ├── resources
    │   └── instance.rb
    ├── templates
    │   └── default
    │       ├── default_tomcat6.erb
    │       ├── logging.properties.erb
    │       ├── manifest.xml.erb
    │       ├── server.xml.erb
    │       ├── setenv.sh.erb
    │       ├── sysconfig_tomcat6.erb
    │       └── tomcat-users.xml.erb
    └── CHANGELOG.md
    └── README.md
    └── metadata.json
    └── metadata.rb
```

The right pane shows the content of the `instance.rb` file:

```
1 action :configure do
2   base_instance = "tomcat#{node['tomcat']['base_version']}"
3
4   # Set defaults for resource attributes from node attributes. We can't do
5   # this in the resource declaration because node isn't populated yet when
6   # that runs
7   [:catalina_options, :java_options, :use_security_manager, :authbind,
8   :max_threads, :ssl_max_threads, :ssl_cert_file, :ssl_key_file,
9   :ssl_chain_files, :keystore_file, :keystore_type, :truststore_file,
10  :truststore_type, :certificate_dn, :loglevel, :tomcat_auth, :user,
11  :group, :tmp_dir, :lib_dir, :endorsed_dir].each do |attr|
12    if not new_resource.instance_variable_get("@#{attr}")
13      new_resource.instance_variable_set("@#{attr}", node['tomcat'][attr])
14    end
15  end
16
17  if new_resource.name == 'base'
18    instance = base_instance
19
20    # If they weren't set explicitly, set these paths to the default
21    [:base, :home, :config_dir, :log_dir, :work_dir, :context_dir,
22     :webapp_dir].each do |attr|
23      if not new_resource.instance_variable_get("@#{attr}")
24        new_resource.instance_variable_set("@#{attr}", node["tomcat"][attr])
25      end
26    end
27  else
28    # Use a unique name for this instance
29  end
```

A recipe is the most basic and fundamental part of the Chef environment that is stored in a cookbook:

```

FOLDERS
└ tomcat
  └ attributes
    default.rb
  └ libraries
    chef_tomcat_cookbook.rb
  └ providers
    instance.rb
  └ recipes
    default.rb
    users.rb
  └ resources
    instance.rb
  └ templates
    default
      default_tomcat6.erb
      logging.properties.erb
      manifest.xml.erb
      server.xml.erb
      setenv.sh.erb
      sysconfig_tomcat6.erb
      tomcat-users.xml.erb
    CHANGELOG.md
    README.md
    metadata.json
    metadata.rb

instance.rb
19
20  # required for the secure_password method from the openssl cookbook
21  ::Chef::Recipe.send(:include, Opscode::OpenSSL::Password)
22
23
24  tomcat_pkgs = value_for_platform(
25    ['smartos'] => {
26      'default' => ['apache-tomcat'],
27    },
28    'default' => ["tomcat#{node['tomcat']['base_version']}"]
29  )
30  if node['tomcat']['deploy_manager_apps']
31    tomcat_pkgs << value_for_platform(
32      '%{ debian ubuntu }' => {
33        'default' => "tomcat#{node['tomcat']['base_version']}-admin",
34      },
35      '%{ centos redhat fedora amazon scientific oracle }' => {
36        'default' => "tomcat#{node['tomcat']['base_version']}-admin-webapps",
37      }
38    )
39  end
40
41  tomcat_pkgs.compact!
42
43  tomcat_pkgs.each do |pkg|
44    package pkg do
45      action :install
46      version node['tomcat'][node['base_version']] to_s if platform.family? 'smartos'

```

Recipes are deployed on the nodes and they are used to configure the node:

```

FOLDERS
└ tomcat
  └ attributes
    default.rb
  └ libraries
    chef_tomcat_cookbook.rb
  └ providers
    instance.rb
  └ recipes
    default.rb
    users.rb
  └ resources
    instance.rb
  └ templates
    default
      default_tomcat6.erb
      logging.properties.erb
      manifest.xml.erb
      server.xml.erb
      setenv.sh.erb
      sysconfig_tomcat6.erb
      tomcat-users.xml.erb
    CHANGELOG.md
    README.md
    metadata.json
    metadata.rb

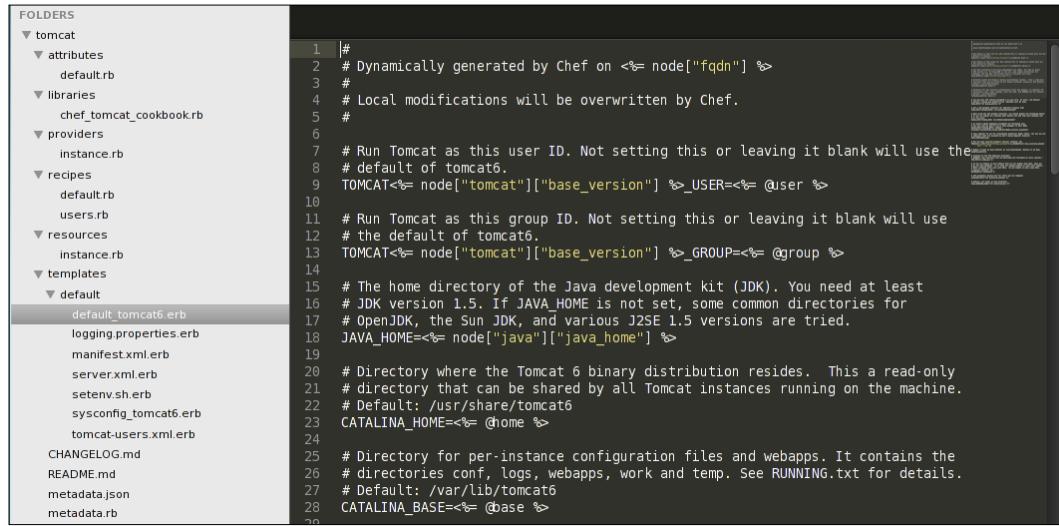
users.rb
22  template "#{node['tomcat']['config_dir']}/tomcat-users.xml" do
23    source 'tomcat-users.xml.erb'
24    owner 'root'
25    group 'root'
26    mode '0644'
27    variables(
28      :users => TomcatCookbook.users,
29      :roles => TomcatCookbook.roles,
30    )
31    notifies :restart, 'service[tomcat]'
32  end
33

```

## Workstation Setup and Cookbook Creation

---

A template is written in an ERB template language and it is used for multifaceted configuration scenarios:



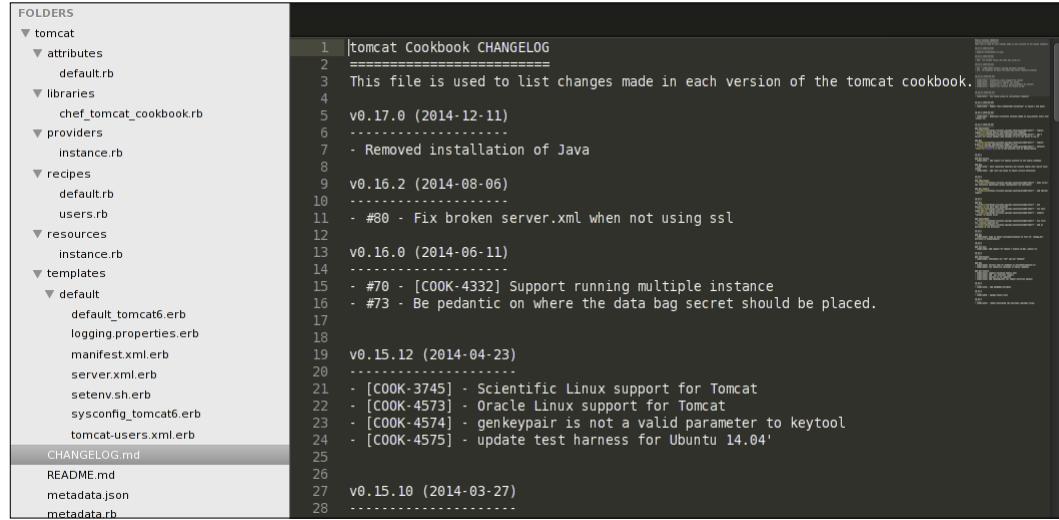
The screenshot shows a terminal window with two panes. The left pane displays the directory structure of the 'tomcat' cookbook, which includes folders for attributes, libraries, providers, recipes, resources, templates, and a 'default' folder containing various erb files like 'default\_tomcat6.erb', 'logging.properties.erb', and 'CHANGELOG.md'. The right pane shows the content of the 'default\_tomcat6.erb' template file. The code is an ERB template with numerous comments explaining the logic for setting up Tomcat. It handles variables like 'fqdn', 'base\_version', 'user', 'group', 'JAVA\_HOME', 'CATALINA\_HOME', and 'CATALINA\_BASE'.

```
FOLDERS
└── tomcat
    ├── attributes
    │   └── default.rb
    ├── libraries
    │   └── chef_tomcat_cookbook.rb
    ├── providers
    │   └── instance.rb
    ├── recipes
    │   ├── default.rb
    │   ├── users.rb
    ├── resources
    │   └── instance.rb
    ├── templates
    │   └── default
    │       ├── default_tomcat6.erb
    │       ├── logging.properties.erb
    │       ├── manifest.xml.erb
    │       ├── server.xml.erb
    │       ├── setenv.sh.erb
    │       └── sysconfig_tomcat6.erb
    └── tomcat-users.xml.erb

CHANGELOG.md
README.md
metadata.json
metadata.rb

# 
# Dynamically generated by Chef on <%= node["fqdn"] %>
#
# Local modifications will be overwritten by Chef.
#
#
# Run Tomcat as this user ID. Not setting this or leaving it blank will use the
# default of tomcat6.
TOMCAT=<%= node["tomcat"]["base_version"] %>_USER=<%= @user %>
#
# Run Tomcat as this group ID. Not setting this or leaving it blank will use
# the default of tomcat6.
TOMCAT=<%= node["tomcat"]["base_version"] %>_GROUP=<%= @group %>
#
# The home directory of the Java development kit (JDK). You need at least
# JDK version 1.5. If JAVA_HOME is not set, some common directories for
# OpenJDK, the Sun JDK, and various J2SE 1.5 versions are tried.
JAVA_HOME=<%= node["java"]["java_home"] %>
#
# Directory where the Tomcat 6 binary distribution resides. This is a read-only
# directory that can be shared by all Tomcat instances running on the machine.
# Default: /usr/share/tomcat6
CATALINA_HOME=<%= @home %>
#
# Directory for per-instance configuration files and webapps. It contains the
# directories conf, logs, webapps, work and temp. See RUNNING.txt for details.
# Default: /var/lib/tomcat6
CATALINA_BASE=<%= @base %>
```

`CHANGELOG.md` is used to keep the information of the changes made in the version of a cookbook:



The screenshot shows a terminal window displaying the `CHANGELOG.md` file for the `tomcat` cookbook. The file lists several versions of the cookbook with their release dates and corresponding changes. The changes include the removal of Java installation, fixes for SSL support, support for Scientific Linux, Oracle Linux, and Ubuntu 14.04, and updates to the test harness.

```
FOLDERS
└── tomcat
    ├── attributes
    │   └── default.rb
    ├── libraries
    │   └── chef_tomcat_cookbook.rb
    ├── providers
    │   └── instance.rb
    ├── recipes
    │   ├── default.rb
    │   ├── users.rb
    ├── resources
    │   └── instance.rb
    ├── templates
    │   └── default
    │       ├── default_tomcat6.erb
    │       ├── logging.properties.erb
    │       ├── manifest.xml.erb
    │       ├── server.xml.erb
    │       ├── setenv.sh.erb
    │       └── sysconfig_tomcat6.erb
    └── tomcat-users.xml.erb

CHANGELOG.md
README.md
metadata.json
metadata.rb

|tomcat Cookbook CHANGELOG
=====
This file is used to list changes made in each version of the tomcat cookbook.

v0.17.0 (2014-12-11)
-----
- Removed installation of Java

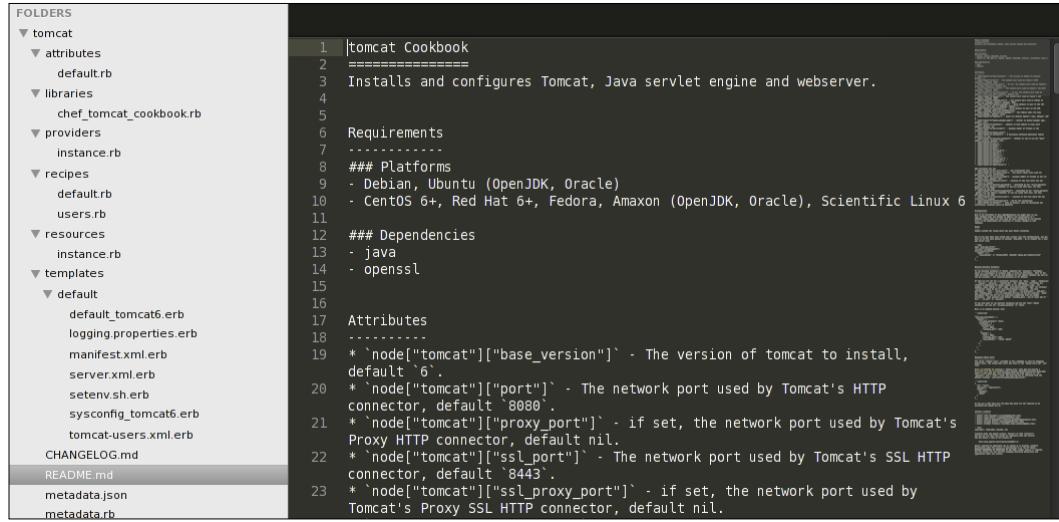
v0.16.2 (2014-08-06)
-----
- #80 - Fix broken server.xml when not using ssl

v0.16.0 (2014-06-11)
-----
- #70 - [COOK-4332] Support running multiple instance
- #73 - Be pedantic on where the data bag secret should be placed.

v0.15.12 (2014-04-23)
-----
- [COOK-3745] - Scientific Linux support for Tomcat
- [COOK-4573] - Oracle Linux support for Tomcat
- [COOK-4574] - genkeypair is not a valid parameter to keytool
- [COOK-4575] - update test harness for Ubuntu 14.04'

v0.15.10 (2014-03-27)
-----
```

`README.md` provides information of the cookbook:



```

FOLDERS
└── tomcat
    ├── attributes
    │   └── default.rb
    ├── libraries
    │   └── chef_tomcat_cookbook.rb
    ├── providers
    │   └── instance.rb
    ├── recipes
    │   ├── default.rb
    │   ├── users.rb
    ├── resources
    │   └── instance.rb
    ├── templates
    │   └── default
    │       ├── default_tomcats.erb
    │       ├── logging.properties.erb
    │       ├── manifest.xml.erb
    │       ├── server.xml.erb
    │       ├── setenv.sh.erb
    │       ├── sysconfig_tomcat6.erb
    │       └── tomcat-users.xml.erb
    ├── CHANGELOG.md
    └── README.md
    └── metadata.json
    └── metadata.rb

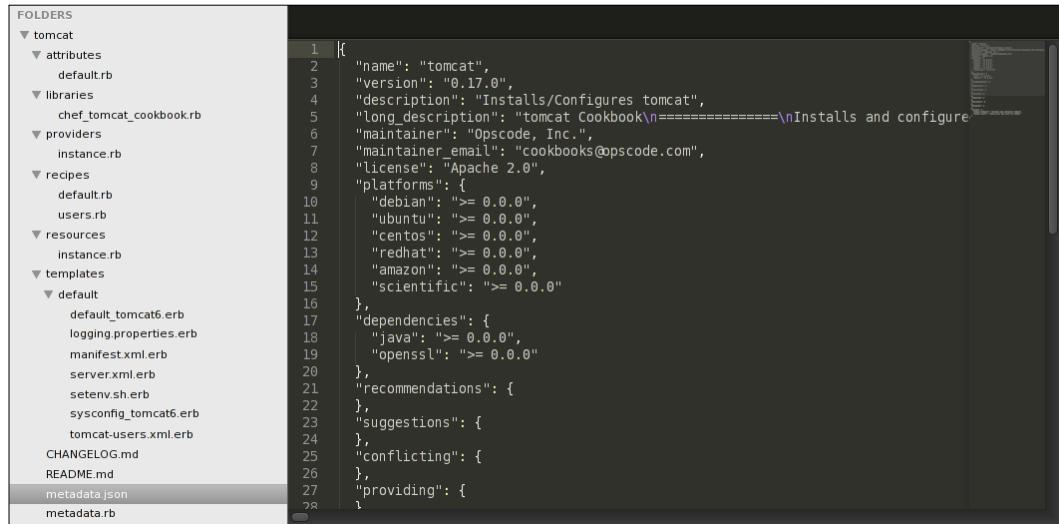
```

```

1 |tomcat Cookbook
2 |=====
3 |Installs and configures Tomcat, Java servlet engine and webserver.
4 |
5 |Requirements
6 |-----
7 |### Platforms
8 |- Debian, Ubuntu (OpenJDK, Oracle)
9 |- CentOS 6+, Red Hat 6+, Fedora, Amazon (OpenJDK, Oracle), Scientific Linux 6
10 |
11 |### Dependencies
12 |- java
13 |- openssl
14 |
15 |
16 |Attributes
17 |-----
18 |* `node["tomcat"]["base_version"]` - The version of tomcat to install,
19 |  default '6'.
20 |* `node["tomcat"]["port"]` - The network port used by Tomcat's HTTP
21 |  connector, default `8080`.
22 |* `node["tomcat"]["proxy_port"]` - if set, the network port used by Tomcat's
23 |  Proxy HTTP connector, default nil.
24 |* `node["tomcat"]["ssl_port"]` - The network port used by Tomcat's SSL HTTP
25 |  connector, default `8443`.
26 |* `node["tomcat"]["ssl_proxy_port"]` - if set, the network port used by
27 |  Tomcat's Proxy SSL HTTP connector, default nil.

```

Metadata gives information of licenses, platforms, constraints, dependencies, and so on:



```

FOLDERS
└── tomcat
    ├── attributes
    │   └── default.rb
    ├── libraries
    │   └── chef_tomcat_cookbook.rb
    ├── providers
    │   └── instance.rb
    ├── recipes
    │   ├── default.rb
    │   ├── users.rb
    ├── resources
    │   └── instance.rb
    ├── templates
    │   └── default
    │       ├── default_tomcat6.erb
    │       ├── logging.properties.erb
    │       ├── manifest.xml.erb
    │       ├── server.xml.erb
    │       ├── setenv.sh.erb
    │       ├── sysconfig_tomcat6.erb
    │       └── tomcat-users.xml.erb
    ├── CHANGELOG.md
    └── README.md
    └── metadata.json
    └── metadata.rb

```

```

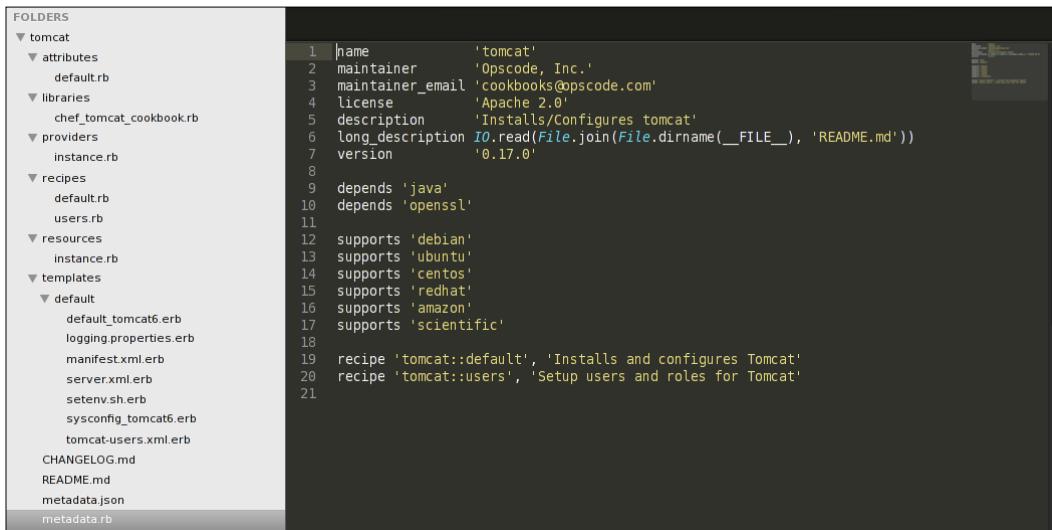
1 |{
2 |  "name": "tomcat",
3 |  "version": "0.17.0",
4 |  "description": "Installs/Configures tomcat",
5 |  "long_description": "tomcat Cookbook\n=====\\nInstalls and configures Tomcat, Java servlet engine and webserver.",
6 |  "maintainer": "Opscode, Inc.",
7 |  "maintainer_email": "cookbooks@opscode.com",
8 |  "license": "Apache 2.0",
9 |  "platforms": {
10 |    "debian": ">= 0.0.0",
11 |    "ubuntu": ">= 0.0.0",
12 |    "centos": ">= 0.0.0",
13 |    "redhat": ">= 0.0.0",
14 |    "amazon": ">= 0.0.0",
15 |    "scientific": ">= 0.0.0"
16 |  },
17 |  "dependencies": {
18 |    "java": ">= 0.0.0",
19 |    "openssl": ">= 0.0.0"
20 |  },
21 |  "recommendations": {
22 |  },
23 |  "suggestions": {
24 |  },
25 |  "conflicting": {
26 |  },
27 |  "providing": {
28 |

```

## *Workstation Setup and Cookbook Creation*

---

The `metadata.rb` file provides a location to store and edit data that is compiled by the Chef server and stored as JSON data.



The screenshot shows a terminal window with two panes. The left pane displays the directory structure of a cookbook named 'tomcat'. The right pane shows the contents of the 'metadata.rb' file, which defines the cookbook's metadata such as name, maintainer, license, description, long\_description, and version.

```
FOLDERS
└ tomcat
  └ attributes
    default.rb
  └ libraries
    chef_tomcat_cookbook.rb
  └ providers
    instance.rb
  └ recipes
    default.rb
    users.rb
  └ resources
    instance.rb
  └ templates
    default
      default_tomcat6.erb
      logging.properties.erb
      manifest.xml.erb
      server.xml.erb
      setenv.sh.erb
      sysconfig_tomcat6.erb
      tomcat-users.xml.erb
    CHANGELOG.md
    README.md
    metadata.json
    metadata.rb

1 name          'tomcat'
2 maintainer    'Opscode, Inc.'
3 maintainer_email 'cookbooks@opscode.com'
4 license        'Apache 2.0'
5 description    'Installs/Configures tomcat'
6 long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))
7 version        '0.17.0'
8
9 depends 'java'
10 depends 'openssl'
11
12 supports 'debian'
13 supports 'ubuntu'
14 supports 'centos'
15 supports 'redhat'
16 supports 'amazon'
17 supports 'scientific'
18
19 recipe 'tomcat::default', 'Installs and configures Tomcat'
20 recipe 'tomcat::users', 'Setup users and roles for Tomcat'
21
```

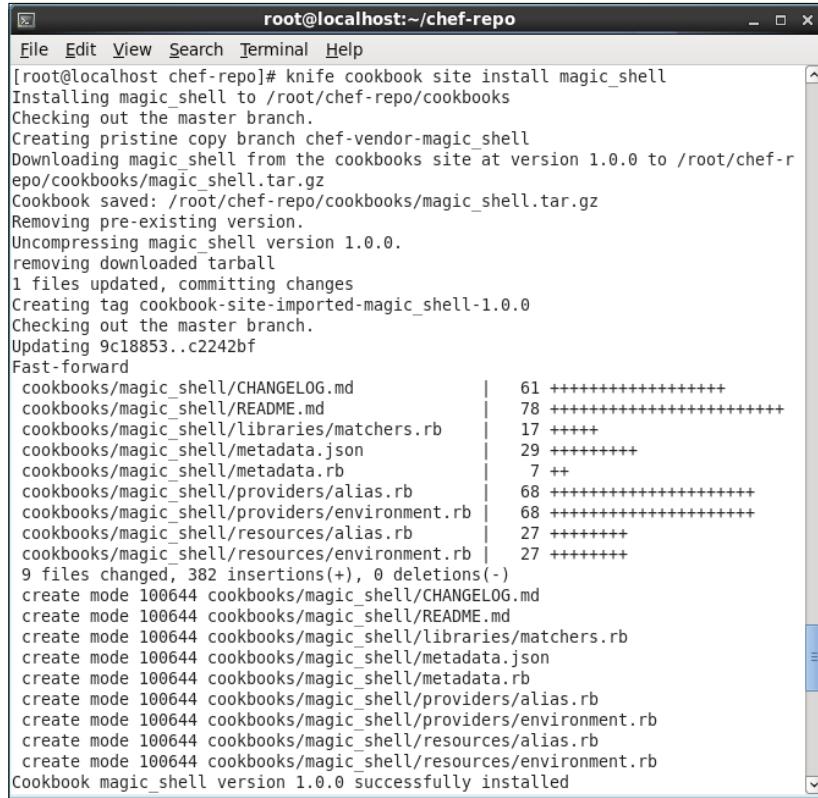
To read more about cookbooks, visit the Chef documentation at <https://docs.chef.io/cookbooks.html>.

Let's take an example of a `magic_shell` cookbook. First, we need to install the `magic_shell` cookbook and make some required changes in the global environment variables before writing our own. Let's begin with the exercise.

As we know, a system that is managed by Chef is referred to as a node.

1. First, install the `magic_shell` cookbook with the help of knife:

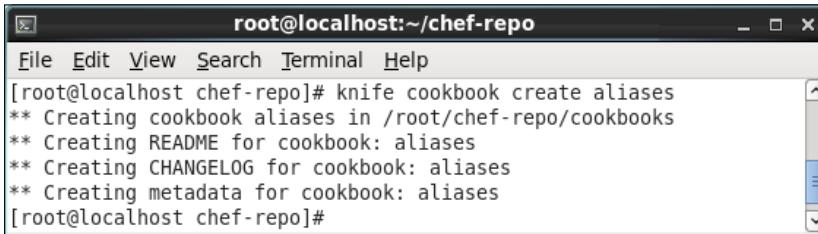
```
$ knife cookbook site install magic_shell
```



```
root@localhost:~/chef-repo
File Edit View Search Terminal Help
[root@localhost chef-repo]# knife cookbook site install magic_shell
Installing magic_shell to /root/chef-repo/cookbooks
Checking out the master branch.
Creating pristine copy branch chef-vendor-magic_shell
Downloading magic_shell from the cookbooks site at version 1.0.0 to /root/chef-repo/cookbooks/magic_shell.tar.gz
Cookbook saved: /root/chef-repo/cookbooks/magic_shell.tar.gz
Removing pre-existing version.
Uncompressing magic_shell version 1.0.0.
removing downloaded tarball
1 files updated, committing changes
Creating tag cookbook-site-imported-magic_shell-1.0.0
Checking out the master branch.
Updating 9c18853..c2242bf
Fast-forward
cookbooks/magic_shell/CHANGELOG.md      |  61 ++++++
cookbooks/magic_shell/README.md          |  78 ++++++
cookbooks/magic_shell/libraries/matchers.rb | 17 +////
cookbooks/magic_shell/metadata.json      | 29 ++++++
cookbooks/magic_shell/metadata.rb        |   7 ++
cookbooks/magic_shell/providers/alias.rb |  68 ++++++
cookbooks/magic_shell/providers/environment.rb |  68 ++++++
cookbooks/magic_shell/resources/alias.rb |  27 +////
cookbooks/magic_shell/resources/environment.rb |  27 +////
9 files changed, 382 insertions(+), 0 deletions(-)
create mode 100644 cookbooks/magic_shell/CHANGELOG.md
create mode 100644 cookbooks/magic_shell/README.md
create mode 100644 cookbooks/magic_shell/libraries/matchers.rb
create mode 100644 cookbooks/magic_shell/metadata.json
create mode 100644 cookbooks/magic_shell/metadata.rb
create mode 100644 cookbooks/magic_shell/providers/alias.rb
create mode 100644 cookbooks/magic_shell/providers/environment.rb
create mode 100644 cookbooks/magic_shell/resources/alias.rb
create mode 100644 cookbooks/magic_shell/resources/environment.rb
Cookbook magic_shell version 1.0.0 successfully installed
```

2. After this, you need to create a new cookbook named aliases:

```
$ knife cookbook create aliases
```



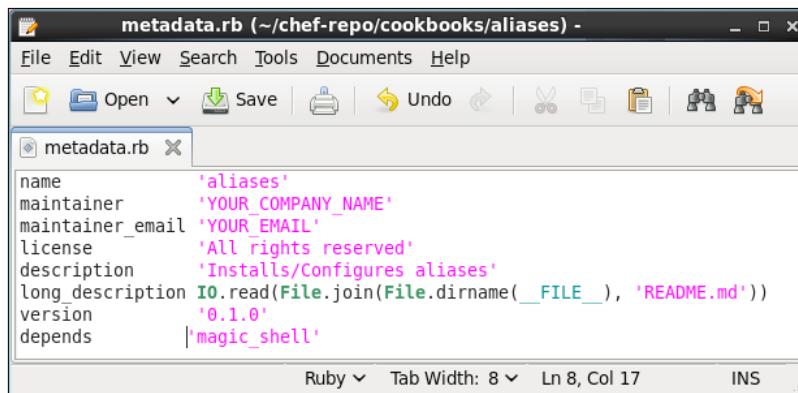
```
root@localhost:~/chef-repo
File Edit View Search Terminal Help
[root@localhost chef-repo]# knife cookbook create aliases
** Creating cookbook aliases in /root/chef-repo/cookbooks
** Creating README for cookbook: aliases
** Creating CHANGELOG for cookbook: aliases
** Creating metadata for cookbook: aliases
[root@localhost chef-repo]#
```

3. The following is the directory structure of the newly created **aliases** cookbook:



4. Now, if you see the directory where all the gathered cookbooks are kept, you will find a new directory. In a plain text editor, open `metadata.rb` and at the end of the file, update the dependency on the `magic_shell` cookbook, which we had installed earlier. Following command is used for update the dependency on the `magic_shell` cookbook:

```
$ depends 'magic_shell'
```



5. For this type of new aliases cookbook, just open the default recipe in the text editor. Now, you can edit magic shell aliases for this recipe. It is open to use and you can also create it, as follows:

```
# Alias `h` to go home  
magic_shell_alias 'h' do  
  command 'cd ~'
```

```
end

# Alias `sites` to cd into apache
magic_shell_alias 'sites' do
  command "cd #{node['apache']['dir']}/sites-enabled"
end

# Set Nano as the default editor
magic_shell_environment 'EDITOR' do
  value 'nano'
end
```

## Uploading cookbooks

After creating a cookbook, we need to upload it on the Chef server, so that the required configuration could be applied to nodes.

First, we need to upload these cookbooks to the Enterprise Chef server using the knife command, only then we can download these cookbooks. Just give the following command:

```
Knife cookbook upload -all
```

## Troubleshooting

Here, we will see all the possible error types which user may encounter during workstation setup and cookbook creation and their troubleshooting steps accordingly.

### Error code – type 1

If you get an error of this type:

```
INFO: Client key /etc/chef/client.pem is not present - registering
INFO: HTTP Request Returned 401 Unauthorized: Failed to authenticate as
ORGANIZATION-validator. Ensure that your node_name and client key are
correct.

FATAL: Stacktrace dumped to c:/chef/cache/chef-stacktrace.out
FATAL: Net::HTTPServerException: 401 "Unauthorized"
```

### Meaning

This means that the organization-validator.pem file is not getting authenticated. It is a Chef 401 "Unauthorized" error.

## Troubleshooting steps

In order to troubleshoot this, the validation key should be regenerated.

In the preceding code, check whether the file is referenced in `validation_key` (usually, `ORGANIZATION-validator.pem`) exists in one of these locations:

```
~/.chef  
~/projects/current_project/.chef  
/etc/chef
```

Check and verify the read permissions and ensure that the permissions are present.

1. A validation key should be regenerated if there is no file present.
2. Go to **Opscode Management Console** and select **Organizations** in the right-hand side of the upper screen; this way, you can create this key again.
3. Next to the organization there is a key named **Regenerate validation key**. Select it for the further process.

## Error code – type 2

You may get an error, as follows:

```
ERROR: Failed to authenticate to https://api.opscode.com/organizations/  
ORGANIZATION as USERNAME with key /path/to/USERNAME.pem  
Response: Failed to authenticate as USERNAME.
```

## Meaning

The previous error code means `client.rb` file is failing to certify `https://api.opscode.com`. It means that after running the `knife` command, your local `client.rb` is not authenticating you to the Chef server.

## Troubleshooting steps

1. You need to ensure and recheck the values in side your `knife.rb` file, especially, `node_name` and `client_key`.
2. In the preceding code, check whether the file `USERNAME.pem` referenced in `client_key` (usually `USERNAME.pem`) exists or not. You need to check this file in the following locations:

```
~/.chef  
~/projects/current_project/.chef  
/etc/chef
```

If a file is present, then you need to ensure the right read permissions. If no file is there, then the client key should be regenerated.

## Error code – type 3

You may get an error, as follows:

```
INFO: Client key /etc/chef/client.pem is not present - registering
INFO: HTTP Request Returned 401 Unauthorized: Failed to authenticate as
ORGANIZATION-validator. Synchronize the clock on your host.
FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out
FATAL: Net::HTTPServerException: 401 "Unauthorized"
```

### Meaning

This means that your system clock has been changed by more than 15 minutes from the actual time. The clock is not properly synchronized on your host machine.

### Troubleshooting steps

This can be fixed by simply synchronizing your clock with an NTP server.

## Error code – type 4

You may get an error, as follows:

```
FATAL: Stacktrace dumped to /var/chef/cache/chef-stacktrace.out
FATAL: Net::HTTPServerException: 403 "Forbidden"
```

### Meaning

This error reflects that there could be an issue with the permissions on the Chef server. It is a Chef 403 Forbidden error.

### Troubleshooting steps

There can be two types of permission issues: one is object-specific and the other is a global permissions. In order to find out the type of permission issue that we are facing, run the Chef-client and set the -l debug to understand the debugging output.

The output of error would be as follows:

```
DEBUG: Sending HTTP Request to https://api.opscode.com/organizations/
ORGNAME/nodes
ERROR: Running exception handlers
```

With the help of the URL, you can easily understand the type of permission issue that you are facing:

- If the URL is /nodes, it means that this is a global permission issue
- If the URL is /nodes/NODENAME, it means that this is an object permission issue

To troubleshoot global permissions' errors, perform the following steps:

1. Navigate to **Opscode Management Console** and click on **Nodes**, which is quite possibly a failed object type.
2. Now, go to the sub tab, **permissions** and click on tab. The required permission issue many depends on the failed request type.
3. Under the group section (**GET**), we need to check whether it has the checked **LIST** permission is checked or not.
4. Under the group section (**POST**), we need to check whether it has the checked **CREATE** permission in checkbox is checked or not.
5. Click on the checkboxes and make sure that they must be always checked; then, click on the **UPDATE** permission.

To troubleshoot object permissions' errors, perform the following steps:

1. Navigate to **Opscode Management Console** and click on **failing object type**.
2. The cause of the error is an object that has to be checked.
3. Now, go to the sub tab, **permissions** and click on it. It depends on the failed request on which permission is needed.
4. Under the group section (**GET**), you should check whether it has the checked **READ** permission.
5. Under the group section (**PUT**), you should check whether it has the checked **UPDATE** permission.
6. **DELETE** permission must be checked.
7. Click on the checkboxes. Keep them checked and click on the **UPDATE** permission.

## Error code – type 5

You may get an error, as follows:

```
Client key /etc/chef/client.pem is not present - registering
WARN: Failed to read the private key /etc/che/validation.pem:
#<Errno::ENOENT: No such file or directory - /etc/chef/validation.pem>
FATAL: Stacktrace dumped to /etc/chef/cache/chef-stacktrace.out
FATAL: Chef::Exceptions::PrivateKeyMissing: I cannot read /etc/chef/
validation.pem
```

### Meaning

This means that Chef is not finding your validation.pem file. There may be no such directory or file as /etc/chef/validation.pem.

### Troubleshooting steps

1. You need to ensure that your validation.pem or ORGANIZATION-validator.pem file is properly downloaded and it is available to the current user.
2. Next, you should ensure that client.rb is matching with the location of your validator.pem file.

## Error code – type 6

You may get an error, as follows:

```
Installing getting-started to /home/jes/chef-repo/.chef/.../cookbooks
ERROR: You have uncommitted changes to your cookbook repo:
  M cookbooks/getting-started/recipes/default.rb
?? .chef/
?? log
```

### Meaning

This is a different kind of warning, not exactly an error. It means that you were installing a cookbook with some changes and those changes have not been committed to Git. Therefore, first, you should commit all your changes and only then import the cookbooks.

## Troubleshooting steps

This issue can be solved by simple changes. For example, the following command will commit the new changes with this message update:

```
git commit -am "Updating for installing a site cookbook"
```

Run the `knife` cookbook again and install the community cookbook again.

## Error code – type 7

You may get an error, as follows:

```
WARN: Cannot find config file: /etc/chef/client.rb, using defaults.  
WARN: No such file or directory - /etc/chef/client.rb  
# ... output truncated ... #  
FATAL: Chef::Exceptions::PrivateKeyMissing: I cannot read /etc/chef/  
validation.pem,
```

## Meaning

This means that the `/etc/chef/client.rb` configuration file is missing.

## Troubleshooting steps

This issue can be solved by providing the full path to your `client.rb` file:

```
chef-client -c /etc/chef/client.rb
```

## Self-test questions

1. What is the command to display all your files in your cookbook's directory?
2. In Chef, where should all of the code be stored?
3. Why do we need the Vagrant installation just after the VirtualBox installation?
4. What is the meaning of the most common error code, Chef 401?
5. Which are the three important authentication files that we download from the Opscode Hosted Chef account?

## Summary

In this chapter, we went through the easy learning exercises that helped us to start working with Chef. We learned about VirtualBox, Vagrant, and Git installation procedures. We also learned about the creation of a Chef repository, how to launch virtual machines, and how to install Ruby and perform the initial settings. We saw the creation and upload of a simple cookbook. We also got detailed practical knowledge of the troubleshooting steps to be taken during the failure of any phase.

In the next chapter, we are going to learn more about cookbook's creation in depth and in detail, as cookbook is the key component of Chef.



# 4

## Learning about Cookbooks

*"We live in a time when automation is ushering in a second industrial revolution, and the powers of the atom are about to be harnessed for ever greater production."*

- Adlai Stevenson

We have already been introduced to cookbooks and how to apply them on running nodes. In this chapter, we are going to see all the minor details of a cookbook because a cookbook is the basic unit of configuration in Chef. To master in Chef, you must know each and every technical aspect of a cookbook.

This chapter will focus on the integral contents of a cookbook. We will come to know the purpose of different components of cookbooks and how to use the following components:

- Cookbook types
- Attributes
- Definitions
- Files
- Libraries
- Resources and providers
- Templates
- Lightweight resources
- Metadata

## Cookbook types

The categorization of a cookbook has come by different user experience. There are many terms used to describe the workings of a cookbook. Broadly, we can categorize cookbooks in three categories: application, library, and wrapper.

### Application cookbooks

Application cookbooks contain complete installation packages according to your company or organization's specification. PostgreSQL, MySQL, Apache2, and Nginx are good examples of an application cookbook. There can be one or more application or different set of application cookbooks used to set up an environment, such as a development or testing or production environment.

### Library cookbooks

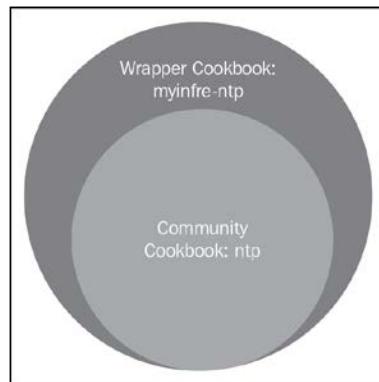
Library cookbooks are used by other cookbooks. Basically, it is a collection of different resources such as definitions, libraries, and **lightweight resources and providers (LWRPs)**. Sometimes, it may include recipes as well, for example, a database cookbook is a type of a library cookbook. If any cookbook acts as a library cookbook, then it should not be directly assigned to nodes. Before assigning a cookbook to nodes, a cookbook should lock the required library version in its `metadata.rb` file.

### Wrapper cookbooks

It is better to use an existing cookbook that suits our requirements rather than writing it from scratch. However, it is always advisable to verify that it is updated properly, based on the latest versions of software that it uses. Often, it is experienced that the existing community cookbooks may not serve all purposes and hence it becomes extremely essential to modify community cookbooks or expand it to suit specific requirements. Hence, a wrapper cookbook is a special cookbook that wraps the community cookbook with custom modifications or additions to suit a specific organization or environment, such as overriding an attribute.

Basically, it changes the behavior of any existing cookbook without forking it. Let's understand this with another example.

If we want to use an `ntp` cookbook to set a server's time, we can download the community `ntp` cookbook from Opscode. After downloading, instead of running it directly, we can modify it according to our infrastructure server's time setting. Following is a representation of wrapper cookbooks for more understanding of the concept:



We can create another `myinfra-ntp` cookbook with the following settings and change the attribute settings in the following manner:

```
myinfra-ntp/attributes/default.rb
default['ntp']['peers'] = ['ntp1.myinfra.com', 'ntp2.myinfra.com']
```

After this, we can add attributes to the recipes:

```
myinfra/recipes/default.rb
include_recipe 'ntp'
```

Now, we can simply run `recipe [myinfra-ntp]` in our running infrastructure, and the default settings from cookbook will automatically come up.

## Components of a cookbook

Suppose that we create a new cookbook with the following command:

```
Knife cookbook create cookbook example1
```

Then, generally, `example1` creates the following directory structure by default:

```
cookbook-example1/
  README.md
  attributes/
  definitions/
```

```
files/
libraries/
metadata.rb
providers/
recipes/
resources/
templates/
```

The following screenshot shows the output for the newly created cookbook named **my\_tomcat** from **Windows PowerShell**:

The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command history and output are as follows:

```
Copyright (C) 2013 Microsoft Corporation. All rights reserved.
PS C:\Users\mitesh> cd E:\chef-repo
PS E:\chef-repo> ls

Directory: E:\chef-repo

Mode                LastWriteTime     Length Name
----                -----          ---- 
d----
```

Let's understand each of these components in detail.

## Attributes

An attribute contains particular information of a node. It can represent the previous state of the node at the end of the Chef-client run, current state of the node, and the state of the node at the current Chef-client run. Nodes, attribute files (for example, `chef-repo/cookbooks/tomcat/attributes/default.rb`), recipes (`chef-repo/cookbooks/tomcat/recipes`), roles, and environments are sources of an attribute.

There are six types of attributes:

Name	Description	Priority	Example of attributes in /attributes/default.rb
default	It resets automatically at the start of every Chef-client run	It has the lowest precedence	<code>default ["tomcat"] ["dir"] = "/etc/ tomcat6"</code>
force_default	It makes sure that cookbook attributes have higher precedence over a role or an environment default attribute set	It has higher precedence over a role or an environment default attribute set	<code>force_ default ["tomcat"] ["dir"] = "/etc/ tomcat6"</code>
normal	It persists in a node object	It has a higher precedence than a default attribute	<code>set ["tomcat "] ["dir"] = "/etc/ tomcat6"  normal ["tomcat "] ["dir"] = "/etc/ tomcat6"</code>
override	It resets automatically at the start of every Chef-client run	It has higher attribute precedence than the default, force_default and normal attributes	<code>override ["tomcat"] ["dir"] = "/etc/ tomcat6"</code>
force_override	It makes sure that cookbook attributes have higher precedence over a role or an environment override attribute set	It has higher precedence over a role or an environment override attribute set	<code>force_ override ["tomcat"] ["dir"] = "/etc/ tomcat6"</code>

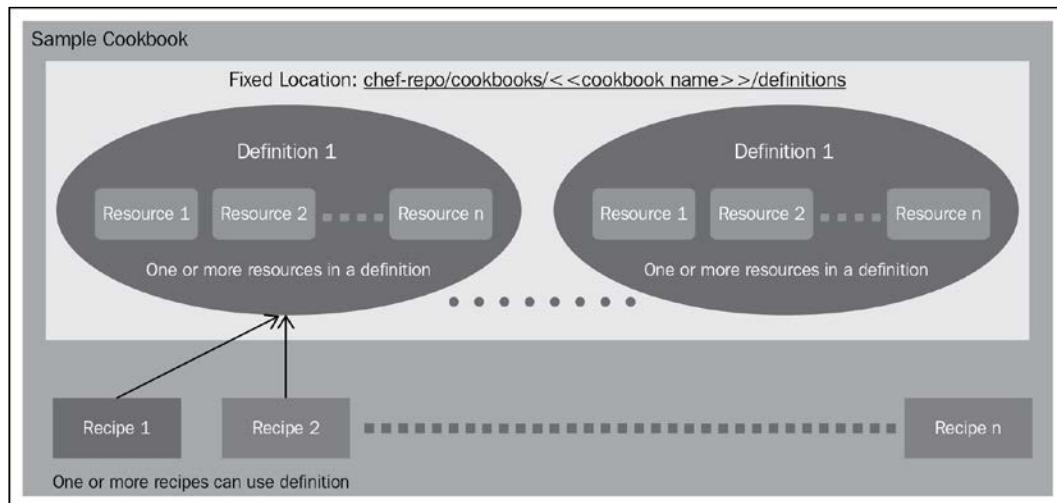
Name	Description	Priority	Example of attributes in /attributes/default.rb
automatic	It has the data that is identified by Ohai at the beginning of every Chef-client run	It has the highest attribute precedence	An automatic attribute contains particular detail of a node, such as domain, hostname, FQDN, macaddress, ipaddress, platform, platform_version, and so on

For more details on the attributes documentation of Chef, visit <http://docs.chef.io/attributes.html>. It gives interesting details in the form of a matrix for precedence of attributes.

## Definitions

A definition is a piece of reusable code. The purpose of a definition is declaration of resources. Definition is a combination of more than two resource declarations. Once the resources are defined, they can be moved to the resource collection. Any number of resources can come under one definition.

There is a specific path to locate a definition; it must be stored in the /definitions folder. Definition can't be declared within a cookbook.



The following are the patterns where definitions are used for effective cookbook writing:

- Where there is the possibility of the same configuration for more than one resources
- When there is the possibility of passing the data to a particular application from more than two recipes
- When an action is preferably sent to a provider, but not directly to a resource

## Syntax of a definition

A definition contains three parts:

- Name of the resource.
- More than one argument is used to define parameters and setting default values. If the default is not specified, then argument would be nil.
- A hash symbol, which is used inside the definition to give access to parameters and values.

Values that we pass in a parameter are taken as a default value.

The following is the syntax for writing a definition:

```
define :resource_name, :parameter => :argument, :parameter =>
:argument do
  params_hash
end
```

For example, a definition named `mysql_site` with a parameter called `action` and an argument for `enable` will look as follows:

```
define :mysql_site, :action => :enable do
  if params[:action] == :enable
    ...
  else
    ...
  end
end
```

When the previous definition is called within a recipe, it looks like the following:

```
mysql_site node['hostname'] do
  port 4000
end
```

You will need to specify the hostname and port number where `Mysql_site` is going to be installed.

## Example of a definition

A definition file can be used to create an object that the Chef-client can use like a resource. For example:

```
mysql_site Definition
define :mysql_site, :enable => true do
  include_recipe "mysql"

  if params[:enable]
    execute "mensite #{params[:name]} do
      command "/usr/sbin/mensite #{params[:name]} "
      notifies :restart, resources(:service => "mysql")
      not_if do
        ::File.symlink?("#{node[:mysql][:dir]}/sites-
enabled/#{params[:name]}") or
        ::File.symlink?("#{node[:mysql][:dir]}/sites-enabled/000-
#{params[:name]}")
      end
      only_if do ::File.exists?("#{node[:mysql][:dir]}/sites-
available/#{params[:name]}") end
    end
  end
```

We are going to install MySQL in the previous example, so we will define it as `mysql_site`. In parameter, we will give the `enable` value. It will execute the `mensite` component in the `/usr/sbin/mensite` location. Once it is installed, the MySQL service restarts and we get a notification. This execution also checks whether the MySQL service is already running on a particular node; if not, only the MySQL installation will get executed.

For more examples, refer to Chef's documentation of definition at <http://docs.chef.io/definitions.html>.

## Files

With the help of files, cookbooks come to know how a distribution will take place according to the platform, node, and version of a file.

## Syntax of a file

Cookbook\_file defines a file's distribution. The following is the syntax for it:

```
Cookbook_file "/usr/local/bin/mysql_module_conf_generate.pl" do
  source "mysql_module_conf_generate.pl"
  mode 0755
  owner "root"
  group "root"
end
```

## Example of a file

The directory structure of /files are as follows:

```
files/
  samplehost.example1.com
  ubuntu-12.04
  ubuntu-10.0
  ubuntu
  redhat-6.4
  redhat-5.2
  -----
  Default
```

The following is the example of a resource type:

```
resource_type "/usr/local/bin/mysql_module_conf_generate.pl" do
  source "mysql_module_conf_generate.pl"
  mode 0755
  owner "root"
  group "root"
end
```

The previous examples explains that the `usr/local/bin/mysql_module_conf_generate.pl` resource would be created from the `mysql_module_conf_generate.pl` file, which is contained in the cookbook. We also need to define the mode, owner, and group permissions.

Here, `resource_type` is acting as `Cookbook_file`. Matching of the resources would be done order wise, as defined in the directory structure. This means that the system will search for the file in its available directory structure in order to execute the resource command:

```
samplehost.example1.com /mysql_module_conf_generate.pl
ubuntu-12.04/mysql_module_conf_generate.pl
```

```
ubuntu-10.0/mysql_module_conf_generate.pl  
ubuntu/mysql_module_conf_generate.pl  
default/mysql_module_conf_generate.pl
```

## Libraries

Libraries help to increase or extend Chef functionalities. With the help of libraries, we can write Ruby code inside a cookbook. This can be done in two ways:

- Implementing of a new class
- Extending the existing class that is being used by a Chef-client

The location of the library for every cookbook is in the `/libraries/library_name.rb` library folder. When we include a library inside the cookbook, it would immediately be available to all the resources, such as definitions, providers, attributes, and recipes.

## Syntax of a library

The following is the syntax of a library:

```
my_Cookbook/libraries/my_example2_library.rb  
# defining a module for including into Chef::Recipe::namespace  
module MyExample2Library  
  def my_function()  
    # ... instructions  
  end  
end  
  
my_Cookbook/recipes/default.rb  
# opens the Chef::Recipe class and combines in the library module  
class Chef::Recipe::namespace  
  include MyExample2Library  
end  
  
my_function()
```

## Example of a library

The following is an example of a cookbook library.

Suppose that a database has various virtual hosts, which are used by different clients. We can create a customized namespace with the help of a library:

```
require 'example'

class Chef::Recipe::ISP
  # calling following with ISP.vhosts
  def system.vhosts
    vh = []
    @db = example1.mysql(
      'web',
      :user => 'name',
      :password => 'name_password',
      :host => 'databaseserver.eg1.com'
    )
    @db [
      "SELECT virtualhost.domainname,
          table1.uid,
          table1.gid,
          table1.homedir
      FROM table1, virtualhosttable
      WHERE table1.uid = virtualhosttable.user_name"
    ].
    vhost_data = {
      :servername => query[:domainname],
      :documentroot => query[:homedir],
      :uid => query[:uid],
      :gid => query[:gid],
    }
    vh.push(vhost_data)
  end
  Chef::Log.debug(" provisioning #{v.length} vhosts")
  vh
end
```

Here, we are creating recipes where the name is `ISP`, which are calling **Internet service provider's (ISPs)** `vhost`. It defines system `vhosts`. Database access parameters are passed into `vhost`, after this, a database query runs, which updates the `vhost` data, such as `servername`, `documentroot`, `uid`, and `gid`.

After creating a custom namespace, we can use it in the recipe.

The following is an example of this:

```
ISP.vhosts.each do |vhost|
  directory vhost[:documentroot] do
    owner vhost[:uid]
    group vhost[:gid]
    mode 0755
    action :create
  end

  directory "#{vhost[:documentroot]}/#{vhost[:domainname]}" do
    owner vhost[:uid]
    group vhost[:gid]
    mode 0755
    action :create
  end
end
```

In the previous example, the custom `vhost` namespace is used in the recipe with the help of a library. While using custom namespace this recipe, we need to pass the user ID (`uid`) of owner, group ID of group (`gid`), and `mode` to permission.

To understand libraries, more examples are available in the Chef documentation at <http://docs.chef.io/libraries.html>.

## Resources and providers

A resource defines the desired state of a node. It describes the actions you want to implement. A resource is a fundamental key unit of a recipe that does the actual work, such as installation of a package or configuration of it, or setting permissions, and so on. Some actions and processes are defined by resources, such as when a package should be installed, where to put the files, what should be the name of a new directory, and so on. A Chef-client run gives each resource and gets its identification done. Later, it relates itself with a provider. Then, the step to complete the action is taken by a resource. Resources are defined in a recipe in an order and they get executed in the same order. A Chef-client makes sure that actions generate the same output result every time. The Ruby language is used to implement a resource in a recipe.

A resource basically represents a piece of a system and also its desired state. A provider is required to describe all the steps that are taken to change the Node's state of a piece of a system from current to desired. The steps which have been undertaken are decoupled from the request itself. The request is performed in a recipe. A lightweight resource describes this request. This lightweight provider defines all the steps.

Each resource is identified and then linked with a provider at the time of the Chef-client execution. A provider's responsibility is to execute the action described by the resources in the given recipe.

## Syntax of resources

The following is the syntax for resource creation. It contains Ruby blocks to represent resources:

```
type "name1" do
  attribute "value1"
  action :describe_action
end
```

A resource has the following four components:

- **type**: Platform resources such as bash, git, package, and so on
- **name**: This is the name of the resource
- **attribute**: There are predefined attributes and default values for most of the attributes
- **action**: There are predefined actions and default values

## Example of resources

The following is an example of how to use a resource to install Apache and start the httpd service after installation:

```
package "httpd" do
  action :install
end

service "httpd" do
  action [:enable, :start]
end
```

To understand resources and providers in detail, refer to the Chef documentation at <http://docs.chef.io/resources.html>.

## Templates

Consider a template as a dynamic file. A cookbook template is an **Embedded Ruby (ERB)** template written using a markup language. The data content of this file is totally based upon a logic (usually complex) or variables. Templates files have Ruby programming expressions and line statements. Across any firm, templates are the best way to arrange configuration files. You can generate a file specific to a recipe, and leverage the metadata of a particular node deployment to produce a custom file.

The following two should be added in a template in order to use it:

- Add a template resource to a recipe
- Add the ERB template to a cookbook

Templates can be found at `/chef-repo/cookbooks/<<cookbook_name>>/templates/`

## Syntax of a template

The following is the syntax of a template:

```
node[:fqdn] = "name1"
template "/tmp/config.conf" do
  source "config.conf.erb"
  variables({
    :x_variable => "are performed"
  })
end
```

## Example of a template

The following is an example of a resource-containing template:

```
template "D:\path1\textfile.txt" do
  source "textfile.txt"
  mode 0755
  owner "root"
  group "root"
end
```

Matching of the resources should be done order wise, as defined in the `/template` directory:

```
/templates
  windows-8.0/textfile.txt
  windows-7.1/textfile.txt
```

```
windows-6.2/textfile.txt  
windows/textfile.txt  
default/textfile.txt
```

Configure a file from a template or a local template or using a variable map. Consider a situation where you want to create a file based on the existence of some attribute on a node. Chef provides the facility to create files based on conditions using templates. It is also possible to use Ruby blocks or strings to provide conditions. Then, use it to create a file based on a template.

More examples of templates are available at [https://docs.chef.io/resource\\_examples.html#template](https://docs.chef.io/resource_examples.html#template).

## An LWRP

The **Lightweight Resources and Providers** (LWRPs) provide an approach to define custom actions of users or system states that are not available in the base Chef installation. A LWRP has two components; it provides a flexible way to define a resource and provider and to implement it with the actual code.

LWRPs express the desired actions, such as installing packages, starting/stopping services, managing firewalls, deploying applications, and so on. It is used in recipes like any other platform resource.

## Components of an LWRP

The following are the components of LWRPs:

- A lightweight resource that describes attributes and actions in a particular set.
- The Chef-client is informed by the lightweight provider about the procedure to handle each configuration/installation action. Lightweight provider also informs the Chef-client about the specific conditions that are fulfilled and so on.

Most of the lightweight providers are developed with the help of basic platform resources, and some lightweight providers are also developed by customizing Ruby code.

After being created once, a LWRP is created again as a Ruby class in the organization. With every Chef-client run, the Chef-client will read the lightweight resources from recipes and process them alongside the other resources. At the time of configuring the node, the Chef-client will utilize another lightweight provider to know the steps that are required to bring the system into the desired state.

A LWRP behaves in the same way with platform resources as it does with providers:

- A lightweight resource is a fundamental key part of a recipe
- A lightweight resource describes the necessary action steps that could be taken
- While a Chef-client runs, each lightweight resource gets identified and then it is associated with a lightweight provider
- A lightweight provider performs the tasks to complete the action requested according to the lightweight resource

To learn more about LWRPs, Inline Compile Mode, and Chef-maintained, visit <http://docs.chef.io/lwrp.html>.

## Metadata

Metadata describes the important information of a cookbook, such as name, description, version, supported operating systems, dependencies, and so on. The filename containing metadata is `metadata.rb`. The data content of the `metadata.rb` file provides hints to the server to correct the deployment of cookbooks to each node.

The following is an example of some configuration settings that are required to configure a cookbook; this type refers to attributes and values of attributes refers to metadata:

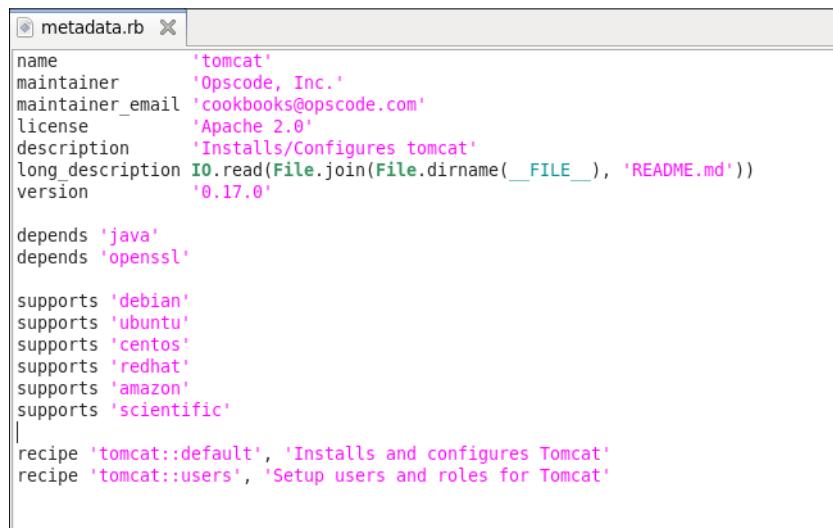
```
attribute 'collection/movies/name',
:display_name => " Movie Name",
:description => "The name of your movies list",
:choice => \[
  'Beautiful mind',
  'Hobbit',
  'Lord of the rings',\]
:type => "string",
:required => " optional",
:recipes => \[ 'movies::watch' \],
:default => " Beautiful mind,"
```

The previous example is just the dummy to understand metadata; here, `display_name`, `description`, `choice`, `type`, `required`, `recipes`, and `default` are the attributes to configure a cookbook and the respective values are metadata.

## The metadata.rb file

The `Metadata.rb` file contains a Ruby DSL. It is used to build the `metadata.json` file. The `metadata.rb` file is a human-readable version of the `metadata.json` file. When you create a new cookbook using `knife`, a `metadata.rb` file will automatically be created. Metadata is compiled and a similar process takes place whenever the `knife cookbook metadata` subcommand is run. When it is run, a cookbook is uploaded onto the Chef-server. Knife creates a `metadata.rb` file and a `knife cookbook` creates a `run` subcommand.

To understand metadata file with a real-time example, let's take the `metadata.rb` file of a Tomcat cookbook available at <https://github.com/opscode-cookbooks/tomcat/blob/master/metadata.rb>.



```

metadata.rb
-----
name          'tomcat'
maintainer    'Opscode, Inc.'
maintainer_email 'cookbooks@opscode.com'
license       'Apache 2.0'
description   'Installs/Configures tomcat'
long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))
version       '0.17.0'

depends 'java'
depends 'openssl'

supports 'debian'
supports 'ubuntu'
supports 'centos'
supports 'redhat'
supports 'amazon'
supports 'scientific'
|
recipe 'tomcat::default', 'Installs and configures Tomcat'
recipe 'tomcat::users', 'Setup users and roles for Tomcat'

```

## The Error message

A cookbook has dependencies at times. For example, a Tomcat cookbook depends on a proper Java installation. If metadata does not include proper dependency details, then the Chef server may ignore it. This scenario results in an error message. It is best practice to verify dependence entries in the `metadata.rb` file if there is an error message from the Chef-server about cookbook distribution.

## **Self-test questions**

1. What are the three different types of cookbooks?
2. How can we write Ruby code inside the cookbook? What are the possible ways to write this?
3. What is the purpose of a library in a cookbook and how can we use it?
4. What is the main purpose of LWRP in a cookbook?
5. What is the syntax for declaring resources in a cookbook?
6. What are the two important elements/resources to use in templates?
7. How can we edit the `metadata.rb` file and what are the ways to edit it?

## **Summary**

In this chapter, we learned about different types of cookbooks and all the prominent components of a cookbook, which are essential skills to become a Chef developer. We learned about definitions, files, libraries, resources and providers, templates, lightweight resources, and metadata with practical examples. Now, we can apply these concepts to develop a cookbook according to our organizational requirement.

In the next chapter, we are going to learn various services and commands to manage nodes.

# 5

## Managing the Nodes

*"Any sufficiently advanced technology is indistinguishable from magic."*

- Arthur C. Clarke

In this chapter, we are going to perform practical exercises with nodes. There are various operations that are applied on nodes, such as adding a new node, deleting a node, and editing nodes. We will see how search queries are executed and how various types of search options are associated with search queries. We will also see details of data bags and how to report some actions when some error message triggers using handlers.

In this chapter, we will cover the following:

- Adding and deleting nodes
- Bootstrapping target nodes
- Introducing search
- Introducing data bags
- Introducing handler

### Adding and deleting a node

As we know, a node could be physical, virtual, or a cloud instance that is configured and maintained by a Chef-client.

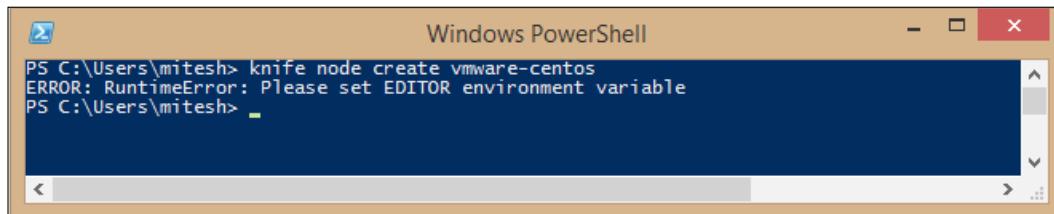
`Knife node` is used as a subcommand, which is used to manage a node on a Chef server.

## Adding a new node

A `create` command is used to create/add a new node to the server. Data of the node is stored in the **JavaScript Object Notation (JSON)** format:

```
knife node create vmware-centos
```

By running the preceding command, we may get the error message or warning message: **ERROR: RuntimeError: Please set EDITOR environment variable.**



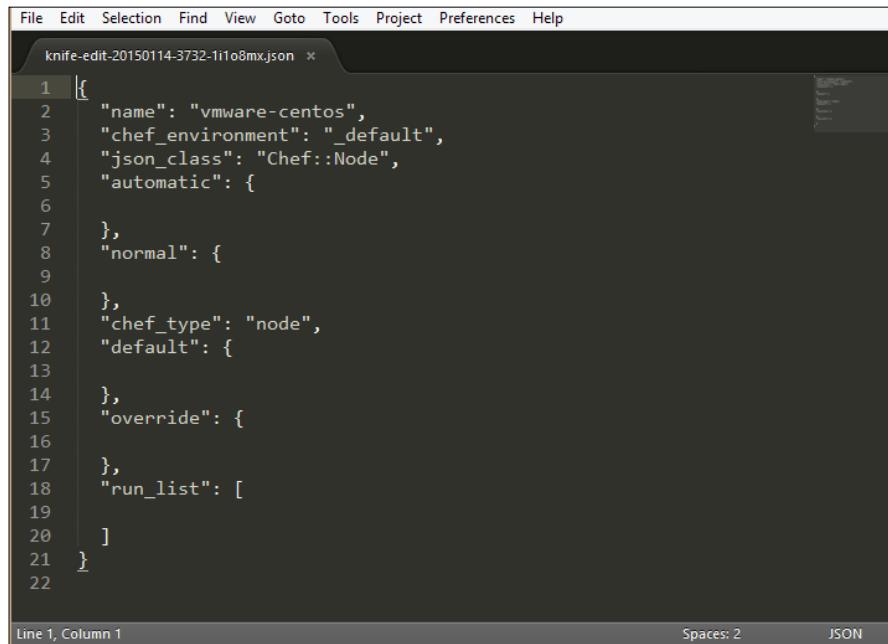
The preceding command requires that information should be edited as JSON data by using a text editor.

Add the following line to the `chef-repo/.chef/knife.rb` file to configure a text editor that needs to be used by knife on Windows:

```
knife[:editor] = '"C:\Program Files\Sublime Text 2\sublime_text.exe"  
-nosession -multiInst'
```

We can also set `EDITOR` environment variable. For more details to set the text editor on different platforms with a different text editor, visit <https://docs.chef.io/chef/knife.html#set-the-text-editor>.

Now, after setting the Sublime Text as a text editor, run `knife node` and create `vmware-centos`. It will open the Sublime Text editor, as follows:



```

File Edit Selection Find View Goto Tools Project Preferences Help
knife-edit-20150114-3732-11108mx.json *
1 {
2   "name": "vmware-centos",
3   "chef_environment": "_default",
4   "json_class": "Chef::Node",
5   "automatic": {
6     },
7     "normal": {
8       },
9       "override": {
10      },
11      "default": {
12        },
13        "run_list": [
14          ]
15      }
16      }
17      }
18      }
19      }
20      }
21      }
22      }

Line 1, Column 1
Spaces: 2
JSON

```

Here, `vmware-centos` is the name of the new node.

After adding the new node, we can add the node data in the JSON format:

```

## vmware-centos
{
  "normal": { },
  "name": "vmware-centos",
  "override": { },
  "default": { },
  "json_class": "Chef::Node",
  "automatic": { },
  "run_list": [
    "recipe[zsh]",
    "role[webserver]"
  ],
  "chef_type": "node"
}

```

## Managing the Nodes

We can also add a new node by using the `file` command, if we already have any pre-existing node data, as a template:

```
$ knife node from file FILEName
```

The setting of a node from a file can be edited to the `knife.rb` file:

Open the `knife.rb` file in text editor and check the `knife [:print_after]` option.

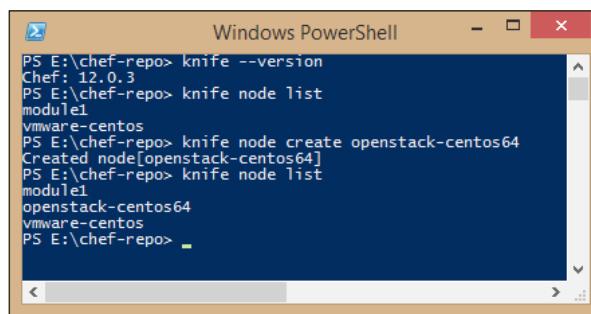
We can add the node using the data that exists in an existing JSON file:

```
$ knife node from file "path to JSON file"
```

The `list` command is used to view all the nodes on a server:

```
$ knife node list
```

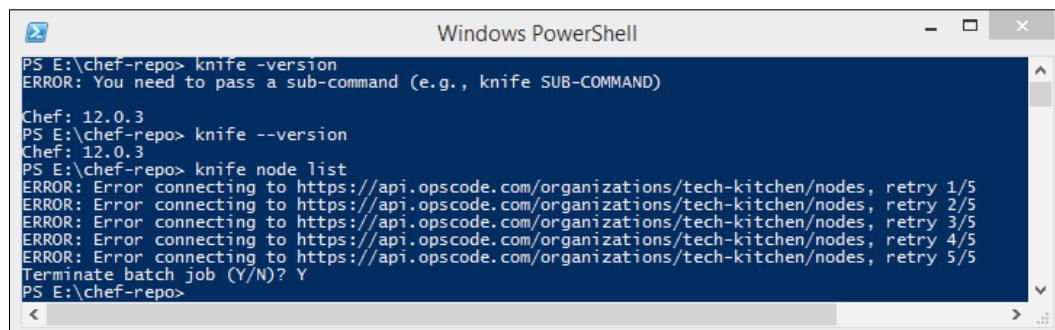
This command will return an output, as follows:



```
Windows PowerShell
PS E:\chef-repo> knife --version
Chef: 12.0.3
PS E:\chef-repo> knife node list
module1
vmware-centos
PS E:\chef-repo> knife node create openstack-centos64
Created node[openstack-centos64]
PS E:\chef-repo> knife node list
module1
openstack-centos64
vmware-centos
PS E:\chef-repo>
```

Command `knife node list` will return the name of the nodes. We can use **Windows PowerShell** or Command Prompt for Windows and **Terminal** for Linux platforms.

While executing the previous command if you get ERROR: Error connecting to `https://api.opscode.com/` while executing the `knife` command when you are using Hosted Chef, then verify your Internet connection.



```
Windows PowerShell
PS E:\chef-repo> knife -version
ERROR: You need to pass a sub-command (e.g., knife SUB-COMMAND)

Chef: 12.0.3
PS E:\chef-repo> knife --version
Chef: 12.0.3
PS E:\chef-repo> knife node list
ERROR: Error connecting to https://api.opscode.com/organizations/tech-kitchen/nodes, retry 1/5
ERROR: Error connecting to https://api.opscode.com/organizations/tech-kitchen/nodes, retry 2/5
ERROR: Error connecting to https://api.opscode.com/organizations/tech-kitchen/nodes, retry 3/5
ERROR: Error connecting to https://api.opscode.com/organizations/tech-kitchen/nodes, retry 4/5
ERROR: Error connecting to https://api.opscode.com/organizations/tech-kitchen/nodes, retry 5/5
Terminate batch job (Y/N)? Y
PS E:\chef-repo>
```

## Deletion of a node

The delete command is used to delete a node from a server:

```
knife node delete NewNode_Name
```

A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command PS E:\chef-repo> knife node list is run, showing three nodes: module1, openstack-centos64, and vmware-centos. The next command PS E:\chef-repo> knife node delete openstack-centos64 is run, followed by a confirmation prompt "Do you really want to delete openstack-centos64? (Y/N) Y". The final command PS E:\chef-repo> knife node list is run again, showing only the two remaining nodes: module1 and vmware-centos.

Here, we are removing a node and verifying the deletion of the `knife node list` command. The `bulk delete` is a command to delete two or more nodes simultaneously. In order to run this command, we have to give a value in a pattern of regular expression:

```
$ knife node bulk delete REGEX
```

`REGEX` is a regular expression, which can be modified according to our search options.

This could be `^ [0-7] {2} $` or `^ [0-5] {3} $` or any other pattern that we wish to delete.

## Editing a node

Using the edit command, we can edit the node details. The following command is used for the same:

```
$ knife node edit NODE_NAME (options)
```

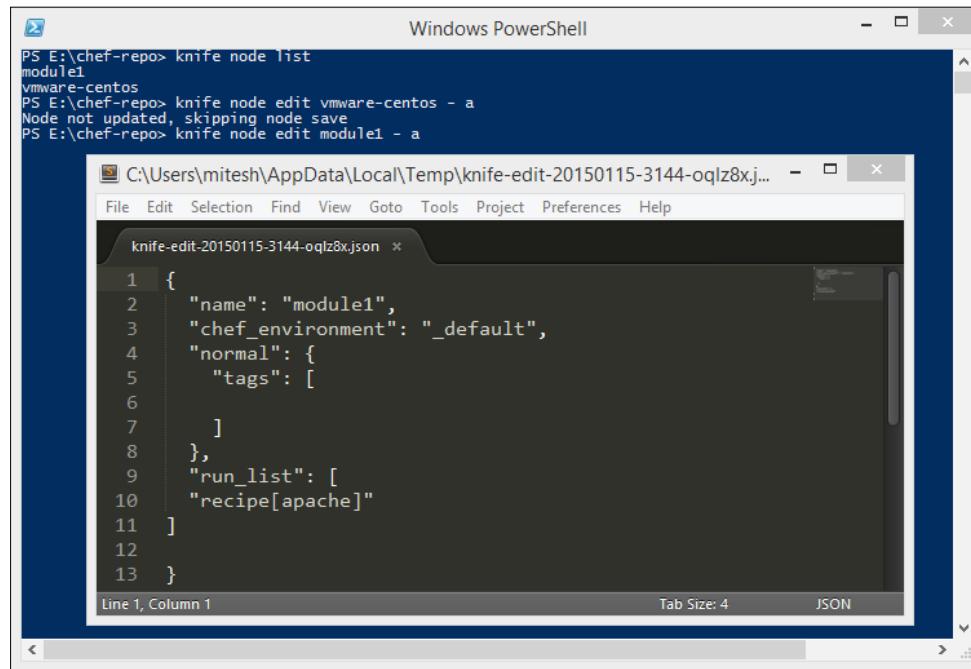
These options could be `-a`, `--all`:

```
$ knife node edit node_name - a
```

## Managing the Nodes

---

After running the preceding command, the default editor is called and we can change or modify the details of a node in the JSON code:



The screenshot shows a Windows PowerShell window and a code editor window. The PowerShell window has the following commands and output:

```
PS E:\chef-repo> knife node list
module1
vmware-centos
PS E:\chef-repo> knife node edit vmware-centos - a
Node not updated, skipping node save
PS E:\chef-repo> knife node edit module1 - a
```

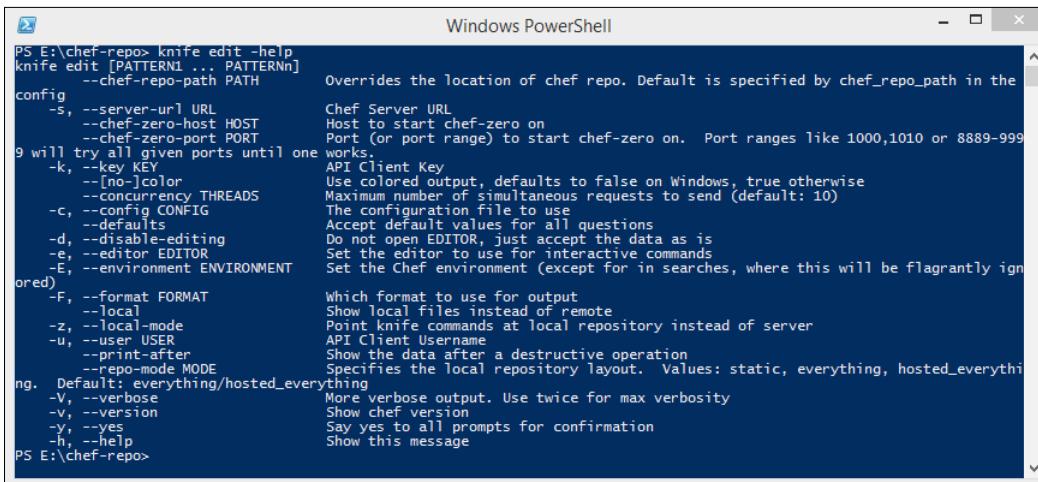
The code editor window shows a JSON file named `knife-edit-20150115-3144-oqlz8x.json`. The JSON content is:

```
1 {
2   "name": "module1",
3   "chef_environment": "_default",
4   "normal": {
5     "tags": [
6       ""
7     ],
8     "run_list": [
9       "recipe[apache]"
10    ]
11 }
```

The code editor has tabs for "File", "Edit", "Selection", "Find", "View", "Goto", "Tools", "Project", "Preferences", and "Help". The status bar at the bottom shows "Line 1, Column 1", "Tab Size: 4", and "JSON".

In a Linux-based OS, previous command will open in a specific editor selected by the user, as configured in `knife.rb`.

To get help for a specific node, execute the `knife <command_name> -help` command.



The screenshot shows a Windows PowerShell window displaying the help output for the `knife edit` command. The output is as follows:

```
PS E:\chef-repo> knife edit -help
knife edit [PATTERN1 ... PATTERNn]           Overrides the location of chef repo. Default is specified by chef_repo_path in the config
  --chef-repo-path PATH                     config
                                             Overrides the location of chef repo. Default is specified by chef_repo_path in the config
                                             --chef-repo-path PATH
                                             Overrides the location of chef repo. Default is specified by chef_repo_path in the config
                                             -s, --server-url URL             Chef Server URL
                                             --chef-zero-host HOST          Host to start chef-zero on
                                             --chef-zero-port PORT         Port (or port range) to start chef-zero on. Port ranges like 1000,1010 or 8889-999
9 will try all given ports until one works.
                                             -k, --key KEY                 API Client Key
                                             --[no-]color                  Use colored output, defaults to false on Windows, true otherwise
                                             --concurrency THREADS        Maximum number of simultaneous requests to send (default: 10)
                                             -c, --config CONFIG          The configuration file to use
                                             --defaults                   Accept default values for all questions
                                             -d, --disable-editing        Do not open EDITOR, just accept the data as is
                                             -e, --editor EDITOR          Set the editor to use for interactive commands
                                             -E, --environment ENVIRONMENT Set the Chef environment (except for in searches, where this will be flagrantly ignored)
                                             -F, --format FORMAT          Which format to use for output
                                             --local                      Show local files instead of remote
                                             -z, --local-mode              Point knife commands at local repository instead of server
                                             -u, --user USER               API Client Username
                                             --print-after                Show the data after a destructive operation
                                             --repo-mode MODE             Specifies the local repository layout. Values: static, everything, hosted_everything
ng. Default: everything/hosted_everything
                                             -V, --verbose                 More verbose output. Use twice for max verbosity
                                             -v, --version                 Show chef version
                                             -y, --yes                     Say yes to all prompts for confirmation
                                             -h, --help                    Show this message
PS E:\chef-repo>
```

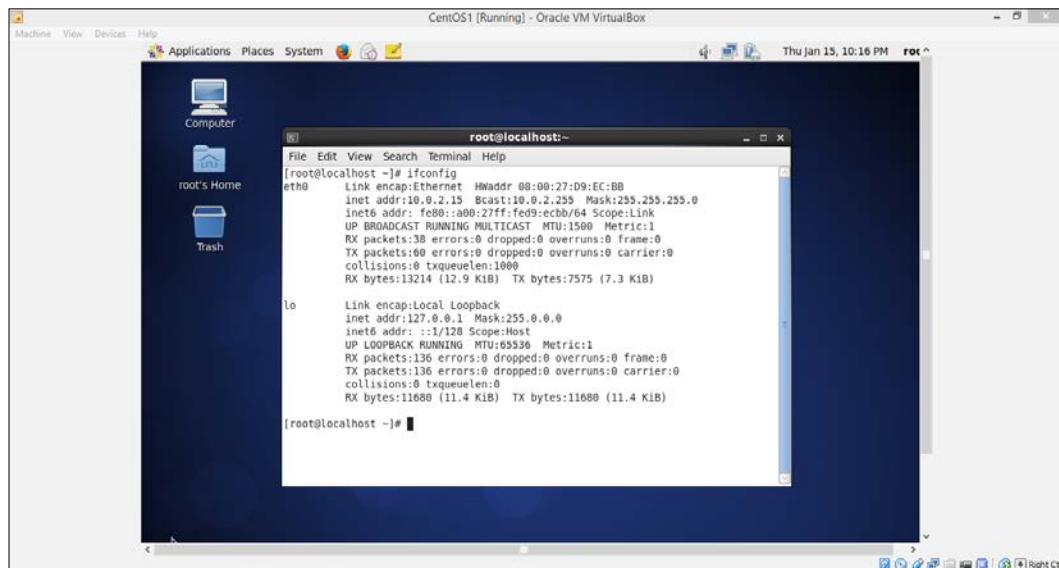
## Bootstrapping target nodes

Bootstrap is a program that initializes an operating system during the startup of a system. In Chef, bootstrapping is the process that is performed by a Chef-client to configure a node. It is the standard and easiest way to install a Chef-client on a node.

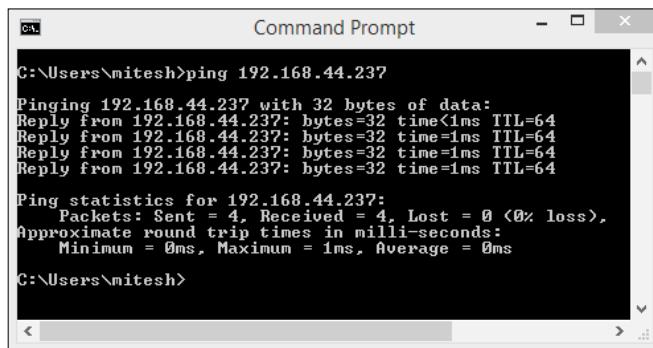
We can configure any number of new nodes by bootstrapping.

To bootstrap a node, you need to follow these steps:

1. Get an IP address for the node by using the `ifconfig` command on a Linux-based OS and the `ipconfig` command on Windows.



Check whether the nodes are accessible from the system or not with the use of the ping command.

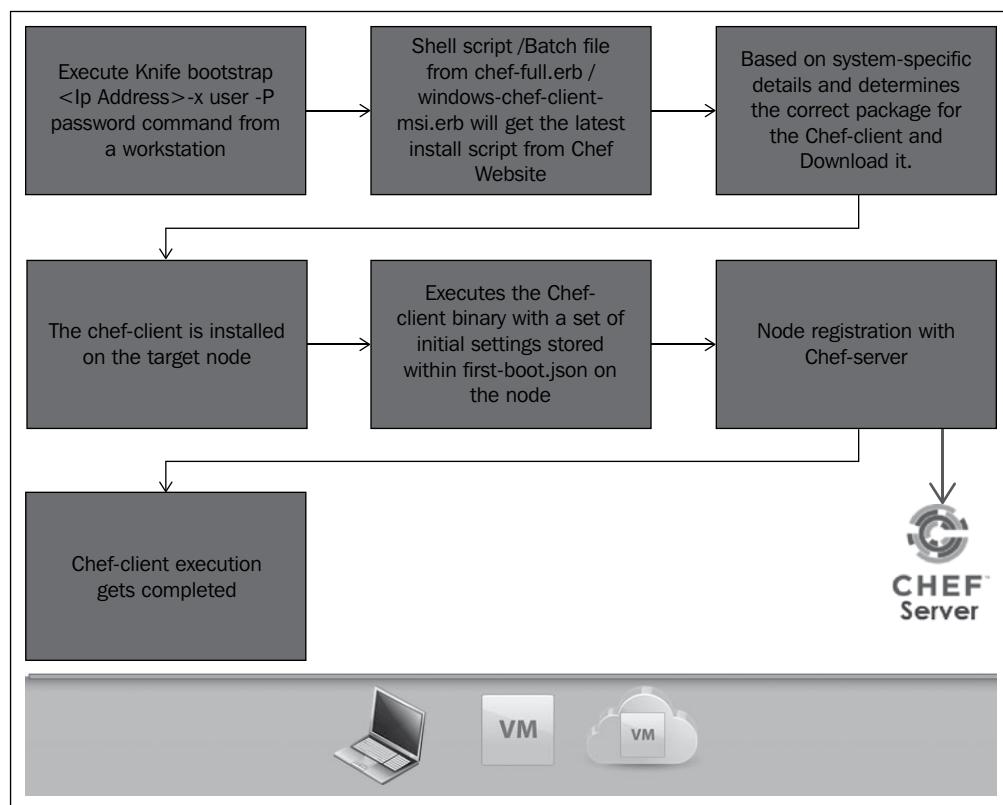


2. Then, the `knife` command is needed to run.
3. The verification process for the node on the server needs to be done, that is, the server should be checked for its presence.

## The Knife.bootstrap command

After configuring a workstation, it is used to install a Chef-client simultaneously on various nodes using the `Knife` command. Bootstrap uses SSH in the target machines. A FQDN or IP address is needed to run the `knife bootstrap` command.

The following figure shows the different phases of a Bootstrap operation:



After this, workstation installs the Chef-client if required, otherwise, it generates keys and then registers a node with the Chef server.

Apart from the IP address of the target machine, we also need SSH credentials, such as a username, password, or an identification file.

Enter the following command from a workstation:

```
$ knife bootstrap 192.168.44.237 -x username -P password --sudo
```

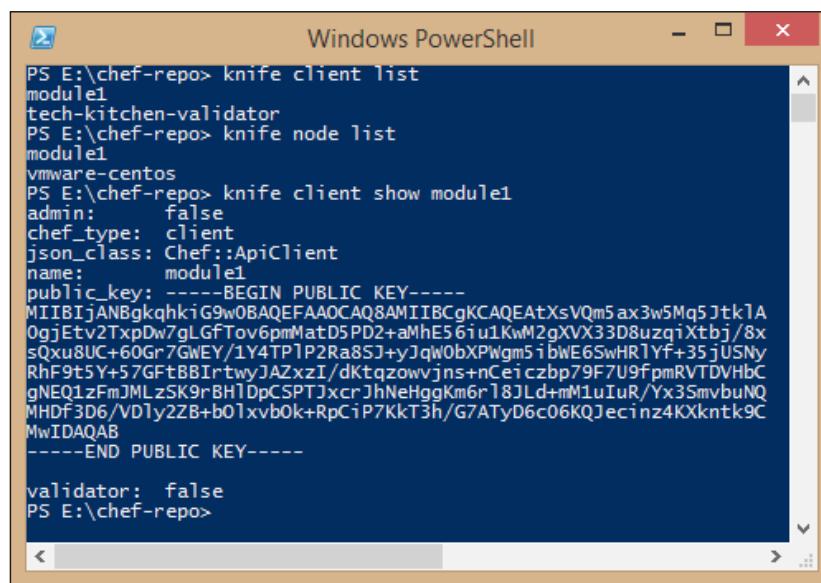
## The verification process for a node

Once bootstrapping is done, verify that the node is recognized by the server. We can run the following `knife` subcommands for this:

In order to verify the node that was bootstrapped, the following command should be run:

```
$ knife client show name_of_node
```

Here, `name_of_node` is the node that is bootstrapped. After running this command, the server will return the following code:



```
Windows PowerShell
PS E:\chef-repo> knife client list
module1
tech-kitchen-validator
PS E:\chef-repo> knife node list
module1
vmware-centos
PS E:\chef-repo> knife client show module1
admin:      false
chef_type:  client
json_class: Chef::ApiClient
name:       module1
public_key: -----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFA0CAQ8AMIIBCgKCAQEAtXsVQm5ax3w5Mq5Jtk1A
0gjEtV2TxpDw7gLgfTov6pmMatD5PD2+aMhE56iu1KwM2gXVX33D8uzqiXtbj/8x
sQxu8UC+60Gr7GNEY/1Y4TP1P2Ra8Sj+yJqWObXPWgm5ibWE6SwHR1Yf+35jUSNy
RhF9t5Y+57GftBBIrtywJAZxzI/dKtqzowvjns+nCeiczb79F7U9fpnRVTDVHbC
gNEQ1zFmJMLzSK9rBHIDpCSPTJxcrJhNeHggKm6r18JLd+nM1uIuR/Yx35mvbuNQ
MHDF3D6/VDlyzzB+b01xvb0k+RpCiP7KkT3h/G7ATyD6c06KQJecinz4KXkntk9C
MwIDAQAB
-----END PUBLIC KEY-----

validator:  false
PS E:\chef-repo>
```

We can see all the nodes that are registered with the server by running the following command:

```
$ knife client list
```

## Introducing search

Search is the procedure that allows users to get any type of information and data that is indexed by the Chef server.

Search is basically a text query that is performed by many locations, for example, /search/INDEX in the Chef server, within a recipe. These locations can be used to run the search command with Knife with the help of the search functionality provided in the Management Console or by creating indexes in the API (the Chef server), such as:

- /search/Index
- /search
- The search command with knife

A Chef search includes searching for nodes, roles, data bags, and environments.

The `knife search` subcommand triggers a search query to get the stored information indexed on a server.

## Syntax of a search query

A `search` query command is basically a combination of two things: key and search.

`key` is the piece of JSON data that we are focusing on and `pattern` is the text string that is being searched.

The syntax of a search query is:

`key: search_pattern`

In this syntax, `key` is the name of the field that is there in the JSON explanation of an object that is indexable on the server, such as a node, client, role, data bag, environment. The `search_pattern` command is the pattern to defining what is there for searching. Using one of the searching patterns, that is, exact, range, wildcard, or fuzzy matching, you will note that `key` and `search_patterns` are case sensitive.

Key has confined its support to a limited extent for wildcard character matching by using an asterisk. However, this asterisk (\*) should not be the first character.

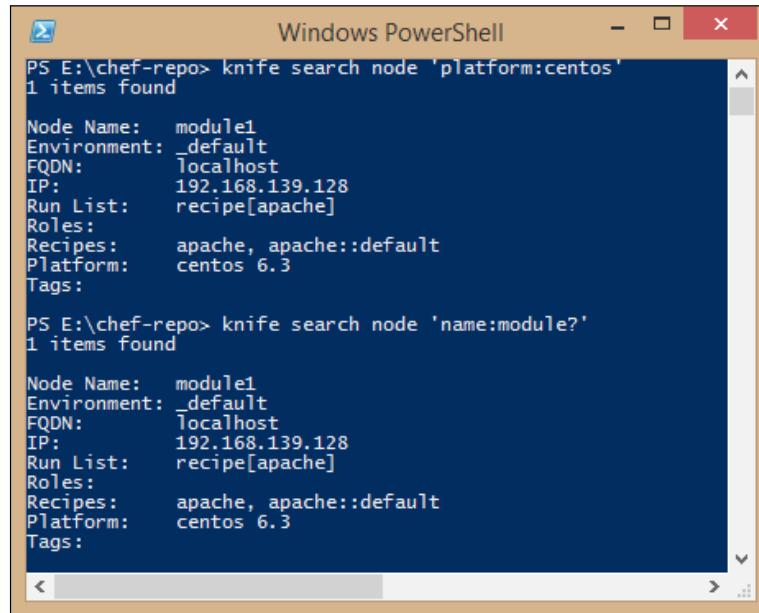
## Search by different options

The Search command can be run using the following options:

### Search by node

To search for all the nodes running under a CentOS operating system, type the following command:

```
$ knife search node 'platform:centos'
```

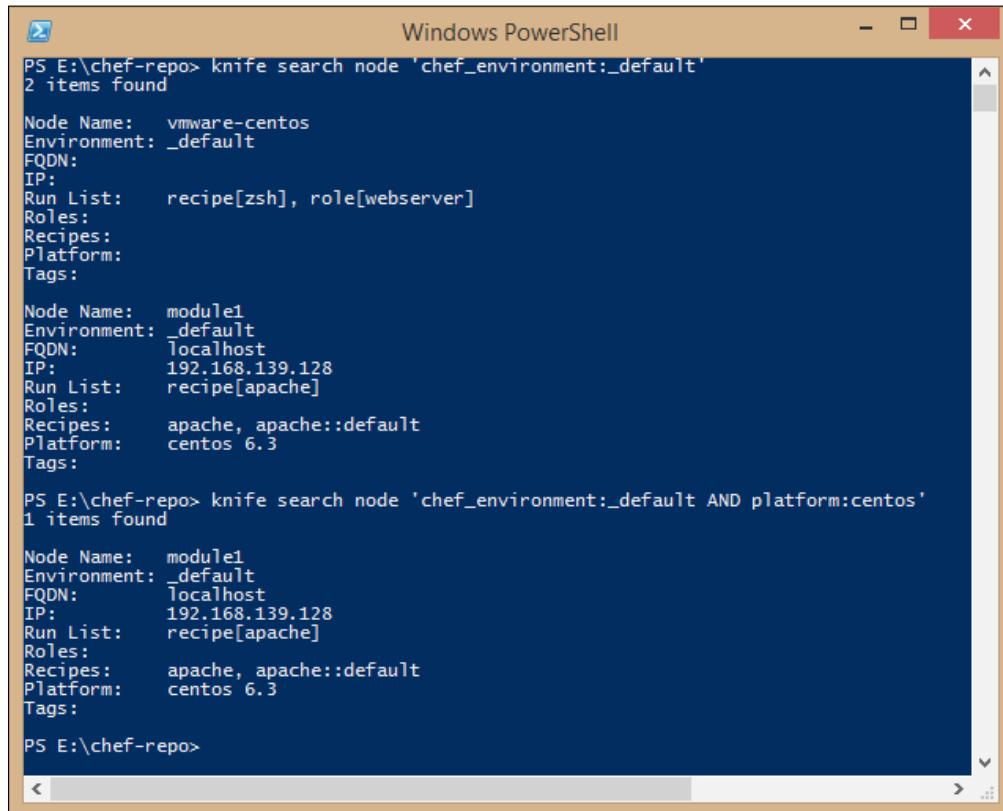


A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window displays two separate command-line executions. The first execution shows the output of the command "knife search node 'platform:centos'". It lists one item found, which is a node named "module1" with the following details: Environment: \_default, FQDN: Localhost, IP: 192.168.139.128, Run List: recipe[apache], Roles: (empty), Recipes: apache, apache::default, Platform: centos 6.3, Tags: (empty). The second execution shows the output of the command "knife search node 'name:module?'" and also lists one item found, which is the same node "module1" with the same details.

## Search by node and environment

To search for all the nodes running under Ubuntu in the development environment, use the following command:

```
$ knife search node 'chef_environment: production AND platform: Ubuntu'
```



A screenshot of a Windows PowerShell window titled "Windows PowerShell". The window displays two sets of Chef node search results. The first search, "knife search node 'chef\_environment:\_default'", finds two items. The first node is named "vmware-centos", running on "localhost" with IP "192.168.139.128", assigned roles "zsh" and "webserver", and using recipes "recipe[zsh]" and "role[webserver]". The second node is named "module1", also running on "localhost" with IP "192.168.139.128", assigned roles "apache" and "apache::default", and using recipe "recipe[apache]". The second search, "knife search node 'chef\_environment:\_default AND platform:centos'", finds one item, which is the "module1" node from the previous search.

```
PS E:\chef-repo> knife search node 'chef_environment:_default'
2 items found

Node Name: vmware-centos
Environment: _default
FQDN:
IP:
Run List: recipe[zsh], role[webserver]
Roles:
Recipes:
Platform:
Tags:

Node Name: module1
Environment: _default
FQDN: localhost
IP: 192.168.139.128
Run List: recipe[apache]
Roles:
Recipes: apache, apache::default
Platform: centos 6.3
Tags:

PS E:\chef-repo> knife search node 'chef_environment:_default AND platform:centos'
1 items found

Node Name: module1
Environment: _default
FQDN: localhost
IP: 192.168.139.128
Run List: recipe[apache]
Roles:
Recipes: apache, apache::default
Platform: centos 6.3
Tags:

PS E:\chef-repo>
```

## Search for nested attributes

To find nested attributes, the following patterns should be used:

```
$ knife search node <query_to_run> -a <main_attribute>.<nested_attribute>
```

For example, the following query is used to search for a kernel machine name in a particular node:

```
$ knife search node name:<node_name> -a kernel.machine
```

A nested search query searches the details within a node.

## Search for multiple attributes

By using an underscore (\_) between the attributes (to separate the attributes), we can run a search query for more than one attribute. For example, in a specific version of the language named Ruby, the query that is needed to run to search for all the nodes is as follows:

```
$ knife search node "languagesrubyversion:1.9.3"
```

The preceding query will search for all the nodes that are running Ruby version 1.9.3.

The following query can be used in the `knife ssh` command for a customized search. We can also exclude the parameters that we don't want in the final result:

```
$ knife search node "role:web AND NOT name:web01"
```

Now, this query searches for all the servers with a web role except the server named `web01`.

## A partial search

If we want to search for some very specific attributes that are stored in the Chef server, then a search query is used, but partially, as `partial_search` returns only those values that match. Partial search queries consume less network bandwidth and memory.

In order to use a partial search query, the first `partial_search` method is used. After this, we will state the key path for different attributes that will return after the execution of the query.

Specifications should be there for every key path. That too as an array of strings and get mapping to a random name in short. For example:

```
partial_search(:node, 'role:database',
  :keys => {ip'    => [ 'ipaddress' ],
             'name' => [ 'name' ],
             }

).each do |result|
  puts result['ip']
  puts result['name']

end
```

A partial query will search for nodes that have a database role. After this, it will pull out the results with a hash key and an IP address.

To understand search in more detail, visit [http://docs.chef.io/knife\\_search.html](http://docs.chef.io/knife_search.html).

## Introducing data bags

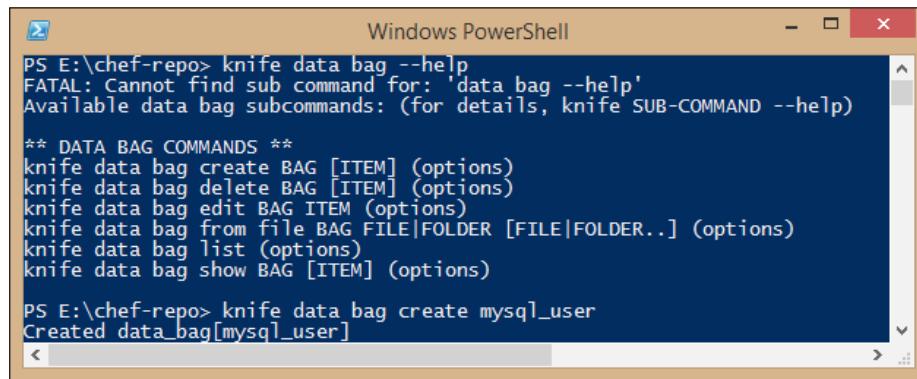
A data bag is JSON's data format global variable that is used as an index for search queries and is accessible from the Chef server. Data bag is loaded through recipes and most of the time, it contains secured information, such as passwords.

Data bag is very good for securing secret information because data bag can be encrypted easily.

The `knife data bag` subcommand is used to create, delete, edit, list, and show data bags.

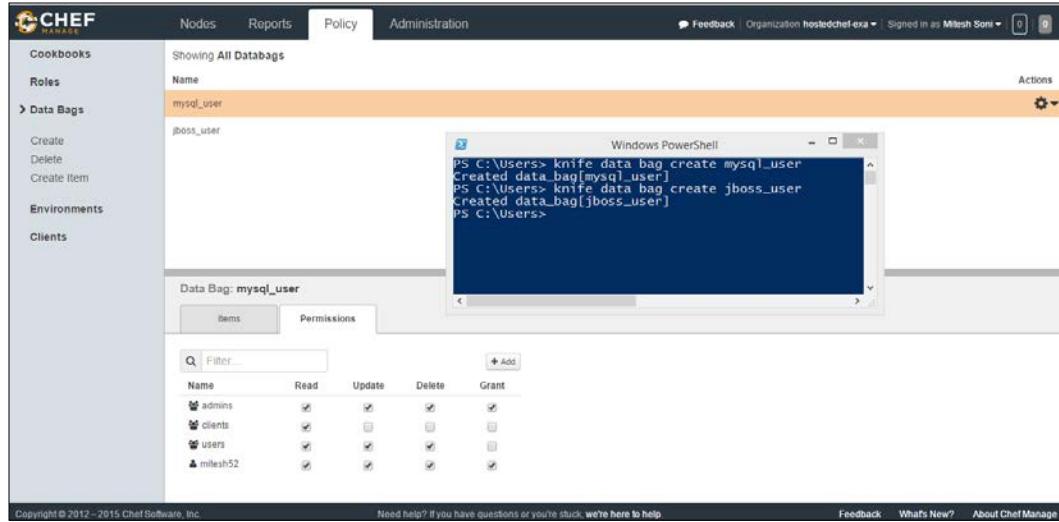
The `create` argument creates a data bag on the Chef server:

```
knife data bag create <>DATA_BAG_NAME>> [DATA_BAG_ITEM] (options)
```

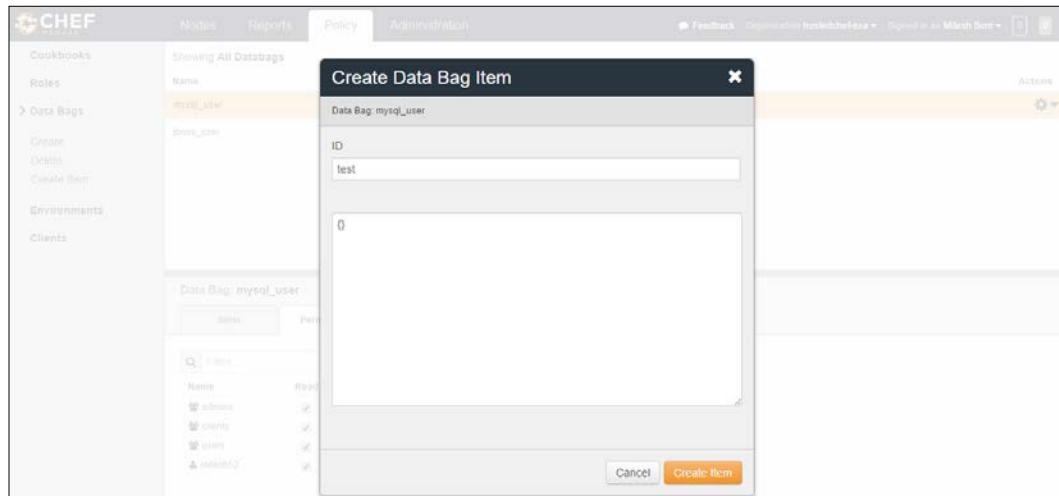


A screenshot of a Windows PowerShell window titled "Windows PowerShell". The command entered is "PS E:\chef-repo> knife data bag create <>DATA\_BAG\_NAME>> [DATA\_BAG\_ITEM] (options)". The output shows an error message: "FATAL: Cannot find sub command for: 'data bag --help' Available data bag subcommands: (for details, knife SUB-COMMAND --help)" followed by a list of subcommands: "\*\* DATA BAG COMMANDS \*\*", "knife data bag create BAG [ITEM] (options)", "knife data bag delete BAG [ITEM] (options)", "knife data bag edit BAG ITEM (options)", "knife data bag from file BAG FILE|FOLDER [FILE|FOLDER..] (options)", "knife data bag list (options)", and "knife data bag show BAG [ITEM] (options)". Below this, the command "PS E:\chef-repo> knife data bag create mysql\_user" is run again, resulting in the output "Created data\_bag[mysql\_user]".

The following is a screenshot of the verification of a data bag creation on the Hosted Chef server:



We can also create a data bag from the dashboard of the hosted Chef server. Go to **Policy**, click on **Data Bags**, and select **Create Item**.



## Managing the Nodes

We can also create an item associated with a data bag from the dashboard of the hosted Chef server.

The screenshot shows the Chef Manage interface. The left sidebar has sections for Cookbooks, Roles, Data Bags, Environments, and Clients. Under 'Data Bags', 'mysql\_user' is selected. The main panel shows a table with one row:

Name	Actions
test	

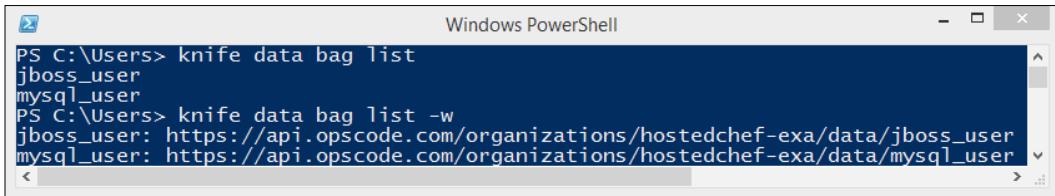
A modal window titled 'Data Bag: mysql\_user' is open, showing the same table with one row. It includes tabs for 'Items' and 'Permissions', and a search bar.

The `create` argument deletes a data bag from the Chef server:

```
knife data bag delete <>DATA_BAG_NAME>> [DATA_BAG_ITEM] (options)
```

The screenshot shows the Chef Manage interface with the 'mysql\_user' data bag selected. A confirmation dialog box titled 'Delete Data Bag' is overlaid on the page, containing the message: 'You are about to delete 1 data bag: mysql\_user'. It includes 'Cancel' and 'Delete' buttons.

The `list` argument lists data bags that are currently available on the Chef server.



```
PS C:\Users> knife data bag list
jboss_user
mysql_user
PS C:\Users> knife data bag list -w
jboss_user: https://api.opscode.com/organizations/hostedchef-exa/data/jboss_user
mysql_user: https://api.opscode.com/organizations/hostedchef-exa/data/mysql_user
```

To find a data bag or item of a data bag, it is necessary to have a specified data bag name. Also, while the search is going on, a query named as the `search` query string should be provided.

If we want to search with the `knife` command within data bag items that are named as `admin_data`, but exclude items that are named as `admin_users`, then the `search` command would be as follows:

```
$ knife search admin_data "(NOT id:admin_users)"
```

We can also include this search query in a recipe, and the code block will be similar to this:

```
search(:admin_data, "NOT id:admin_users")
```

There can be confusion between data bag items about what is needed and what is not. A search query is the best way to avoid this confusion, although, all essential data is returned by using this search query. However, it can be mandatory to save everything in a data bag, but whole data is not known for information. Some examples are here to show that a recipe uses a list of search queries in a specified way inside a data bag, such as `admins`. For example, in order to search for administrator, use the following command:

```
search (:admins, "*:*")
```

Search for an administrator named `mark`:

```
search (:admins, "id:mark")
```

Search for an administrator that has a group identifier named `dev`:

```
search(:admins, "gid:dev")
```

In the same way, we can also search for an administrator whose name starts with the letter `b`:

```
search(:admins, "id:b*")
```

After successful execution of a search query, we get the items of a data bag as a result, and these items can be used as hash values.

We got admins and id as a search result for the preceding query. These could be used for the following query:

```
mark = search(:admins, "id:mark").first
# => variable 'mark' is set to the mark data bag item
mark["gid"]
# => "ops"
mark["shell"]
# => "/bin/zsh"
```

The following example demonstrates how the same thing can be done using explicit access of data bag fields as can be done by using the search mechanism.

A specific recipe is used to align a user with each administrator by saving all the items with the name of the data bag as admins. By looping all admin in the data bag, and after that, by creating a user resource for each admin.

The recipe to do this is as follows:

```
admins = data_bag('admins')
admins.each do |login|
  admin = data_bag_item('admins', login)
  home = "/home/#{login}"
  user(login) do
    uid      admin['uid']
    gid      admin['gid']
    shell    admin['shell']
    comment  admin['comment']
    home     home
    supports :manage_home => true
  end
end
```

We can use the preceding recipe with some changes, such as we can make an array to store search results. The following query will search for all admin users and the result will be stored in the array:

```
admins = []

search(:admins, "*:*").each do |admin|
  login = admin["id"]
  admins << login
```

```

home = "/home/#{login}"
user(login) do
  uid      admin['uid']
  gid      admin['gid']
  shell    admin['shell']
  comment  admin['comment']
  home     home
  supports :manage_home => true
end
end

```

## Introducing handler

Sometimes, we need to perform certain actions in response to some specific situations, for example, sending an e-mail when a Chef-client run fails. In such instances, a handler is used to trigger a specific response. A handler collects data from a Chef-client run and that data is used for different types of analytics in various industries or organizations.

## Types of handlers

There are basically three types of handlers:

Exception	Report	Start
<ul style="list-style-type: none"> <li>It identifies situations which causes chef-client run to fail</li> </ul>	<ul style="list-style-type: none"> <li>It provides details about Chef-client when it succeeds</li> </ul>	<ul style="list-style-type: none"> <li>It provides a way to run events at the start of the Chef-client run</li> </ul>

- **Report handler:** This is triggered when a Chef-client runs successfully
- **Exception handler:** This is triggered when a Chef-client run fails
- **Start handler:** This is triggered to run events at the beginning of the chef-client run

## Installation and configuration of a handler

We can install a handler manually and with the help of a `chef_handler` LWRP.

First, we will take a look at the manual installation of a handler.

## The manual installation

To install and configure a handler, first, the `client.rb` file must be edited. Using RubyGems, we can install the handler in a simple way. After installing it on the system, we will have to allow it in the `client.rb` file using the required command.

For example, if we are using RubyGems to install handlers, then we need to extend a code in the `client.rb` file that will make it work:

```
require "rubygems"
require "name_of_handler"
```

It is also possible that a handler has been installed with another method, and now, it only needs a full path to the file:

```
require "/var/chef/handlers/email_me" # the path of installation
```

In order to write a simple handler, we just need to add the `client.rb` file. It will be moved together as a single code block.

```
require "/var/chef/handlers/email_me" # the path of installation
email_handler = MyOrganization::EmailMe.new # a simple handler
report_handlers<<email_handler # these trigger at the end of a successful run
exception_handlers<<email_handler # these trigger at the end of a failed run
```

After installation, a handler also requires some additional configuration, which may differ from handler to handler.

## Using `chef_handler`

Chef handler is a very lightweight resource. Recipes that use the Ruby language have inbuilt exception handlers which is used by `chef_handlers`.

Using `chef_handler`, we can enable exception and report handlers within recipes, and after that, these recipes can be added to any node's run-list, where it is required to be run. In the following code, we are using `chef_handler` in a recipe:

```
chef_handler "MyOrganization::EmailMe" do
  source "/var/chef/handlers/email_me"
  action :enable
end
```

## Writing a simple handler

In case a Chef-client run fails, a simple handler sends a mail for the description of the failure. Handler uses a pony library for this process (<https://github.com/benprew/pony>). The easy way is to send an e-mail in Ruby. Also, for this work to be done, a mail can be sent to localhost (via /usr/sbin/sendmail or via SMTP).

One example to better understand this process is to learn how e-mail is sent in a Pony library:

```
require 'rubygems'
require 'pony'
module MyOrganization
  class EmailMe < Chef::Handler
    def initialize(from_address, to_address)
      @from_address = from_address
      @to_address = to_address
    end
    def report
      # The Node is available as +node+
      subject = "Chef run is failed on #{node.name}\n"
      # +run_status+ is a value object with all run status data
      message = "#{run_status.formatted_exception}\n"
      # Join the backtrace lines. Coerce to an array just in case.
      message<< Array(backtrace).join("\n")
      Pony.mail(
        :to => @to_address,
        :from => @from_address,
        :subject => subject,
        :body => message)
    end
  end
end
```

We can customize or modify this example according to our requirements.

## Open source handlers

We can also use open source handlers while working with Chef. Here are the examples of some open source handlers:

- **Syslog:** This handler contains basic log-related information, such as information related to success or failure of a Chef-client run
- **Simple E-mail:** This handler collects data of exception and report handlers. Then, based on the Erubis template format, it uses Pony to send e-mails that contain reports

- `SNSHandler`: This handler receives an exception, and then, it makes a report that sends it to the subscribed SNS topic
- `Cloudkick`: This handler takes exception and report handler's data together and then it sends it to Cloudkick (which is a combination of cloud server monitoring and management tools)
- `Updated Resource`: This handler is used for displaying purpose; it displays all updates of all Chef-client run processes

## Self-test questions

1. How do you configure a text editor that needs to be used by Knife?
2. Which command is used to view all the nodes on a Chef server?
3. What is bootstrapping? What are the different phases of a Bootstrap operation?
4. What is the use of data bags?
5. Explain the types of handlers, and give examples of open source handlers.

## Summary

Here, we learned how to add new nodes, delete existing nodes, and edit the nodes in detail. We also went through the bootstrapping process on target nodes. We understood search, data bags, and the usage of different types of handlers in Chef.

In the next chapter, we are going to learn about open source Chef servers in detail, such as installation of an open source Chef server, installation of an open source Chef server on virtual machines, and upgradation of an open source Chef server.

# 6

## Working with an Open Source Chef Server

*"Technology makes it possible for people to gain control over everything, except over technology."*

- John Tudor

An open source Chef version is free for all and anyone can download and use it for testing and learning purposes. We don't need to purchase an enterprise license for an open source Chef server. Although we get some limited functionalities with open source Chef, it is better to get familiar with an open source Chef server first. Here, we are going to learn installation and configuration of an open source Chef server through some easy steps.

We already learned that the Chef server is the central point for all the activities during a Chef run. It stores cookbooks, policies, and metadata, which contain information about the connected nodes.

Although a Chef-client works on the node to get the configuration details from the Chef server, a Chef-client gives the details of recipes, file distribution, and templates to the node. We can add as many nodes as we want depending on our organizational needs. As a result, this approach provides efficiency to manage our infrastructure, irrespective of the number of nodes.

In this chapter, we are going to learn about the following topics:

- Installation of an open source Chef server
- Installation of an open source Chef server on a virtual machine (VMware Fusion and VMware Workstation)
- Installation of a workstation and node on a CentOS 6.x virtual machine
- Using a community cookbook of Tomcat to upload on an open source Chef server and installing Tomcat cookbook on a node
- Upgradation of an open source Chef server

## System requirements

Following are the system requirements to install Chef server:

System requirements to install open source Chef server	
<b>System architecture</b>	x86_64 compatible system architecture Enterprise Linux/CentOS/ Oracle Linux (version 5, 6) and Ubuntu (version 10.04, 10.10, 11.04, 11.10, 12.04, 12.10). Install updates before the Chef server installation.
<b>FQDN or an IP address</b>	A <b>fully-qualified domain name (FQDN)</b> is the domain name of a specific computer, or host on the network. It is also known as the absolute domain name. It has two parts: the domain name and the hostname. A virtual machine with a local hostname, my_chef_server and a parent domain name myorg . com has the FQDN my_chef_server.myorg . com.
<b>NTP</b>	This is a networking protocol for clock synchronization. Its current version is NTPv4. Direct connection to NTP to prevent the effects of variable network latency.
<b>Apache Qpid daemon disablement</b>	It is necessary to disable Apache Qpid daemon on CentOS and Red Hat Enterprise Linux
<b>Mail transfer agent</b>	Mail relay, to allow Chef server to send e-mail notifications
<b>User and group account</b>	This refers to a local user account and group account that has privileges to some required services
<b>Firewall configuration</b>	Open port 80 and 443. Nginx service uses it.
<b>Libraries</b>	libfreetype and libpng
<b>Git</b>	To keep the revision of services
<b>Cron</b>	Cron performs periodic maintenance tasks

<b>Network accessibility</b>	Nodes must be able to communicate with the Chef server
<b>Hardware requirement for standalone deployment</b>	<p>More details on hardware requirements are available at <a href="https://docs.chef.io/install_server_pre.html">https://docs.chef.io/install_server_pre.html</a>.</p> <ul style="list-style-type: none"> <li><b>RAM:</b> 4 GB</li> <li><b>Cores:</b> 4 total cores, 2.0 GHz AMD 41xx/61xx or Intel Xeon 5000/E5 CPUs or faster</li> <li><b>Free disk space:</b> 5 GB each in /opt and /var</li> </ul>

An open source Chef server installation has the following prerequisites:

## Installing an open source Chef server

In this section, we are going to learn about the installation procedure for an open source Chef server. The following are the steps for installation:

1. First, navigate to <https://www.chef.io/download-open-source-chef-server-11/>.
2. Select the appropriate operating system, then the version of the operating system, and then the architecture of the selected OS.

The screenshot shows the 'Open Source Chef Server 11' download page. At the top, there's a navigation bar with links for 'WHAT IS CHEF?', 'LEARN CHEF', 'RESOURCES', and 'GET CHEF'. Below the navigation, the title 'Open Source Chef Server 11' is displayed. A sub-header explains that this is the open source version of Chef Server 11 and provides a link to the Chef Server download page. On the left, there's a sidebar with instructions for selecting the kind of system (Enterprise Linux 6), version (11.16.1), and architecture (x86\_64). The main content area is titled 'Downloads' and contains a form where the user has selected 'chef-server-11.16.1.el6.x86\_64.rpm'. Below this, there's an 'Instructions' section with a note about manual installation and a link to documentation. A callout box highlights the selected file path: 'chef-server-11.16.1.el6.x86\_64.rpm'.

3. After selecting the right options, download the link for Chef server 11.x from the given link.
4. Downloading the file may take some time. Once that is done, the open source Chef server is ready for execution.
5. After executing the open source Chef server, we need to configure the Chef server with the help of the following command:  

```
$ sudo chef server-ctl reconfigure
```
6. Now, all the Chef server supporting modules (RabbitMQ, erchef, PostgreSQL) will be installed automatically. This will also install all the necessary cookbooks that are required to maintain the Chef server.
7. In order to verify that all the components of Chef server 11.x have been installed successfully, we need to run the following command:  

```
sudo chef server-ctl test
```
8. The preceding command will run the test for all the installed components of the Chef server and will revert to the previous state if something is not installed properly.
9. Now, we need to set up our workstation in the same way as we did in *Chapter 3, Workstation set up and cookbook creation*.
10. After setting up the workstation, we need to check whether the Chef-client and registration of the user has been done properly. To do so, type the following command:

```
$ knife client list  
$ knife user list
```

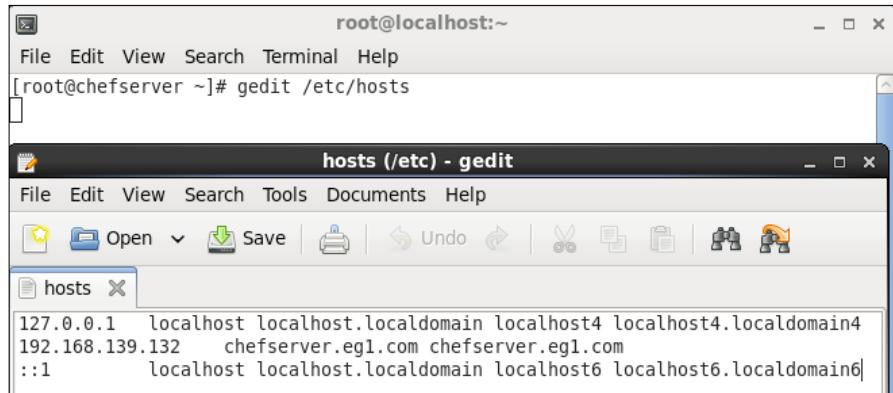
## FQDN and hostnames configuration

The following requirements must be completed by the hostname for the open source Chef server:

Host name configuration should be inclusive of the domain suffix and the hostname must be an FQDN. This means that the hostname is resolvable, such as:

`chefserver.eg1.com`

In the production environment, adding the hostname for the server to the DNS will ensure that the hostname is resolvable. In some cases, if the server is in a testing environment, then hostname is added to the `/etc/hosts` file is there to give assurance that the hostname is resolvable.



## Restarting the virtual machine

The `api_fqdn` setting can be added to the `private-chef.rb` file. If these settings are not there by default, then the value of FQDN should be equal to the FQDN for the service URI used by the server. Set the same value for the bookshelf, `['vip']` before installing Enterprise Chef:

```
api_fqdn "chefserver.egl.com"
```

Now, we need to verify that the hostname is correctly mapped with the FQDN by entering the following command:

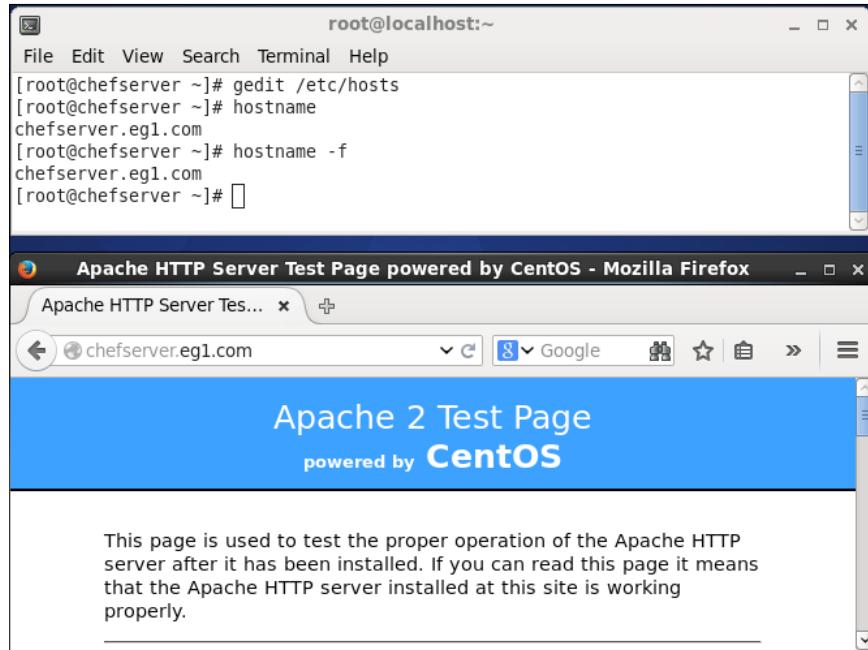
```
$ hostname
```

If the hostname is not correctly mapped with the FQDN, then it should first be mapped properly. To check whether the hostname is resolvable or not, we need to run the following command:

```
$ hostname -f
```

If it is resolvable, then we will get the following output:

```
$ chefserver.eg1.com
```



## Changing the hostname

If the hostname is not resolvable, then we need to map it correctly. Updation of the hostname for the server is necessary. Processes can be varied according to the platform where the server is running. The instructions for changing the hostname can be decided after seeing the manual for the platform, or we can even contact the local administrator for any special and specific information/guidance. The following example shows how a hostname can be changed while running Red Hat or CentOS:

```
$ sudo hostname 'chefserver.eg1.com'
```

Then, run the following command:

```
$ echo'chefserver.eg1.com' |sudo tee /etc/hostname
```

## **Installing an open source Chef server on a VMware Fusion virtual machine – Ubuntu 12.04**

We have studied in *Chapter 1, An Overview of Automation and Advent of Chef*, that the Chef installation can be done on physical servers, Cloud instances, and on virtual machines. Here, we are going to learn how to install Chef on a virtual machine. For this, we will use VMware Fusion 5x, even though we can use any type of virtual machine to install the Chef server. VMware products are most commonly used and they specialize in virtualization, therefore, we prefer them.

### **VM machine settings**

In order to install the Chef server 11.x, there are some initial settings that should be applied on the virtual machine:

- The system must be run under VMware Fusion 5x
- Here, we will use 64-bit Ubuntu 12.04 version; it must be installed properly inside the virtual machine
- After the Ubuntu installation with all the default settings, we should have a small footprint in the virtual machine
- A web browser should be installed on the same virtual machine where the Chef sever is going to be installed
- The URL that is going to access the open source server must include the IP address or FQDN of the virtual machine, for example,  
`https://127.93.32.12`
- The network's setting must be set to a bridged adapter
- In an Ubuntu-supported virtual machine, there should not be any other application or software installed, apart from the Chef server

## **Installing an open source Chef server on a VM machine**

After completing the initial settings in the VM machine, we are now going to install an open source Chef server on a virtual machine.

The following are the sequence of steps that we need to follow:

1. Again, we are going to install the open source Chef server 11.x version. Therefore, we need to follow the same procedure as we did before. Now, we need to repeat the process from step 1 to step 6 as we previously did, while installing the open source Chef server.
2. After following the mentioned steps, we will be able to install the open source Chef server on a VM machine. Now, we need to verify the settings that the network adapter must be set to bridge the mode in the VM machine.
3. After completing the recommended settings, we need to shut down the adapter using the following command:  
`$ sudoifdown eth0`
4. Now, restart it:  
`$ sudoifup eth0`
5. We need to get the IP address of the open source Chef server by entering the following command:  
`$ ifconfig`
6. The IP address will be reflected in the `inetaddr` field.
7. Now, the open source server's IP addresses, that we got from the preceding command should be entered in the URL. Note that it gets accessed through HTTPS only. For example, `154.74.3.189`.
8. Finally, you will get the login page of the open source Chef server. Now, all you need to do is enter the default username and password provided by the Opscode:

`Username: admin`

`Password: p@ssw0rd1`

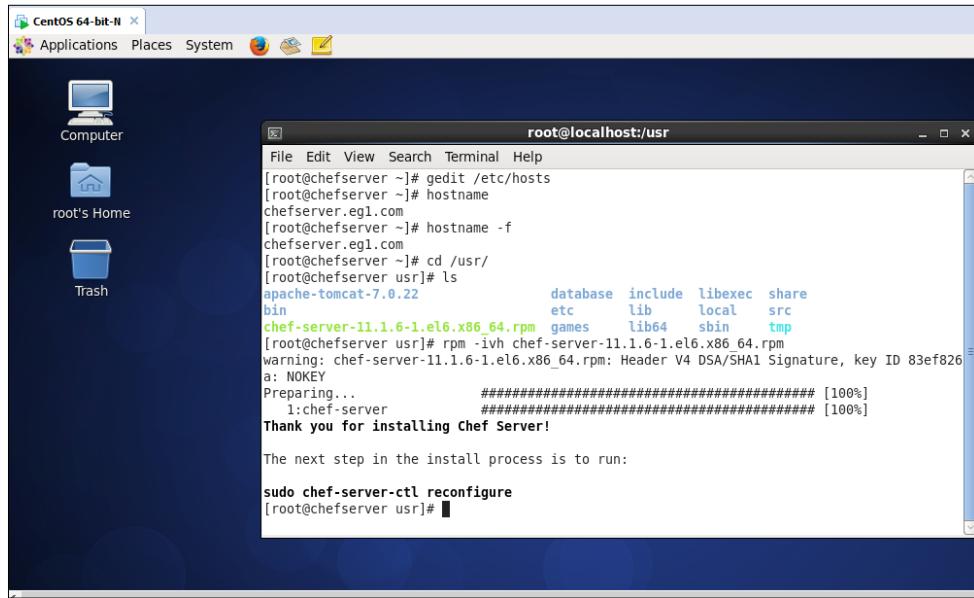
## **Installing an open source Chef server on a VMware Workstation virtual machine – CentOS 6.x**

Now, we will install the Chef server on a CentOS 6.x virtual machine created on a VMware Workstation.

Earlier in this chapter, we saw how to download `chef-server-11.1.6-1.el6.x86_64.rpm`.

Open the terminal and execute the following command:

```
[root@chefserver usr]# rpm -ivh chef-server-11.1.6-1.el6.x86_64.rpm
```



Configure the Chef server by executing the following command:

```
[root@chefserver usr]# chef-server-ctl reconfigure
```

It will take some time to configure.

```
[root@localhost:~]#
File Edit View Search Terminal Help
+      "checkpoint_timeout": "5min",
+      "checkpoint_completion_target": 0.9,
+      "checkpoint_warning": "30s"
+
}
+
],
+
"run_list": [
+ "recipe[chef-server]"
+
+
+]
- change mode from '' to '0600'
- change owner from '' to 'chef_server'
- change group from '' to 'root'
- restore selinux security context

Recipe: chef-server::erchef
* service[erchef] action restart
- restart service service[erchef]

Running handlers:
Running handlers complete

Chef Client finished, 273/286 resources updated in 98.441251623 seconds
chef-server Reconfigured!
[root@chefserver usr]#
```

Once the Chef server is configured successfully, use the `chef-server-ctl test` command to check whether the Chef server is working fine or not.



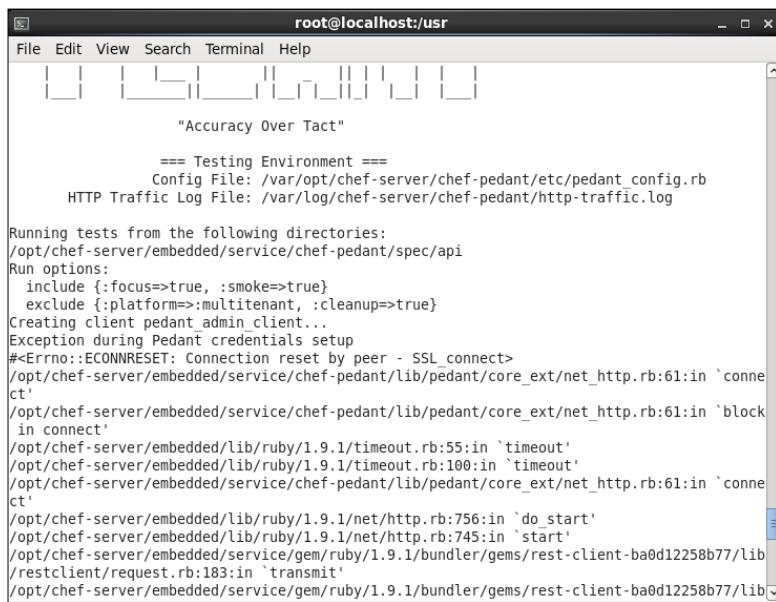
```
root@chefserver:~# chef-server-ctl test
The source :rubygems is deprecated because HTTP requests are insecure.
Please change your source to 'https://rubygems.org' if possible, or 'http://rubygems.org'
if not.
Configuring logging...
Creating platform...
Configured URL: https://chefserver.egl.com:443
Starting Pedant Run: 2015-01-23 17:32:18 UTC
setting up rspec config for #<Pedant::OpenSourcePlatform:0x00000028f69c8>
Configuring RSpec for Open-Source Tests

[Progress Bar] Accuracy Over Tact [Progress Bar]

"Accuracy Over Tact"

== Testing Environment ==
Config File: /var/opt/chef-server/chef-pedant/etc/pedant_config.rb
```

The previous command will run tests from the `/opt/chef-server/embedded/service/chef-pedant/spec/api` directory.

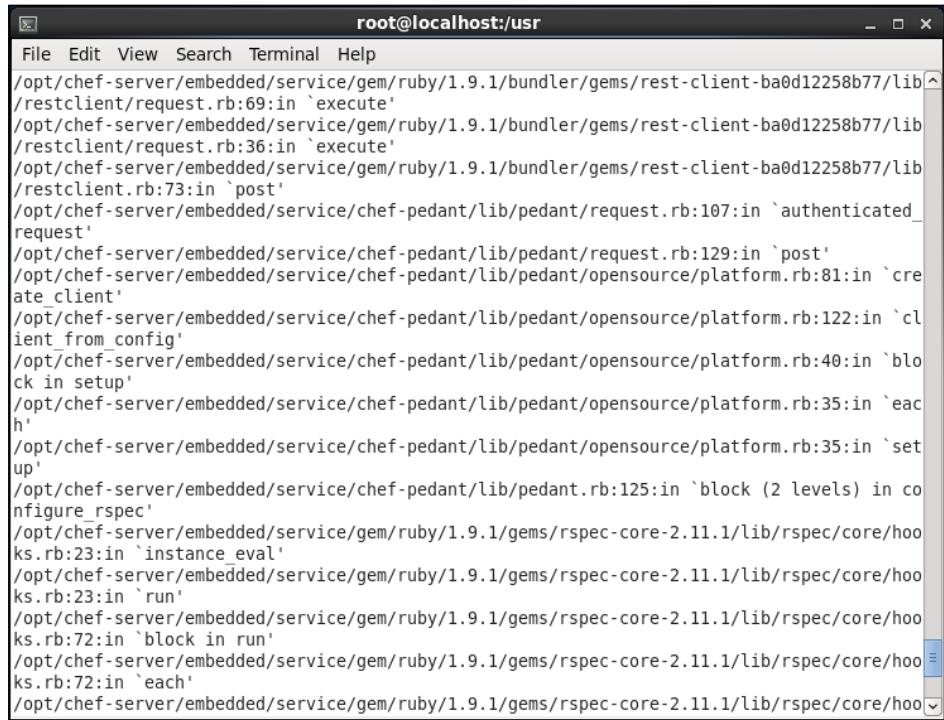


```
root@localhost:~# chef-server-ctl test
[Progress Bar]
"Accuracy Over Tact"

== Testing Environment ==
Config File: /var/opt/chef-server/chef-pedant/etc/pedant_config.rb
HTTP Traffic Log File: /var/log/chef-server/chef-pedant/http-traffic.log

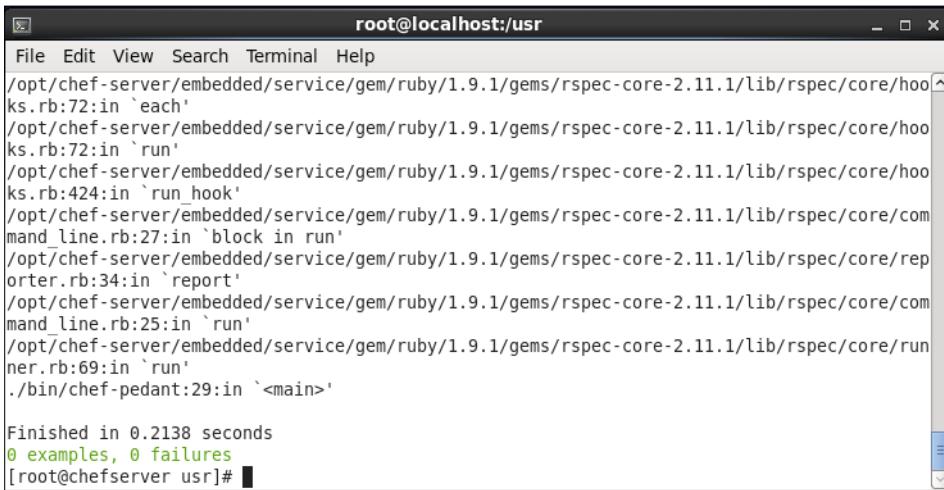
Running tests from the following directories:
/opt/chef-server/embedded/service/chef-pedant/spec/api
Run options:
  include {:focus=>true, :smoke=>true}
  exclude {:platform=>:multitenant, :cleanup=>true}
Creating client pedant_admin_client...
Exception during Pedant credentials setup
#<Errno::ECONNRESET: Connection reset by peer - SSL_connect>
/opt/chef-server/embedded/service/chef-pedant/lib/pedant/core_ext/net_http.rb:61:in `connect'
/opt/chef-server/embedded/service/chef-pedant/lib/pedant/core_ext/net_http.rb:61:in `block_in_connect'
/opt/chef-server/embedded/lib/ruby/1.9.1/timeout.rb:55:in `timeout'
/opt/chef-server/embedded/lib/ruby/1.9.1/timeout.rb:100:in `timeout'
/opt/chef-server/embedded/service/chef-pedant/lib/pedant/core_ext/net_http.rb:61:in `connect'
/opt/chef-server/embedded/lib/ruby/1.9.1/net/http.rb:756:in `do_start'
/opt/chef-server/embedded/lib/ruby/1.9.1/net/http.rb:745:in `start'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/bundler/gems/rest-client-ba0d12258b77/lib
/restclient/request.rb:183:in `transmit'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/bundler/gems/rest-client-ba0d12258b77/lib
```

It will run tests from the `/opt/chef-server/embedded/service/gem/` directories as well.



```
root@localhost:/usr
File Edit View Search Terminal Help
/opt/chef-server/embedded/service/gem/ruby/1.9.1/bundler/gems/rest-client-ba0d12258b77/lib
/restclient/request.rb:69:in `execute'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/bundler/gems/rest-client-ba0d12258b77/lib
/restclient/request.rb:36:in `execute'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/bundler/gems/rest-client-ba0d12258b77/lib
/restclient.rb:73:in `post'
/opt/chef-server/embedded/service/chef-pedant/lib/pedant/request.rb:107:in `authenticated_
request'
/opt/chef-server/embedded/service/chef-pedant/lib/pedant/request.rb:129:in `post'
/opt/chef-server/embedded/service/chef-pedant/lib/pedant/opensource/platform.rb:81:in `cre
ate_client'
/opt/chef-server/embedded/service/chef-pedant/lib/pedant/opensource/platform.rb:122:in `cl
ient_from_config'
/opt/chef-server/embedded/service/chef-pedant/lib/pedant/opensource/platform.rb:40:in `blo
ck in setup'
/opt/chef-server/embedded/service/chef-pedant/lib/pedant/opensource/platform.rb:35:in `eac
h'
/opt/chef-server/embedded/service/chef-pedant/lib/pedant/opensource/platform.rb:35:in `set
up'
/opt/chef-server/embedded/service/chef-pedant/lib/pedant.rb:125:in `block (2 levels) in co
nfigure_rspec'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/hoo
ks.rb:23:in `instance_eval'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/hoo
ks.rb:23:in `run'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/hoo
ks.rb:72:in `block in run'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/hoo
ks.rb:72:in `each'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/hoo
```

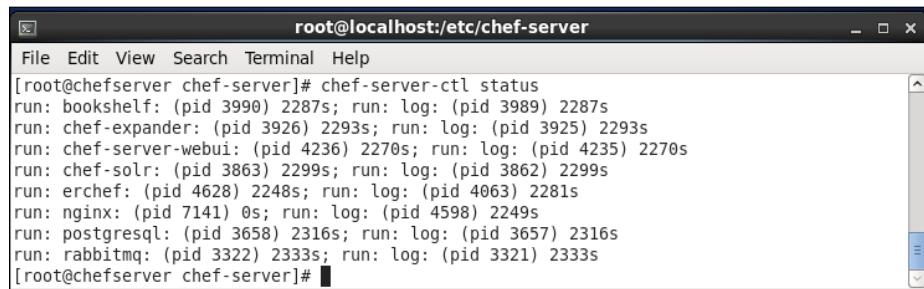
Check whether you get `0 failures` in your final output, as shown in the following screenshot:



```
root@localhost:/usr
File Edit View Search Terminal Help
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/hoo
ks.rb:72:in `each'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/hoo
ks.rb:72:in `run'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/hoo
ks.rb:424:in `run_hook'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/com
mand_line.rb:27:in `block in run'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/repor
ter.rb:34:in `report'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/com
mand_line.rb:25:in `run'
/opt/chef-server/embedded/service/gem/ruby/1.9.1/gems/rspec-core-2.11.1/lib/rspec/core/run
ner.rb:69:in `run'
./bin/chef-pedant:29:in `<main>'

Finished in 0.2138 seconds
0 examples, 0 failures
[root@chefserver usr]#
```

To verify the status of all the services of the Chef server, execute the `chef-server-ctl status` command, as shown in the following screenshot:



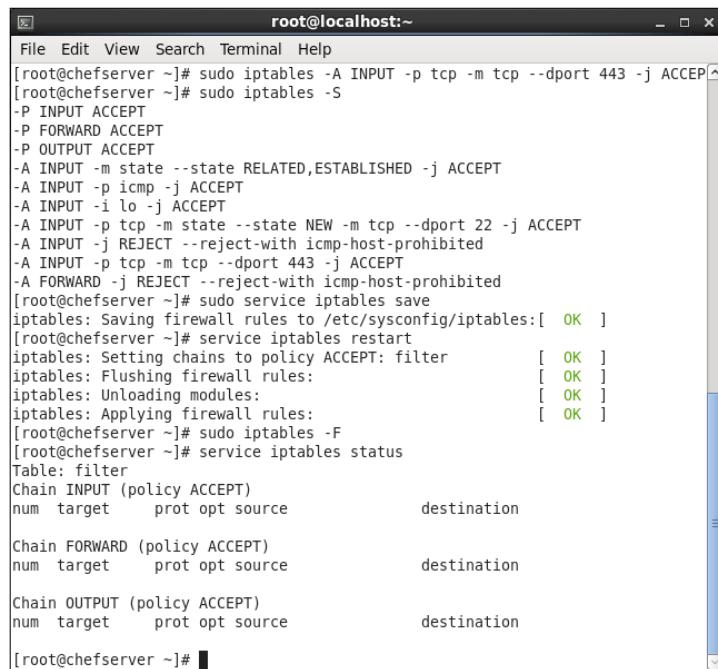
```
root@chefserver chef-server]# chef-server-ctl status
run: bookshelf: (pid 3990) 2287s; run: log: (pid 3989) 2287s
run: chef-expander: (pid 3926) 2293s; run: log: (pid 3925) 2293s
run: chef-server-webui: (pid 4236) 2270s; run: log: (pid 4235) 2270s
run: chef-solr: (pid 3863) 2299s; run: log: (pid 3862) 2299s
run: erchef: (pid 4628) 2248s; run: log: (pid 4063) 2281s
run: nginx: (pid 7141) 0s; run: log: (pid 4598) 2249s
run: postgresql: (pid 3658) 2316s; run: log: (pid 3657) 2316s
run: rabbitmq: (pid 3322) 2333s; run: log: (pid 3321) 2333s
[root@chefserver chef-server]#
```

Open the web browser and visit <https://chefserver.eg1.com>.

If you get a connection error, then run the following command:

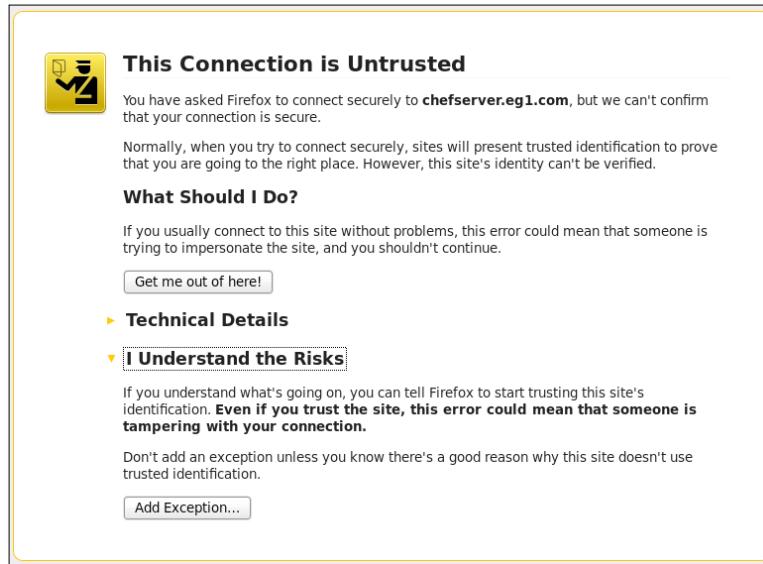
```
sudo iptables -A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
sudo iptables -S
sudo service iptables save
service iptables restart
```

If it still does not work, then run the `sudo iptables -F` command.



```
root@chefserver ~]# sudo iptables -A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
[root@chefserver ~]# sudo iptables -S
-A INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT
-A INPUT -j REJECT --reject-with icmp-host-prohibited
-A INPUT -p tcp -m tcp --dport 443 -j ACCEPT
-A FORWARD -j REJECT --reject-with icmp-host-prohibited
[root@chefserver ~]# sudo service iptables save
iptables: Saving firewall rules to /etc/sysconfig/iptables:[ OK ]
[root@chefserver ~]# service iptables restart
iptables: Setting chains to policy ACCEPT: filter          [ OK ]
iptables: Flushing firewall rules:                         [ OK ]
iptables: Unloading modules:                               [ OK ]
iptables: Applying firewall rules:                        [ OK ]
[root@chefserver ~]# sudo iptables -F
[root@chefserver ~]# service iptables status
Table: filter
Chain INPUT (policy ACCEPT)
num target     prot opt source               destination
Chain FORWARD (policy ACCEPT)
num target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
num target     prot opt source               destination
[root@chefserver ~]#
```

Open <https://chefserver.eg1.com>, click on **I understand the Risks** and then click on **Add Exception...**



Click on **Confirm Security Exception**.



## *Working with an Open Source Chef Server*

---

The Chef server web UI will be available for use after this. Enter the default **Username** and **Password**, as given on the page.

The screenshot shows a web browser window titled "Chef Server". The URL in the address bar is <https://chefserver.eg1.com/users/login>. The page has a blue header bar with the title "Chef Server". Below the header, there is a "Messages" section containing a message: "You don't have access to that, please login.". To the right of this message is a "Where do I get a Login?" section with instructions for creating new users. The main part of the page is a "Login" form. It includes fields for "Username" (containing "admin") and "Password" (containing "\*\*\*\*\*"). Below the password field is a "login" button. On the right side of the login form, there is a note: "Please change the default password immediately after logging in!". At the bottom of the page, there is a copyright notice: "Copyright © 2009-2015 Opscode".

Click on the **Login** button. Enter a new password, confirm it, and click on the **Regenerate Private Key** checkbox if you want to regenerate the key. Then, click on **Save User**.

The screenshot shows a "Edit user: admin" page from the Chef Server web UI. The top navigation bar includes links for Environments, Search, Status, Roles, Nodes, Cookbooks, Databases, and Clients. On the far right of the top bar are "Edit account" and "Logout admin (admin)" links. The main content area is titled "Edit user: admin" and shows a "List", "Create", "Show", "Edit", and "Delete" menu. Below this, there are two password input fields: "Password" (containing "\*\*\*\*\*") and "Password confirmation" (containing "\*\*\*\*\*"). A note below the first password field says: "New password for the User. Keep blank if you do not want to change password." A "Regenerate Private Key" checkbox is present, with a note below it stating: "Existing one will no longer work!". At the bottom of the form is a "Save User" button.

If the **Regenerate Private Key** checkbox is clicked, then it will generate a new private key, as given in the following screenshot:

The screenshot shows the Chef Server interface with the 'User: admin' page selected. At the top, there is a message: "Updated user admin. Please copy the following private key as the user's private key." Below this, the 'Public Key' section displays a long string of characters representing the generated public key.

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFADCAQ8AMIIIBcQKCADEA010/jH+YDl3lNORb0TR5
w0q1Xe0KtEkbj14o71504a4Ks37fETVhs/8CT06940998x+0PK9Q002Fy10U
FB4/kch0Ht1rvb2EMtUAJUR0a01NCF275kczjYSME1AG9mIEJMsA1afysT63ksY
0B29NB2T1nls1XkRqDeCK+dJb0mKkUX+pZL6m0w9WBACxxq91cafgn4jndu
e/An4nAe2P7PhqMr-d54CuTyhd721Lh-Kt810B36551e/P)F5J51c57ZB2y1g1
qvzJubHf1cqapT4XMHFnYFEIqbbU0CkQup3WngRbC3Hy/Mw50MKELFc/JsgdA
10ID4048
-----END PUBLIC KEY-----
```

The following figure shows the **Environments** tab after a fresh installation of the Chef server:

The screenshot shows the Chef Server interface with the 'Environments' tab selected. It displays a single environment named 'default'. The interface includes standard navigation links like 'List' and 'Create'.

## *Working with an Open Source Chef Server*

---

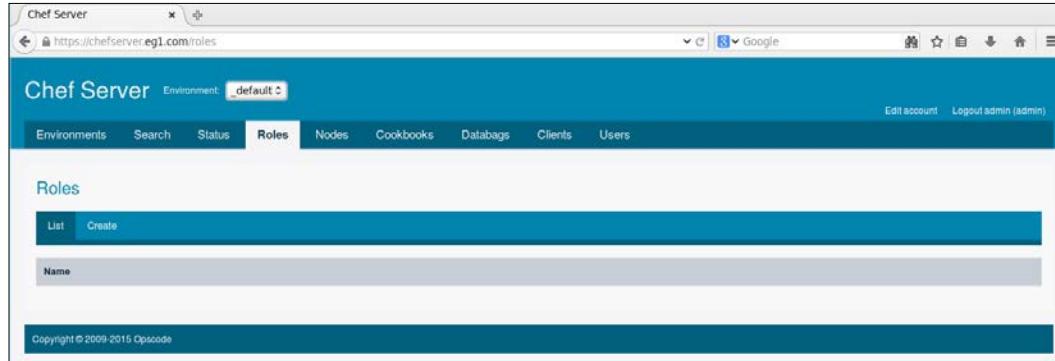
The following figure shows the **Search Indexes** tab after a fresh installation of the Chef server:

A screenshot of a web browser displaying the 'Search Indexes' page of a Chef Server. The URL in the address bar is <https://chefserver.eg1.com/search>. The page has a dark blue header with the text 'Chef Server' and 'Environment: default'. Below the header is a navigation bar with tabs: Environments, Search (which is selected), Status, Roles, Nodes, Cookbooks, Databags, Clients, and Users. On the right side of the header are links for 'Edit account' and 'Logout admin (admin)'. The main content area is titled 'Search Indexes' and contains six search input fields, each with a 'Query (i.e. attribute:value, leave empty to search all)' placeholder and a corresponding 'Search [entity]' button: 'Search client', 'Search environment', 'Search node', and 'Search role'. At the bottom of the page is a dark blue footer with the text 'Copyright © 2009-2015 Opscode'.

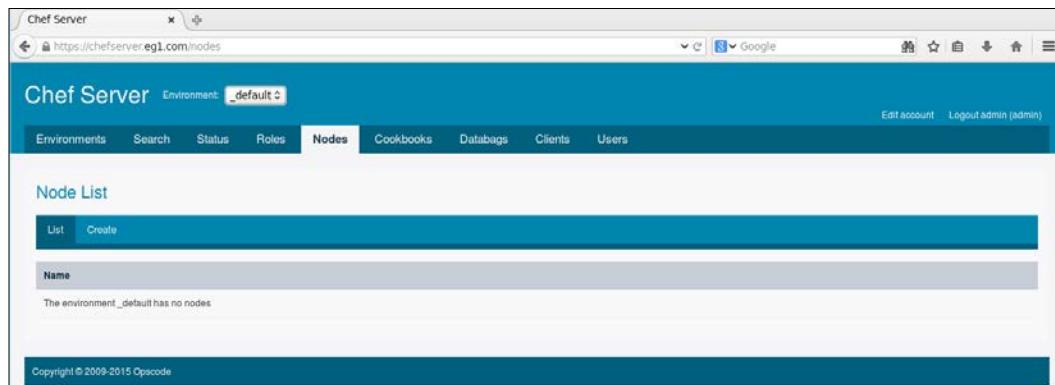
The following figure shows the **Status** tab after a fresh installation of the Chef server:

A screenshot of a web browser displaying the 'Status' page of a Chef Server. The URL in the address bar is <https://chefserver.eg1.com/status>. The page has a dark blue header with the text 'Chef Server' and 'Environment: default'. Below the header is a navigation bar with tabs: Environments, Search, Status (which is selected), Roles, Nodes, Cookbooks, Databags, Clients, and Users. On the right side of the header are links for 'Edit account' and 'Logout admin (admin)'. The main content area is titled 'Status' and displays a table with columns: Node Name, Platform, FQDN, IP Address, Uptime, Last Check-in, and Run List. A message at the top of the table says: 'You appear to have no nodes in the \_default environment - try connecting one, or creating or editing a [client](#)'. At the bottom of the page is a dark blue footer with the text 'Copyright © 2009-2015 Opscode'.

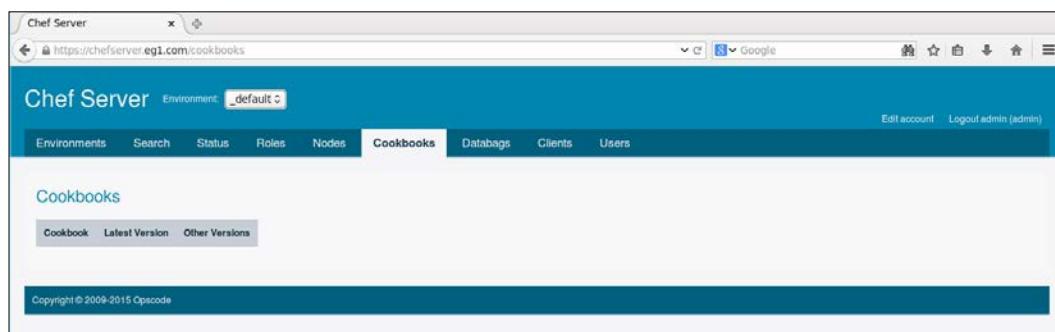
The following figure shows the **Roles** tab after a fresh installation of the Chef server:



The following figure shows the **Nodes List** tab after a fresh installation of the Chef server:



The following figure shows the **Cookbooks** tab after a fresh installation of the Chef server:

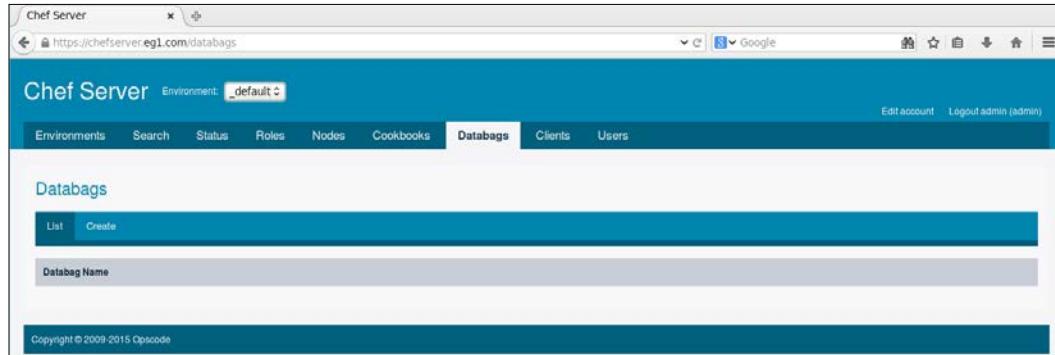


---

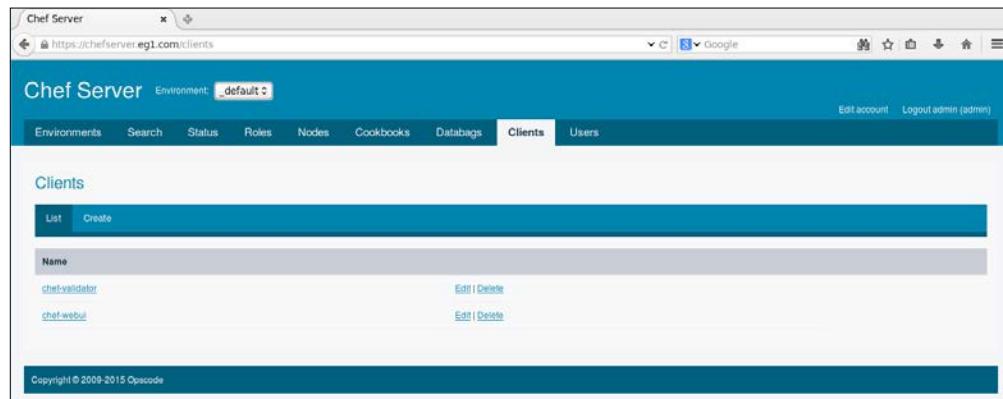
## *Working with an Open Source Chef Server*

---

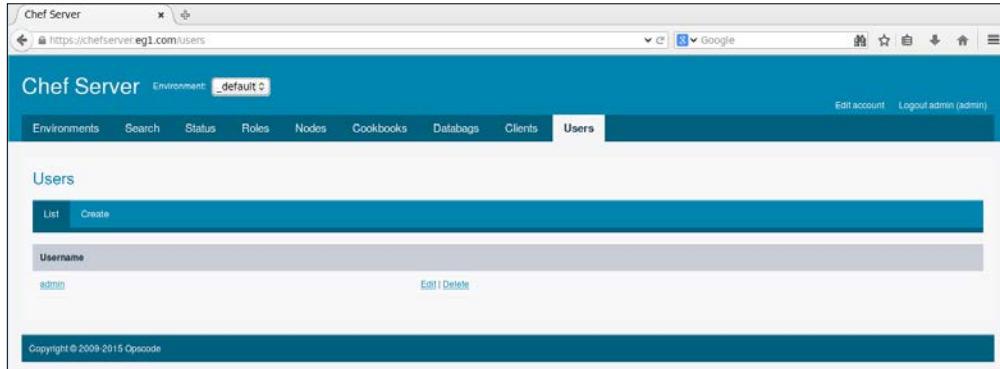
The following figure shows the **Databags** tab after a fresh installation of the Chef server:



The following figure shows the **Clients** tab after a fresh installation of the Chef server. Note the **Name** of the Clients:



The following figure shows the **Users** tab after a fresh installation of the Chef server:



## Installing an open source Chef server on Amazon Web Services ( AWS )

A step-by-step procedure is available in the official Chef documentation on how to set up the Chef server on an Amazon infrastructure considering high availability of Chef server.

## Setting up the workstation

A workstation has a chef-client installed and knife configured. It contains a local repository for the Chef server. The development work takes place at the workstation. Workstation interacts with the Chef server to upload artifacts from a local repository, to create cookbooks, roles, and environments, to install a Chef-client on nodes by knife bootstrapping, and to manage nodes.

## System requirements

Chef client installation has the following prerequisites:

System Architecture	OS	Version	Architecture
	(Version and supported Architecture are in sequence)		
	<b>AIX</b>	6.1	powerpc
	<b>Debian</b>	6, 7	i686 and x86_64 are supported for all versions
<b>Enterprise Linux</b>	5	i686 x86_64	
	6	i686, x86_64	
	7	x86_64	
	<b>FreeBSD</b>	9	(i686, amd64)
	<b>OS X</b>	10.6, 10.7, 10.8, 10.9, 10.10	Only x86_64 is supported for all versions
	<b>Suse Enterprise</b>	11.2	(i686, x86_64)
<b>Solaris</b>	5.9	sparc	
	5.10	i686, sparc	
	5.11	i686, sparc	
	<b>openSUSE</b>	12.1	(i686, x86_64)
	<b>Ubuntu</b>	10.04, 10.10, 11.04, 11.10, 12.04, 12.10, 13.04, 13.10, 14.04	i686 and x86_64 are supported for all versions
<b>Windows</b>	7	x86_64	
	8	x86_64	
	2003r2	i686, x86_64	
	2008	i686, x86_64	
	2008r2	x86_64	
	2012	x86_64	
	2012r2	x86_64	

<b>Ruby</b>	1.9.3 (or higher)	
<b>Firewall Configuration</b>	Proper configuration of network and firewall settings; Access to Chef server via HTTPS	
<b>Hardware Requirement for standalone deployment</b>	More details are available at <a href="https://docs.chef.io/chef_system_requirements.html">https://docs.chef.io/chef_system_requirements.html</a>	
RAM		512MB
Free Disk Space		200MB to /opt/chef to store chef-client binaries 5GB to /var/chef/cache; it stores downloaded cookbooks, packages required by those cookbooks, and other large files

Open <https://www.chef.io/download-chef-client/>.

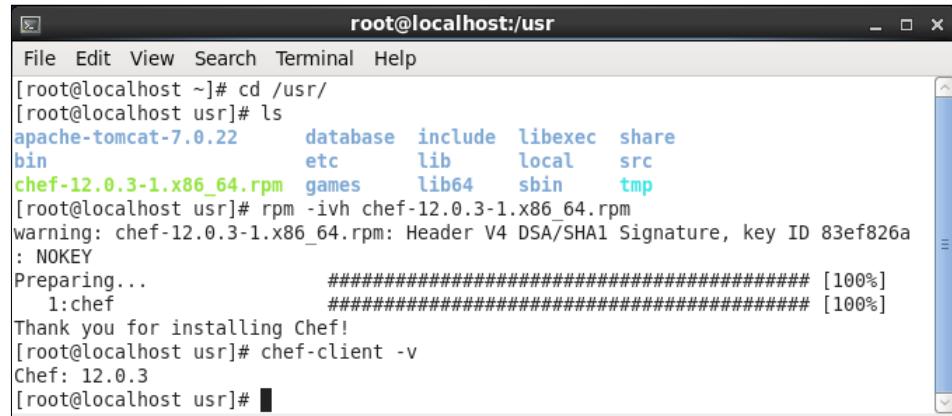
Select **Operating system, version, and architecture** based on your requirements, as shown in the following screenshot:

Select **version** of the Chef-client. If you click on the link below the box, the link will get downloaded:

The screenshot shows the 'Chef Client' download page. At the top, there are navigation links: 'WHAT IS CHEF?', 'LEARN CHEF', 'RESOURCES', and a prominent orange 'GET CHEF' button. Below these, the title 'Chef Client' is displayed. A descriptive text explains that the Chef client and server work together to manage nodes using policies (recipes). To the left, there's a note about supported systems: 'Select the kind of system you would like to install the Chef Client on. The versions listed have been tested and are supported.' Three dropdown menus are highlighted with blue boxes: 'Enterprise Linux' (set to '6'), 'Version' (set to '12.0.3-1'), and 'Architecture' (set to 'x86\_64'). To the right, under 'Quick Installation Instructions', there's a command-line instruction: 'curl -L https://www.chef.io/chef/install.sh | bash'. Below this, the 'Downloads' section is shown, with a note: 'You can install manually by downloading the package below after you have selected a Chef version. For more information about manual installation, [please read the documentation](#)'. A final dropdown menu for 'Architecture' is also highlighted.

Copy **chef-12.0.3-1.x86\_64.rpm** to the CentOS-based virtual machine.

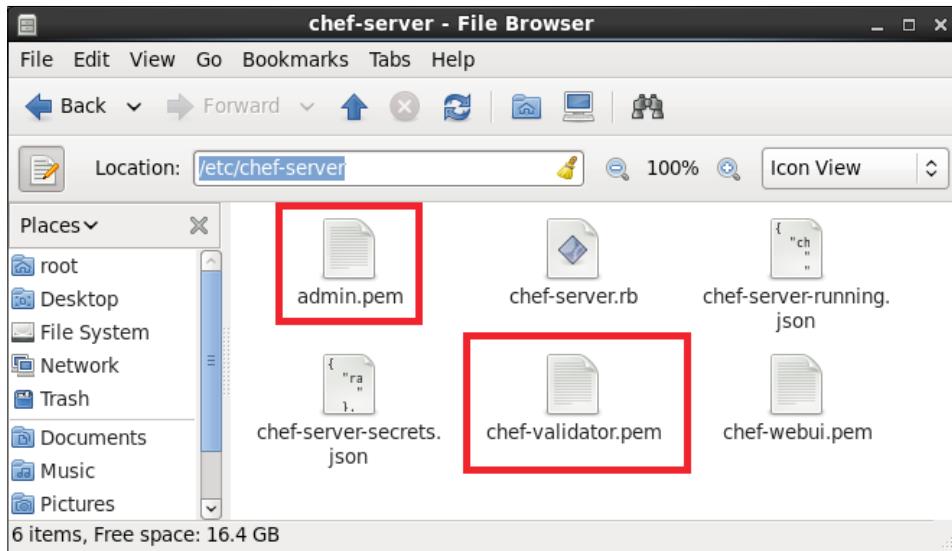
Now, we will install the Chef-client and check whether it is installed properly or not.



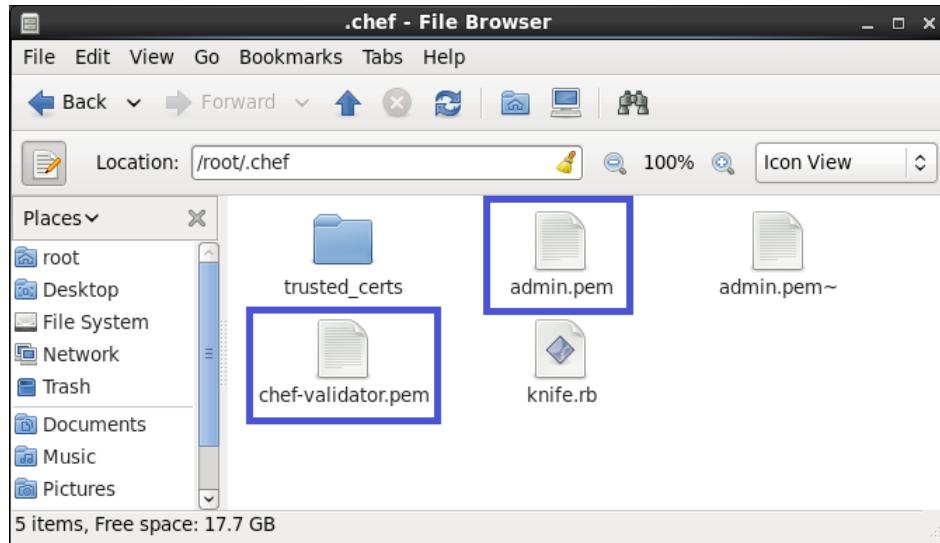
The screenshot shows a terminal window titled "root@localhost:/usr". The command "cd /usr/" is run, followed by "ls" which lists directory contents including "apache-tomcat-7.0.22", "bin", "chef-12.0.3-1.x86\_64.rpm", "etc", "lib", "local", "sbin", and "tmp". The command "rpm -ivh chef-12.0.3-1.x86\_64.rpm" is then run, which installs the Chef client. A warning message about a missing key ID is shown, followed by progress bars for preparing and installing the package. Finally, the command "chef-client -v" is run to verify the installation, showing the version "12.0.3".

```
[root@localhost ~]# cd /usr/
[root@localhost usr]# ls
apache-tomcat-7.0.22      database  include  libexec  share
bin                         etc       lib      local   src
chef-12.0.3-1.x86_64.rpm   games     lib64    sbin    tmp
[root@localhost usr]# rpm -ivh chef-12.0.3-1.x86_64.rpm
warning: chef-12.0.3-1.x86_64.rpm: Header V4 DSA/SHA1 Signature, key ID 83ef826a
: NOKEY
Preparing...           ##### [100%]
1:chef                  ##### [100%]
Thank you for installing Chef!
[root@localhost usr]# chef-client -v
Chef: 12.0.3
[root@localhost usr]#
```

The workstation needs keys and configuration files to connect and authenticate with the Chef server. Open **Chef-server**, go to `etc/chef-server`, and copy `admin.pem` and `chef-validator.pem`.



Put `admin.pem` and `chef-validator.pem` in the `/root/.chef` folder.



Check whether the Chef server is accessible from the workstation with a ping command and execute the command `knife configure`.

The following screenshot shows the content from `knife.rb`, in our example:

```

log_level :info
log_location STDOUT
node_name 'admin'
client_key '/root/.chef/admin.pem'
validation_client_name 'chef-validator'
validation_key '/root/.chef/chef-validator.pem'
chef_server_url 'https://chefserver.eg1.com:443'
syntax_check_cache_path '/root/.chef/syntax_check_cache'

```

Run `sudo iptables -F` on the Chef server if you get the following:

**ERROR: Errno::EHOSTUNREACH: No route to host - connect(2) for "chefserver.eg1.com" port 443**

We may get the following error:

**ERROR: SSL Validation failure connecting to host: chefserver.egl.com - SSL\_connect returned=1 errno=0 state=SSLv3 read server certificate B: certificate verify failed**

**ERROR: Could not establish a secure connection to the server.**

```
root@localhost:~# ping chefserver.egl.com
PING chefserver.egl.com (192.168.139.132) 56(84) bytes of data.
64 bytes from chefserver.egl.com (192.168.139.132): icmp_seq=1 ttl=64 time=0.405 ms
64 bytes from chefserver.egl.com (192.168.139.132): icmp_seq=2 ttl=64 time=0.343 ms
64 bytes from chefserver.egl.com (192.168.139.132): icmp_seq=3 ttl=64 time=0.455 ms
64 bytes from chefserver.egl.com (192.168.139.132): icmp_seq=4 ttl=64 time=0.728 ms
64 bytes from chefserver.egl.com (192.168.139.132): icmp_seq=5 ttl=64 time=0.391 ms
64 bytes from chefserver.egl.com (192.168.139.132): icmp_seq=6 ttl=64 time=0.662 ms
64 bytes from chefserver.egl.com (192.168.139.132): icmp_seq=7 ttl=64 time=0.735 ms
^C
--- chefserver.egl.com ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6358ms
rtt min/avg/max/mdev = 0.343/0.531/0.735/0.158 ms
[root@localhost ~]# knife client list
ERROR: Errno::EHOSTUNREACH: No route to host - connect(2) for "chefserver.egl.com" port 443
[root@localhost ~]# knife client list
ERROR: SSL Validation failure connecting to host: chefserver.egl.com - SSL_connect returned=1 errno=0 state=SSLv3 read server certificate B: certificate verify failed
ERROR: Could not establish a secure connection to the server.
Use `knife ssl check` to troubleshoot your SSL configuration.
If your Chef Server uses a self-signed certificate, you can use
`knife ssl fetch` to make knife trust the server's certificates.

Original Exception: OpenSSL::SSL::SSLError: SSL_connect returned=1 errno=0 state=SSLv3 r
ead server certificate B: certificate verify failed
[root@localhost ~]#
```

Run the `knife ssl check` on the workstation virtual machine.

```
root@localhost:~# knife ssl fetch
WARNING: Certificates from chefserver.egl.com will be fetched and placed in your trusted cert directory (/root/.chef/trusted_certs).

Knife has no means to verify these are the correct certificates. You should verify the authenticity of these certificates after downloading.

Adding certificate for chefserver.egl.com in /root/.chef/trusted_certs/chefserver_egl_com.crt
[root@localhost ~]#
```

Run the `knife configure` command again.

```
root@localhost:~# knife configure
Overwrite /root/.chef/knife.rb? (Y/N) Y
Please enter the chef server URL: [https://localhost:443] https://chefserver.eg1.com:443
Please enter an existing username or clientname for the API: [root] admin
Please enter the validation clientname: [chef-validator]
Please enter the location of the validation key: [/etc/chef-server/chef-validator.pem] /root
/.chef/chef-validator.pem
Please enter the path to a chef repository (or leave blank):
*****
You must place your client key in:
/root/.chef/admin.pem
Before running commands with Knife!
*****
You must place your validation key in:
/root/.chef/chef-validator.pem
Before generating instance data with Knife!
*****
Configuration file written to /root/.chef/knife.rb
[root@localhost ~]#
```

Now, run the command `knife client command list` and compare it with the list of clients we noted down from the Chef server web UI's **client** tab.

```
root@localhost:~# knife client list
chef-validator
chef-webui
```

Verify the `knife` commands available for use by using the following command:

`knife -help`

Parameter	Description
<code>-s, --server-url</code> URL	Chef server URL
<code>--chef-zero-host</code> HOST	Host to start chef-zero on
<code>--chef-zero-port</code> PORT	Port (or port range) to start chef-zero on. Port ranges like 1000,1010 or 8889-9999 will try all given ports until one works

[root@localhost ~]# knife -help

ERROR: You need to pass a sub-command (e.g., `knife SUB-COMMAND`)

Usage: `knife sub-command (options)`

<code>-k, --key KEY</code>	API client key
<code>--[no-] color</code>	Use colored output, defaults to false on Windows, true otherwise
<code>-c, --config CONFIG</code>	The configuration file to use
<code>--defaults</code>	Accept default values for all questions
<code>-d, --disable-editing</code>	Do not open editor, just accept the data as it is
<code>-e, --editor EDITOR</code>	Set the editor to use for interactive commands
<code>-E, --environment ENVIRONMENT</code>	Set the Chef environment (except for in searches, where this will be flagrantly ignored)
<code>-F, --format FORMAT</code>	Which format to use for output
<code>-z, --local-mode</code>	Point knife commands at local repository instead of server
<code>-u, --user USER</code>	API client username
<code>--print-after</code>	Show the data after a destructive operation
<code>-V, --verbose</code>	More verbose output. Use twice for max verbosity
<code>-v, --version</code>	Show Chef version
<code>-y, --yes</code>	Say yes to all prompts for confirmation
<code>-h, --help</code>	Show this message
<b>Available subcommands:</b> (for details, <code>knife SUB-COMMAND --help</code> )	
<b>** USER COMMANDS **</b>	
<code>knife user create USER (options)</code>	
<code>knife user delete USER (options)</code>	
<code>knife user edit USER (options)</code>	
<code>knife user list (options)</code>	
<code>knife user reregister USER (options)</code>	
<code>knife user show USER (options)</code>	
[root@localhost ~]#	

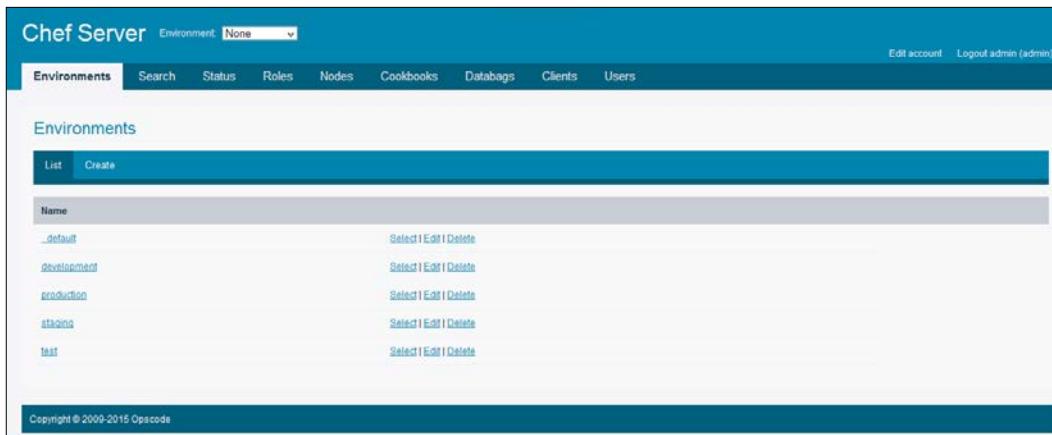
Create different environments with the use of the knife command.

If you get `RuntimeError: Please set EDITOR environment variable`, add `knife[:editor] = "/usr/bin/vim"` in the `knife.rb` file.



```
root@localhost:~#
File Edit View Search Terminal Help
[root@localhost ~]# knife environment create development
ERROR: RuntimeError: Please set EDITOR environment variable
[root@localhost ~]# knife environment create development
Created development
[root@localhost ~]# knife environment create test
Created test
[root@localhost ~]# knife environment create staging
Created staging
[root@localhost ~]# knife environment create production
Created production
[root@localhost ~]#
```

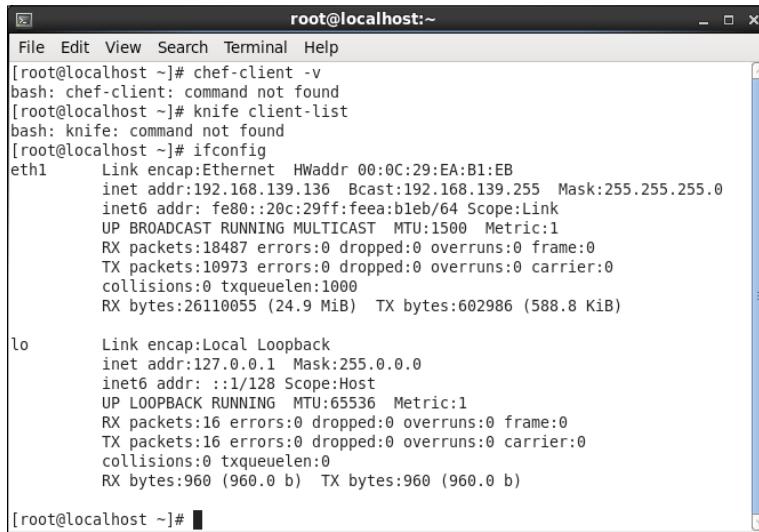
Verify the Chef server web UI.



The screenshot shows the Chef Server web interface. The top navigation bar includes "Chef Server", "Environment: None", "Edit account", and "Logout admin (admin)". Below the navigation is a horizontal menu with tabs: Environments, Search, Status, Roles, Nodes, Cookbooks, Databags, Clients, and Users. The "Environments" tab is selected. A sub-menu below it has "List" and "Create" buttons, with "List" currently active. The main content area displays a table titled "Environments" with columns "Name" and actions "Select | Edit | Delete". The environments listed are: default, development, production, staging, and test. At the bottom of the page is a footer with the text "Copyright © 2009-2015 Opscode".

## Bootstrapping a node

Let's Bootstrap a node. Check whether the Chef-client is already installed on a node or not.



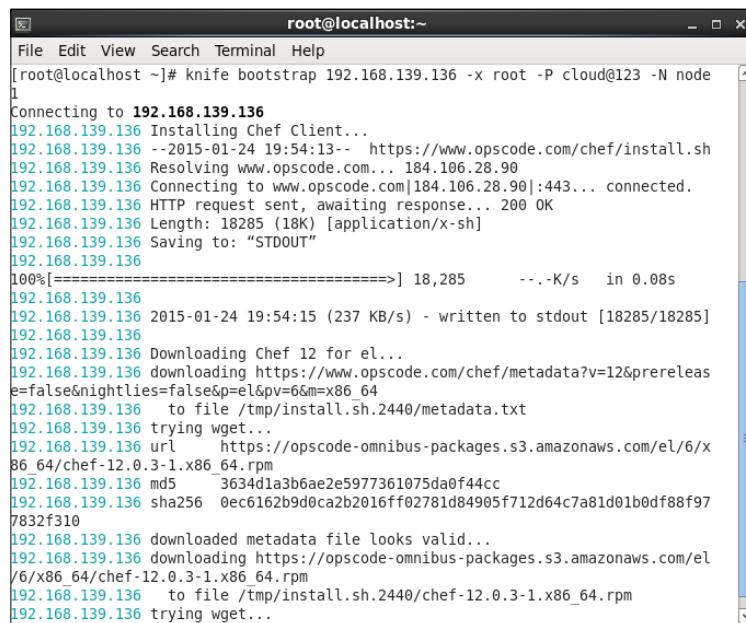
A terminal window titled "root@localhost:~" showing the output of the "ifconfig" command. The output shows two interfaces: eth1 (Ethernet) and lo (Loopback). The eth1 interface has an IP address of 192.168.139.136 and a broadcast address of 192.168.139.255. The lo interface has an IP address of 127.0.0.1. Both interfaces are up and running.

```
[root@localhost ~]# chef-client -v
bash: chef-client: command not found
[root@localhost ~]# knife client-list
bash: knife: command not found
[root@localhost ~]# ifconfig
eth1      Link encap:Ethernet HWaddr 00:0C:29:EA:B1:EB
          inet addr:192.168.139.136 Bcast:192.168.139.255 Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:feaa:bleb/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:18487 errors:0 dropped:0 overruns:0 frame:0
            TX packets:10973 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:26110055 (24.9 MiB) TX bytes:602986 (588.8 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING MTU:65536 Metric:1
            RX packets:16 errors:0 dropped:0 overruns:0 frame:0
            TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:0
            RX bytes:960 (960.0 b) TX bytes:960 (960.0 b)

[root@localhost ~]#
```

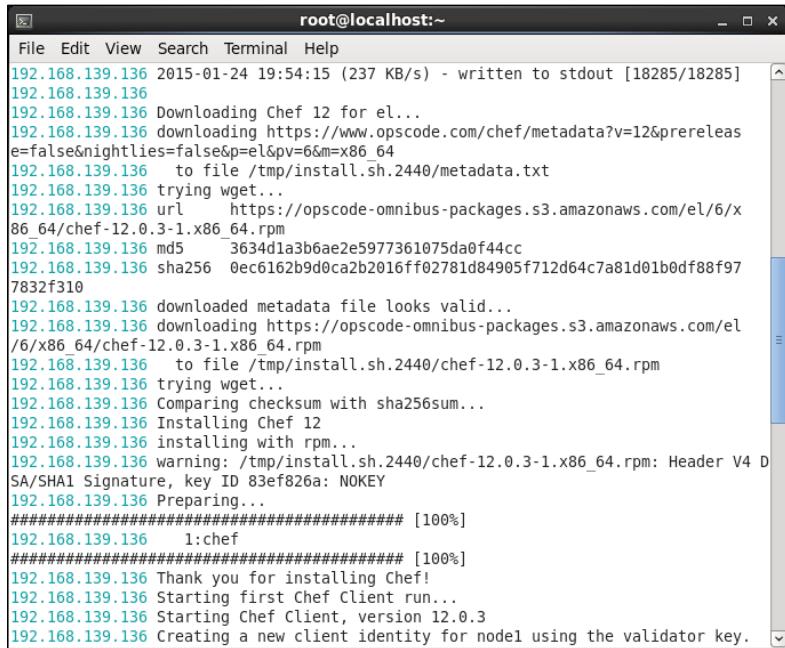
Try other knife commands. Open the workstation and run the bootstrap command on the node that has the 192.168.139.136 IP address.



A terminal window titled "root@localhost:~" showing the output of the "knife bootstrap" command. The command is run with arguments: 192.168.139.136, -x root, -P cloud@123, -N node1. The output shows the process of connecting to the node, installing the Chef Client, downloading the Chef 12 package, and saving it to STDOUT. The progress bar indicates the download is at 100%.

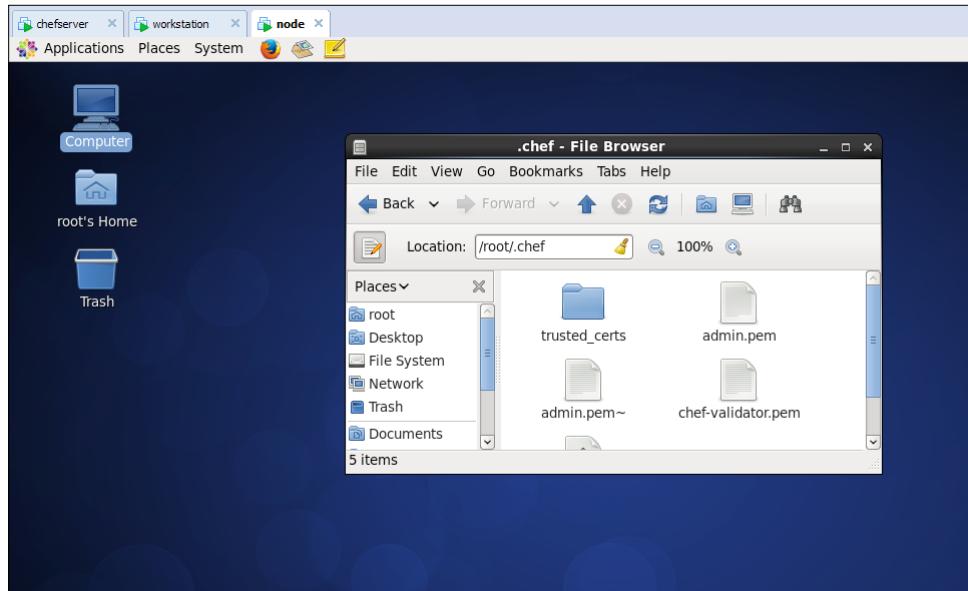
```
[root@localhost ~]# knife bootstrap 192.168.139.136 -x root -P cloud@123 -N node1
Connecting to 192.168.139.136
192.168.139.136 Installing Chef Client...
192.168.139.136 --2015-01-24 19:54:13-- https://www.opscode.com/chef/install.sh
192.168.139.136 Resolving www.opscode.com... 184.106.28.90
192.168.139.136 Connecting to www.opscode.com|184.106.28.90|:443... connected.
192.168.139.136 HTTP request sent, awaiting response... 200 OK
192.168.139.136 Length: 18285 (18K) [application/x-sh]
192.168.139.136 Saving to: "STDOUT"
192.168.139.136
100%[=====] 18,285  --.-K/s   in 0.08s
192.168.139.136
192.168.139.136 2015-01-24 19:54:15 (237 KB/s) - written to stdout [18285/18285]
192.168.139.136
192.168.139.136 Downloading Chef 12 for el...
192.168.139.136 downloading https://www.opscode.com/chef/metadata?v=12&prerelease=false&nightlies=false&p=el&p=v=x86_64
192.168.139.136   to file /tmp/install.sh.2440/metadata.txt
192.168.139.136 trying wget...
192.168.139.136 url   https://opscode-omnibus-packages.s3.amazonaws.com/el/6/x86_64/chef-12.0.3-1.x86_64.rpm
192.168.139.136 md5   3634d1a3b6ae2e5977361075da0f44cc
192.168.139.136 sha256 0ec6162b9d0ca2b2016ff02781d84905f712d64c7a81d01b0df88f977832f310
192.168.139.136 downloaded metadata file looks valid...
192.168.139.136 downloading https://opscode-omnibus-packages.s3.amazonaws.com/el/6/x86_64/chef-12.0.3-1.x86_64.rpm
192.168.139.136   to file /tmp/install.sh.2440/chef-12.0.3-1.x86_64.rpm
192.168.139.136 trying wget...
```

It will download the Chef-client and install it on the node that has the 192.168.139.136 IP address.



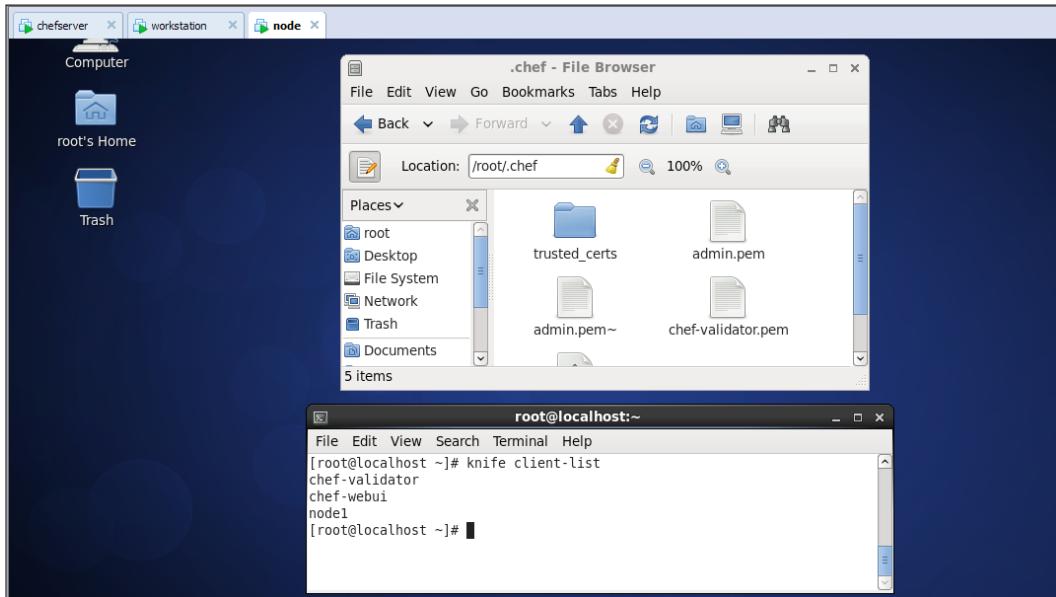
```
root@localhost:~#
File Edit View Search Terminal Help
192.168.139.136 2015-01-24 19:54:15 (237 KB/s) - written to stdout [18285/18285]
192.168.139.136
192.168.139.136 Downloading Chef 12 for el...
192.168.139.136 downloading https://www.opscode.com/chef/metadata?v=12&prerelease=false&nightlies=false&p=el&pv=6&m=x86_64
192.168.139.136   to file /tmp/install.sh.2440/metadata.txt
192.168.139.136 trying wget...
192.168.139.136 url   https://opscode-omnibus-packages.s3.amazonaws.com/el/6/x86_64/chef-12.0.3-1.x86_64.rpm
192.168.139.136 md5   3634d1a3b6ae2e5977361075da0f44cc
192.168.139.136 sha256 0ec6162b9d0ca2b2016ff02781d84905f712d64c7a81d01b0df88f977832f310
192.168.139.136 downloaded metadata file looks valid...
192.168.139.136 downloading https://opscode-omnibus-packages.s3.amazonaws.com/el/6/x86_64/chef-12.0.3-1.x86_64.rpm
192.168.139.136   to file /tmp/install.sh.2440/chef-12.0.3-1.x86_64.rpm
192.168.139.136 trying wget...
192.168.139.136 Comparing checksum with sha256sum...
192.168.139.136 Installing Chef 12
192.168.139.136 installing with rpm...
192.168.139.136 warning: /tmp/install.sh.2440/chef-12.0.3-1.x86_64.rpm: Header V4 DSA/SHA1 Signature, key ID 83ef826a: NOKEY
192.168.139.136 Preparing...
#####
1:chef
#####
Thank you for installing Chef!
Starting first Chef Client run...
Starting Chef Client, version 12.0.3
Creating a new client identity for node1 using the validator key.
```

Copy the /root/.chef folder from the workstation to the node.



## Working with an Open Source Chef Server

Verify the Chef installation and configuration with the `knife client-list` command.



Verify the Chef server web UI.

## The Nodes tab

The **Nodes** tab provides list of nodes. Following is the screenshot:

A screenshot of the Chef Server web interface. The URL in the address bar is https://chefserver.eg1.com/nodes. The page has a blue header with the text "Chef Server" and "Environment: None". It features a navigation menu with tabs for Environments, Search, Status, Roles, Nodes (which is currently selected), Cookbooks, Databags, Clients, and Users. The main content area is titled "Node List" and contains two tabs: "List" (which is selected) and "Create". Below these tabs is a table with a single row. The table has a header row labeled "Name" and a data row containing the entry "node1". To the right of "node1" are links for "Edit | Delete". At the bottom of the page, there's a footer bar with the text "Copyright © 2009-2015 Opscode".

## The Clients tab

The Clients tab provide list of clients registered with Chef server. Following is the screenshot:

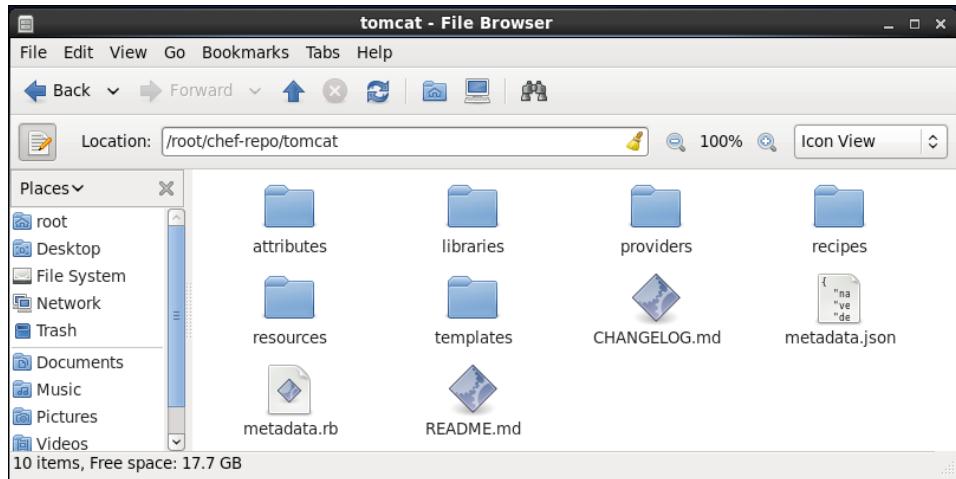
The screenshot shows a web browser window for the Chef Server at <https://chefserver.eg1.com/clients>. The page has a blue header bar with the title "Chef Server" and an "Environment: None" dropdown. On the right side of the header are links for "Edit account" and "Logout admin (admin)". Below the header is a navigation menu with tabs: Environments, Search, Status, Roles, Nodes, Cookbooks, Databags, Clients (which is highlighted in blue), and Users. The main content area is titled "Clients" and contains a sub-menu with "List" and "Create" buttons. The "List" button is selected. Below this is a table with a single column labeled "Name". Three entries are listed: "chef-validator", "chef-webui", and "node1", each with "Edit | Delete" links next to them. At the bottom of the page is a footer bar with the text "Copyright © 2009-2015 Opscode".

## Using community cookbooks

Download the **Tomcat** cookbook with the `knife` command. More details are available at <https://github.com/opscode-cookbooks/tomcat>.

The screenshot shows a terminal window titled "root@localhost:~/chef-repo" with the command `knife cookbook site download tomcat` being run. The output of the command is displayed in the terminal window, showing the download of the Tomcat cookbook from the opscode repository. In the background, there is a file browser window titled "chef-repo - File Browser" showing a directory structure with a folder named "tomcat" and a file named "tomcat-0.17.0.tar.gz". The file browser also shows a sidebar with various places like root, Desktop, and Downloads.

Extract the content of the **Tomcat** cookbook.

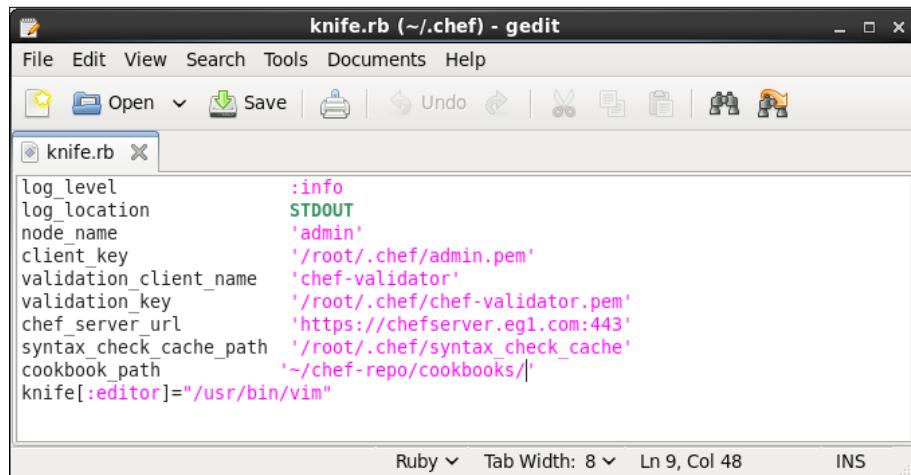


Try to upload Tomcat cookbook on the Chef server with the `knife cookbook upload tomcat` command

If you get an error, as shown in the following figure, then add `cookbook_path` in the `knife.rb` file:

```
[root@localhost cookbooks]# knife cookbook upload tomcat
ERROR: Could not find cookbook tomcat in your cookbook path, skipping it
ERROR: Failed to upload 1 cookbook.
```

The following screenshot shows the example of the `knife.rb` file:



Try to upload the Tomcat cookbook again. A Tomcat cookbook depends on two other cookbooks and hence, we need to get and upload Java and OpenSSL before uploading a Tomcat cookbook. OpenSSL cookbook also depends on the Chef Sugar cookbook, hence we need to get and upload it as well. Once we upload all these dependent cookbooks, we will be able to upload the Tomcat cookbook to the Chef server.

```
File Edit View Search Terminal Help
[root@localhost cookbooks]# knife cookbook upload tomcat
Uploading tomcat [0.17.0]
ERROR: Cookbook tomcat depends on cookbooks which are not currently
ERROR: being uploaded and cannot be found on the server.
ERROR: The missing cookbook(s) are: 'java' version '>= 0.0.0', 'openssl' version
'>= 0.0.0'
[root@localhost cookbooks]# knife cookbook site download java
Downloading java from the cookbooks site at version 1.29.0 to /root/chef-repo/cookbo
oks/java-1.29.0.tar.gz
Cookbook saved: /root/chef-repo/cookbooks/java-1.29.0.tar.gz
[root@localhost cookbooks]# knife cookbook site download openssl
Downloading openssl from the cookbooks site at version 2.0.2 to /root/chef-repo/cook
books/openssl-2.0.2.tar.gz
ERROR: Error connecting to https://supermarket.chef.io/api/v1/cookbooks/openssl/vers
ions/2.0.2/download, retry 1/5
Cookbook saved: /root/chef-repo/cookbooks/openssl-2.0.2.tar.gz
[root@localhost cookbooks]# knife cookbook upload java
Uploading java [1.29.0]
Uploaded 1 cookbook.
[root@localhost cookbooks]# knife cookbook upload openssl
Uploading openssl [2.0.2]
ERROR: Cookbook openssl depends on cookbooks which are not currently
ERROR: being uploaded and cannot be found on the server.
ERROR: The missing cookbook(s) are: 'chef-sugar' version '>= 0.0.0'
[root@localhost cookbooks]# knife cookbook site download chef-sugar
Downloading chef-sugar from the cookbooks site at version 2.5.0 to /root/chef-repo/c
ookbooks/chef-sugar-2.5.0.tar.gz
Cookbook saved: /root/chef-repo/cookbooks/chef-sugar-2.5.0.tar.gz
[root@localhost cookbooks]# knife cookbook upload chef-sugar
Uploading chef-sugar [2.5.0]
Uploaded 1 cookbook.
[root@localhost cookbooks]# knife cookbook upload openssl
Uploading openssl [2.0.2]
Uploaded 1 cookbook.
[root@localhost cookbooks]# knife cookbook upload tomcat
Uploading tomcat [0.17.0]
Uploaded 1 cookbook.
[root@localhost cookbooks]#
```

Verify the Chef server web UI, as shown in the following screenshot:

The screenshot shows the Chef Server web interface. At the top, there's a header bar with the title "Chef Server" and a dropdown menu labeled "Environment: default". Below the header is a navigation bar with tabs: Environments, Search, Status, Roles, Nodes, Cookbooks (which is currently selected), Databags, Clients, and Users. The main content area is titled "Cookbooks". Under this, there's a table with three columns: "Cookbook", "Latest Version", and "Other Versions". The table lists four cookbooks: "chef-sugar" (version 2.5.0), "java" (version 1.29.0), "openssl" (version 2.0.2), and "tomcat" (version 0.17.0). At the bottom of the page, there's a footer bar with the text "Copyright © 2009-2015 Opscode".

Add the Tomcat recipe to the run-list of a node.

The screenshot shows a terminal window with the title "root@localhost:~/cookbooks". The window has a standard Linux-style menu bar with options like File, Edit, View, Search, Terminal, and Help. The main pane of the terminal shows the following command being run:

```
File Edit View Search Terminal Help
[root@localhost cookbooks]# knife node run_list add node1 'recipe[tomcat]'
node1:
  run_list: recipe[tomcat]
[root@localhost cookbooks]#
```

Verify the run-list of a node in Chef server dashboard as shown in the following screenshot:

The screenshot shows the Chef Server dashboard with the 'Nodes' tab selected. The main content area displays the details for 'Node node1'. The 'Run List' section shows a single item: 'tomcat' at position 0. The 'Recipes' section indicates that 'tomcat' is the only recipe applied to this node.

Position	Name	Version
0	tomcat	

Position	Name
0	tomcat

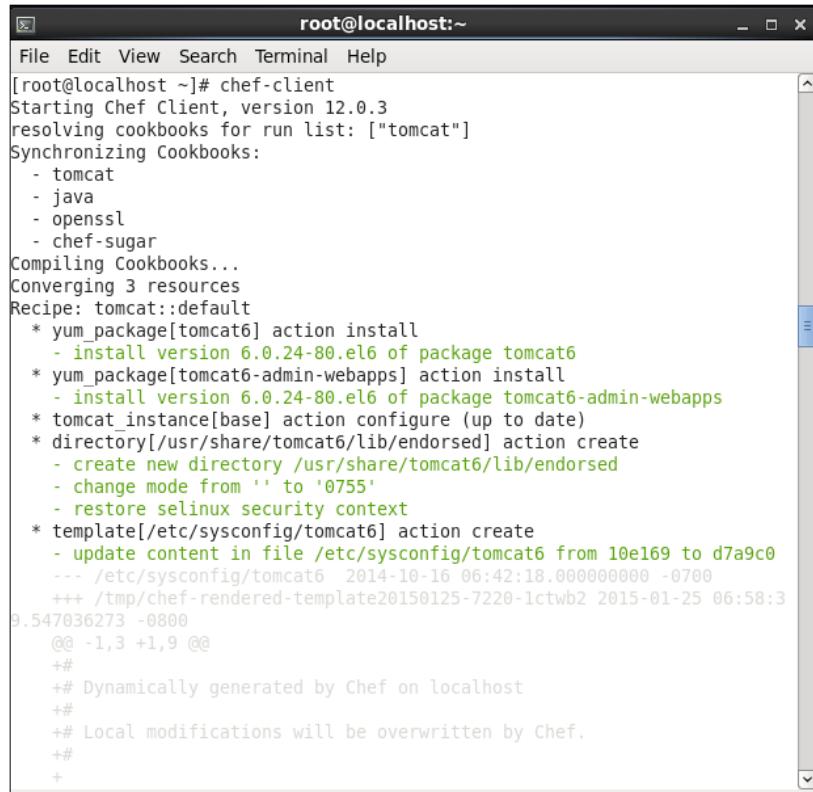
Verify the **Status** tab of a node, as shown in the following screenshot:

The screenshot shows the Chef Server dashboard with the 'Status' tab selected. The main content area displays the status for 'Node node1'. The table shows the following information:

Node Name	Platform	FQDN	IP Address	Uptime	Last Check-in
<a href="#">node1</a>	centos 6.6	localhost	192.168.139.136	<a href="#">8 hours</a>	<a href="#">37 minutes</a>

Copyright © 2009-2015 Opscode

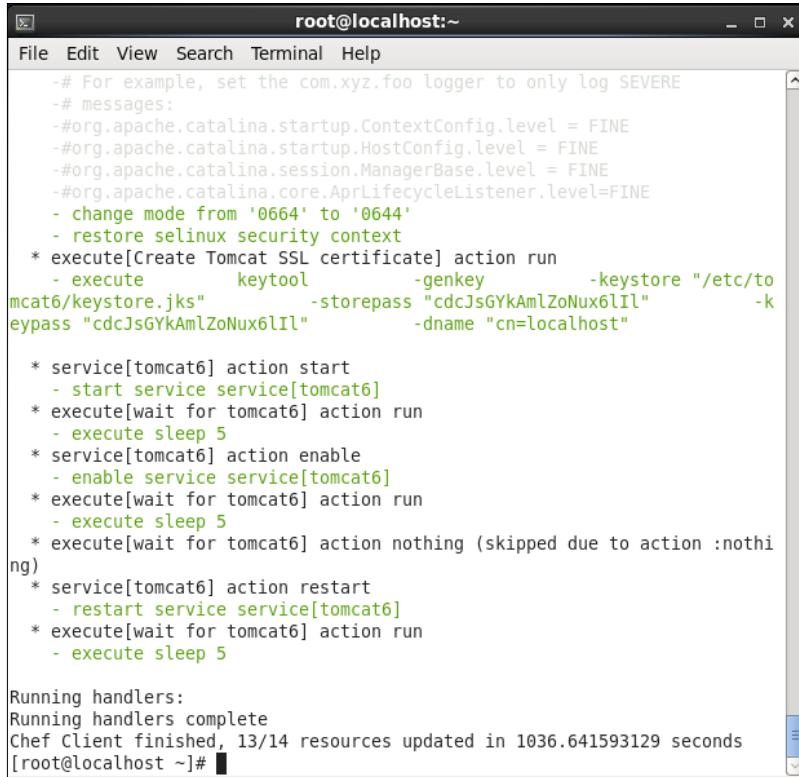
Now, open the **Terminal** tab in the node's virtual machine, and run the `chef-client` command.



The screenshot shows a terminal window titled "root@localhost:~". The window displays the output of the `chef-client` command. The output shows the client starting, resolving cookbooks for a run list containing "tomcat", synchronizing cookbooks (including tomcat, java, openssl, and chef-sugar), compiling cookbooks, and converging 3 resources. The resources listed include actions like `install` for yum\_packages (tomcat6 and tomcat6-admin-webapps) and `configure` for tomcat instances. It also shows the creation of a directory at `/usr/share/tomcat6/lib/endorsed`, changing its mode to 0755, and restoring SELinux security context. A template resource for `/etc/sysconfig/tomcat6` is updated, showing a timestamp from 2014-10-16 to 2015-01-25. The terminal ends with a note about local modifications being overwritten by Chef.

```
[root@localhost ~]# chef-client
Starting Chef Client, version 12.0.3
resolving cookbooks for run list: ["tomcat"]
Synchronizing Cookbooks:
  - tomcat
  - java
  - openssl
  - chef-sugar
Compiling Cookbooks...
Converging 3 resources
Recipe: tomcat::default
* yum_package[tomcat6] action install
  - install version 6.0.24-80.el6 of package tomcat6
* yum_package[tomcat6-admin-webapps] action install
  - install version 6.0.24-80.el6 of package tomcat6-admin-webapps
* tomcat instance[base] action configure (up to date)
* directory[/usr/share/tomcat6/lib/endorsed] action create
  - create new directory /usr/share/tomcat6/lib/endorsed
  - change mode from '' to '0755'
  - restore selinux security context
* template[/etc/sysconfig/tomcat6] action create
  - update content in file /etc/sysconfig/tomcat6 from 10e169 to d7a9c0
    --- /etc/sysconfig/tomcat6 2014-10-16 06:42:18.000000000 -0700
    +++ /tmp/chef-rendered-template20150125-7220-1ctwb2 2015-01-25 06:58:3
9.547036273 -0800
@@ -1,3 +1,9 @@
+#
+## Dynamically generated by Chef on localhost
+#
+## Local modifications will be overwritten by Chef.
+#
+
```

Installation process will take some time.

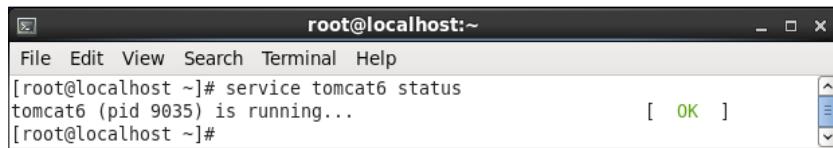


```
# For example, set the com.xyz.foo logger to only log SEVERE
# messages:
-#org.apache.catalina.startup.ContextConfig.level = FINE
-#org.apache.catalina.startup.HostConfig.level = FINE
-#org.apache.catalina.session.ManagerBase.level = FINE
-#org.apache.catalina.core.AprLifecycleListener.level=FINE
- change mode from '0664' to '0644'
- restore selinux security context
* execute[Create Tomcat SSL certificate] action run
- execute      keytool -genkey -keystore "/etc/tomcat6/keystore.jks" -storepass "cdcJsGYkAmlZoNux6l1L" -keypass "cdcJsGYkAmlZoNux6l1L" -dname "cn=localhost"

* service[tomcat6] action start
- start service service[tomcat6]
* execute[wait for tomcat6] action run
- execute sleep 5
* service[tomcat6] action enable
- enable service service[tomcat6]
* execute[wait for tomcat6] action run
- execute sleep 5
* execute[wait for tomcat6] action nothing (skipped due to action :nothing)
* service[tomcat6] action restart
- restart service service[tomcat6]
* execute[wait for tomcat6] action run
- execute sleep 5

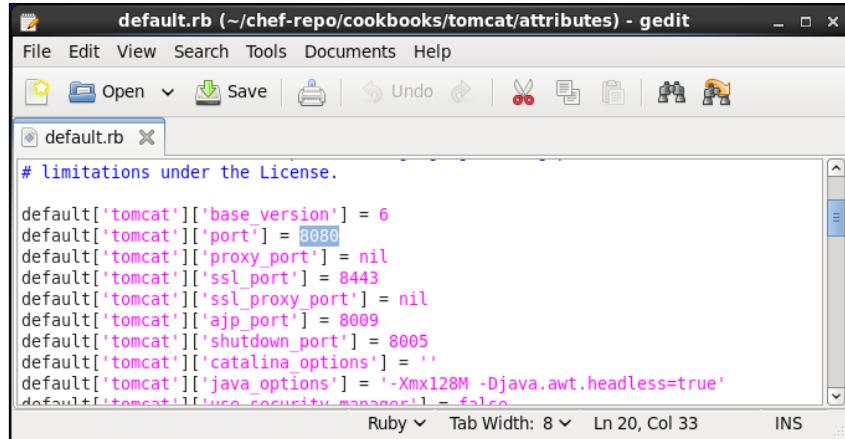
Running handlers:
Running handlers complete
Chef Client finished, 13/14 resources updated in 1036.641593129 seconds
[root@localhost ~]#
```

Let's verify the status of the Tomcat service.



```
[root@localhost ~]# service tomcat6 status
tomcat6 (pid 9035) is running...
[ OK ]
```

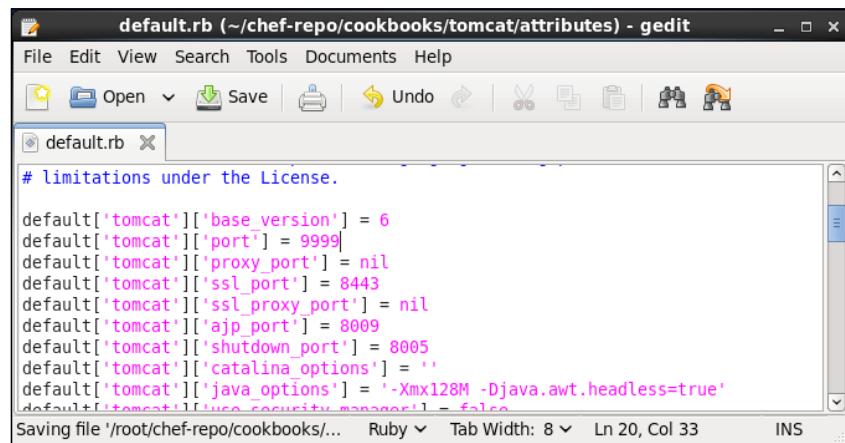
Let's change the default port available in `attributes/default.rb`.



```
# limitations under the License.

default['tomcat']['base_version'] = 6
default['tomcat']['port'] = 8080
default['tomcat']['proxy_port'] = nil
default['tomcat']['ssl_port'] = 8443
default['tomcat']['ssl_proxy_port'] = nil
default['tomcat']['ajp_port'] = 8009
default['tomcat']['shutdown_port'] = 8005
default['tomcat']['catalina_options'] = ''
default['tomcat']['java_options'] = '-Xmx128M -Djava.awt.headless=true'
default['tomcat']['use_security_manager'] = false
```

Change the default `['tomcat']['port'] = 8080` to `9999` in `default.rb` file as shown in the following screenshot:



```
# limitations under the License.

default['tomcat']['base_version'] = 6
default['tomcat']['port'] = 9999
default['tomcat']['proxy_port'] = nil
default['tomcat']['ssl_port'] = 8443
default['tomcat']['ssl_proxy_port'] = nil
default['tomcat']['ajp_port'] = 8009
default['tomcat']['shutdown_port'] = 8005
default['tomcat']['catalina_options'] = ''
default['tomcat']['java_options'] = '-Xmx128M -Djava.awt.headless=true'
default['tomcat']['use_security_manager'] = false
```

Save the file and upload the cookbook again to the Chef server and verify the change in the web UI.

```

Chef Server Environment: _default ▾

Environments Search Status Roles Nodes Cookbooks Databases Clients Users

tomcat [0.17.0 ▾]

attributes

default.rb
#
# Cookbook Name:: tomcat
# Attributes:: default
#
# Copyright 2010, Opscode, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

default['tomcat']['base_version'] = 6
default['tomcat']['port'] = 8080
default['tomcat']['proxy_port'] = nil
default['tomcat']['ssl_port'] = 8443
default['tomcat']['ssl_proxy_port'] = nil
default['tomcat']['ajp_port'] = 8009
default['tomcat']['shutdown_port'] = 8005
default['tomcat']['catalina_options'] =
default['tomcat']['java_options'] = '-Xmx128M -Djava.awt.headless=true'
default['tomcat']['use_security_manager'] = false
default['tomcat']['authbind'] = 'no'
default['tomcat']['deploy_manager_apps'] = true
default['tomcat']['max_threads'] = nil
default['tomcat']['ssl_max_threads'] = 150
default['tomcat']['ssl_cert_file'] = nil
default['tomcat']['ssl_key_file'] = nil

```

Run the `chef-client` command from the terminal of the node's virtual machine. Run-list will execute again and the port number for Tomcat will be changed.

```
File Edit View Search Terminal Help
+     Define a non-SSL HTTP/1.1 Connector on port 9999
-->
- <Connector port="8080" protocol="HTTP/1.1"
+ <Connector port="9999" protocol="HTTP/1.1"
    connectionTimeout= 20000
    URIEncoding="UTF-8"
    redirectPort="8443"
@@ -79,7 +79,7 @@
    <!-- A "Connector" using the shared thread pool-->
    <!--
        <Connector executor="tomcatThreadPool"
-         port="8080" protocol="HTTP/1.1"
+         port="9999" protocol="HTTP/1.1"
            connectionTimeout= 20000
            redirectPort="8443" />
    -->
    - restore selinux security context
* template[/etc/tomcat6/logging.properties] action create (up to date)
* execute[Create Tomcat SSL certificate] action run (up to date)
* service[tomcat6] action start (up to date)
* service[tomcat6] action enable (up to date)
* execute[wait for tomcat6] action nothing (skipped due to action :nothing)
g)
* service[tomcat6] action restart
- restart service service[tomcat6]
* execute[wait for tomcat6] action run
- execute sleep 5

Running handlers:
Running handlers complete
Chef Client finished, 3/12 resources updated in 50.258479933 seconds
[root@localhost ~]#
```

## Upgradation of the open source Chef server

Chef updates its feature and services frequently in order to make our automation process up to date. We require to upgrade our current system with the latest one.

Version upgradation is simpler when it comes to upgradation of the Chef server 0.10.x to Chef server 11.x. For this to be done, install the Chef server and then transfer all the data from the older server to the new one. Data can't be moved directly.

Some knife subcommands have to be used for the upgradation, as the database for Chef server 0.10.x is CouchDB and the database for Chef server 11.x is PostgreSQL. The knife subcommands are:

```
Knife download  
Knife list  
Knife upload
```

These are used to download the data from the Chef server 0.10.x and upload on the Chef server 11.x. This approach solves the data storage problem.

Now, we need to install the latest version of the knife-essentials plugin:

```
$ gem install knife-essentials
```

## Existing requirements

You should have the following requirements for the upgradation:

- A running Chef server 0.10.x
- A live Chef server 11.x
- Knife-enabled workstations, which can access both the Chef servers (0.10 x and 11.x)
- To check whether the download of knife and the upload knife works efficiently or not

## Accessing the Chef server 0.10.x

In order to communicate with the Chef server 0.10.x, the following configuration is required to be done on the workstation.

Make a directory, so that it can be referred to as the location where the data will be downloaded.

We will use the `~/transfer` directory name.

Now, in the `transfer` directory, we need to create a new file, such as `chef/knife-chef10.rb`.

The contents of the newly created file should be as follows:

```
transfer_repo=File.expand_path('..',File.dirname(__FILE__))  
chef_server_url "http://chef-10.example.com:4000"
```

```
node_name 'chef-webui'
client_key "#{transfer_repo}/.chef/chef-webui.pem"
repo_mode 'everything'
versioned_cookbooks true
chef_repo_path transfer_repo
```

Here, `chef_server_url` will be replaced by the URL of the Chef server 0.10.x, which is used to download the data.

The `node_name` parameter will be replaced by the name of the workstation.

Now, we need to copy the private key from the `etc/chef/webui.pem` location to the `.chef` directory of the Chef server 0.10.x by using the following command:

```
$ cp <local_webui.pem> .chef/chef-webui.pem
```

By running the following command, you can verify the configuration:

```
$ knife list /clients
```

## Downloading data from the Chef server 0.10.x

We need to download the data from the Chef server 0.10.x using the following command:

```
$ knife download -c .chef/knife-chef10.rb /
```

This command will move all the data to the `transfer` directory of the Chef server 0.10.x.

## Accessing the Chef server 11.x

In order to communicate with the Chef server 11.x, the following configurations need to be performed on the workstation.

Now, using the `~/transfer` directory, which we have created, we need to create a new file named `.chef/knife.rb`.

The contents of the newly created file should be as follows:

```
transfer_repo=File.expand_path('..', File.dirname(__FILE__))
chef_server_url "https://chef-11.example.com"
node_name 'admin'
client_key "#{transfer_repo}/.chef/admin.pem"
```

```
repo_mode'everything'  
versioned_cookbookstrue  
chef_repo_pathtransfer_repo
```

Here, `chef_server_url` will be replaced by the URL of the Chef server 11.x, which is used to upload the data.

Now, we need to copy the private key from the `admin.pem` location to the `.chef` directory of the Chef server 11.x by using the following command:

```
cp<admin.pem> .chef/admin.pem
```

By running the following command, you can verify that the configuration is done:

```
$ knife list /users
```

## Updating Chef-validator settings

Now, we don't require `chef-validator`. Chef server 11.x needs the `chef-validator` flag, which should be set properly for creating `chef-validator`. Now, go to the `~/transfer` directory and Open the file named as `clients/chef-validator.json` and just add the entry of "validator": true

```
{  
  
  "name": "chef-validator",  
  "public_key": "-----BEGIN PUBLIC KEY-----\  
  nMIIBIjANBgkqHJkijuyhNMJK89OCAQ8AM235gKCgAQEA8l0+sy05G6YX/SaVsU2k\  
  ndwOTIZKLhvfuhp/ghyt34As456HJfghjukiITD+mgUqkF4ox/zIwhLG5nyHMLa\  
  nFKsKPxUQ1S1Jsf2gaoP+RhnswmSPJffhF21593DwSsg1TLNtDw5cqhF6YYo7b7cB\  
  nywHaWL+O3cSFLd0UShjuki897ghnfgTyhiji876Hiki89783Ottn83V8BUCfpnbi\  
  nNetytGDnE1Ms91vYswsW2EqEnzQ+afvlDq5tXu72b1XBs7Y/8JqQz8+31VHNGKys\  
  nh5U6VdI5Br0ulle00LcffgrgE4@#$fs7/THjijknmhjkhngt1a+3siu3HAa8lslo\  
  noQIDAQAB\n-----END PUBLIC KEY-----\n",  
  "_rev": "1-72a9f16a92108bd794704c075261aeb5",  
  "validator": true  
  
}
```

Now, by running the following command, you can verify the configuration:

```
$ knife list /clients
```

## Verifying the admin public key

There will be a key to access the Chef server. That key is called the admin public key. This private admin key must be correct for every workstation, which can access the Chef server 11.x. The Chef server 11.x has admin as a new username. Many of the instances of the Chef server 0.10.x have got the name admin as their admin client.

For Chef 11.x, `admin.pem` (private key) is required by knife. If the client does not match the name of the private key, these mismatching in names can be a matter of issue.

Replace the Chef 11.x admin private key with the old private key or copy `admin.pem` for every workstation. To do this, the following command should be run:

```
$ knife download users/admin.json  
$ grep public_key clients/admin.json
```

The final step's results will be available in the latest content of `users/admin.json`:

```
$ knife upload users/admin.json  
$ cp <Chef 10 admin.pem> .chef/admin.pem
```

Remove the following command:

```
$ rm clients/admin.json
```

Verification of the configuration can be done by running the following command:

```
$ knife list /users
```

## Verification of user passwords

While using Knife download or Knife upload subcommands, if the user-hashed passwords are not transferred to or from the server system. Then, after using these commands for upgradation to a newer version of the open source Chef server ,every user should run the following command:

```
$ knife user edit user_name
```

The following should be added in the JSON data:

```
"password": "password_value"
```

## **Uploading data to the Chef server 11.x**

We need to upload the data to the Chef server 11.x by using the following command:

```
$ knife upload /
```

Now, all of the data will be transferred in the `transfer` directory of the Chef server 11.x.

## **The last steps**

At this point, the Chef server 11.x should have all of the data that used to be on the Chef server 0.10.x. At this point, go to DNS or point to the load balancer at the upgraded server. The Chef-client should run appropriately on every node and every workstation should be able to manage objects on the server using Knife. If the issues remain, try the IRC channel or e-mail the discussion alias to `chef@lists.opscode.com`. If knife-essentials still have the issue, then file an issue in GitHub or check the IRC channel.

## **Self-test questions**

1. What are the system requirements for an open source Chef server installation?
2. How to configure a hostname? and why?
3. How to check whether the Chef server is working fine or not?
4. What is the default username and password for the Chef server web UI?
5. What are the system requirements for a Chef-client installation?
6. Which keys and configuration files does a workstation require to connect to and authenticate with the Chef server?
7. What is bootstrapping and how to bootstrap a node?

## **Summary**

We learned the step-by-step installation process of an open source Chef server and its installation on a virtual machine. We also learned about VMware's initial settings and the configuration to install an open source Chef server. An infrastructure automation administrator can learn to upgrade an open source Chef server from a lower version to the latest version.

In the next chapter, we are going to learn about the Private Chef-server, its benefits, various types of Private Chef installations, and their prerequisites.



# 7

## Working with the On-premises Chef Server Setup

*"The science of today is the technology of tomorrow."*

- Edward Teller

In this chapter, we will learn about the on-premises Chef server in detail. It describes the detailed necessity of on-premises Private Chef deployment, benefits, different types of on-premises Chef server installations and management. We will see the various types of installations in a step-by-step manner.

In this chapter, we will cover the following topics:

- Enterprise Chef or on-premises Chef
- The benefits of on-premises Chef
- Types of on-premises Chef installations
- Prerequisites for a standalone Private Chef installation
- Installation of standalone Private Chef
- Prerequisites for a tiered Private Chef installation
- Installation of a tiered Private Chef
- Prerequisites for a high-availability Chef installation
- Installation of a high-availability Private Chef
- Managing Private Chef

## The on-premises Chef server

In *Chapter 2, Different Components of Chef's Anatomy*, we discussed a variety of Chef server installations, in which, the on-premises Chef server installation is one of the most important.

As per the official blog of Chef posted on August 19, 2013 at <https://www.chef.io/blog/2013/08/19/introducing-enterprise-chef/>, Private Chef and Hosted Chef were combined and renamed as Enterprise Chef, which offered both as on-premise software and as a hosted service of Chef. This move was strategic to align with the Fortune 1000 businesses that represent the majority of Opscode's total sales.

In on-premises Chef, different types of services work together to provide on-premises automated solutions. On-premises Chef works like a Private Cloud; meaning, we can manage and automate our infrastructure within our firewall and have a flexibility of choice.

## Benefits of on-premises Chef

As we can customize on-premises Chef to suit our organizational usages or demand/requirements, there are some key benefits of on-premises Chef server as follows:

### Simple to scale

On-premises Chef enables the feature of rapid scalability of automation and configuration in on-demand cloud instances. We can easily manage thousands of instances using on-premises Chef. It has the capability to manage all instances simultaneously.

### Completely automotive solution

Using on-premises Chef, we can create a clone of any environment, such as clones of testing, production, or development environments and customize our deployment and configuration-related changes according to the environment's needs.

## **Fast and easy configuration management**

Since you are managing this on-premises Chef server within your organization, your data is inside your firewall and it gives you speed and promptness during configuration management.

## **Reduced complexity within infrastructure**

Using on-premises Chef, we can search for data within recipes themselves, such as searching for a memcached server and MySQL servers. The required data is indexed and is quite easy to search within on-premises Chef.

## **Improved data encryption policies**

Since on-premises Chef is centrally controlled, we can customize data encryption according to a particular environment. It provides full control to the system administrator or manager who can see what type of data is present at what time.

## **Types of on-premises Chef installations**

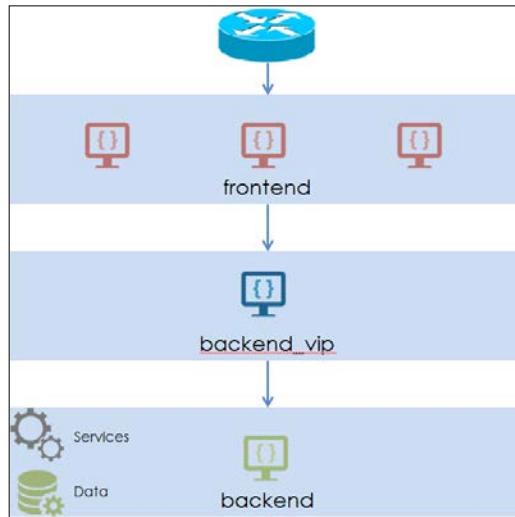
There are three flavors of on-premises Chef installations. Depending on our business requirements, we can select any one of them. We can set a on-premises Chef server in the following ways:

### **Standalone on-premises Chef**

When on-premises Chef is installed only on a single machine, it is known as standalone on-premises Chef. It is also called a solo installation of a on-premises Chef server. As the name suggests, it works only on a single server and this single server could be any physical machine, virtual machine, or cloud instance. This type of on-premises Chef installation is mostly used for testing, learning, or **proof of concept (PoC)**.

## Tiered on-premises Chef

When on-premises Chef is installed on many machines, it is known as tiered on-premises Chef. It is also called the tiered installation of the Chef server.



Using the tiered on-premises Chef installation, we can install on-premises Chef on many machines and scale up resources easily. Although it does not guarantee high availability of data and services, but with tiered installation, it is quite possible to take a quick back up from the server and then restore the desired state of configuration.

It is worth noticing that all the servers that are going to be a part of the Chef installation must form a cluster first, so that they can be viewed as a on-premises Chef cluster.

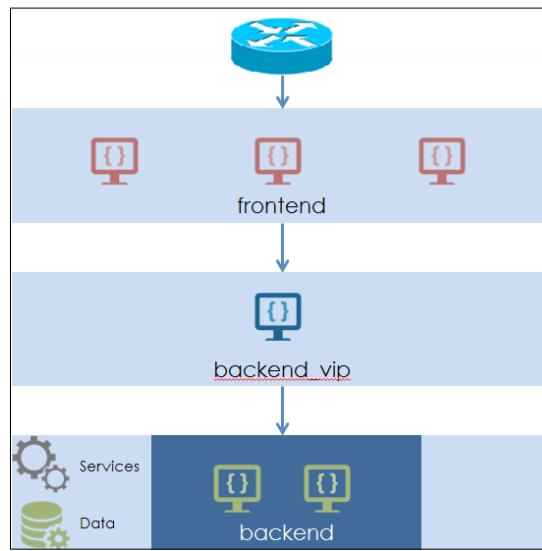
In the tiered Chef installation topology, we have many frontend servers and one backend server. Frontend servers communicate with the backend server through an API's request. This provides a high level of concurrency, but for commercial usage. We should go for the high availability on-premises Chef installation, because it is not possible to horizontally scale up the backend server if the I/O load increases. Here, only the vertical scaling of the backend server is possible; meaning, we can increase the RAM and Processor capacity only when the I/O load is excessive.

## High-availability on-premises Chef

When the Chef server is installed with one extra data storage layer to provide greater accessibility of data on many different servers, it is known as high availability for on-premises Chef installation.

High availability Chef is used for most of the commercial businesses and high demand production systems. Using the high availability version, we can install Chef on multiple servers and can horizontally scale up on both backend and frontend servers at any time.

Before going for the on-premises Chef installation, there are some prerequisites that must be fulfilled in the particular system where on-premises Chef is going to be installed.



Once the on-premises Chef installation is done, we need to create users and organizations in Hosted Chef dashboard.

## Downloading the installation package

To install Enterprise Chef, go to <http://downloads.chef.io/enterprise-chef/>.

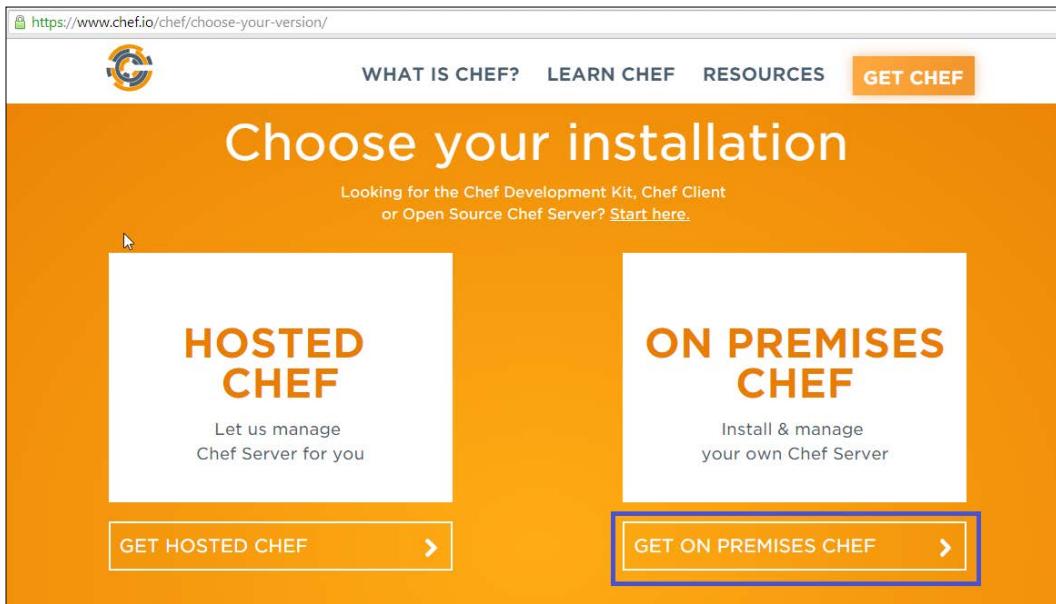
It has the following message:

*"This is the download page for Enterprise Chef 11. You probably only need to be here if you are already running Enterprise Chef 11 and need to download a critical security update. The current version of Chef server is Chef server 12."*

For a standalone installation, we will use **Chef server 12**.

## *Working with the On-premises Chef Server Setup*

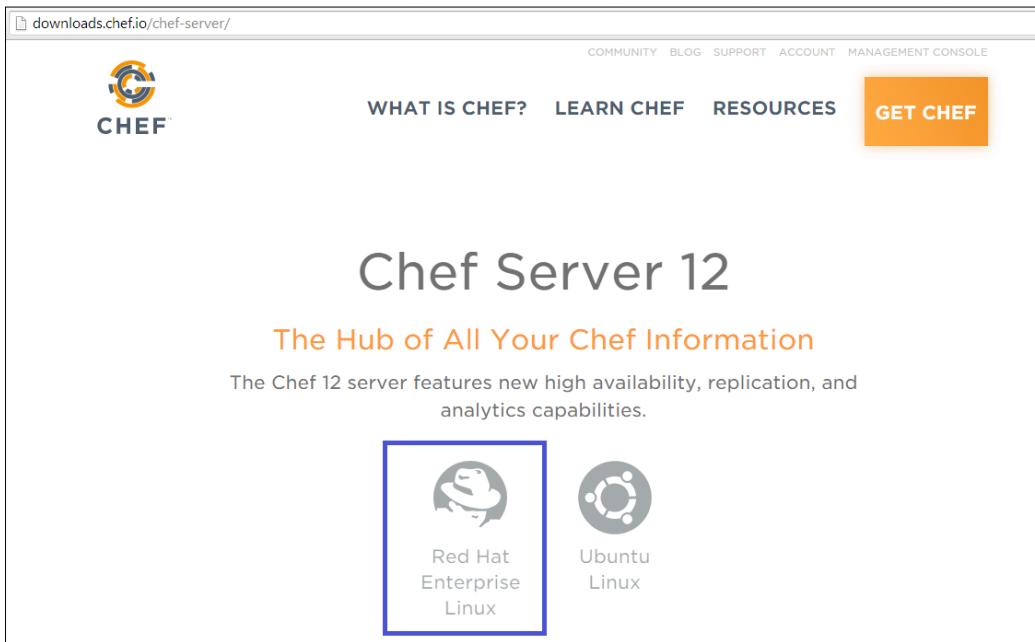
Go to <https://www.chef.io/chef/choose-your-version/> and select the **GET ON PREMISES CHEF** link.



Fill in the information to get the details and download instructions in the next page.

The screenshot shows the 'Install Enterprise Chef On-Premises' contact form page. At the top, there's a navigation bar with links for 'WHAT IS CHEF?', 'LEARN CHEF', 'RESOURCES', and a prominent orange 'GET CHEF' button. Below the navigation, the main heading 'Install Enterprise Chef On-Premises' is displayed. A sub-instruction below it reads: 'Fill out the information below to receive download details and installation instructions for Enterprise Chef Server.' The form itself consists of four text input fields: 'First Name:', 'Last Name:', 'Email:', and 'Comments:'. To the right of each input field is a small red asterisk indicating it's required. Below the comments field is a checkbox labeled 'I agree to the [Terms of Service](#) and the [Master License and Services Agreement](#)'. At the bottom of the form is a blue 'Submit' button.

Select the operating system. We will install it on CentOS 6.6, hence select **Red Hat Enterprise Linux**.



Click on the **Download** link for **Red Hat Enterprise Linux 6**.



## *Working with the On-premises Chef Server Setup*

---

Provide the required details and submit it to get the `chef-server-core-12.0.1-1.x86_64.rpm` file.

The screenshot shows a web browser window with the URL `downloads.chef.io/chef-server/redhat/#/`. A modal dialog box titled "Download Chef Server" is open. It contains fields for "First Name", "Last Name", "Email", and "Comments", all of which have been redacted. Below these fields is a checkbox labeled "I agree to the Terms of Service and the Master License and Services Agreement." A "Submit" button is at the bottom of the modal. In the background, there's a dark banner with the text "The Chef 12" and "Red Hat Enterprise Linux 5" and "Red Hat Enterprise Linux 6" with "Download" buttons. A top navigation bar has tabs for "SOURCES" and "GET CHEF". A status bar at the bottom right says "Version 12.0.1".

Now, we are going to discuss the prerequisites.

## **Prerequisites for the standalone on-premises Chef installation**

The following is a list of the prerequisites for standalone on-premises Chef:

Resource	Minimum Requirement
Processor	2.0 GHz AMD processor with 4 cores or Intel Xeon E5 processor
Memory	4GB RAM
Storage	5 GB free space is required in both the /opt and /var directory

## **Firewall requirements**

In *Chapter 2, Different Components of Chef's Anatomy*, we learned that Nginx is a load balancer used for the Chef server. Port 80 (HTTP) and port 443 (HTTPS) must be open if a host-based firewall is being used.

After applying the required settings, our system is now ready for the on-premises Chef installation.

We need to record the location of the filesystem where the on-premises Chef server package is going to be installed. By default, it is installed in the `/tmp` directory.

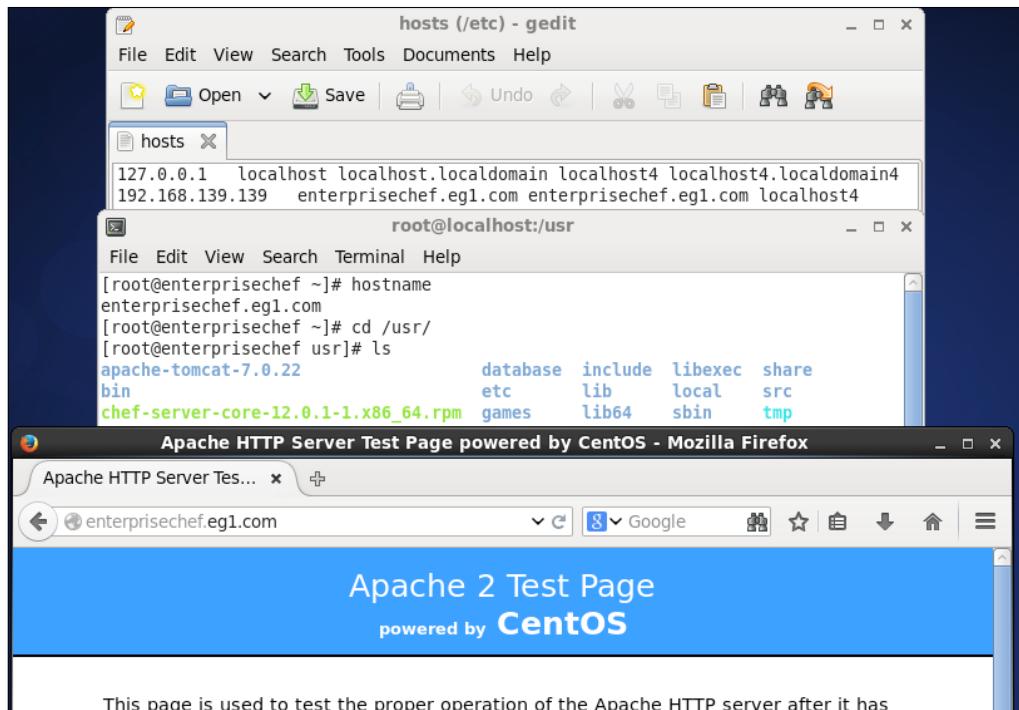
Standalone Chef has different commands for different Linux distributions. For more details, visit [http://docs.chef.io/enterprise/install\\_server\\_standalone.html](http://docs.chef.io/enterprise/install_server_standalone.html).

## Installing standalone on-premises Chef

After applying all the initial prerequisite settings, we are going to install standalone on-premises Chef.

### Installing the on-premises Chef package on CentOS and Red Hat

Set FQDN and verify it. To set FQDN, open the `/etc/hosts` file and make the necessary changes, as shown in the following screenshot:

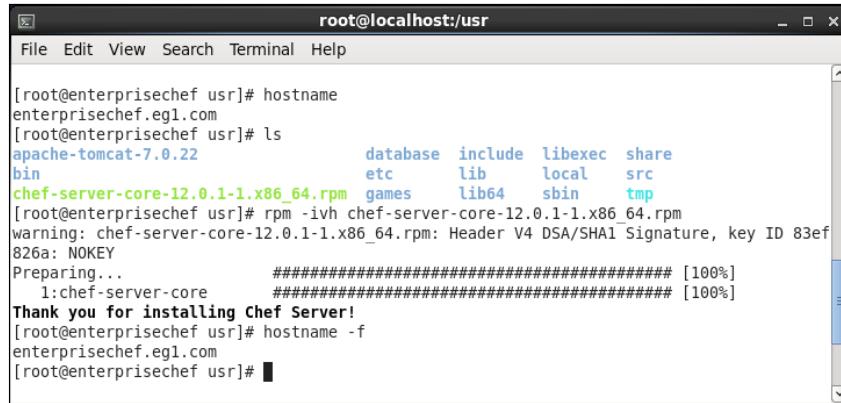


## *Working with the On-premises Chef Server Setup*

---

The following is the command for installing the Chef package on CentOS and Red Hat:

```
$ rpm -uvh chef-server-core-12.0.1-1.x86_64.rpm
```



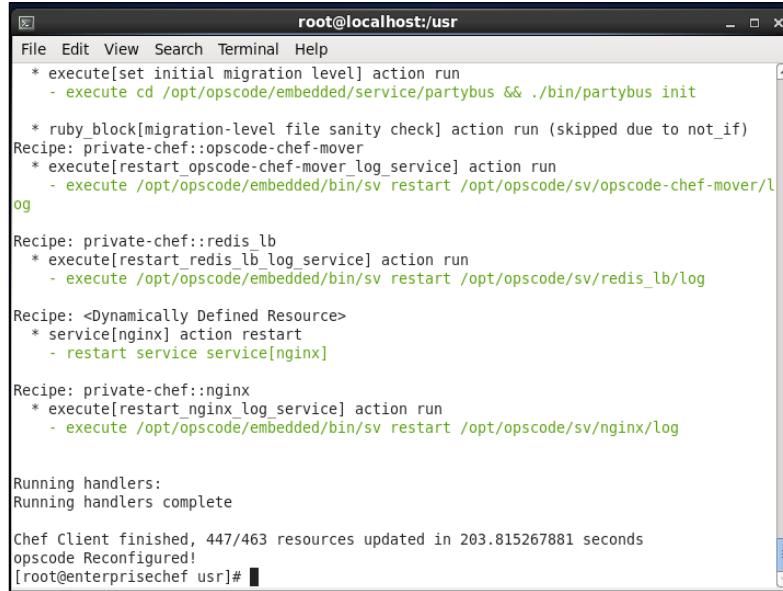
A screenshot of a terminal window titled "root@localhost:/usr". The window shows the command "rpm -uvh chef-server-core-12.0.1-1.x86\_64.rpm" being run. The output includes the file listing, the warning about the header signature, the progress bar for the download, and the message "Thank you for installing Chef Server!". The terminal prompt "[root@enterprisechef usr]#" is visible at the bottom.

Enterprise Chef is now installed. To configure it for a standalone installation, run the following command:

```
$ private-chef-ctl reconfigure
```

Enterprise Chef might take several minutes to get installed on your system. Once it does, we need to configure it with the following command:

```
$ private-chef-ctl reconfigure
```



A screenshot of a terminal window titled "root@localhost:/usr". The window shows the command "private-chef-ctl reconfigure" being run. The output displays the configuration steps for various services, including "partybus", "opscode-chef-mover", "redis\_lb", "nginx", and "nginx". It also shows the execution of handlers and the final statistics: "Chef Client finished, 447/463 resources updated in 203.815267881 seconds" and "opscode Reconfigured!". The terminal prompt "[root@enterprisechef usr]#" is visible at the bottom.

Here, `chef-server-ctl` is the command-line utility that is used to manage various services in on-premises Chef. It comes with the omnibus installer, which is specifically used for Chef installation.

After running the preceding commands, we should wait for some more minutes to complete the final installation of on-premises Chef. After completion, we will see the following message:

**Chef Server Reconfigured!**

Verify the status of the installation by using the following command:

```
$ private-chef-ctl status
```

```
root@enterprisechef ~]# private-chef-ctl status
the ffi-yajl and yajl-ruby gems have incompatible C libyajl libs and should not
be loaded in the same Ruby VM
falling back to ffi which might work (or might not, no promises)
ffi-yajl/json_gem is deprecated, these monkeypatches will be dropped shortly
run: bookshelf: (pid 1102) 6608s; run: log: (pid 1082) 6609s
run: nginx: (pid 1095) 6608s; run: log: (pid 1067) 6609s
run: oc_bifrost: (pid 1100) 6608s; run: log: (pid 1076) 6609s
run: oc_id: (pid 1097) 6608s; run: log: (pid 1069) 6609s
run: opscode-erchef: (pid 1098) 6608s; run: log: (pid 1070) 6609s
run: opscode-expander: (pid 1101) 6608s; run: log: (pid 1086) 6609s
run: opscode-expander-reindexer: (pid 1103) 6608s; run: log: (pid 1085) 6609s
run: opscode-solr4: (pid 1099) 6608s; run: log: (pid 1075) 6609s
run: postgresql: (pid 1112) 6608s; run: log: (pid 1072) 6609s
run: rabbitmq: (pid 1074) 6609s; run: log: (pid 1066) 6609s
run: redis_lb: (pid 1096) 6608s; run: log: (pid 1071) 6609s
[root@enterprisechef ~]#
```

## Prerequisites for the tiered on-premises Chef installation

Here is the list of the prerequisites for tiered on-premises Chef. The following are the minimum requirements:

Resource	Minimum Requirement
Processor	2.0 GHz AMD processor with 8 cores or Intel Xeon E5 processor is required
Memory	16 GB RAM is required
Storage	20 GB free space is required in /opt and 40 GB space is required in the /var directory
Hardware	2 x 300GB SAS RAID1 drives and one RAID Card
Network	One Gigabit Ethernet NIC interface card
Server	One backend server is mandatory and the remaining will be a frontend server

## Load balancer requirements

In a tiered installation, various frontend servers get connected as a cluster. Therefore, there must be an efficient load balancing mechanism to balance the load throughout the cluster. Based on the requirements, we can implement a hardware-specific load balancer.

There is one cluster configuration file named as `private-chef.rb`. Inside this file, there is one entry `api_fqdn`.

First, we need to make a DNS entry for all the virtual IP addresses. Once this is done, we will pass that entry into the FQDN.

## Configuring `api_fqdn`

To configure `api_fqdn`, we need to pass one entry of FQDN in the `private-chef.rb config` file:

```
api_fqdn "FQDN"
```

In the previous command, `FQDN` will be replaced by the FQDN of load balanced virtual IP addresses.

## Firewall requirements

We need to open the following ports in the iptables:

### Ports for frontend servers

The following ports must be open for frontend entries:

Port	Open for
80	nginx
443	nginx
9672	nrpe

### Ports for backend servers

The following ports must be open for backend entries:

Port	Open for
80	nginx
443	nginx

Port	Open for
4321	bookshelf
4369	opscode-org-creator
5140	opscode-certificate
5432	postgresql
9000	nagios
9460	opscode-chef
9462	opscode-webui
9463	opscode-authz
9465	opscode-account
9466	estatsd
5672	rabbitmq
5984	couchdb
6379	redis
7788	drbd
8000	opscode-erchef
8983	opscode-solr
9670	nagios
9671	nagios
9672	nrpe
9671	nginx
9680	nginx
9683	nginx

## Configuring the private-chef.rb file

As discussed previously, the `private-chef.rb` file is the main configuration file of the cluster. This file is located in every server at the `/etc/opscode/private-chef.rb` location. This configuration file contains the information of the topology that is being used for the entire cluster. Therefore, we first need to configure this file before the tiered installation.

First, mention the name of the topology in the `private-chef.rb` file:

```
topology "tier"
```

Then, make this cluster horizontally scalable. As discussed previously, the following settings are required:

1. A backend server is a must for the tiered installation, so the server that is going to be used for standalone Chef will work as a backend server.
2. The IP address of the backend server must be included in the `private-chef.rb` file.

## The required settings for the backend server

Open the `private-chef.rb` file in editor and make the following changes:

```
server "FQDN",
:ipaddress => "IPADDRESS",
:role => "backend",
:bootstrap => true
```

We need to replace these entries according to our backend server. Here, `FQDN` will be FQDN of the backend server and `ipaddress` will be `IPADDRESS` of the backend server.

This backend server will be used to bootstrap the tiered Chef installation. So, we need to add one more entry for this:

```
backend_vip "FQDN"
:ipaddress => "IPADDRESS",
```

Here, `FQDN` will be FQDN of the backend server.

## The required settings for the frontend server

The same type of configuration changes need to be done in the `private-chef.rb` file for the frontend server:

```
server "FQDN",
:ipaddress => "IPADDRESS",
:role => "frontend"
```

Here, `FQDN` will be FQDN of the frontend server, `ipaddress` will be `IPADDRESS` of the frontend server, and `role` will be `frontend`.

After getting all the required configurations, the `private-chef.rb` configuration file will look as follows:

```
topology "tier"

server "ab1.myexample.com",
:ipaddress => "192.168.139.139",
```

```
:role => "backend",
:bootstrap => true

backend_vip"ab.myexample.com",
:ipaddress => "192.168.139.139"

server"ab2.myexample.com",
:ipaddress => "192.168.139.140",
:role => "frontend"

server"ab3.myexample.com",
:ipaddress => "192.168.139.141",
:role => "frontend"

server"ab4.myexample.com",
:ipaddress => "192.168.139.142",
:role => "frontend"

api_fqdn"service.myexample.com"
```

## Adding on-premises Chef packages to servers

We need to upload the on-premises Chef package in the /tmp directory of every server where we want to install on-premises Chef. We need to remember its location to configure the Bootstrap server.

## Installing tiered on-premises Chef

After performing all the prerequisite settings and uploading Chef packages on every server, now, finally, install on-premises Chef. The tiered on-premises Chef installation happens in two parts. First, the on-premises Chef installation is done on the bootstrap server, after that, on the frontend servers.

## Configuring Bootstrap and installing on-premises Chef

Now, we need to copy the `private-chef.rb` file to `/etc/opscode/private-chef.rb` on the Bootstrap server.

After this, we need to set the Bootstrap server with the following command:

```
$ private-chef-ctl reconfigure
```

When the system is starting up, you will see full CPU utilization due to many Ruby processes. Therefore, this command takes some time to complete the configuration. Once it is completed, we will see the following output:

```
Chef Server Reconfigured!
```

## Configuring the frontend server and installing on-premises Chef

The following configuration is required for the frontend server.

You need to ensure that you are logged in as a root user in the Bootstrap server. Once the Bootstrap process is completed, then you need to copy the `/etc/opscode` file in all the frontend servers:

```
$ scp -r /etc/opscode FQDN:/etc
```

Here, `FQDN` will be replaced by a particular frontend server's FQDN.

## Installing on-premises Chef packages

Now, we need to install the on-premises Chef package on every frontend server. There are different commands for different Linux distributions, which are as follows:

- The command for the installation of the on-premises Chef package on Ubuntu:

```
$ dpkg -i /tmp/chef-server-core-<version>.deb
```

- The command for the installation of the on-premises Chef package on CentOS and Red Hat:

```
$ rpm -Uvh /tmp/ chef-server-core-<version>.rpm
```

After the on-premises Chef package installation is done, we need to run the configuration command in frontend servers:

```
$ private-chef-ctl reconfigure
```

When the system is starting up, you may see full CPU utilization due to many Ruby processes. Thereby, the configuration command takes some time to complete the configuration. Once this is complete, we will see the following output:

```
Chef Server Reconfigured!
```

We have learned all about tiered Chef installation. Now, we can manage it according to our requirements, followed by the user management section.

## Prerequisites for the high-availability Chef installation

Here is a list of the prerequisites for the high-availability on-premises Chef. The following are the minimum requirements:

Resource	Minimum Requirement
Processor	2.0 GHz AMD processor with 8 cores or Intel Xeon E5 processor
Memory	16 GB RAM
Storage	20 GB free space is required in /opt and 40 GB space is required in the /var directory
Hardware	2 x 300GB SAS RAID1 drives and one RAID Card
Network	One Gigabit Ethernet NIC interface card
Server	Two backend servers are mandatory and the remaining server will be the frontend server

## Load balancer requirements

In the high availability Chef installation, various frontend servers get connected in the form of a cluster. Therefore, there must be an efficient load balancing mechanism to balance the load throughout the cluster. We can implement a hardware-based load balancer depending on our requirements.

There is a cluster configuration file named as `private-chef.rb`. Inside this file, there is one entry named as `api_fqdn`.

First, we need to make the DNS entry for all the virtual IP addresses. After which, we will pass that entry into FQDN.

## Configuring api\_fqdn

To configure `api_fqdn`, we need to pass one entry of FQDN in the `private-chef.rbconfig` file:

```
api_fqdn "FQDN"
```

In the previous command, `FQDN` will be replaced by the FQDN of load balanced virtual IP address's firewall requirement.

We need to open the following ports in the iptables:

## Ports for frontend servers

The following ports must be open for frontend entries:

Port	Open For
80	nginx
443	nginx
9672	nrpe

## Ports for backend servers

The following ports must be open for backend entries:

Port	Open for
80	Nginx
443	Nginx
9671	Nginx
9680	Nginx
9685	Nginx
9683	nginx
9672	nrpe
5984	couchdb
8983	opscode-solr
5432	postgresql
5672	rabbitmq
6379	Redis
7788	Drbd

## Configure the private-chef.rb file

As discussed previously, the `private-chef.rb` file is the main configuration file of the cluster. This file is located in every server at the `/etc/opscode/private-chef.rb` location. This configuration file contains the information of the topology that is being used for the entire cluster.

Therefore, we need to configure this file first before installing the high-availability Chef server.

First, mention the name of the topology in the `private-chef.rb` file:

```
topology "ha"
```

In order to make the high-availability backend server, a cluster must be horizontally scalable. Therefore, our backend server will work as the Bootstrap server and the IP of this backend server must be included in the `private-chef.rb` file.

## The required settings for the backend server with Bootstrapping

Open the `private-chef.rb` file in editor and make the following changes:

```
server "FQDN",
:ipaddress => "IPADDRESS",
:role => "backend",
:bootstrap => true
:cluster_ipaddress => "CLUSTER_IPADDRESS"
```

Now, we need to replace these entries according to our backend server. Here, `FQDN` will be `FQDN` of the backend server, `ipaddress` will be `IPADDRESS` of the backend server.

This backend server will be used for Bootstrapping, so `cluster_ipaddress` will be the interface's IP address. This IP address of the interface will be used for the entire cluster communication. This can be excluded if there is no interface.

## The required settings for other backend servers

For other backend servers, just add the following block:

```
server "FQDN",
:ipaddress => "IPADDRESS",
:role => "backend",
:cluster_ipaddress => "CLUSTER_IPADDRESS"
```

All the replacements will be the same as before, except Bootstrapping.

Here, we need to assign one entry for the virtual IP address as shown following:

```
backend_vip "FQDN",
:ipaddress => "IPADDRESS",
:device => "eth0",
:heartbeat_device => "eth1"
```

We need to replace the preceding entries according to our backend server. Here, FQDN will be FQDN of the server, ipaddress will be IPADDRESS of the virtual IP address.

We require some additional parameters, such as a device and heartbeat\_device. A device is used to bind the floater virtual IP address, so it works like an Ethernet interface for floater virtual IP address. Another parameter heartbeat\_device will be working like an Ethernet interface, but for cluster\_ipaddress.

## The required changes for frontend entries

For all the frontend servers, we need to make the following changes in the private-chef.rb file on each server:

```
server "FQDN",
:ipaddress=>"IPADDRESS",
:role=>"frontend"
```

Here, FQDN will be FQDN of the frontend server, ipaddress will be IPADDRESS of the frontend server, and role will be frontend.

## Configuring api\_fqdn

To configure api\_fqdn, we need to pass one entry of FQDN in the private-chef.rbconfig file:

```
api_fqdn "FQDN"
```

Here, FQDN will be the load balanced virtual IP address's FQDN.

## Examples

```
topology "ha"
```

```
server "ab1.example.com"
:ipaddress => "192.168.139.139",
```

```
:role => "backend",
:bootstrap => true,
:cluster_ipaddress => "10.2.3.15"

server"ab2.myexample.com",
:ipaddress => "192.168.139.146",
:role => "backend",
:cluster_ipaddress => "10.2.3.16"

backend_vip"ab.myexample.com",
:ipaddress => "192.168.139.147",
:device => "eth0",
:heartbeat_device => "eth1"

server"ab3.myexample.com",
:ipaddress => "192.168.139.142",
:role => "frontend"

server"ab4.myexample.com",
:ipaddress =>"192.168.139.143",
:role => "frontend"

server"ab5.myexample.com",
:ipaddress => "192.168.139.144",
:role => "frontend"

api_fqdn"service.myexample.com"
```

## **Adding on-premises Chef packages to servers**

First, we need to upload the on-premises Chef package in the /tmp directory of every server where we want to install on-premises Chef. We need to remember its location.

Now, we need to copy the private-chef.rb file to /etc/opscode/private-chef.rb on the Bootstrap server.

## Installing the high-availability Chef server

As we know, in the high-availability on-premises Chef installation, we have two backend servers in which one works as a Bootstrap server. First, we will have to install on-premises Chef on both the backend servers, then configure on-premises Chef on the frontend server.

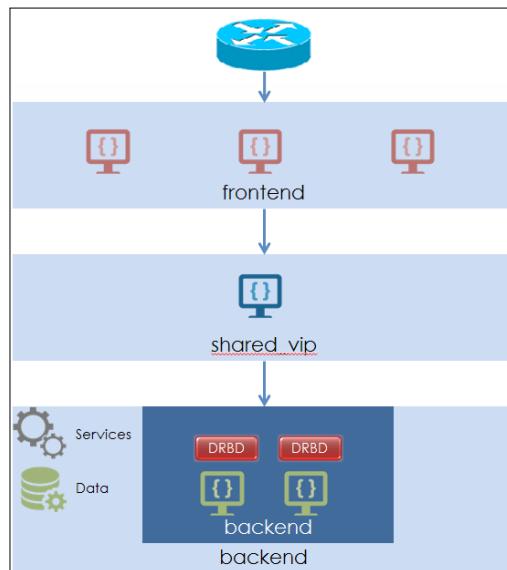
### Installing on-premises Chef on the backend server

As discussed, we need to install on-premises Chef on both the backend servers. There are different commands for different Linux distributions, which are as follows:

- Installing the on-premises Chef package on Ubuntu:  
`$ dpkg -i /tmp/chef-server-core-<version>.deb`
- Installing the on-premises Chef package on CentOS and Red Hat:  
`$ rpm -Uvh /tmp/chef-server-core-<version>.rpm`

In the high-availability Chef server, we also require the DRBD installation.

**Distributed Replicated Block Device (DRBD)** works like a building block to create high-availability clusters. The working of DRBD is the same as network raid-1.



## Installing DRBD on the backend servers

The DRBD installation is done on both the backend servers. There are different commands for different Linux distributions.

### The DRBD installation on CentOS and Red Hat

The following is the command for the DRBD installation on CentOS and Red Hat:

```
$ rpm --import http://elrepo.org/RPM-GPG-KEY-elrepo.org
$ yum install -y http://elrepo.org/elrepo-release-6-5.el6.elrepo.noarch.rpm
$ yum install -y drbd84-utils kmod-drbd84
```

### The DRBD installation on Ubuntu

The following is the command for the DRBD installation on Ubuntu:

```
$ apt-get install drbd8-utils
```

## The DRBD configuration on the backend Bootstrap server

The high-availability Chef server installation is done in two steps. First, we need to configure DRBD with the following commands:

```
$ drbdadm create-md pc0
$ drbdadm up pc0
```

Then, run the following command:

```
$ private-chef-ctl reconfigure
```

The preceding command will configure the high-availability Chef server successfully on the backend server. However, if we run preceding command first, then installation will give the message to complete the DRBD replication. Then, we can exit this message by pressing *Ctrl + c* and give the remaining DRBD configuration commands.

## The DRBD configuration on the backend non-Bootstrap server

After configuring DRBD on the backend Bootstrap server, the same configuration needs to be done on another backend server, which is not bootstrapped.

Now, we need to copy the contents from the `/etc/opscode` location of the backend Bootstrap server to the non-Bootstrap backend server by using the following command:

```
$ scp -r FQDN:/etc/opscode /etc
```

Here, FQDN will be replaced by the Bootstrap's server FQDN.

Now, the DRBD configuration setup needs to be configured on the non-Bootstrap backend server by using the following command:

```
$ private-chef-ctl reconfigure
```

However, as we discussed previously, this installation will not be successful until we configure the DRBD configuration. Therefore, we will exit by pressing *Ctrl + c* and then give the remaining DRBD configuration commands:

```
$ drbdadm create-md pc0
$ drbdadm up pc0
```

Now that we have configured both the servers for DRBD, the cluster should know that the Bootstrap server is the primary server for the shared device.

We can make the Bootstrap server as a primary server by using the following commands:

- DRBD on Ubuntu:

```
$ drbdadm -- --overwrite-data-of-peer primary pc0
```

- DRBD on CentOS and Red Hat:

```
$ drbdadm primary --force pc0
```

## Checking the filesystem for the DRBD server

Now, we need to create the filesystem for DRBD on the Bootstrap server if it is `ext4`, then we need to provide the following commands:

```
$ mkfs.ext4 /dev/drbd0
$ mkdir -p /var/opt/opscode/drbd/data
$ mount /dev/drbd0 /var/opt/opscode/drbd/data
```

DRBD must be completely synchronized with all the devices before installation. To check the synchronization process, run the following command:

```
$ watch -n1 cat /proc/drbd
```

The output is seen by the following command:

```
cat /proc/drbd output
```

```
version: 8.4.1 (api:1/proto:86 [STRIKEOUT:100])
GIT-hash: 92b4d048c1b0e06775b5f66d3a2d38d47abbea9a build by
dag@Build64R6, 2012]13 [STRIKEOUT:21 09:19:23
 0: cs:SyncSourcer0:Primary/Secondary ds:UpToDate/Inconsistent C r]--
 ns:3071457 nr:0 dw:0 dr:3084621 al:0 bm:167 lo:0 pe:13 ua:4 ap:0 ep:1
wo:b oos:12976632
[==>.....]sync'ed: 18.4% (11248/15372)M
finish: 0:10:00 speed: 18,141(23,238) K/sec
```

In the preceding output, we need to observe the `ds` section, where the Up-To-Date output means that the synchronization is successful.

This procedure ensures that the entire data of all the shared devices must be synchronized. If we fail to follow the previous operation before installing high-availability on-premises Chef, then DRBD allocates only a portion of the bandwidth for the initialization and synchronization process.

Now, we can speed up the initial synchronization process by using the following command to increase more bandwidth utilization:

- Increasing the synchronization on Ubuntu:  
  \$ `drbdsetup /dev/drbd0 syncer -r 1100M`
- Increasing the synchronization on CentOS and Red Hat:  
  \$ `drbdadm disk-options --resync-rate=1100M pc0`

After this synchronization is finished successfully, the next step is for on-premises Chef to know that DRBD is completely ready for the Bootstrap server. Now, run the following command:

```
$ touch /var/opt/opscode/drbd/drbd_ready
```

## Configuring on-premises Chef on the Bootstrap backend server

Now, we need to run the following, final on-premises Chef configuration command on the Bootstrap server:

```
$ private-chef-ctl reconfigure
```

This command takes some time to complete the configuration. Once it is completed, we will see the following as an output:

```
Chef Server Reconfigured!
```

## Configuring on-premises Chef on the non-Bootstrap backend server

Before moving on to the non-Bootstrap backend server, it is highly recommended that on-premises Chef must be successfully installed on the Bootstrap server and DRBD's synchronization is finished properly.

After this synchronization is finished successfully, the next step is that on-premises Chef knows that DRBS is completely ready for the non-Bootstrap server. Chef should know this because it may so happen that the Bootstrap server will not be allowed to finish the initialization until the non-Bootstrap setting is not completed.

Now, to check the status of non Bootstrap server run the following command:

```
$ touch /var/opt/opscode/drbd/drbd_ready
```

We need to run the final on-premises Chef configuration on the non-Bootstrap server by using the following command:

```
$ private-chef-ctl reconfigure
```

This command takes some time to complete the configuration. Once it is completed, we will see the following as an output:

```
Chef Server Reconfigured!
```

## Configuring and installing on-premises Chef on the frontend servers

First, you need to ensure that you are logged as a root user in the Bootstrap server once the Bootstrap process is completed. Now, we need to copy the contents from the /etc/opscode location of the backend Bootstrap server to the frontend server by using the following command:

```
$ scp -r /etc/opscode FQDN:/etc
```

Here, FQDN will be replaced by a particular frontend server's FQDN.

## Installing on-premises Chef packages

Now, we need to install the on-premises Chef package on every frontend server. There are different commands for different Linux distributions, which are as follows:

- Installing the on-premises Chef package on Ubuntu:

```
$ dpkg -i /tmp/chef-server-core-<version>.deb
```

- Installing the on-premises Chef package on CentOS and Red Hat:

```
$ rpm -Uvh /tmp/chef-server-core-<version>.rpm
```

After the on-premises Chef package installation is done, we need to run the configuration command in the frontend servers:

```
$ private-chef-ctl reconfigure
```

This command takes some time to complete the configuration. Once it is completed, we will see the following as an output:

```
Chef Server Reconfigured!
```

This is all about the high-availability on-premises Chef installation. Now, in the next section, we will learn about the management of the on-premises Chef server.

## Managing on-premises Chef

After the successful installation of the different types of Chef servers, we have a choice to select any kind of on-premises Chef installation according to our organizational or personal requirements. However, a simple installation is not enough. As a Chef administrator or learner, we must have the knowledge of various management and administration commands for the on-premises Chef server. Here, we are going to learn some commands that are frequently used by Chef administrators.

Practically, Chef installation is done with the help of the omnibus installer. We have seen before that `chef-server-ctl` is the command-line utility that is used to manage various services in on-premises Chef, such as starting, stopping, reconfiguring, uninstalling something, and so on. This utility makes our work very easy as we can give just one command and the service action will be done.

## **Service commands**

Now, we will see a list of commands to manage on-premises Chef using the `chef-server-ctl` utility.

### **Viewing Chef commands**

To view the list of the available on-premises Chef control commands, the `help` subcommand is used.

The syntax of the `help` subcommand is as follows:

```
$ private-chef-ctl help
```

### **Uninstalling on-premises Chef**

In order to uninstall the Chef application except removal of any data, we can give one subcommand named as `uninstall`. This immediately shuts down all the running services.

The syntax of the `uninstall` subcommand is as follows:

```
$ private-chef-ctl uninstall
```

### **View configuration**

In order to verify all the stages of deployment, ensure that all the services are installed successfully. We can give a single subcommand named as `show-config`.

The syntax of the `show-config` subcommand is as follows:

```
$ private-chef-ctl show-config
```

### **Reconfiguring Chef**

Whenever any changes are done in the `private-chef.rb` file, the server needs to be reconfigured every time after every change. These changes can't be applicable without giving the `reconfigure` subcommand.

The syntax of the reconfigure subcommand is as follows:

```
$ private-chef-ctl reconfigure
```

## Service subcommands

On-premises Chef has got a default in-process supervisor, which ensures that all the services will be in the appropriate state at any point of time. The supervisor initiates a dual process for every service.

Here are a few of the subcommands:

### The hup subcommand

The hup subcommand is applicable to send SIGINT for the whole of the services. If you write all the names of the services, this command can be run for an individual service.

The syntax for the subcommand is as follows:

```
$ private-chef-ctl hup name_of_service
```

Here, `name_of_service` means the name of the service that is being noted in a list after the service list subcommand is run.

### The int subcommand

The int subcommand is applicable to send a SIGINT to all of these services. If you write all the names of the services, this command can be run for an individual service.

The syntax for the subcommand is as follows:

```
$ private-chef-ctl int name_of_service
```

Here, `name_of_service` means the name of the service that is being noted in a list after the service list subcommand is run.

### The kill subcommand

The kill subcommand is applicable to send a SIGINT to all of these services. If you write all the names of the services, this command can be run for an individual service.

The syntax for the subcommand is as follows:

```
$ private-chef-ctl kill name_of_service
```

Here, `name_of_service` means the name of the service that is being noted in a list after the service list subcommand is run.

## The once subcommand

The supervisor for Enterprise Chef is configured to restart any service that fails unless and until that service has been asked to change its state. The `once` subcommand is given to inform the supervisor not to attempt to restart any service that fails.

This command is applicable while troubleshooting configuration errors that stop a service from initiating. We will run the `once` subcommand followed by the `status` subcommand to look for services in a down state and/or to identify which services are not in the normal state. By identifying the name of the service in the command, this command is made to run.

The syntax for the subcommand is as follows:

```
$ private-chef-ctl once name_of_service
```

Here, `name_of_service` means the name of the service that is being noted in a list after the `service-list` subcommand is run.

## The service-list subcommand

The `service-list` subcommand is applicable to show/display a list of all the available services. An enabled service is labeled with an asterisk symbol (\*).

The syntax for this subcommand is as follows:

```
$ private-chef-ctl service-list
```

## The start subcommand

The `start` subcommand is given to initiate all the services that are enabled in Enterprise Chef. If you write all the names of the services, this command can be run for an individual service.

For example, if we are configuring any file, we may need to start a particular service accordingly.

The syntax for this subcommand is as follows:

```
$ private-chef-ctl start name_of_service
```

Here, `name_of_service` means the name of a service that is being noted in a list after the `service-list` subcommand is run.

When the service gets initiated successfully, the outcome shall be similar to:

```
$ ok: run: service_name: (pid 12345) 1s
```

## The restart subcommand

The `restart` subcommand is given to start all the services once again on Enterprise Chef. If you write all the names of the services, this command can be run for an individual service.

The syntax for this subcommand is as follows:

```
$ private-chef-ctl restart name_of_service
```

Here, `name_of_service` means the name of a service that is being noted in a list after the `service-list` subcommand is run:

```
$ ok: run: service_name: (pid 12345) 1s
```

## The stop subcommand

The `stop` subcommand is applicable to stop all the services enabled on Enterprise Chef. By specifying the name of the service, this command can be run for an individual service.

The syntax for the subcommand is as follows:

```
$ private-chef-ctl stop name_of_service
```

Here, `name_of_service` symbolizes the name of any service that is listed after the `service-list` command is run. When a service is halted successfully, the outcome should be the same as the following:

```
$ ok: diwb: service_name: 0s, normally up
```

Take the following code as an example:

```
$ private-chef-ctl stop
```

It would return the same as:

```
ok: down: couchdb: 385s, normally up
ok: down: nginx: 385s, normally up
ok: down: opscode-account: 386s, normally up
ok: down: opscode-authz: 386s, normally up
ok: down: opscode-certificate: 392s, normally up
ok: down: opscode-chef: 387s, normally up
ok: down: opscode-erchef: 387s, normally up
ok: down: opscode-expander: 388s, normally up
```

```
ok: down: opscode-expander-reindexer: 388s, normally up
ok: down: opscode-org-creator: 388s, normally up
ok: down: opscode-solr: 389s, normally up
ok: down: opscode-webui: 389s, normally up
ok: down: postgresql: 390s, normally up
ok: down: rabbitmq: 390s, normally up
ok: down: redis: 387s, normally up
```

## The status subcommand

The status subcommand is given to show the status of all the services that are existing to avail Enterprise Chef.

The syntax for the subcommand is as follows:

```
$ private-chef-ctl status
```

The previous command will give the status for services. If you write all the names of the services, this command can be run for an individual service:

```
$ private-chef-ctl status name_of_service
```

Here, `name_of_service` means the name of the service that is being noted in a list after the service-list subcommand is run.

When the service status is requested, the outcome shall be similar to:

```
$ run: service_name: (pid 12345) 12356s; run: log: (pid 1234) 67890s
```

Where meaning of specific parts are given as follows:

- `run` is the state of the service (run or down)
- `service_name` is the service name in which the status is returned
- `(pid12345)` is the process identifier
- `12356s` is the uptime for the service (in seconds)

## The tail subcommand

The tail subcommand is applicable for Enterprise Chef logs. It is done on all services. By specifying the name of the service, this command can be run for an individual service.

The syntax for the subcommand is as follows:

```
$ private-chef-ctl tail name_of_service
```

Where `name_of_service` symbolizes the name of the service that is listed after the `service-list` subcommand is run.

## The term subcommand

The `term` subcommand is applicable to send a SIGTERM to all services. By specifying the name of the service, this command can be run for an individual service.

The syntax of the subcommand is as follows:

```
$ private-chef-ctl term name_of_service
```

Here, `name_of_service` symbolizes the name of any service that is listed after the `service-list` subcommand is run.

## Log files

A typical status line for a service that runs in Enterprise Chef in a standalone, or standalone, topology is similar to the following:

```
run: name_of_service: (pid 1486) 7819s; run: log: (pid 1485) 7819s
```

Here, `run` explains a situation in which the supervisor keeps an attempt to tell the status of processing. This is either in the running state or in the down condition. If the service is in state of **Down**, it should be halted.

Here, `name_of_service` is the service name, for example, `opscode-solr`.

`(pid1486) 7819s` refers to the process identifier of running services and it tells the amount of time taken to run the process.

`run:log: (pid1485) 7819s` refers to the log process. It is specific for a log process that has run longer than the service time. It is done so that the supervisor need not start the log process again, which leads to the connection of the supervised process.

## Self-test questions

1. What is Enterprise Chef or on-premises Chef?
2. Why do we need to use on-premises Chef?
3. Which are the three different types of on-premises Chef installations?
4. How do you uninstall on-premises Chef?

## **Summary**

In this chapter, we learned about the benefits of the on-premises Chef server, different types of on-premises Chef servers, all the prerequisites and initial settings for every type of on-premises Chef installation, and their step-by-step installation procedures. We also learned various service commands and subcommands used to manage on-premises Chef services. Now, we can easily install and manage on-premises Chef in our organization or laboratory according to our usage. In the next chapter, we will learn the Chef installation of virtual machines and Chef installation on cloud instances.

# 8

## Managing Chef on Cloud Infrastructure

*"The technologies which have had the most profound effects on human life are usually simple."*

- Freeman Dyson

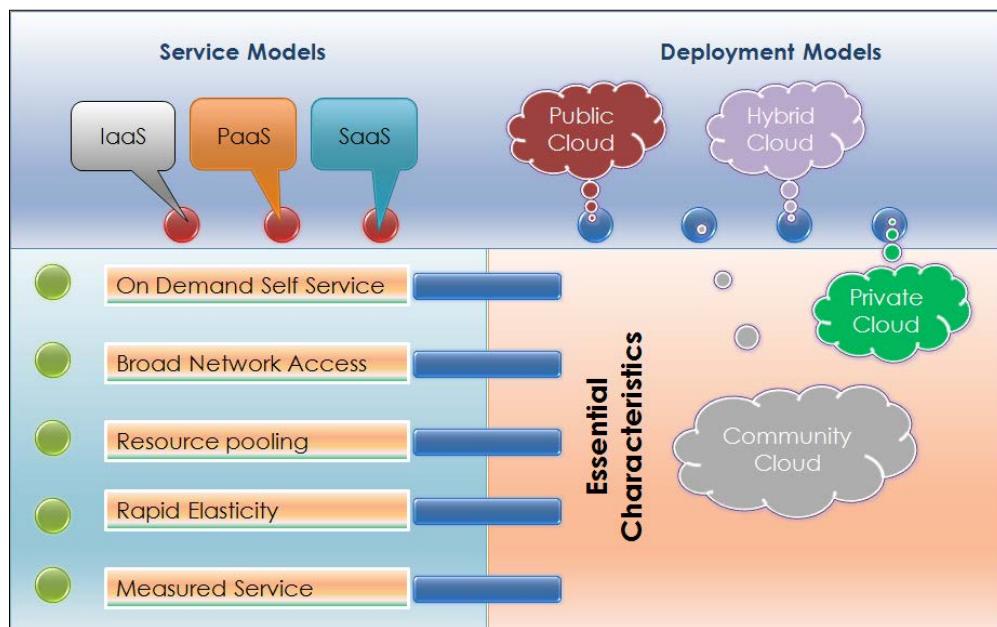
This chapter gives you a clear understanding of public cloud infrastructure services, such as Amazon EC2 and Rackspace. Chef is one of the most demanding automation scripts used to manage and automate Cloud infrastructures, either for public or private Clouds. Chef is well suited in both public and private deployment models of a cloud. In this chapter, we will see the detailed working of Chef automation in a Cloud.

We are also going to cover the most popular and commonly-used public Cloud infrastructure, specifically, **Amazon Web Service (AWS)** and Rackspace. Here, we are going to use a Hosted Chef scenario to manage Cloud instances. We are going to learn the following topics:

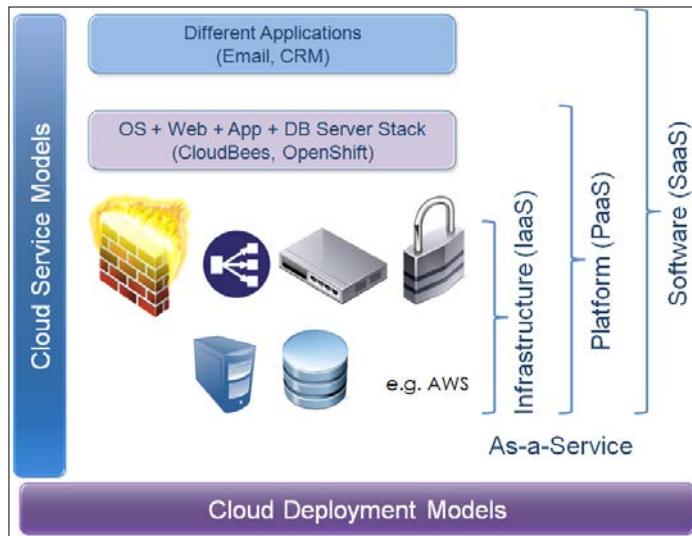
- What is cloud computing?
- Why use Chef with cloud infrastructure?
- AWS EC2 bootstrapping using Chef
- Rackspace Cloud server bootstrapping
- Knife Cloud plugin

## What is cloud computing?

According to National Institute of Standards and Technology (NIST), cloud computing is a model used to enable ubiquitous (a cloud provider's capabilities are available over the network and can be accessed through standard mechanisms), convenient, and on-demand network access to a shared pool of configurable computing resources, such as networks, servers, and storage that can be rapidly provisioned and released with minimal management effort or service provider interaction. Cloud computing is one of the greatest revolutions in the IT industry where everything comes in the form of a service and we pay according to our usage (the pay per use model). Among the various services of cloud, such as **Infrastructure as a Service (IaaS)**, **Platform as a Service (PaaS)**, and **Software as a Service (SaaS)**, the infrastructure service is the most demanding and high-valued service.



The following figure depicts more details on service models. It gives a basic understanding of what each model contains:



There are four cloud deployment models that address specific requirements:

- **Public cloud:** Here, the cloud infrastructure is available to the general public
- **Private cloud:** In this model, the cloud infrastructure is operated for a single organization
- **Community cloud:** Here, the cloud infrastructure is shared by a specific community that has similar regulatory or compliance concerns
- **Hybrid Cloud:** In this model, the cloud infrastructure has a composition of two or more cloud models

To use this service, we don't need to pay any upfront cost to build our own infrastructure. Entire servers, storage, processors, memory, bandwidth, network management, and administration can be built within a few minutes and every service is on demand basis. Agility and elasticity (it is an ability to scale resources both up and down as needed. To the Cloud consumer, the Cloud appears to be infinite and the consumer can purchase as much or as little computing power as they require) are the basic properties of the Cloud services.

## Why Chef with cloud infrastructure?

In *Chapter 1, An Overview of Automation and Advent of Chef*, we already learned about the benefits and detailed usage of Chef automation. Chef is the integral and necessary part of every Cloud infrastructure as well as it is one of the most preferred choice of architects and experts nowadays.

In a Cloud environment, we can scale up any number of cloud instances at any time using an infrastructure service. This provides us with an on-demand scalability. A Cloud instance is same as our computing machine, which can be spun up for us by just a few clicks on the Cloud environment. However, only getting instances is not sufficient; we will have to setup and prepare the required services for those instances. Therefore, Chef perfectly fits our requirement for managing and configuring initial services on each Cloud instance. We can easily maintain, automate, and scale any type of environment within Cloud instances, such as development, testing, and production.

Chef supports all types of cloud deployment scenarios, whether it is a public, private, or hybrid cloud. According to our organizational requirement, Chef automation efficiently works in all types of Cloud deployment. There are many successful case studies of successful implementation of Chef on a Cloud infrastructure, which will be discussed in the final chapter of this book. For any type of Cloud provider, there are specific plugins and APIs available that make Chef's installation quite flexible and easy. For example, we are going to see the knife-ec2 plugin configuration in next section of this chapter.

## AWS EC2 bootstrapping using Chef

AWS is the pioneer in providing IaaS. **Elastic cloud computing (EC2)** is the most popular and widely adopted Infrastructure as a service.

Here, we are going to see step-by-step procedures to bootstrap AWS EC2 instances. Let's start with the first step to set up a workstation.

### Preparing your workstation

First, we need to prepare your workstation to install the knife-ec2 plugin, so that you can use the `knife` command on AWS instances. Here, we are considering to install the Ubuntu operating system on your workstation.

If Ruby is not installed on your workstation, kindly go ahead with the Ruby installation first:

```
sudo apt-get update  
sudo apt-get install ruby1.8-dev ruby1.8ri1.8rdoc1.8irb1.8
```

```
sudo apt-get install libreadline-ruby1.8libruby1.8libopenssl-ruby  
sudo apt-get install libxslt-dev libxml2-dev
```

After this is done, the Gem installation is also required on Ubuntu because Gems contain all of the package information with files for installation:

```
sudo apt-add-repository ppa:maco.m/ruby  
sudo apt-get update  
sudo apt-get install rubygems  
sudo gem install net-ssh net-ssh-multi fog highline --no-rdoc --no-ri  
--verbose
```

## Installing the knife-ec2 plugin

You can install the knife-ec2 plugin with the help of this command; it downloads and installs the ec2 Gem for knife:

```
sudo gem install knife-ec2 --no-rdoc --no-ri
```

If we are installing the knife-ec2 plugin with the help of the Omnibus installer, the following command should be used:

```
sudo /opt/chef/embedded/bin/gem install knife-ec2
```

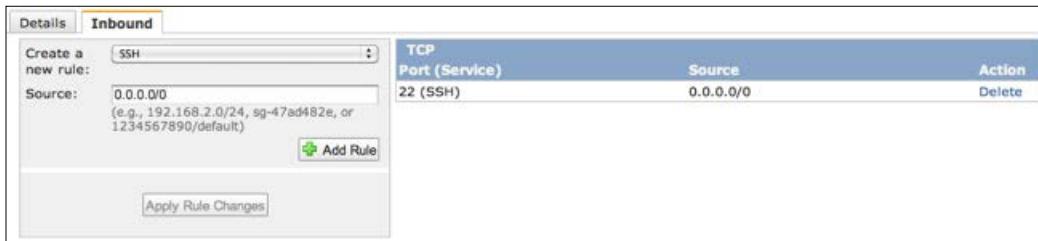
## Configuring the AWS settings and the knife.rb file

Amazon follows the security group concept to access particular ports on EC2 instances. The following are the steps to manage your security group on EC2 instances.

In order to connect from external locations, you need to have SSH access to EC2 instances using PuTTY (for Linux instances) and remote desktop connection (for Windows instances). You can do this by following these steps:

1. Sign in to the **AWS management console** and choose the **EC2** tab.
2. Choose **Security Groups** from the menu (left-hand side of the screen) and click on it.
3. Choose the default security group or you may create a newer group. However, in that condition, you need to add this new security group to your knife-ec2 server to create a command followed by -g because otherwise, it would still use the default security group. Because in AWS, When you have one or more security groups. Then you must specify the security group by ID.

4. Choose the **Inbound** tab and click on it.
5. Select **SSH** and enter `0.0.0.0/0` because the source is to accept all the incoming traffic of web and then click on **Add Rule**.



Create your SSH key pair for EC2 and save your identity file on the local workstation.

The following is the step-by-step procedure to create the SSH key pair for EC2 and to save your identity file on your local workstation:

1. Sign into the **AWS Management Console** and choose the **ec2** tab.
2. From the menu on the left-hand side of the screen, go to **Key Pairs** and click on it.
3. Click on the **Create New Pair** button.
4. Edit the **Name for your key pair**, press *Enter* and create it. It will download a `.pem` file to the local machine.
5. Store the key pair to `~/.ssh/` and memorize the name of it.
6. Be sure that `ssh-agent` is in succession. You have to execute `ssh` from the console, not from PuTTY or any other access. To initiate it, run the following command:

```
exec ssh-agent /usr/bin/bash
```

- ° In case you don't use `exec`, you will get the following error message:

```
"Could not open a connection to your authentication  
agent"
```

7. Alter the permissions of the key pair file to write. To do this, modify it by replacing `KEY_PAIR_NAME` with the name of your key pair:

```
chmod 400 ~/.ssh/KEY_PAIR_NAME.pem
```

8. You can add this key pair to `ssh` with the `KEY_PAIR_NAME` command for the name of key pair:

```
ssh-add ~/.ssh/KEY_PAIR_NAME.pem
```

9. The previous step can be avoided if you add the name of your identity file to your knife-ec2 server and create a command later with `-i` eg.`-i:`  
`/.ssh/KEY_PAIR_NAME.pem.`
10. You will also need to paste this line at the bottom of `knife.rb`, substituting `KEY_PAIR_NAME` with the name of the key pair:  
`knife[:aws_ssh_key_id] = "KEY_PAIR_NAME"`

The previous step can be avoided if you add the name of your identity file to your knife-ec2 server and create a command later with `-S`. For example, `-S "KEY_PAIR_NAME".`

This key pair is only valid in the region where you have generated it. Therefore, setting the region and availability zone in the `knife.rb` file is required:

```
knife[:availability_zone]      = 'eu-west-1a'  
knife[:region]                = 'eu-west'
```

## Configuring `knife.rb` with your AWS Cloud credentials

Now, you need to open the `knife.rb` file and add the AWS Cloud credentials at the end of the file:

```
knife[:aws_access_key_id]      = "Your AWS Access Key"  
knife[:aws_secret_access_key]  = "Your AWS Secret Access Key"
```

## Bootstrapping the EC2 instance

To bootstrap the instance, first, you will have to select an **Amazon Machine Image (AMI)** for use. You will need to get the login credentials and Bootstrap the file for the AMI.

The template will be searched for in the following locations (and order) during the boot sequence:

- The `bootstrap` directory in the installed Chef Knife library (the location varies depending on how you installed Chef).
- The `bootstrap` directory in the `$PWD/.chef/bootstrap`, for example, in `~/chef-repo/.chef/bootstrap`.
- The `bootstrap` directory in the users `$HOME/.chef/bootstrap`.

Now, you can launch the instance with a command, such as the following command. You need to replace the AMI\_ID, the LOGIN\_NAME and the BOOTSTRAP\_FILE according to your credentials.

```
knife-ec2 server create -I ami-XXXXXXX -x LOGIN_NAME
```

Knife would create a node name based on the instance ID. In case, you would like to add a name specifying it, then you can do it with the -N node name .

A bootstrap file can also be specified by you with -d.

However, by default, it will use chef-full.erb, which works alongside most of the operating systems to install the Omnibus version of Chef.

## Various configuration options

By reading the `readme` file for the `knife-ec2` plugin, the availability of the full list of options that you can add to the `knife.rbconfig` file for seeing purposes will be present. You will also see the complete list of switches available at the command line by using the following command:

```
knife ec2 server create -help
```

The `help` command will give the list of various configuration options, which we can select according to our requirements:

- -I: It displays the **Amazon machine Image (AMI)**.
- -x: It displays ssh username (for example, `root`, by default).
- -N: It displays the name of the Chef node. If a node is new and the Chef-client is not installed yet, then it will display the instance name (for example, `i-7gi46ad5`).
- -r: It displays the list of roles or recipes that are ready to apply.
- -f: It displays the type of server (for example, `m1.small`, by default).
- --region: It displays your current region using region of AWS instance, for example, `us-east-1`, by default.
- -G: It displays the security group of the server (for example, `default`, by default).

## The expected output

After running these commands, you are likely to see the following type of output:

`Instance ID: i-XXXXXXX`

`Flavor: m1.small`

```
Image: ami-XXXXXXX
Region: us-east-1
Availability Zone: us-east-1b
Security Groups: default
SSH Key: KEY_PAIR_NAME

Waiting for server.....
Public DNS Name: ec2-XXX-XXX-XXX-XXX.compute-1.amazonaws.com
Public IP Address: XXX.XXX.XXX.XXX
Private DNS Name: ip-XXX-XXX-XXX-XXX.ec2.internal
Private IP Address: XXX.XXX.XXX.XXX
```

**Waiting for sshd**

This step may take some time to turn SSH into instances. If process gets stuck at this stage, it means that the security group is not properly setup. If you want to customize your security group rather than use the default, then you can specify this with the -G switch.

If you get a message for a password prompt here, it means that SSH is not working. Then, you will get the following message code:

```
Waiting for sshd.done
Bootstrapping Chef on ec2-184-72-162-197.compute-1.amazonaws.com
Failed to authenticate ubuntu - trying password auth
Enter your password:
```

Now, you need to recheck and confirm all the steps to create and save your SSH key. Once you have logged in successfully, it will start running the bootstrap script and Ruby will be installed:

```
Waiting for sshd...done
Bootstrapping Chef on ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com --2013-09-0913:08:30-- http://
opscode.com/chef/install.sh
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com Resolving opscode.com (opscode.
com)...
ec2-95-19-171-28.compute-1.amazonaws.com 173.116.24.9
```

## Running the Chef-client on the new client node (cloud instance)

To run the Chef-client on the new client node, we just need to change SSH into a new node and run the Chef-client. You should first ensure that you are logged in as a root user. If you have multiple nodes and you want to run the Chef-client on all the nodes simultaneously, use wildcard (\*) as NODENAME.

The Chef-client can be installed using the knife command from your workstation and it is quite easy:

```
Knife ssh name:NODENAME -x ubuntu "sudo chef-client"
```

Generally, the internal FQDN is used to connect the nodes by the knife command, but it is highly recommended that you use the external FQDN when you are connecting to your Cloud instances from an outside location. This avoids extra traffic charges. Otherwise, all the traffic will route through **Network Address Translation (NAT)** and it will increase the instance charges.

You can do changes easily with the -a option. The -a option ensures that you will connect to the public hostname and then not need to specify the FQDN to connect, for example, -a ec2.public\_hostname. The following command is used for this purpose:

```
Knife sshname:NODENAME -x ubuntu "sudo chef-client"-a ec2.public_hostname
```

After some time, you will see that Chef is installed successfully. Now, Chef-client will run for the first time:

The following will be the output of successful Chef Installation:

```
ec2-95-19-171-28.compute-1.amazonaws.com Thank you for installing Chef!
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com Processing triggers
for initramfs-tools ...
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com update-initramfs: Generating /
boot/initrd.img-3.2.0-23-virtual
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:24+0000]
INFO: *** Chef 11.X.X ***
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201397:07:30+0000]
INFO: Client key /etc/chef/client.pem is not present - registering
ec2-95-19-171-28.compute-1.amazonaws.com
```

```
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:31+0000]
INFO: Setting the run_list to [] from JSON
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:31+0000]
INFO: Run List is []
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:31+0000]
INFO: Run List expands to []
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:31+0000]
INFO: Starting Chef Run for i-XXXXXXX
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:31+0000]
INFO: Running start handlers
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:31+0000]
INFO: Start handlers complete.
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:32+0000]
INFO: Loading cookbooks []
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:32+0000]
WARN: Node i-XXXXXXX has an empty run list.
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:32+0000]
INFO: Chef Run complete in 0.752311845seconds
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:32+0000]
INFO: Running report handlers
ec2-95-19-171-28.compute-1.amazonaws.com
ec2-95-19-171-28.compute-1.amazonaws.com [Fri, 13Sep] 201223:08:32+0000]
INFO: Report handlers complete
ec2-95-19-171-28.compute-1.amazonaws.com
```

After this, it will show all the details of your instance:

```
Instance ID: i-XXXXXXX
Flavor: m1.small
Image: ami-b89842d1
Region: us-east-1
```

```
Availability Zone: us-east-1b
Security Groups: default
Tags: Name=i-XXXXXXX
SSH Key: KEY_PAIR_NAME
Root Device Type: instance-store
Public DNS Name: ec2-XXX-XXX-XXX-XXX.compute-1.amazonaws.com
Public IP Address: XXX.XXX.XXX.XXX
Private DNS Name: ip-XXX-XXX-XXX-XXX.ec2.internal
Private IP Address: XXX.XXX.XXX.XXX
Environment: _default
```

The current version of Chef will be displayed in the output, `Chef 11.x`. This means that the Chef-client has been installed in the AWS EC2 instance. Now, you can add a recipe on it. The `run_list` command can also be added using the `-r` switch while creating the instance.

If you add a recipe in the run-list, then the Chef-client service can be run as a daemon. We will see this process in the last step.

## Verification of the complete installation

After the successful registration of the client with Hosted Chef, the next step is to verify it. Registration details should be immediately reflected in the EC2 sever list and node list. The following `knife` command is used to display the EC2 server's list:

```
% knife ec2 server list
```

It will display the following details:

Instance ID	Public IP	Private IP	Flavor	Image
SSH Key	Security Groups	State		
i-XXXXXXX	XXX.XXX.XXX.XXX	XXX.XXX.XXX.XXX	m1.small	
ami-XXXXXXX	KEY_PAIR_NAME	default	running	

The `i-XXXXXXX` refers to your expected instance ID. Here, the real output will be your instance ID of the newly registered node. If the instance ID is not displayed, it means that there is some problem in the creation of EC2 instances.

Now, an EC2 instance has been created and configured with the Chef-client properly. You should know the procedure to add recipes on the EC2 node from your workstation, it is same as we discussed in *Chapter 3, Workstation Setup and Cookbook Creation*.

If you need more EC2 instances, you can repeat the EC2 bootstrapping procedure multiple times.

## Managing recipes on the new client node

After configuring the client, the next step would be to add recipes on it. These commands can be executed with the help of a management workstation.

You will need to download the sample cookbook (the Getting started recipe) from the Opscode community site, <https://supermarket.chef.io/cookbooks>.

The following are the commands for installing getting started recipe:

```
% cd ~/chef-repo
% knife cookbook site install getting-started
```

The output will be as follows:

```
Installing getting-started to /home/username/chef-repo/.chef/.../cookbooks
...
Cookbook getting-started version 0.3.0 successfully installed
```

After downloading, the next step is to upload recipes to the Hosted Chef, so that it will be accessible to our nodes:

```
% knife cookbook upload getting-started
```

The output is as follows:

```
Uploading getting-started [0.3.0]
upload complete
```

You need to add this recently uploaded recipe in the run-list of the new node:

```
% knife node run_list add NODENAME 'recipe[getting-started]'

run_list: recipe[getting-started]
```

The Chef-client is running, meaning that it will also run the getting started recipe. Now, the getting started recipe has added a template to the node in `~/chef-getting-started.txt`. You can verify the installed version of Chef by using the following command:

```
% cat ~/chef-getting-started.txt
```

It will show the following output:

```
Welcome to Chef!
This is Chef version 11.x.x
Running on ubuntu.
Version 12.04.
```

Here, the output depends on the data. Whatever you are going to store on your nodes. From the management console of the workstation, you can extract the code of the recipe after opening the following code in the text editor:

```
cd ~/chef-repo  
vi cookbooks/getting-started/recipes/default.rb
```

## Running the Chef-client as a daemon

It is recommended that you run the Chef-client as a daemon. Therefore, we don't need to perform SSH on nodes every time and we will get automatic updates. This can be set up with the use of the cookbook of the Chef-client. With this process, the Chef-client configuration will be performed as a daemon. The following command is used to add Chef-client as a daemon:

```
knife cookbook site install chef-client  
knife cookbook upload chef-client  
knife node run_list add NODENAME 'recipe[chef-client]'
```

After adding this, the Chef-client is needed to run on a node in order to configure the recipe to the server:

```
Knife ssh name:NODENAME -x ubuntu "sudo chef-client"
```

For your convenience, it is recommend that if the next time you perform bootstrapping for the new client, then you should include the cookbook of the Chef-client in the knife bootstrap command itself (as we did in preceding). Thereby, Chef-client will be added to the run-list of the node and will run automatically while installing Chef. This can be done by the following command:

```
knife ec2 server create -I ami-b89842d1 -x ubuntu -r 'recipe[chef-client]'
```

In summary, we can perform the following operations in AWS with Chef:

- Create base **Amazon Machine Images (AMIs)**
- Incorporate Chef into a cloud formation template
- Install and configure new server/application configurations
- Manage EC2 instances
- Manage EBS volumes and raid devices

For more information on knife-ec2, visit [https://docs.chef.io/plugin\\_knife\\_ec2.html](https://docs.chef.io/plugin_knife_ec2.html).

## Rackspace Cloud server bootstrapping

Here, we will manage Rackspace Cloud instances with the help of Hosted Chef, as we did for AWS EC2. The same infrastructure computing service is known as the Cloud server in Rackspace. We are familiar with most of the initial steps, therefore, we will include only the specific features that are mainly related to the Rackspace Cloud server settings.

### The prerequisite to work with Rackspace Cloud

The following are the prerequisites to start Chef with Rackspace Cloud:

- Login credentials for Rackspace Cloud
- Login credentials for Opscode
- Workstation setup and Chef Repository setup on your local system as guided in *Chapter 3, Workstation Setup and Cookbook Creation*

### Installing plugins for knife-rackspace

Here, we are assuming that you have already set up a workstation with the Chef repository, Opscode's Chef, and RubyGems.

With the help of RubyGems, the knife-rackspace plugin can be easily installed. You need to run the following command:

```
$ /opt/chef/embedded/bin/gem install knife-rackspace
```

Here, `/opt/chef/embedded/bin/` is the location where the knife plugin must be located.

Now, we need to customize the knife configuration file in order to interact with the Rackspace Cloud API.

### Preparing the workstation with Rackspace credentials

After installing the knife-rackspace plugin, you need to add the following things in the `~/chef-repo/.chef/knife.rb` file:

```
knife(  
  :rackspace_api_key => "Cloud API key of your Rackspace account",
```

```
:rackspace_api_username => "Cloud username of Rackspace"  
)
```

We can now prepare cookbooks or download them from GitHub. After preparing cookbooks, we need to upload them to the Opscode server by giving the following command:

```
knife cookbook site vendor chef --dependencies  
rakeupload_cookbooks
```

Now, the next step is to learn how to bootstrap Rackspace Cloud with the Chef-client demon.

## Bootstrapping the Rackspace Cloud server with the Chef-client

As we know, running the Chef-client as a daemon is the best practice to install the Chef-client.

In order to communicate with the Opscode server, the Chef-client must run like a persistent daemon that talks to the Chef server in every 30 minutes. Therefore, you will need to create a `.rb` file in the Chef repository, for example, `chefsample.rb`. The following is the exact path to create the `chefsample.rb` file:

```
~/chef-repo/roles/chefsample.rb  
name "chefsample"  
description "chef node Bootstrapping "  
run_list("recipe[chef::bootstrap_client]")  
default_attributes "chef" =>{ "server_url" => "https://api.chef.io/  
organizations/ORGNAME" }
```

Here, `ORGNAME` will be your organization name. Now, you need to run these commands:

```
cd ~/chef-repo  
rake roles  
knife rackspace server create 'role[chefsample]' --server-name chefsample  
--image 49 --flavor 2
```

In the previous code, `image 49` refers to the Ubuntu version 10.0.04 (LTS), Flavor refers to the different size of the server. There are different flavors, where 2 means 20 GB hard disk with 512 RAM for 0.03 \$/hr.

We can choose any type of flavor according to our requirement.

## **Deleting Rackspace servers**

The following command is used for deletion of a Rackspace server:

```
Knife rackspace server delete <ID>
```

Here, `ID` will be your Rackspace server ID.

The server ID can be discovered by the following command:

```
Knife rackspace server list
```

## **The Knife-cloud plugin**

Opscode's Chef also provides various Cloud compatible plugins that can be easily integrated with the `knife` command. Currently, Chef supports most of the popular Cloud infrastructure services. It also supports private Cloud services, such as Eucalyptus and OpenStack.

Here is the list of the Knife plugins that are maintained by Chef:

- Knife-azure
- Knife-bluebox
- Knife-ec2
- Knife-eucalyptus
- Knife-google
- Knife-hp
- Knife-linode
- knife-rackspace
- knife-openstack
- knife-terremark
- Knife-windows

## **VMware and Chef**

Chef can also be used to configure and manage VMware infrastructure. We can use Chef to provision and deprovision VMs with VMware vCenter. Chef's `knife-vsphere` plugin is available at <https://github.com/chef-partners/knife-vsphere>. It allows you to integrate Chef with the existing vSphere Client installation. The `knife-vsphere` plugin can be used to list out folders, datastores, resource pools and clusters, hosts in a pool or cluster, templates, VMs, VLANs, and customization specifications.

It allows the facility to Power on/off VMs, clone VMs, migrate VMs, delete VMs, and connect/disconnect network.

The Chef's Knife plugin, knife-vcair is available for VMware's vCloud Hybrid Service (vcair). It gives the ability to create, bootstrap, and manage instances on vcair-based public and private cloud deployment models. We can also list vCloud Air templates or images, deployed vCloud Air VMs and vApps, networks, NAT rules, firewall rules, and so on. Details are available at <https://github.com/chef-partners/knife-vcair>.

## **Self-test questions**

1. Why is Chef more preferred to manage a Cloud infrastructure?
2. While creating a security group in AWS, why do we give source as 0.0.0.0/0 in the inbound traffic?
3. What is the first recipe that needs to be run immediately just after the Chef-client is installed in the node?
4. Where can we see all the available options of the knife-ec2 plugin, which can be added in the knife.rbconfig file?
5. List all the operations that can be performed on AWS with the use of Chef.
6. What is the meaning of the flavor list argument in the Rackspace server?
7. List out the knife plugins maintained by Chef

## **Summary**

In this chapter, we learned the necessary information regarding Cloud infrastructure services and we learned about how Chef can be used on a Cloud infrastructure to manage the automation process. Chef installing and bootstrapping procedures on Cloud instances of AWS EC2 and the Rackspace Cloud server have been explained in a detailed, step-by-step manner. We can also simplify our work by using the appropriate knife-cloud plugin according to our Cloud requirement.

In the next chapter, we are going to learn about the industry standard best practices of Chef.

# 9

## Best Practices while Using Chef

*"In this electronic age we see ourselves being translated more and more into the form of information, moving toward the technological extension of consciousness."*

- Marshall McLuhan

This chapter explores the industry's standard best practices for optimal and effective utilization of Chef. It sheds light on Chef's anti-patterns and patterns. As we know that testing is an important phase of a software project's life cycle to ensure quality, the same principle applies to Chef as well. There are various ways to test cookbooks. They give details of all the types of tests taken by Chef. As best practices are always evolving and come from people's experiences, this chapter provides tips for effective utilization of Chef in our industry and organization.

We are going to cover the following topics:

- Chef's anti-patterns and patterns
- Testing of cookbooks
- Best practices for effective usage of Chef

### Chef anti-patterns and patterns

In our daily life, we often face quite a few chronic challenging situations. To solve these recurring problems, we often adopt the same type of solution without understanding its long term effects. This situation results in patterns of failure.

According to the book *Anti-patterns: Refactoring Software, Architectures, and Projects in Crisis*, anti-patterns are repeated practices in a software architecture, software design, and software project management that initially appear to be beneficial, but ultimately result in bad consequences that outweigh the hoped-for advantages.

For example, in programming, we often come across various anti-patterns, such as introducing unnecessary complexity in a solution, retaining functionality of a system that no longer has any use, hardcoding assumptions about an environment of a system in its implementation, and so on. In configuration management, there are various anti-patterns, such as dependency hell. While using Chef automation in organizations or industries, when one observes some problem or difficulty with any operations of Chef, and if its evident solution has side effects, then it comes under the category of Chef anti-patterns. The existing best solution of these problems are known as Chef patterns.

Chef anti-patterns and patterns are ever-evolving processes as they are the outcomes of user experiences. In this section, we are going to see some popular Chef anti-patterns and patterns that have come under observation until now.

The following framework describes Chef anti-patterns and patterns:

Parameter	Description
Pattern name	This is a unique name that helps to identify and refer to a pattern
Problem	This is the problem scenario
Intent/usual solution or pattern	This is a description of the goal behind the pattern and the reason to use it
Side effect/consequences	These are the side effects and consequences because of the usage of the usual solution results in an anti-pattern
Structure	This refers to the graphical representation of a pattern
Anti-pattern solution and implementation	This is the description of the implementation of a pattern; the solution part of the pattern
Also known as	Another name of pattern if any

## A wrapper cookbook

Wrapper cookbook is used to customize an existing library cookbook. Following table describes more details on wrapper cookbook:

Parameter	Description
Pattern name	A wrapper cookbook
Problem	There are more than 130 cookbooks provided by the Opscode community ( <a href="https://supermarket.chef.io/cookbooks">https://supermarket.chef.io/cookbooks</a> ). Anyone can download and add changes in the <code>metadata.rb</code> file. You can also add more recipes, various attributes, and LWRP. The problem is how to manage these changes?
Intent/Usual solution or pattern	To fork community cookbooks
Side effects/consequences	<p>Now, the problem is that when the cookbook is under the deployment state in GitHub, you can get merged conflicts. Hence it may not be possible to get benefit of upstream bug fixes and enhancements</p> <p>Consider a scenario of production environment, where you have the role of one <code>web_server</code> that has the properties of two websites, which are going to host on that web server and attributes for names. Now, we need to create distinct roles for each website such as <code>web_server</code>, and <code>app_server</code>. In that case, how would you test your development environment without affecting the production environment? Either you would have to make the changes again in all environments or you would have to override those attributes in all environment starting from development, testing and staging followed by cleaning up those attributes after reaching production.</p> <p>Another side effect arises, if you wish to share the changed cookbook with the community, then you need to create a feature branch with the changes in the Git repository. It then needs to perform a pull request.</p>
Structure	<p>The diagram illustrates the structure of a Wrapper Cookbook. It features two overlapping circles: a green circle labeled "org_tomcat" and an orange circle labeled "tomcat". Two arrows point from these circles to separate blue boxes. One arrow points to a blue box containing the text "Wrapper Cookbook from with custom modifications or additions". The other arrow points to a blue box containing the text "Existing Cookbook from Chef community".</p>

Parameter	Description
Anti-pattern solution and implementation	<p>The solution for anti-pattern is that, rather than using the Opscode community cookbook or upstream community cookbook, we should use an organization- or company-specific cookbook. This method is known as a wrapper cookbook. To use a wrapper cookbook, all you need to do is add metadata dependency on the community cookbook and add <code>include_recipe</code> in order to run recipes. We can override the default configuration of a community cookbook by setting customized attributes according to the requirement of our organization in our wrapper cookbook.</p> <p>Therefore, it is recommended that we set custom attributes inside the wrapper cookbook because cookbooks are versioned, but roles are not versioned. Using custom attributes, we can point and set our specific environment, such as testing, development or production of a specific version of a cookbook.</p> <p>For reference and more examples of wrapper patterns, visit <a href="https://www.chef.io/blog/2013/12/03/doing-wrapper-cookbooks-right/">https://www.chef.io/blog/2013/12/03/doing-wrapper-cookbooks-right/</a>.</p>
Also known as	An application/library cookbook

## A default cookbook

Default cookbook installs and configures any prerequisite utilities. Following table describes more details on default cookbook:

Parameter	Description
Pattern name	A default cookbook
Problem	In order to build the server from scratch, a run-list of all the required recipes must be specified in a particular role, such as creating a <code>web_server</code> role. <code>name: app_server</code> <code>description: Role of web servers</code>

Parameter	Description
Intent/Usual solution or pattern	<p>Roles are suited to denote the run-list of all the recipes needed to build the server:</p> <pre>name "app_server" description "Role of app servers" run_list(   "recipe[base_server::disk_configuration]",   "recipe[base_server::dhcp_reservation]",   "recipe[base_server::pagefile]",   "recipe[app_server::web_sites]",   "recipe[base_server::ssl_certs]",   recipe[utility::install_tools]" )</pre>
Side effects/consequences	This means that removing or adding recipes from the run-list is going to affect all the servers that have that role. Therefore, this may affect all the servers of the production environment as well
Structure	<p>For example, a cookbook of an application contains recipes of a web server and application server. While building such a type of application server, the following should be your role:</p> <pre>name "app_server" description "Role of app servers" run_list(   "role[base]",   "recipe{app_server}" )</pre> <p>The default recipe of an application cookbook containing the app_server role will be as follows:</p> <pre># app_server cookbook recipes/default.rb  include_recipe "base_server::disk_configuration", include_recipe "base_server::dhcp_reservation", include_recipe "base_server::pagefile", include_recipe "app_server::web_sites", include_recipe "base_server::ssl_certs", include_recipe "utility::install_tools"</pre>
Anti-pattern solution and implementation	Try to keep the role as lightweight as possible. Instead of declaring roles, a list of recipes must be included in the cookbook.

In short, we can summarize the preceding framework in the following table:

Chef Anti-pattern	Chef Pattern
Versioning of an upstream cookbook	Creation of a wrapper cookbook
Applying more role attributes	Usage of custom attributes in a wrapper cookbook
A role run-list contains a long list of specific recipes	A role run-list contains a list of recipes in the wrapper cookbook

## Testing cookbooks

Before making the configuration changes in the production environment, the best practice is to perform proper tests on cookbooks in all lower environments.

### Types of cookbook tests

There are different types of cookbook tests that must be performed at various stages.

#### Checking the syntax

To test syntax errors in a cookbook, a test argument is used. A Ruby syntax check is used by this argument for the verification of each file in a cookbook that finishes in .rb and **Embedded Ruby (ERB)**.

The following is the command for testing a cookbook's syntax:

```
$ knife cookbook test COOKBOOK_NAME
```

#### Integration testing

Integration testing is an essential type of testing in Chef. There are some popular testing tools used for integration testing in Chef.

**Minitest** is a testing tool for integration, which lets you to make assertions regarding the final condition of the machine after the run of the cookbook. This should have a clean machine to run your cookbooks on to get the desired results.

The following is an example of Minitest integration testing:

```
describe 'mysql::server' do
  it 'runs as a daemon' do
    service(node['mysql']['service_name']).must_be_running
  end
end
```

The preceding example will check whether MySQL is running, provided that the attribute is populated.

**Cucumber Chef** is a testing tool for integration, which utilizes a compatible way to read and write syntax for further describing scenarios for testing and infrastructure. It lets all utilize Gherkin syntax to define infrastructure and test the output result of your recipes.

The following is an example of Cucumber Chef integration testing:

Suppose that the Chef-client is running as a daemon; then, the following process should be run as an output:

```
run "ps aux | grep chef-client"

Then check for "chef-client" in the o/p
Then check for "-d" in the o/p
Then check for "-i 1800" in the o/p
Then check for "-s 20" in the o/p
```

If the previous processes are running fine, it means that the chef-client is running.

## Checking the result

In order to check what changes have been made by a cookbook on a particular node, Chef why-run mode is used. However, this facility is available only in Chef 10.14 and higher versions.

## Checking the consistency

Foodcritic is a linting tool that is used to check the consistency of code is maintained and it also ensures industry best practices. There are two additional sets of rules that you can add to it, **etsy** and **customink**; more details about these two rules are available at <https://github.com/etsy/foodcritic-rules> and <https://github.com/customink-webops/foodcritic-rules>:

```
$ location_of_custom_rules
$ cookbook_path/$cookbook_name
```

## Checking the performance

Test kitchen is a very good tool used to check a cookbook's performance on different platforms and versions.

Test kitchen easily runs your cookbooks against a few clean VirtualBox nodes that are running on different versions of different operating systems. It does not have any external verification, but you can use Minitest to get it.

Using this test kitchen, it is very easy to know the outcomes. It provides great validation of cookbook behavior before running in live environments. It may speed up a bunch of different platforms for testing your cookbook on separate operating systems.

## **Best practices for effective usage of Chef**

In this section, we are going to discuss the key points of best practices, which we should keep in mind while developing cookbooks.

### **Planning in advance**

It is recommended that before planning Chef deployment for your organization or industry, you must plan the following things in advance:

- All the cookbooks you are going to use
- All the recipes you are going to use
- Roles and their names
- The number of environments you are going to use
- If there is any clustering going to be used within the environments
- Naming conventions and the hierarchy of data bag items

### **Designing a cookbook**

We can design efficient cookbooks if we take care of the following points:

- It is recommended that while designing a cookbook, we should use all the dependencies of a cookbook appropriately and include recipes.
- We should try to add thin roles, as many as possible. Basically, roles are just names, they get executed only when a run-list is attached to those names.
- If it is mandatory to add roles, then try to add only attributes.
- We should try to use wrapper cookbooks, as using wrapper cookbooks enables us to customize the settings of upstream cookbooks without any forking.

## Using a private recipe

A private recipe is an easy convention to utilize an underscore to prefix a recipe name, for example, `recipes/common_setup.rb`. Instructions to create a private recipe are:

- It is meant for code separation (logical)
- It is included only in recipes from the same cookbook
- It is not to be included directly in the run-list

## Avoiding the use of one giant cookbook

It is not recommended that you use only a single giant cookbook for your organization or company. As configuration of a cookbook is a high-priority service in Chef, there is a possibility of mixing other things or data with that cookbook. There is a greater possibility of misconfiguration when a scenario of a quick change and deployment appears.

The following pattern should be followed while designing a cookbook for any organization:

```
+cookbooks
+ your_organization_name
+ Recipes
+mainsite-apache-virtualhost.rb
+anothersite-apache-virtualhost.rb
+anothersite1-apache-virtualhost.rb
+spring-properties.rb
```

In the previous example, there is a hierarchy of cookbook ordering; we are not using one giant cookbook for all operations.

## Avoid overloading of a Chef environment

An environment concept in Chef is purely logical. You should not use a lot of environment declarations in your cookbook.

For example, if we are using more Couchdb database clusters in our production environment, we generally use the following:

```
node.set ['couchdb'][cluster_name] = 'couchcluster1'
Couchs = search (:node, "role: couchdb AND
Chef_environments :#{node.chef_environment} AND
couchdb.cluster_name =#{node['couchdb']['cluster_name']}")
```

However, the preceding code is not the best practice. If we are using more than one Couchdb clusters in our production environment, then we should code it as follows:

```
node.set['warehouse']['data_center'] = 'data_center_name'  
node.set ['couchdb'] [cluster_name''] = 'couchcluster1'  
Couchs = search (:node, "role: couchdb AND  
Chef_environments :#{node.chef_environment } AND  
mongodb.cluster_name =#{node ['couchdb'] ['cluster_name']} AND  
warehouse.data_center=#{node['warehouse'] ['data_center']}")
```

Now, we have the names of different data centers in one type of environment (here, this is the production environment) and in each data center, we have different clusters, which can be named as cluster 1, cluster 2, and so on.

## Self-test questions

1. What is the command for testing a cookbook's syntax?
2. To use a wrapper cookbook, what are the things that you need to define?
3. What are the instructions to use a private recipe?
4. What is the purpose of a test kitchen?
5. Why should you not use a giant cookbook for an organization?

## Summary

In this chapter, we got a clear understanding of Chef anti-patterns and how to use Chef patterns.

However, Chef anti-patterns and patterns are being evolved and it is quite possible that system administrators and infrastructure automation professionals may use different patterns according to their own skills and experience.

We have got an initial idea of the various types of Chef testing, such as a syntax check, performance testing, integration testing tools, consistency checking, and result checking. These testing tools and techniques will be very useful to ensure the quality before production. Apart from this, we also learned about some concrete tips for effective utilization of Chef in our organization.

In the next chapter, we are going to see a variety of use cases of Chef deployment and how various industries are being benefited benefitting by the use of Chef.

# 10

## Case Studies on Different Chef Deployments

This chapter describes the various successful Chef deployment case studies of different organizations and industries. These case studies bring a variety of typical case scenarios, so that you can think and design the Chef deployment for any new organization. This chapter will help you to understand how Chef is currently used.

In this final chapter, we are going to study some successful stories of Chef implementation in organizations. We are also going to see their challenges and how they come up with solutions. Extensive details are available at <https://www.chef.io/customers/>. We have provided sample diagrams based on the description available on the Chef website.

As we have learned from the previous chapters, there are various types of Chef deployments, such as Hosted, Private and open source Chef. They provide a variety of choices to all those who are planning to automate with Chef in their respective organizations. Usually, the question is not whether the Chef deployment is good or bad; it indeed depends upon your requirement and use cases. Before planning to choose any type of Chef deployment model, first, we should identify the requirement clearly. Also, the architecture and budget of our existing infrastructure must be identified before the deployment.

In this chapter, we will learn from different case studies of a variety of industries that are reaping the benefits of the Chef implementation. After going through all these case studies, we would get a clear picture in our mind about which type of Chef deployment model will be suitable for our requirement. In this way, we will attain the approach to implement Chef in our organization. Following are the categories of case studies covered:

- Case studies of the Hosted Chef deployment

### *Case Studies on Different Chef Deployments*

---

- Case studies of the Private Chef deployment
- Case study of the Open Source Chef deployment
- Case study of the Chef-solo deployment

For better understanding and to have a crystal clear vision of each of these case studies, we will follow the pattern shown in the following figure:



## **Case studies of Hosted Chef deployments**

We found that many start-up and mid-size organizations prefer the Hosted Chef deployment for their organization. Let's see some of the successful, industry case studies of the Hosted Chef deployment.

### **Admeld**

Admeld is a leading company located in New York that provides advertising network services. Their average advertisement transactions are around 40 billion every month. Hosted Chef made their infrastructure automation process on an on demand basis.

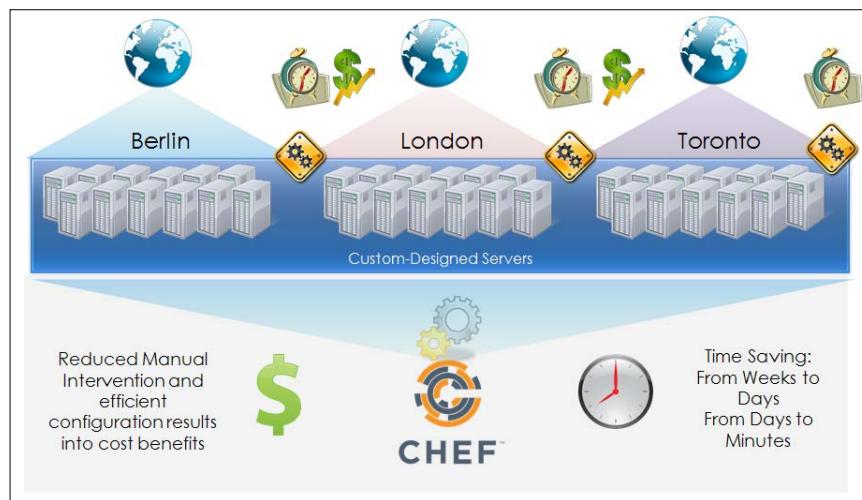
### **Challenges with the infrastructure of Admeld**

The headquarters of Admeld is located in New York and has six data centers across the globe. The company makes around 40 billion transactions per month. All the designing and configuration of every server of all the data centers was done manually, which involved lots of human interaction and efforts.

Manual updates for each and every server made it difficult to manage its efficiency and maintenance. As the number of servers grew in number, these manual efforts were also increased.

## The solution with Hosted Chef

Opscode's Hosted Chef provides hosted software as a service platform for configuration management. Admeld's technical team decided to go with Hosted Chef for this challenging problem. Hosted Chef acted as a central location for all the installation components, updates, and patches for all the nodes. Hosted Chef provides an easily manageable interface that can be quickly scaled on demand basis. Any new server gets ready with the help of a few clicks and in a few minutes of time. Following is the graphical representation of requirements, implementations, and benefits achieved by Chef implementation:



## The final outcome

After the successful Hosted Chef deployment in Admeld, the following benefits were achieved:

- Focus on innovation rather than maintenance
- Time consumption for server readiness reduced from weeks to just 15-20 minutes
- Increase in agility and assurance of true scalability

For more details on Admeld's case study, visit <https://www.chef.io/customers/admeld/>.

## Fanhattan

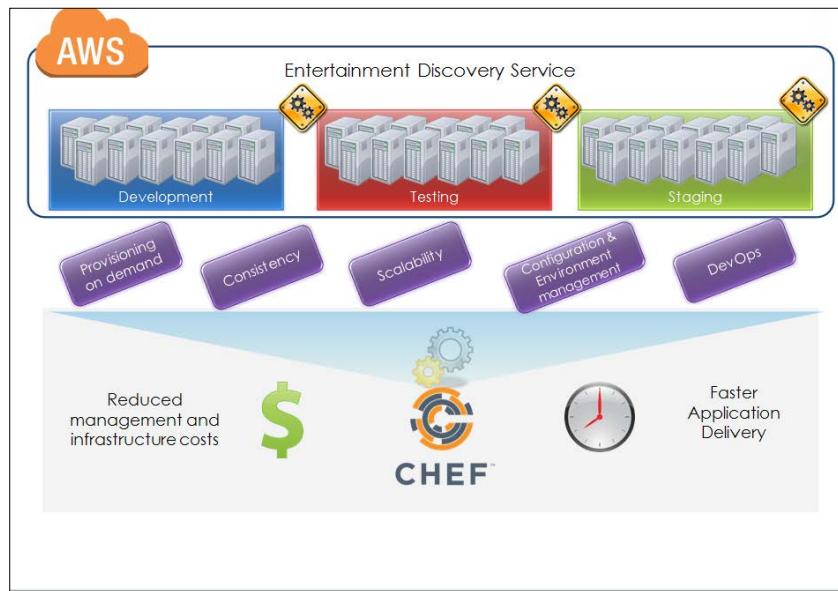
Fanhattan is an organization that has redefined entertainment discovery. It is located in San Mateo, CA. They have remarkably increased their productivity by 10X automating their Amazon EC2 Cloud with Hosted Chef.

### Challenges with the infrastructure of Fanhattan

Fanhattan, a leader in providing services, follows parallelism for the development, engineering, and testing processes, as all the processes happen simultaneously. To maintain consistency across all the environments while maintaining the scalability was the critical challenge that Fanhattan faced. The foremost requirement of Fanhattan was the acceleration of the application deployment process with optimal cost and resource.

### The solution with Hosted Chef

Fanhattan deployed Opscode's Hosted Chef that automates almost everything from basic configurations to application deliveries. Productivity now accelerated to a higher extent around, ten times, and the time take for application was reduced along with the optimal cost of operation. As we can see in following diagram that development, testing and staging environments are being managed on AWS cloud and automated using Hosted Chef:



## The final outcome

After the successful Hosted Chef deployment in Fanhattan, the following benefits were achieved:

- Hosted Chef deployment now provides best practices for business agility
- It has reduced the significant time in resource configuration, cost enabling management and operational work together
- The production process is faster now

For more details on the Fanhattan case study, visit <https://www.chef.io/customers/fanhattan>.

## Zumba Fitness

Zumba Fitness is the world's largest dance fitness group. It is located in Hallandale, Florida. Zumba has increased the fitness program flexibility and thus has a better management. Zumba classes are located in more than 125 countries. Zumba automated its Amazon EC2 and Rackspace Cloud infrastructure with Hosted Chef challenges with the infrastructure of Zumba.

## Challenges with the infrastructure of Zumba Fitness

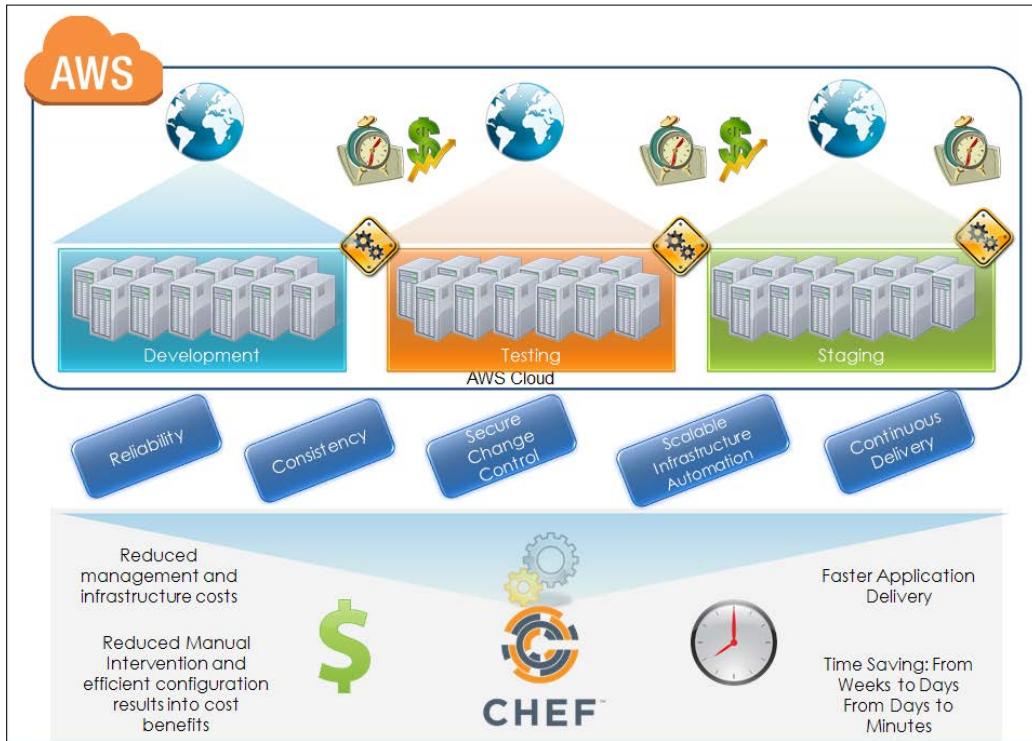
Zumba Fitness is one of the biggest and most successful dance party programs. Weekly, the number of people attending the classes across 110,000 locations is 12 million. The Zumba website gives the service of handles entire business operations. The challenge is to maintain consistency and secure control with less time consumption and efficient management.

## The solution with Hosted Chef

Zumba Fitness deployed Opscode's Hosted Chef for automation purposes to manage the whole cloud infrastructure, from the development environment to changing the control management in its Amazon EC2 and Rackspace clouds. The manual efforts were now eliminated as the high-traffic management site is now automated. Every component of both its Amazon and Rackspace clouds is automated now.

### *Case Studies on Different Chef Deployments*

As we can see in following diagram that development, testing and staging environments are being managed on AWS cloud and automated using Hosted Chef:



## **The final outcome**

After the successful Hosted Chef deployment in Zumba, the following benefits are achieved:

- The configuration control is much more efficient with optimal management of resources
- It saves time as the management process is compressed from weeks to days
- It now has a secured system with the automation process, as well as there is large saving of resources by the automation of resource management

For more details on the Zumba Fitness case study, visit <https://www.chef.io/customers/zumba>.

## The Limelight video platform

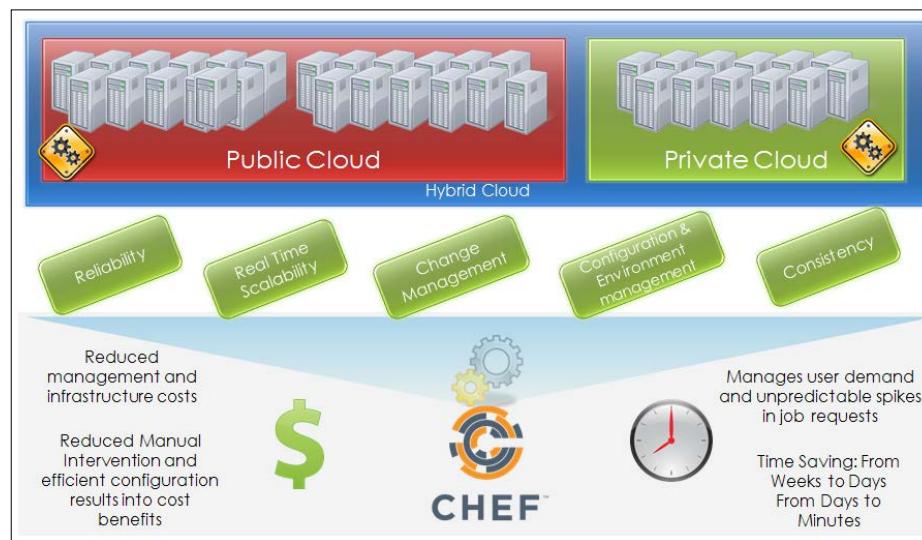
The Limelight video platform does management in the field of entertainment as it is involved in the analysis of high-quality video on the Web. It is located in Seattle, Washington. It deploys Hosted Chef to automate its infrastructure (data center and hybrid cloud).

### Challenges with the infrastructure of Limelight

The Limelight video platform is specifically used for management and analysis of high-quality video on the Internet. It is a part of the leading company Limelight Networks. A large number of components are there in existing infrastructure that are necessary to provide its specified service. So, the problem is to maintain the consistency in this architecture that contains a huge number of components, such as new software upgrades, patch installations, and configurations.

### The solution with Hosted Chef

The Limelight video platform was deployed by Opscode's Hosted Chef for the purpose of automation in the world of configuration management. With the help of a single click, the scalability can grow at a good pace as Hosted Chef enables real-time scalability by decreasing the chances of manual errors in the infrastructure configuration. Across the service-oriented architecture, both efficiency, and consistency of resource configuration increase simultaneously.



## The final outcome

After the successful Hosted Chef deployment in Limelight, the following benefits are achieved:

- From data management to real-time analytics, Limelight increases the value of video content online
- Better infrastructure management and output high quality video output and quick response time of process from testing to production after deploying Hosted Chef
- Ensures that the scalability increases with accelerating efficiency in the deployment process and productivity with the reduced risk of misconfigurations

For more details on the Limelight case study, visit <http://www.prnewswire.com/news-releases/limelight-networks-orchestrates-brilliance-in-online-media-delivery-with-opscode-166732096.html>.

## Imagination

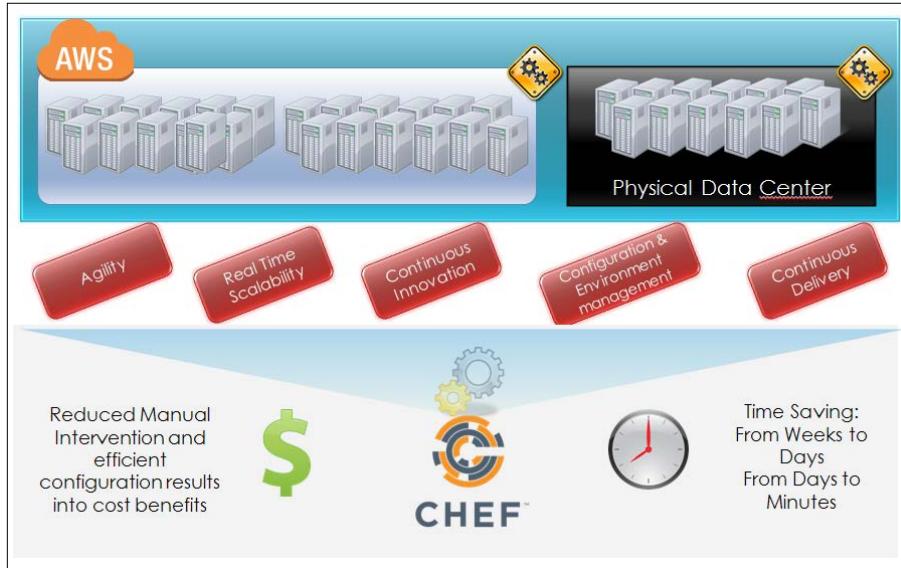
Imagination is one of the leading creative agencies. It is located in London. To accelerate productivity, Imagination adapted automation by deploying its hybrid event infrastructure with new Hosted Chef.

## Challenges with the infrastructure of Imagination

Imagination is globally a leader among the creative agencies that provides a wide range of communications, media services in digital terms to spread services for the leading brand names, such as Canon, Ford, and many more on the list. Management of the sophisticated infrastructure with an increased speed and less time is a critical measure that needs to be taken by Imagination. Imagination was facing a barrier between devolvement (Dev) and operations (Ops) in terms of agility and rapidity.

## The solution with Hosted Chef

Imagination utilizes Hosted Chef, which automates configuration management, resource management, and metadata management with the delivery of the data intensive applications. Deployment of Hosted Chef increases the time efficiency by elimination of manual efforts in the infrastructure management.



## The final outcome

After the successful Hosted Chef deployment in Imagination, the following benefits were achieved:

- The whole development and operational stack is under automation control that provides the best of all practices.
- The team can focus on the best way to implement new features by innovating new ideas rather than managing the deployment and configuration process for the entire infrastructure. Hosted Chef allows Imagination to adapt without having any impact on the services that it provides, which in return, ensures maximum flexibility.

For more details on the Imagination case study, visit <https://www.chef.io/customers/imagination>.

## Getaroom

Getaroom is a famous online company for booking rooms in hotels (<http://getaroom.com/>). Getaroom provides services in the major destinations of the United States.

## Challenges with the infrastructure of Getaroom

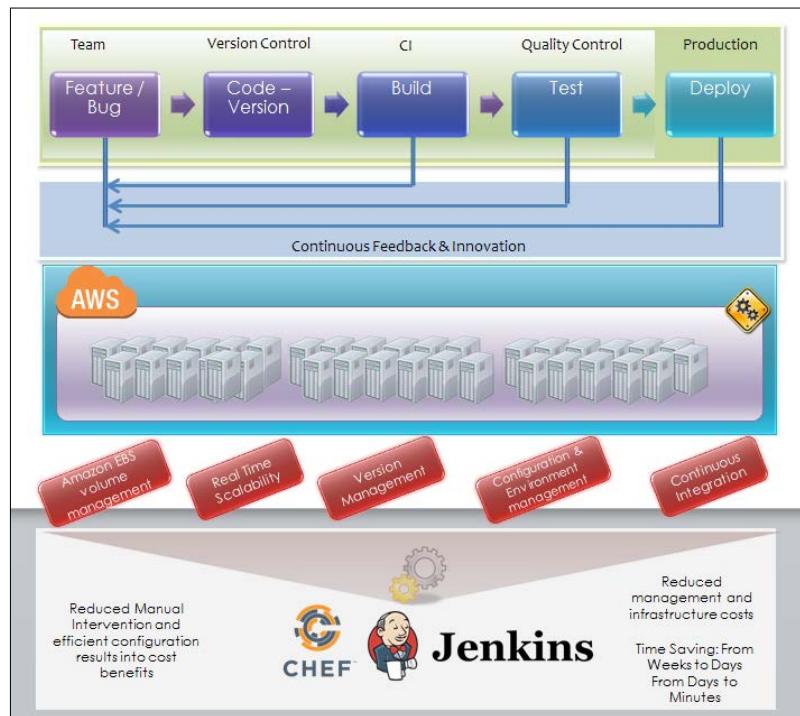
Getaroom is currently using a public cloud service of Amazon EC2. Approximately, more than 100 EC2 instances are currently used by Getaroom.

There are a variety of server roles and the management of each type of role is quite complicated due to the multiple versions of Ruby in different environments, such as preprod, prod, dev, and testing. Apart from this, managing a number of accounts in the development and operation group is also a critical and time-consuming task.

## The solution with Hosted Chef

By using the Chef Community's cookbook feature for **Ruby Version Management (RVM)**, Getaroom's technical team was able to manage the version problem of Ruby in all the different types of environments.

After deploying the Hosted Chef solution, Getaroom also made user authentication an easy process in the production and development environment. They facilitated Hosted Chef as a data center repository to store user information, so that they can manage user accounts easily in the entire infrastructure.



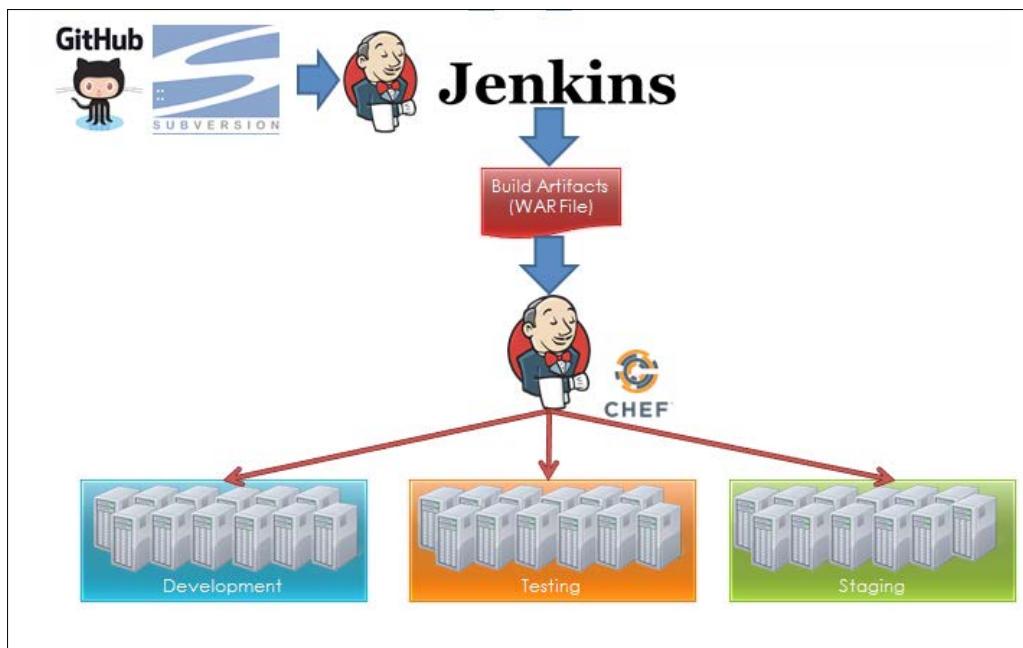
## The final outcome

After the successful Hosted Chef deployment in Getaroom, the following benefits were achieved:

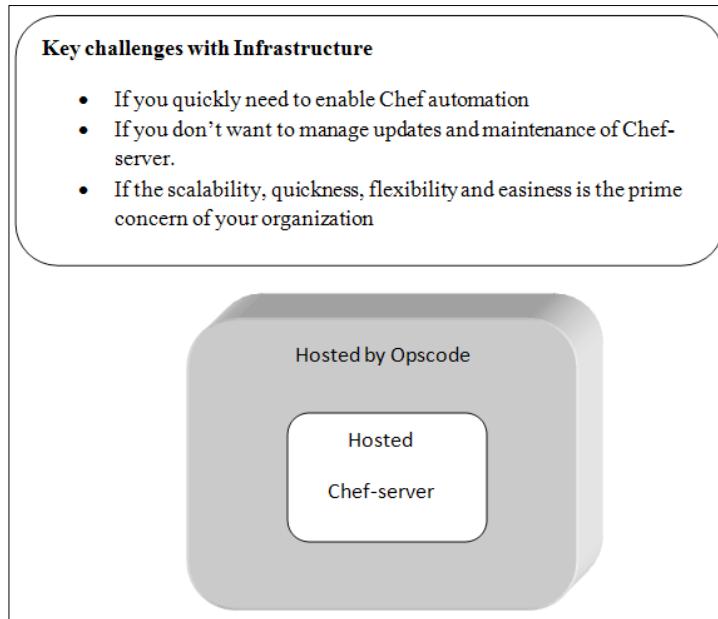
- With the help of Hosted Chef, Getroom's cloud infrastructure looks like any traditional software, but with the Hosted Chef solution, we can configure resources once and replicate as many times as required according to the new environment
- It was able to build highly consistent and repeatable infrastructure with minimal risk
- It automated the entire infrastructure requirement, including Jenkins, MongoDB, MySQL, and HAProxy.

For more details on the Getaroom case study, visit <https://www.chef.io/customers/getaroom>.

In general, Jenkins can be used for continuous integration and Chef can be used for continuous delivery, as shown in the following screenshot:



The following diagram explains the main challenges addressed by Hosted Chef:



## **Case studies of Private Chef deployment**

In this section, we are going to see a few case studies of the Private Chef deployment. The Private Chef deployment is very specific and related to a special requirement of the infrastructure.

### **Ancestry.com**

Ancestry Inc (<http://www.ancestry.com/>) is located in Utah, US. It has 1.8 million subscribers throughout the world. It helps many users to search and share their family history with a number of different options for all types of users.

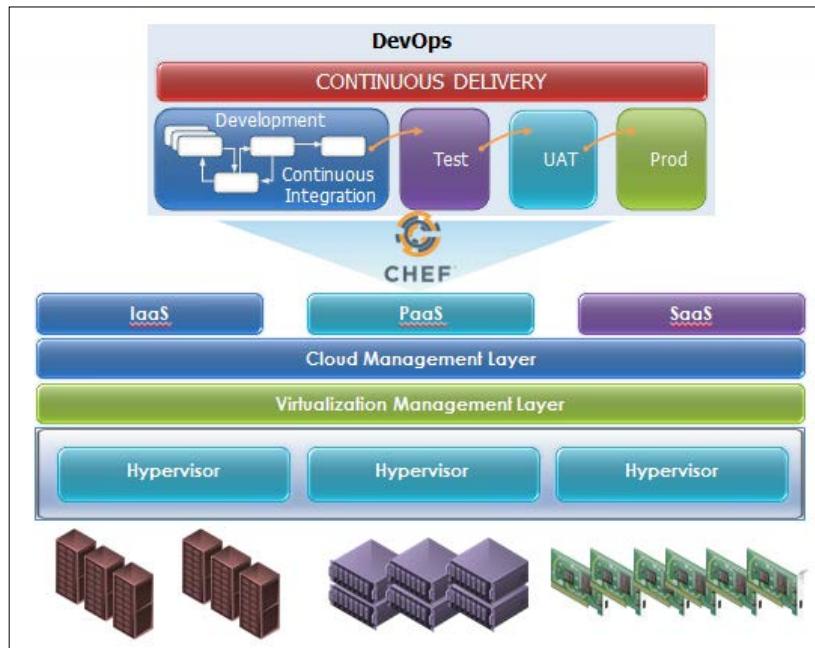
### **Challenges with the infrastructure of Ancestry.com**

In order to serve quickly to all their subscribers, [www.ancestry.com](http://www.ancestry.com) facilitates thousands of servers across many data centers. They work on service-oriented architecture with critical management scenario. Their business has grown so rapidly that they need to develop, test, and deploy the applications quickly in the growing infrastructure. Opscode's Private Chef provides fast infrastructure automation solution to <http://www.ancestry.com/>.

## The solution with Private Chef

In order to enable the DevOps methodology, continuous improvement and quick configuration management, [www.ancestry.com](http://www.ancestry.com) selected the Private Chef deployment in their data centers. Their approach leverages the special features of the Private Chef deployment where the center automation server is placed behind the firewall (on-premises).

After selecting the Private Chef deployment, configuration management data, including applications and resource usage information, now has a high accuracy and is even described with deep detail. In execution, the development with [www.ancestry.com](http://www.ancestry.com) is now specifically more improved, which leads to better utilization of the existing resources that helps in bringing new innovative techniques to offered services.



## The final outcome

After the successful Private Chef deployment in [www.ancestry.com](http://www.ancestry.com), the following benefits are achieved:

- A new service launch in the market is done in a few days rather than weeks
- Application deployment has become quite easy and fast

- The new server becomes ready with a few clicks
- The Agile development and DevOps methodology is enabled successfully

For more details on the <http://www.ancestry.com> case study, visit <https://www.chef.io/customers/ancestry>.

## **Facebook**

Facebook is one of the most popular web-based social networking service site, which is actively working for registered users. It was founded on February 4, 2004. It is headquartered at Menlo Park, California, US. It was created by Mark Zuckerberg (who is among the 100 wealthiest people of the world) with his college roommates and Harvard University students.

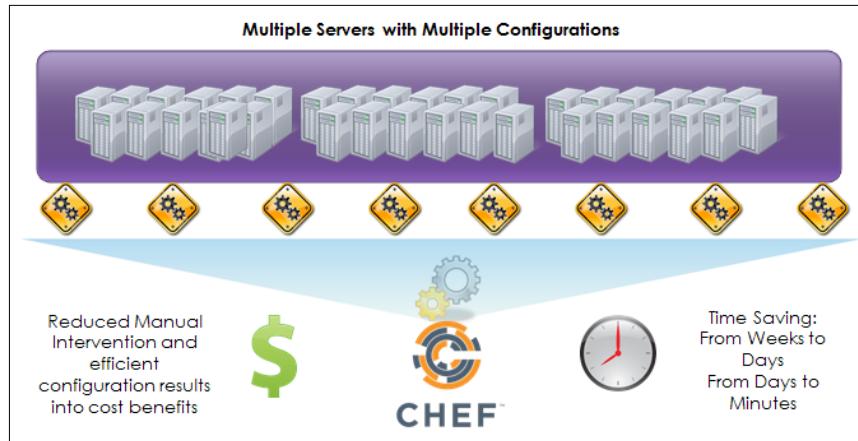
### **Challenges with the infrastructure of Facebook**

The infrastructure engineers of Facebook had to face the criticality of configuration management, administration and policy distribution across thousands of servers. Facebook's infrastructure team has to manage thousands of servers, configurations, and administrative access policies across a very dynamic computing environment.

### **The solution with Private Chef**

Facebook, the biggest social networking company, selected Opscode's Private Chef for their large infrastructure automation tasks across their multiple data centers. A huge network that connects all the users worldwide is now remarkably and efficiently managed.

Using Chef, Facebook was able to manage a quite flexible and huge system with a variety of different configurations. In order to manage the overall operation, they only require a very small team due to the Chef automation and configuration management.



## The final outcome

After the successful Private Chef deployment in Facebook, the following benefits were achieved:

- Facebook built easy automation, fast configurations, and flexible and secured administrative access policies across thousands of servers
- Private Chef provided flexible automation scenario without changing the existing workflow

For more details on the Facebook case study, visit <https://www.chef.io/customers/facebook/>.

## DreamHost

DreamHost is a web hosting and domain name register owned by the New Dream Network, LLC company located in Los Angeles.

In 2012, Dream Host and Inktank worked together so that they could launch new services in the cloud storage. The service was named as Dream Objects. It was made open for public in 2012.

## Challenges with the infrastructure of DreamHost

DreamHost is a global service provider of cloud solutions and web hosting. They wanted to build their offered services using open source products, such as the open source Ceph storage, to provide storage as a service, leveraging OpenStack to provide private cloud services. Among the multiple services offered on the large scale infrastructure, security and privacy are the top concerns for DreamHost.

## The solution with Private Chef

Private Chef provides high scalability with faster management. It empowers us to deliver all the unmatched services with negligible manual efforts. This automated system almost eliminates the risk of manual errors that leads to a secured infrastructure. It also gives the power of reusability of all the infrastructure components.

Private Chef has provided a management that requires almost negligible human efforts. Now, only a small number of employees are enough to manage virtual and physical infrastructure. This remarkable reduction of cost and the improved management accelerates the innovation in the system, which in turn increases the efficiency. Private Chef here provides a very easy approach. It provides automated framework for Ceph and OpenStack. It gives full management control of the delivery quality services in much less time.



## The final outcome

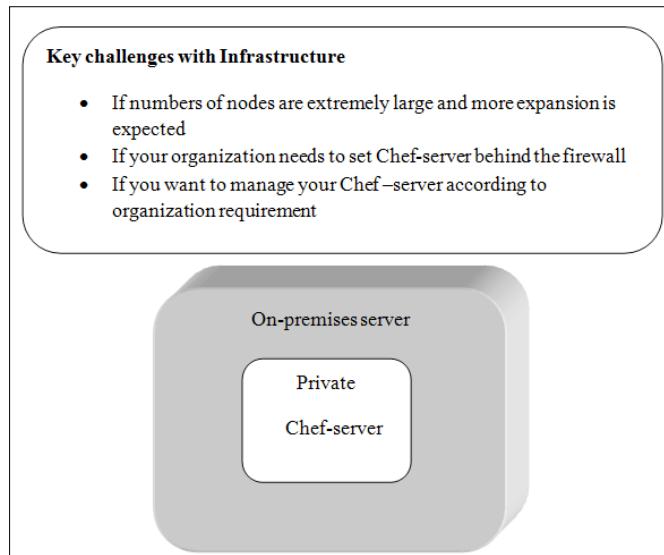
After the successful Private Chef deployment in DreamHost, the following benefits were achieved:

- Elimination of all manual infrastructure management tasks
- Enabled continuous delivery of new applications and services with highly integrated scenarios

- Enforced full utilization of resources, once they are developed.  
Resources can be reused
- Great reduction in the company's overall operational cost

For more details on the DreamHost case study, visit <https://www.chef.io/customers/dreamhost/>.

The following diagram describes the key challenges addressed by the Private Chef server:



## Case studies of the open source Chef deployment

Generally, the open source Chef deployment is preferred by research communities for testing, analysis, and learning purposes.

If any organization or industry has plans for Chef automation, first, they want to see a successful **proof of concept (POC)** and a kind of a running demo. The open source Chef server is absolutely free and we can test any number of nodes with open source Chef. However, both the Hosted and Private Chef deployment are restricted up to 5 nodes. After that, we have to acquire an enterprise license for effective utilization of Chef. Before making the final decision to go with the enterprise license, it is recommended that we test all the features of Chef. Hence, choosing open source Chef for testing is one of the best practices, which is followed by organizations.

Here, we are covering a case study where the organization first tested all the features of Chef with the open source Chef deployment and finally moved on to Hosted Chef.

## SolutionSet

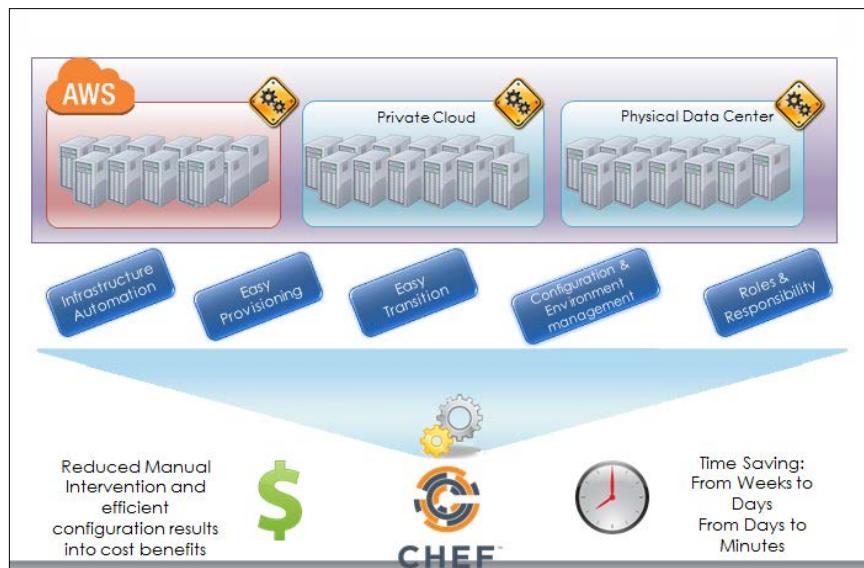
SolutionSet is one of the biggest digital consultancy companies that deals with digital marketing solutions, such as web development, mobile and social marketing. SolutionSet is present in 19 locations across the United States of America.

## Challenges with the infrastructure of SolutionSet

SolutionSet's technical team wanted to go with the hybrid cloud solution for their infrastructure setup; it means a combination of Public and Private clouds. The challenging task here is to manage the physical infrastructure as well as the hybrid cloud infrastructure. Basically, they have a heterogeneous IT infrastructure and they want to implement the Agile project development lifecycle.

## The solution with open source Chef

SolutionSet initially selected the open source Chef deployment to manage the heterogeneous infrastructure setup. They successfully automated approximately 400 physical servers, 100 cloud instances of Amazon EC2 public cloud, and some instances on private clouds as well. They experienced quick deployment, fast configuration management, and effective usage of the Agile methodology in the entire infrastructure.



## The final outcome

After the successful open source Chef deployment in SolutionSet, the following benefits were achieved:

- The SolutionSet technical team is now able to focus on the client's needs and not on infrastructure management
- Deployment of server resources is done in just a few minutes
- They now have enabled strong disaster recovery plans
- They incorporated the security best practices that have taken care of the client privacy model

For more details on the SolutionSet case study, visit <https://www.chef.io/customers/solutionset>.

## Case studies of the Chef-solo deployment

As we learned in the earlier chapters, Chef-solo is a special type among all the cases of deployment where the Chef server, Chef-client, and workstation are all located on a single node. With the help of some plugins and third-party tools, we can manage the required infrastructure and automation related tasks. This type of deployment is preferred by students and learners of Chef. It is also true that Chef-solo does not support all the advanced features of Chef.

Chef-solo acts like a standalone tool, as it does not require access to the Chef server and runs locally. Let's see a case study of the Chef-solo deployment.

## Wharton School - University of Pennsylvania

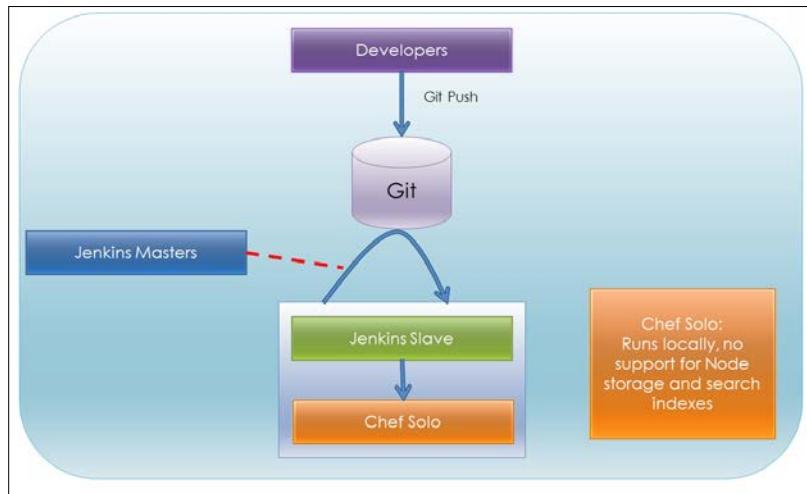
The Wharton School of the University of Pennsylvania is famous worldwide for its research and innovation work in multiple disciplines. It usually happens that Wharton receives the highest reputation scores from academic sections and managers of the recruiters worldwide.

## Challenges with the infrastructure of Wharton School

The globally popular Wharton school is involved in research. There is quite a lot of work to do with the **Elastic Block Storage (EBS)** volumes and instances. However, without automating, multiple tasks performed to maintain the heterogeneous infrastructure were actually not done in a manageable manner. Also, time consumption was high as the financial data research was done for more than 30,000 users across the globe.

## The solution with Chef-solo

They used the Knife Amazon - EC2 plugin and several of the community cookbooks to manage the EBS volume and instances. The Wharton School uses the Vagrant tool for management and automation of a virtual machine.



## The final outcome

After the successful open solo deployment in the Wharton School, the following benefits were achieved:

- The usage of the Vagrant tool increased the efficiency, as well as decreased the consumption of time
- The management is now up to the mark; Wharton School has realized the following benefits: increasing in the scalability results into improved functionality; flexibility, customizability, and resource performance
- Chef-solo provides an ideal balance between resources and the overall cost

For more details on the Wharton School, University of Pennsylvania case study, visit <https://www.chef.io/customers/whartonschool/>.

## Self-test questions

1. Which kind of Chef deployment is fit for small- and medium-sized organizations?
2. What is the criterion to choose the Private Chef deployment?
3. Why did SolutionSet select the open source Chef deployment initially?
4. What is the working scenario of Chef-solo and when should one choose Chef-solo?
5. What is the common outcome of the Hosted Chef deployment in all types of industries?

## Summary

In this chapter, we saw a variety of case studies on Hosted Chef, Private Chef, open source, and Chef-solo. If we take a general survey of all the industries, we will find that many of the small- and medium-sized organizations prefer Hosted Chef for problems related to infrastructure automation. If they never want to move any data outside the firewall, then Private Chef could be a secured option. If you require for testing purposes, open source is the solution. However, if the organization is research-based, then Chef-solo is the solution.

Hosted Chef is deployed in small- and medium-sized organizations for automation, which leads to better management and scalability with negligible human efforts.

If you need a highly scalable solution when it is confirmed that your business will keep on increasing rapidly and you would need a higher number of servers as the demand has increased, then Private Chef is a good option.

Any number of nodes can be tested with the open source Chef server. It is used in heterogeneous infrastructure setups and for **proof of concept (POC)**. It is better to use the open source Chef server initially as it is absolutely free.

It is meant for the cases where the focus is on research and innovation. A minimal set of operations can be handled quickly with Chef-solo.



# Index

## A

**advance automation tools, IT industry**  
comparing 23  
URL, for article 24

**agile methodology** 3

**Amazon Machine Image (AMI)** 235, 236, 242

**Amazon Web Service (AWS)**  
about 229  
open source Chef server, installing on 167

**Ancestry Inc**  
URL 268

**Ansible**  
about 21  
features 21

**application cookbooks** 110

**Application Programming Interface (API)** 19

**attributes**  
about 38, 39, 113  
automatic 114  
default 113  
force\_default 113  
force\_override 113  
normal 113  
override 113  
reasons, for using 38  
URL, for documentation 114

**authentication procedure**  
between workstation, and Chef server 42

**AutoIt**  
about 13  
features 13

**automated system** 2

**automation**  
about 2  
benefits 2  
benefits, to IT industry 4  
need for 2, 3

**automation tools**  
Ansible 21  
AutoIt 13  
Bcfg2 15  
cdist 17  
CFEngine 14  
Cobbler 16  
Crowbar 20  
Fabric 20  
Glu 19  
InstallShield 13  
Juju 22  
Mina 22  
Pallet 18  
Puppet 15  
Rex 18  
RunDeck 19  
SaltStack 21  
Sprinkle 17  
Windows PowerShell scripting 14

**AWS Cloud credentials**  
knife.rb file, configuring with 235

**AWS EC2 bootstrapping, with Chef**  
about 232  
AWS settings, configuring 233-235  
Chef-client, running as daemon 242  
Chef-client, running on new client node 238-240  
EC2 instance, bootstrapping 235, 236

knife-ec2 plugin, installing 233  
knife.rb file, configuring 233-235  
recipes, managing on new client node 241  
workstation, preparing 232, 233

## B

**barclamps** 20

**Bcfg2**

about 15  
features 15

**best practices, effective usage of Chef**

about 254  
advance planning 254  
cookbook, designing 254  
overloading, avoiding of Chef environment 255  
private recipe, using 255  
usage, avoiding of giant cookbook 255

**Bookshelf** 36

## C

**cdist**

about 17  
features 17

**CFEngine**

about 14  
features 14  
versus Chef 25

**Chef**

about 8  
and VMware 245  
automation tools 12  
automation with 11  
components 8  
features 9, 10  
need for 9  
URL 10  
versus CFEngine 25  
versus Puppet 24  
with cloud infrastructure 232

**Chef anti-patterns**

about 247, 248  
default cookbook 250, 251  
wrapper cookbook 249, 250

**Chef automation**

about 30  
infrastructure scenario 30

**Chef-client**

about 44  
running, as daemon 242  
running, on new client node 238-240  
URL, for downloading 51

**Chef-client version based**

URL 74

**Chef community**

about 46  
cloud related cookbooks 48  
database related cookbooks 47  
monitoring related cookbooks 48  
package management related cookbooks 48  
process management related cookbooks 47  
programing languages related cookbooks 48  
virtualization related cookbooks 48  
web server related cookbook 47

**Chef components**

about 32  
Chef server 32  
nodes 43  
workstations 40

**Chef-expander** 35

**Chef framework**

about 30  
components 31

**chef\_handler**

handler, installing with 146

**Chef patterns**

about 247, 248  
default cookbook 251  
wrapper cookbook 249, 250

**Chef repository**

about 41  
creating 80  
setting up 87-89

**Chef script** 30

**Chef server**  
about 32

functionalities 32

**Chef server 0.10.x**

accessing 189, 190  
data, downloading from 190

**Chef server 11.x**  
accessing 190  
data, uploading to 193  
initial settings, applying on virtual machine 155

**Chef-server API**  
about 45  
prerequisites 45

**Chef server tools**  
about 33, 34  
Bookshelf 36  
cookbook 36  
ErChef 34  
message queues 35  
Nginx 35  
node objects 38  
policies 39  
PostgreSQL 36  
search index 35  
WebUI 35

**Chef server, types**  
about 32  
Hosted Chef-server 32  
open source Chef-server 33  
Private Chef 33

**Chef-solo 46**

**Chef-solo deployment, case studies**  
about 275  
Wharton School 275, 276

**Chef-solr 35**

**Chef tools 31**

**chef-validator settings**  
updating 191

**cloud computing 4, 230, 231**

**cloud deployment models**  
community cloud 231  
hybrid cloud 231  
private cloud 231  
public cloud 231

**cloud instances 43**

**Cloudkick, open source handler 148**

**Cobbler**  
about 16  
features 16

**community cloud 231**

**community cookbooks**  
using 179-188

**cookbook**  
about 36  
attributes 38  
components 111, 112  
creating 92  
downloading 87-89  
features 36  
high-level structure 92  
recipe 37  
testing 252  
uploading 101  
URL, for documentation 98

**cookbook library**  
example 119, 120

**cookbook tests, types**  
about 252  
consistency, checking 253  
integration testing 252, 253  
performance, checking 253  
results, checking 253  
syntax, checking 252

**cookbook, types**  
about 110  
application 110  
library 110  
wrapper 110, 111

**Create command**  
used, for adding node 128-130

**Crowbar**  
about 20  
features 20

**Cucumber Chef 253**

**customink 253**

**D**

**daemon**  
Chef-client, running as 242

**data bag 40, 140-144**

**database related cookbooks**  
Elasticsearch 47  
mongodb 47  
mysql 47  
postgresql 47

**default cookbook** 250, 251

**definition**

- about 114
- example 116
- syntax 115
- URL, for documentation 116

**delete command**

- used, for deleting node 131

**Dell Cloud Manager** 5

**DevOps**

- about 5, 6
- benefits 6
- necessities 7

**directory structure, tomcat cookbook**

- about 93
- attributes 93
- CHANGELOG.md 96
- library 94
- metadata 97
- provider 94
- README.md 97
- recipe 95
- template 96

**Distributed Replicated Block Device  
(DRBD)** 216

**Domain Specific Language (DSL)** 24

**DRBD configuration, on backend Bootstrap  
server**

- filesystem, checking 218, 219

**DreamHost case study**

- URL 273

## E

**EC2 instance, bootstrapping**

- about 235, 236
- configuration options 236
- expected output 236

**Elastic Block Storage (EBS)** 275

**Elastic cloud computing (EC2)** 232

**Embedded Ruby (ERB)** 252

**Enstratius** 5

**ErChef** 34

**error types, troubleshooting steps** 101-106

**etsy** 253

**exception handler** 145

## F

**Fabric**

- about 20
- features 20

**Facebook case study**

- URL 271

**Fanhattan case study**

- URL 261

**files**

- about 116
- example 117
- syntax 117

**firewall requirements**

- for backend servers 206
- for frontend servers 206

**Foodcritic** 253

**fully-qualified domain name (FQDN)** 150

## G

**Getaroom case study**

- URL 267

**Git**

- about 69
- installing 70-74
- URL 69
- used, for setting up workstation on  
Windows 8 80, 81

**GitHub** 45

**Glu**

- about 19
- features 19

## H

**handler**

- about 145
- configuring 145
- installing 145
- installing, manually 146
- installing, with chef\_handler 146

**handler, types**

- about 145
- exception handler 145
- report handler 145

**help command**  
configuration options 236

**high-availability Chef installation**  
load balancer requirements 211  
prerequisites 211  
private-chef.rb file, configuring 213

**high-availability Chef server**  
DRBD configuration, on backend Bootstrap server 217  
DRBD configuration, on backend non-Bootstrap server 217, 218  
DRBD, installing on backend servers 217  
installing 216  
on-premises Chef, configuring on Bootstrap backend server 220  
on-premises Chef, configuring on frontend servers 221  
on-premises Chef, configuring on non-Bootstrap backend server 220  
on-premises Chef, installing on backend server 216  
on-premises Chef, installing on frontend servers 221

**Hosted Chef deployments, case studies**  
about 258  
Admeld 258, 259  
Fanhattan 260, 261  
Getaroom 265, 267  
Imagination 264, 265  
Limelight video platform 263, 264  
Zumba Fitness 261, 262

**Hosted Chef server**  
working 50-55

**hybrid cloud** 231

**I**

**Imagination case study**  
URL 265

**Infrastructure as a Service (IaaS)** 230

**installation package, Enterprise Chef**  
downloading 199-202

**InstallShield**  
about 13  
features 13

**Internet service provider (ISP)** 119

**J**

**JavaScript Object Notation (JSON)** 128

**Juju**  
about 22  
features 22

**K**

**Knife**  
about 41  
functionalities 41

**Knife.bootstrap command** 134

**Knife-cloud plugin** 245

**knife configuration options**  
reference link 52

**knife data bag subcommand** 140

**knife-ec2 plugin**  
installing 233

**Knife GitHub plugin**  
URL, for installing 45

**Knife node subcommand** 127

**Knife plugins, Chef** 245

**Knife-Rackspace**  
plugins, installing for 243

**knife.rb file**  
configuring, with AWS Cloud credentials 235

**knife search subcommand**  
reference link 136

**knife-vcair**  
about 246  
URL 246

**knife-vsphere plugin**  
URL 245

**L**

**libraries**  
about 118  
syntax 118  
URL, for documentation 120

**library cookbooks** 110

**Lightweight Resources and Provider (LWRP)**  
about 110, 123  
components 123, 124  
URL 124

**Limelight case study**  
URL 264  
**load balancer requirements,**  
    **high-availability Chef**  
    api\_fqdn, configuring 212  
    for backend server 212  
    for frontend server 212

## M

**magic\_shell cookbook**  
example 98-100  
**message queues** 35  
**metadata**  
    about 124  
    error message 125  
**metadata.rb file**  
    about 125  
    URL 125  
**Mina**  
    about 22  
    features 22  
**Minitest** 252  
**monitoring related cookbooks**  
    collectd 48  
    nagios 48  
    newrelic\_monitoring 48

## N

**National Institute of Standards and Technology (NIST)** 230  
**Network Address Translation (NAT)** 238  
**Nginx** 35  
**node objects**  
    about 38  
    attributes 39  
    run-list 39  
**nodes**  
    about 43, 127  
    adding, Create command used 128-130  
    bootstrapping 133, 176-178  
    Cloud instances 43  
    deleting, delete command used 131  
    editing 131  
    physical nodes 43  
    verification process 135  
    virtual nodes 43

**nodes, tools**  
    Chef-client 44  
    Ohai 44

## O

**Ohai** 44  
**on-premises Chef**  
    managing 221  
    service commands 222  
    service subcommands 223  
**on-premises Chef installations**  
    standalone on-premises Chef 197  
    tiered on-premises Chef 198  
    types 197  
**on-premises Chef package, installing**  
    about 210-221  
    on CentOS 203-205  
    on Red Hat 203-205  
**on-premises Chef server**  
    about 196  
    benefits 196, 197  
**open source Chef deployment, case studies** 273-275  
**open source Chef server**  
    installing 151, 152  
    installing, on Amazon Web Services (AWS) 167  
    installing, on VM machine 155  
    installing, on VMware Fusion virtual machine (Ubuntu 12.04) 155  
    installing, on VMware Workstation virtual machine (CentOS 6.x) 156-167  
    system requisites 150  
    upgrading 188, 189  
    URL 151  
**open source Chef server, requisites**  
    FQDN and hostnames configuration 152  
    hostname, modifying 154  
    virtual machine, restarting 153  
**open source Chef sever** 33  
**open source handlers**  
    about 147  
    Cloudkick 148  
    Simple E-mail 147  
    SNSHandler 148

Syslog 147  
Updated Resource 148  
**Opscode** 8  
**Opscode's Cookbook Tomcat**  
URL 92

## P

**package management related cookbooks**  
Apt 48  
Yum 48  
Yumrepo 48  
**Pallet**  
about 18  
features 18  
**partial search** 139  
**physical nodes** 43  
**Platform as a Service (PaaS)** 230  
**policies**  
about 39  
data bag 40  
environment 40  
roles 39  
**pony library**  
URL 147  
**PostgreSQL** 36  
**Private Chef** 33  
**Private Chef deployment, case studies**  
about 268  
Ancestry.com 268, 269  
DreamHost 271, 272  
Facebook 270, 271  
**private-chef.rb file**  
configuring 207  
on-premises Chef packages, adding  
to servers 209  
required settings, for backend server 208  
required settings, for frontend server 208  
**private-chef.rb file, high-availability Chef**  
required changes, for frontend  
entries 214, 215  
required settings, for backend server with  
Bootstrapping 213  
required settings, for other  
backend servers 213  
**private cloud** 231

**process** 39  
**process management related cookbooks**

Bluepil 47  
monit\_bin 47  
runit 47

**programming related cookbooks**

java 48  
nodejs 48  
php 48  
python 48

**proof of concept (POC)** 273

**providers** 120

**public cloud** 231

**Puppet**

about 15  
features 15  
versus Chef 24

## R

**RabbitMQ** 35  
**Rackspace Cloud server bootstrapping**  
about 243  
plugins, installing for Knife-Rackspace 243  
prerequisites 243  
with Chef-client 244  
workstation, preparing with Rackspace  
credentials 243

**Rackspace servers**

deleting 245

**recipe**

about 37  
characteristics 37  
managing, on new client node 241

**report handler** 145

**resources**

about 120  
example 121  
syntax 121  
URL, for documentation 121

**resources, components**

action 121  
attribute 121  
name 121  
type 121

**Rex**  
about 18  
features 18  
URL 18

**RightScale** 5  
role 39

**Ruby Version Management (RVM)** 266

**RunDeck**  
about 19  
features 19

**S**

**SaltStack**  
about 21  
features 21

**Scalr** 5

**search**  
about 136  
reference link 140

**search index** 35

**search, options**  
about 137  
partial search 139  
search, by node 137  
search, by node and environment 138  
search, for multiple attributes 139  
search, for nested attributes 138

**search query command**  
syntax 136

**service commands, on-premises Chef**  
Chef commands, viewing 222  
Chef, reconfiguring 222  
uninstalling 222  
view configuration 222

**service subcommands, on-premises Chef**  
hup subcommand 223  
int subcommand 223  
kill 223  
log files 227  
once 224  
restart subcommand 225  
service-list subcommand 224  
start subcommand 224  
status subcommand 226  
stop subcommand 225  
tail subcommand 226

term subcommand 227

**Simple E-mail, open source handler** 147

**simple handler**  
writing 147

**SNSHandler, open source handler** 148

**Software as a Service (SaaS)** 230

**Software Development Life Cycle (SDLC)** 4

**SolutionSet case study**  
URL 275

**Sprinkle**  
about 17  
features 17

**standalone on-premises Chef**  
about 197  
firewall requirements 202  
installing 203  
prerequisites 202

**Syslog, open source handler** 147

**T**

**templates**  
about 122  
example 122  
syntax 122  
URL, for example 123

**text editor**  
reference link 128

**tiered on-premises Chef** 198

**tiered on-premises Chef installation**  
about 209  
Bootstrap, configuring 210  
firewall requirements 206  
frontend server, configuring 210  
load balancer requirements 206  
prerequisites 205  
private-chef.rb file, configuring 206, 207

**U**

**Updated Resource, open source handler** 148

**upgrade, open source Chef server**  
about 188  
admin public key, verifying 192  
Chef server 0.10.x, accessing 189, 190  
Chef server 11.x, accessing 190

chef-validator settings, updating 191  
data, downloading from  
  Chef server 0.10.x 190  
data, uploading to Chef server 11.x 193  
requisites 189  
user passwords, verifying 192

## V

**Vagrant**  
  about 50, 66  
  installing 66-68  
  URL, for installation procedure 66  
  virtual machine, launching with 90-92  
**Vagrant boxes**  
  references 90  
**verification process, node** 135  
**Version control system (VCS)** 69  
**VirtualBox**  
  about 61  
  installing 62-66  
  URL, for installation procedure 62  
**virtualization, and cloud related cookbooks**  
  aws 48  
  openstack 48  
  vmware 48  
  vmware\_workstation 48  
**virtual machine**  
  launching, with Vagrant 90-92  
  launching, with workstation setup 90-92  
**virtual nodes** 43  
**VM machine**  
  open source Chef server, installing on 155  
**VMware**  
  and Chef 245  
**VMware Fusion virtual machine  
(Ubuntu 12.04)**  
  open source Chef server, installing on 155  
**VMware Workstation virtual machine  
(CentOS 6.x)**  
  open source Chef server, installing  
    on 156-167

## W

**web server related cookbooks**  
  apache2 47  
  glassfish 47  
  jboss 47  
  nginx 47  
  tomcat 47  
**WebUI** 35  
**Wharton School, University of Pennsylvania case study**  
  URL 276  
**Windows PowerShell scripting**  
  about 14  
  features 14  
**workstation**  
  about 40  
  configuring 74-80  
  functionalities 40  
  installing 74-80  
  setting up 80, 167-175  
**workstation setup**  
  virtual machine, launching with 90-92  
**workstation setup, on Windows 8**  
  Git used 80, 81  
**workstation setup without Git, on CentOS**  
  about 81  
  Ruby installation 81-86  
**workstations, tools**  
  Chef repository 41  
  Knife 41  
**wrapper cookbooks** 110, 111, 249, 250  
**wrapper patterns**  
  reference link 250

## Z

**Zumba Fitness case study**  
  URL 262





**Thank you for buying  
Learning Chef**

## About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike.

For more information, please visit our website: [www.packtpub.com](http://www.packtpub.com).

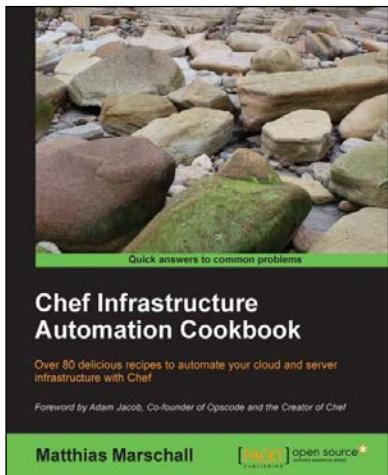
## About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licenses, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

## Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to [author@packtpub.com](mailto:author@packtpub.com). If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

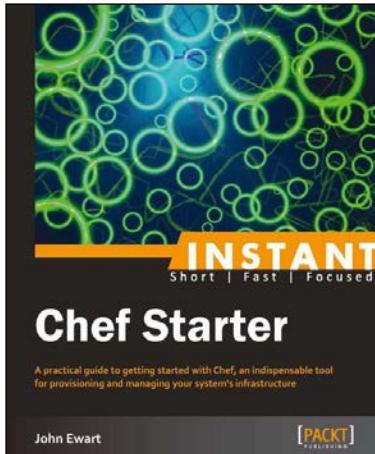


## Chef Infrastructure Automation Cookbook

ISBN: 978-1-84951-922-9      Paperback: 276 pages

Over 80 delicious recipes to automate your cloud and server infrastructure with Chef

1. Configure, deploy, and scale your applications.
2. Automate error prone and tedious manual tasks.
3. Manage your servers on-site or in the cloud.
4. Solve real world automation challenges with task-based recipes.
5. The book is filled with working code and easy-to-follow, step-by-step instructions.



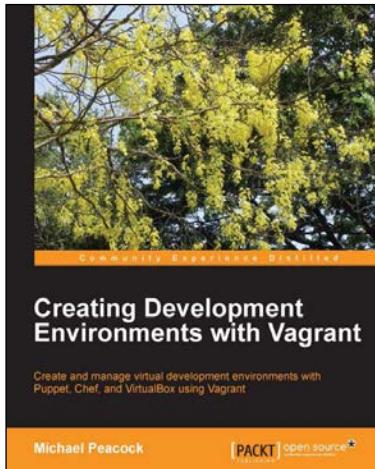
## Instant Chef Starter

ISBN: 978-1-78216-346-6      Paperback: 70 pages

A practical guide to getting started with Chef, an indispensable tool for provisioning and managing your system's infrastructure

1. Learn something new in an Instant! A short, fast, focused guide delivering immediate results.
2. Learn the core capabilities of Chef and how it integrates with your infrastructure.
3. Set up your own Chef server for managing your infrastructure.
4. Provision new servers with ease and develop your own recipes for use with Chef.

Please check [www.PacktPub.com](http://www.PacktPub.com) for information on our titles

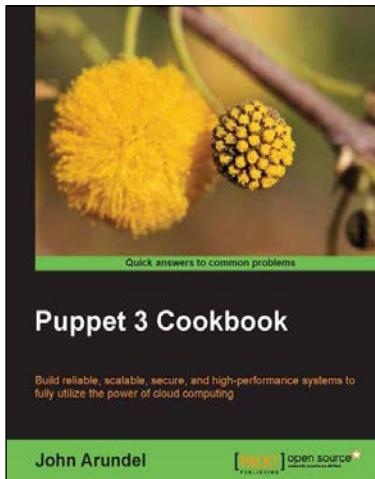


## Creating Development Environments with Vagrant

ISBN: 978-1-84951-918-2      Paperback: 118 pages

Create and manage virtual development environments with Puppet, Chef, and VirtualBox using Vagrant

1. Provision virtual machines using Puppet and Chef.
2. Replicate multi-server environments locally.
3. Set up a virtual LAMP development server.



## Puppet 3 Cookbook

ISBN: 978-1-78216-976-5      Paperback: 274 pages

Build reliable, scalable, secure, and high-performance systems to fully utilize the power of cloud computing

1. Use Puppet 3 to take control of your servers and desktops, with detailed step-by-step instructions.
2. Covers all the popular tools and frameworks used with Puppet: Dashboard, Foreman, and more.
3. Teaches you how to extend Puppet with custom functions, types, and providers.
4. Packed with tips and inspiring ideas for using Puppet to automate server builds, deployments, and workflows.

Please check [www.PacktPub.com](http://www.PacktPub.com) for information on our titles