

Community Experience Distilled

Managing Windows Servers with Chef

Harness the power of Chef to automate management of Windows-based systems using hands-on examples

John Ewart

[PACKT] open source*
PUBLISHING

Table of Contents

Managing Windows Servers with Chef
Credits
About the Author
About the Reviewers
www.PacktPub.com
    Support files, eBooks, discount offers, and more
        Why subscribe?
        Free access for Packt account holders
Preface
    What this book covers
    What you need for this book
    Who this book is for
    Conventions
    Reader feedback
    Customer support
        Downloading the example code
        Errata
        Piracy
        Questions
1. Chef and Windows
    Working with Windows
        Interacting with end hosts
        Bootstrapping Windows hosts
            Scaling with cloud providers
            Scripting with PowerShell
    Integrating with Linux-based systems
    Working with Windows-specific resources
        Supported platforms
    Summary
2. Installing the Client – an Overview of Chef Concepts
    Getting to know Chef better
    An overview of Chef's architecture
    Installing the Chef client on Windows
        Preparing to bootstrap Windows hosts
            Enabling Windows Remote Management
            Configuring firewall ports
            Enabling basic authentication
            Bootstrapping a Windows host
        Installing via MSI
    Summary
3. Windows-specific Resources
    Working with Windows-specific resources
        Platforms supported by Chef
    Managing roles and features

Installing roles using different mechanisms
Executing batch scripts
Running scripts at startup
Installing software packages
Manipulating printers
Managing printer ports
Managing printers
Interacting with the Windows Registry
The Chef 10.x resource
The Chef 0.11.x resource
Managing the system path
Scheduling tasks
Interacting with Windows pagefiles
ZIP files
Rebooting Windows
Summary
4. Provisioning an Application Stack
Examining the cookbook
Installing the cookbook
Fetching from GitHub
Examining the recipe
Installing the prerequisites
Preparing the IIS service
Fetching the application
Configuring the application
Generating an IIS pool and site
Performing the installation
Bootstrapping the host
Creating the role
Applying the role to the node
Summary
5. Managing Cloud Services with Chef
Working with Microsoft Azure
Downloading the management certificate
Configuring knife for Azure
Creating a new Azure virtual machine
Bootstrapping your Azure node
Creating a reusable image
Managing Amazon EC2 instances
Installing the EC2 knife plugin
Setting up EC2 authentication
Provisioning an EC2 instance
Executing custom user scripts in EC2
Writing the user script
Providing a custom user script
Terminating an EC2 instance
Removing the Chef node
Interacting with Rackspace Cloud

Provisioning a Rackspace instance
Injecting configuration into the virtual machine
Terminating the instance
Removing the Chef node
Summary
6. Going Beyond the Basics
Chef's declarative language
Handling multiple platforms
Declaring support in your cookbook
Multiplatform recipes
Reducing complexity
Versioning and source control
Testing recipes
RSpec and ChefSpec
Testing basics
RSpec
ChefSpec
Getting started with ChefSpec
Executing tests
Understanding failure
Expanding your tests
Summary
Index

Managing Windows Servers with Chef

Managing Windows Servers with Chef

Copyright © 2014 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 author, 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: May 2014

Production Reference: 1160514

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78398-242-4

www.packtpub.com

Cover Image by Neha Rajappan (<neha.rajappan1@gmail.com>)

Credits

Author

John Ewart

Reviewers

Joshua Black

Thabo Fletcher

Lauren Malhoit

Commissioning Editor

Edward Gordon

Acquisition Editor

Llewellyn Rozario

Content Development Editor

Athira Laji

Technical Editors

Arwa Manasawala

Ankita Thakur

Copy Editor

Laxmi Subramanian

Project Coordinator

Puja Shukla

Proofreader

Paul Hindle

Indexer

Mehreen Deshmukh

Graphics

Ronak Dhruv

Production Coordinator

Conidon Miranda

Cover Work

Conidon Miranda

About the Author

John Ewart is a system architect, software developer, and lecturer. He has designed and taught courses at a variety of institutions including the University of California, California State University, and local community colleges covering a wide range of computer science topics including Java, data structures and algorithms, operating systems fundamentals, Unix and Linux systems administration, and web application development. In addition to working and teaching, he maintains and contributes to a number of open source projects. He currently resides in Redmond, Washington, with his wife, Mary, and their two children.

About the Reviewers

Joshua Black has been working with computers professionally for 20 years. He has a broad range of experience and expertise including systems and network administration, mobile app development, and production web applications. He earned a BS in Computer Science with a minor in Math from California State University, Chico, in 2005. He currently resides in Chico, California, with his wife, Rachel, and their four children.

Thabo Fletcher is a software developer at Degreed and co-founder of Coderbits with a background in network appliance engineering, web application development, and JavaScript injection.

Lauren Malhoit has been in the IT field for over 10 years and has acquired several data center certifications. She's currently a technical virtualization architect specializing in virtualization and storage in the data center. She has been writing for a few years for TechRepublic, TechRepublic Pro, and VirtualizationSoftware.com. As a Cisco Champion, EMC Elect, VMware vExpert, and PernixPro; Lauren stays involved in the community. She also hosts a bi-weekly technology podcast called AdaptingIT (http://www.adaptingit.com/). She has been a delegate for Tech Field Day several times as well. She recently published her first book, *VMware vCenter Operations Manager Essentials*, *Packt Publishing*.

www.PacktPub.com

Support files, eBooks, discount offers, and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

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 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> for more details.

At 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.



http://PacktLib.PacktPub.com

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

Free access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Preface

Welcome to *Managing Windows Servers with Chef*. This book is designed to familiarize you with the concepts, tools, and features available to help you manage Windows hosts with Chef. Inside the book, you will learn what you can expect from Chef on Windows, how to get started using it, and what Chef provides for managing Windows hosts that differs from Linux systems. Included are examples of deploying a complete .NET/IIS application stack, cloud integration, and some information on developing and testing for heterogeneous networks of Windows and Linux-based hosts.

What this book covers

Chapter 1, *Chef and Windows*, serves as an introduction to Chef's support for Windows, what sort of features you can expect from Chef on the Windows platform, and how to get started.

Chapter 2, *Installing the Client – an Overview of Chef Concepts*, provides coverage of how to install the client on a Windows host as well as a quick overview of Chef's architecture and terminology and other important information to get you started with managing Windows.

Chapter 3, *Windows-specific Resources*, introduces you to the resources that are unique to managing Windows via Chef. This chapter provides descriptions and examples of each resource, including roles, features, package installers, batch scripts, the Windows registry, and many more.

Chapter 4, *Provisioning an Application Stack*, provides a hands-on guide to provisioning a complete application stack (the .NET framework, IIS configuration, database server installation, and so on), including defining roles, setting up configuration data, installing requirements, and configuring the application.

Chapter 5, *Managing Cloud Services with Chef*, covers integrating with various cloud providers such as AWS, Rackspace Cloud, and Azure.

Chapter 6, *Going Beyond the Basics*, focuses on the integration of existing systems in heterogeneous networks, how to deal with recipes and multiple platforms, testing, organization, and publishing of recipes.

What you need for this book

This book expects that you have access to some important components in order to be successful. In order to execute the examples in the book, the following prerequisites are needed:

- Access to a Chef server for managing your configuration; a self-hosted installation or a Chef-hosted account will work equally well
- A workstation where you can install software including knife (Windows or Linux host)
- A text editor of your choice

Additionally, if you wish to try out any of the cloud-computing examples, you will need an account with the cloud hosting providers you are trying out.

Who this book is for

This book is designed for system administrators who have had some exposure to Chef, either casually or in-depth. It covers the Windows-specific facets of Chef and expects that you have a working installation of the Chef server available for you to use, either hosted or self-managed. A working knowledge of some programming language, preferably Ruby, will be needed to get the most out of the examples and to build your own recipes.

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, cookbook names, recipe names, scripts, database table names, folder names, filenames, file extensions, and pathnames are shown as follows: "In the same way that you would leverage `knife ssh` for Linux systems, `knife winrm` is available to execute commands remotely on a Windows host using the WinRM protocol."

A block of code is set as follows:

```
search(:node, 'role:web_server).each do |node|
  ip = node[:external_ip]
  firewall_rule "#{ip}" do
    source "#{ip}"
    action :allow
  end
end
```

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

```
knife ssh "role:mysql" "chef-client"  --sudo -x
ubuntu
```

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: "If you check the **Chef Client Service** box during the installation phase, the service will be set to run automatically."

> ### Note
>
> Warnings or important notes appear in a box like this.

> ### Tip
>
> 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 may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to <feedback@packtpub.com>, and mention the book title via 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 on 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 for all Packt books you have purchased from your account at http://www.packtpub.com. 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 would 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 on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from http://www.packtpub.com/support.

Piracy

Piracy of copyright 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.

Please contact us at <copyright@packtpub.com> with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at <questions@packtpub.com> if you are having a problem with any aspect of the book, and we will do our best to address it.

Chapter 1. Chef and Windows

If you are reading this book, the presumption is that you are already familiar with Chef, at least conceptually, and that you have some Windows-based systems you need to manage. If you have used Chef to manage some Linux-based systems, or have interacted with chef-solo to manage virtual machines, you will be ready to hit the ground running. If not, don't worry—there are plenty of explanations and examples to make you feel right at home.

If you need help installing the Chef server, there are a number of other resources available to help you do that, or you can sign up for the free tier of hosted Chef to get you started right away without managing your own server.

In this chapter, we will cover the following topics:

- Working with Chef and Microsoft Windows
- Ways to integrate Chef and Windows
- Supported platforms and technologies

Working with Windows

For those who are new to Chef, the client-side components of it are written in Ruby, a very popular language. Due to the cross-platform nature of Ruby, support for Windows is as straightforward as support for Linux and UNIX-like systems and has been around for quite some time now, dating back to the release of the **knife-windows** gem circa 2011.

Chef uses Ruby as the scripting language on client systems, and because of this, it is capable of running anywhere Ruby is supported. This alone makes Chef a very capable tool for managing a combination of different operating systems. Chef goes one step further by providing you with a **domain-specific language** (**DSL**) that makes writing recipes for interacting with Windows hosts look no different than UNIX-like platforms. With the exception of some resource names and paths and the existence of Windows-specific resources such as the Windows Registry, recipes are almost drop-in compatible with either platform. This means that the wide variety of existing cookbooks and recipes available for Chef are available to use with little or no modification.

It's also important to note that while a lot of conversations focus on server management, cloud scaling, and so on, Chef is not reserved solely for managing servers; it can be used to manage client workstations as well. With the available resources, you can just as easily install and configure

desktop applications, import registry settings, manage users, set up printers, and so on.

Interacting with end hosts

Where Linux-based systems can execute commands over SSH, Windows platforms have an additional mechanism called **Windows Remote Management** (**WinRM**). In the same way that you would leverage `knife ssh` for Linux systems, `knife winrm` is available to execute commands remotely on a Windows host using the WinRM protocol.

For example, one might execute the following command for connecting to Linux hosts:

```
knife ssh "role:mysql" "chef-client"  --sudo -x
ubuntu
```

The following command would connect to Windows hosts in the same role:

```
knife winrm "role:mysql" "chef-client" -x
Administrator
```

As you can see, the `winrm` subcommand supports executing a command on any number of hosts that match the supplied search criteria just like the `ssh` subcommand. While the protocol for communicating with the hosts may be different, the mechanism for interacting with them via `knife` remains consistent.

> **Tip**
>
> **Downloading the example code**
>
> You can download the example code files for all Packt Publishing books you have purchased from your account at http://www.packtpub.com. 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.

Bootstrapping Windows hosts

Bootstrapping a host is intended to prepare a host for, and complete registration with, the Chef service (be it your own Chef server or a hosted installation). Hosts that are being bootstrapped typically contain nothing

more than a bare OS installation; however, it is also possible to bootstrap hosts that have existing software configurations. The bootstrapping script is responsible for performing the following functions:

- Installing Ruby 1.8.7 with RubyGems
- Installing the RubyInstaller Development Kit (DevKit)
- Installing Windows-specific gems to support Chef
- Installing Chef from RubyGems.org
- Writing out the validation key into `C:\chef\validation.pem`
- Optionally writing out an encrypted data bag secret
- Writing the default configuration file for Chef in `C:\chef\client.rb`
- Creating the initial run-list JSON file in `C:\chef\first-boot.json`
- Running `chef-client` for the first time to register the node with Chef

An example of bootstrapping a Windows host using the Windows Remote Management protocol might look like the following command:

```
knife bootstrap windows winrm
windowshost.domain.com -x Administrator
```

This command will connect to `windowshost.domain.com` as Administrator via the Windows Remote Management protocol and then run the commands in the Windows bootstrap script. For a complete view of the commands being run, you can find the Windows bootstrap script at https://github.com/opscode/knife-windows/blob/master/lib/chef/knife/bootstrap.

Scaling with cloud providers

By combining the ability to automatically bootstrap a Windows system with a provider that supplies Windows virtual hosts, you can integrate cloud servers into your infrastructure with ease. Chef has existing support for using Azure, AWS, and Rackspace APIs to manage cloud infrastructure including starting, stopping, and provisioning new instances with those services automatically. If you are using a service that is currently unsupported, it is entirely possible to develop a plugin to provide integration with that provider. Through Chef, you can manage a collection of on-site and off-site hosts with a mix of physical and virtual servers with ease. This means that you can bring up new servers in a much shorter period of time when you need them and do away with them when they are not in use, saving you both time and money.

Scripting with PowerShell

Modern Windows systems come with the PowerShell runtime, an incredibly powerful tool for interacting with the system. Naturally, as Chef is a developer-oriented way of managing systems, writing scripts to execute on

end hosts is a convenient and flexible way of extending Chef's functionality. Chef provides a mechanism for executing PowerShell scripts in Windows in the same way it supports running Bash scripts on a Linux host. A very trivial example might be the following PowerShell script that writes a line of text into a file:

```
powershell "say-hello" do
   code <<-EOH
   $stream = [System.IO.StreamWriter] "C:\hello.txt"
   $stream.WriteLine("Hello world!")
   $stream.Close()
   EOH
end
```

The preceding code allows you to exercise the full power of PowerShell from within your recipes by executing scripts you define on the managed systems. These scripts can even be dynamically generated from configuration data and other variables in your recipes.

Integrating with Linux-based systems

Having a heterogeneous network is becoming more common as time goes by. Certain pieces of software either don't exist or are not as well supported on one platform as they are on another. As a result, administrators encounter situations where they are required to deploy and manage hosts running multiple operating systems side-by-side. Integrating Windows and Linux-based systems comes with its own set of challenges, and Chef helps to address these issues by providing a consistent way to interact with both Linux and Windows-based systems.

For anyone who manages such infrastructure (specifically a collection of systems running some combination of Windows and Linux), Chef has some amazing features. Because it is capable of modeling both Windows and Linux systems with the same declarative language and configuration data, you can easily configure both your Linux and Windows systems using the same tool.

For example, you could have a cookbook with a recipe that configures the firewalls of your network hosts. That recipe can search Chef's configuration data for all other hosts in your network, gather up a list of their IP addresses, and open up traffic to all those IP addresses. If you provision a new host, all of the hosts being managed will automatically know about the new host and add a new firewall rule. Additionally, because Chef provides you with its own declarative language that hides the implementation details, you can focus on what you want to achieve and not on how to achieve it. Chef knows what platform it is running on and how to load the system-specific implementation of your resource such as a network interface, firewall, user, file, and more. It is also entirely possible to write recipes to install Apache or MySQL that are capable of working on both Linux and Windows platforms.

This makes it much easier to integrate any number of Windows and Linux systems without having to maintain multiple sets of scripts to achieve the same end goal. For example, assuming you had a firewall cookbook for both Windows and Linux, it would be possible to write a recipe similar to the following:

```
search(:node, 'role:web_server).each do |node|
  ip = node[:external_ip]
  firewall_rule "#{ip}" do
    source "#{ip}"
    action :allow
  end
end
```

In the preceding code, we are searching for all nodes that have the role of `web_server` and which call the `firewall_rule` resource to allow traffic to originate from that source. Notice that the recipe does not refer to the Windows Firewall software or Linux's firewall tool, iptables. Rather, Chef's custom language allows us to describe what we were doing, not how to achieve our goal. The "how" is implemented in a provider and the "what" is described by a resource, which are both provided in our firewall cookbook.

Working with Windows-specific resources

There are a handful of resources that Chef provides on a Windows system that are specific to Windows. Chef can automatically determine which type of host a recipe is being executed on and perform a different set of actions based on the host type. For example, the installation of a particular software package such as MySQL may be mostly identical between hosts but requires slightly different settings or needs to store Registry settings on a Windows system. Some of the resources that are specific to Windows include the following:

- Batch scripts
- PowerShell scripts
- Autorun scripts
- Software packages (MSIs, installers, and so on)
- Printers
- Windows Registry
- Network paths
- System tasks

Supported platforms

Chef for Windows supports recent versions of Windows as of the time of writing. This includes the following (but may work on other, newer releases as well):

- Windows Server 2003 R2
- Windows Server 2008
- Windows 8
- Windows 7
- Windows Vista

Summary

As you can see, Chef has a lot to offer to Windows administrators both in managing Windows-only infrastructure as well as heterogeneous Windows and Linux infrastructure.

Now that you have got a feeling for how Chef can benefit you when managing Windows systems, let's take a look at how to install the client and how it fits into the overall architecture of the Chef ecosystem in the next chapter.

Chapter 2. Installing the Client – an Overview of Chef Concepts

As with all guides, the journey must begin somewhere. In this chapter, we will start from the beginning, covering some important information about using Chef with Windows. It contains a brief refresher of some material you may already know about if you have used Chef before, and then we continue with getting Chef installed onto a Windows host.

In this chapter, we will cover the following topics:

- Reviewing key Chef terminology
- Describing the overall Chef system architecture
- Installing the Chef client on a Windows system manually
- Using the bootstrap script provided as part of Chef to install the client

Getting to know Chef better

As with any other technology, Chef has its own set of terminologies which are used to identify the various components of the Chef ecosystem. The following are some key terms along with their definitions that are used throughout this book:

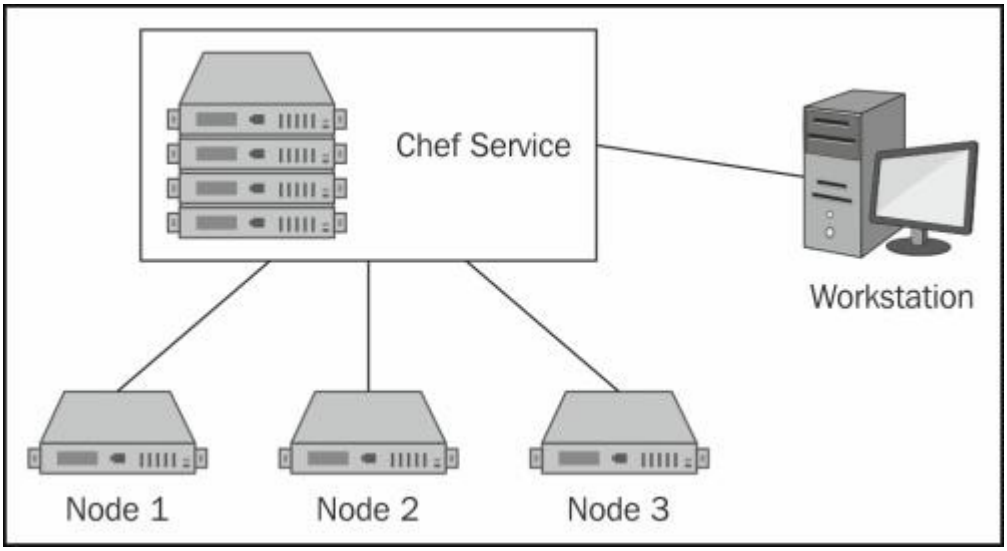
- **Node**: A node is a system that is managed by Chef. These can be servers, desktop systems, routers, or anything else that is capable of running the Chef client and has a supported operating system.
- **Workstation**: A workstation is a special node that is used by a system administrator to interact with the Chef server and with nodes. This is where the command-line tools are executed, specifically the `knife` command-line tool.
- **Bootstrap**: Bootstrap is the process of setting up a node to be used as a Chef client. This involves performing any work required to install the dependencies for Chef as well as Chef itself.
- **Bootstrap script**: There are a number of possible ways to install Chef, Ruby, and other core requirements as well as any additional configuration that is needed for your specific systems. To provide this level of flexibility, the bootstrap process is scripted; on Windows, this is a batch file.
- **Recipe**: Recipes provide the instructions required to achieve a goal such as installing a software package, configuring a firewall, provisioning users and printers, or managing other system resources. These are written in Ruby and executed on the nodes specified by the system administrator through the Chef console.
- **Cookbook**: A cookbook is a collection of recipes. Typically, a cookbook provides one specific group of actions such as installing

Apache or MySQL, providing Chef resources for a specific software tool, and so on.

- **Attributes**: Various components of the system have their own attributes; properties that describe how the software needs to be configured. These properties are defined at various levels, ranging from node-specific settings to general defaults for a cookbook or a role.
- **Role**: A role is a data structure that describes how a node that has the role applied to it should be configured. It contains a list of recipes that are to be run and the configuration data to be applied to nodes that are associated with that role. Examples of roles might include MSSQL Servers, Exchange servers, IIS servers, file servers, and so on. Note that the role itself knows nothing about nodes; the association between the node and its role is created by the administrator by applying roles to nodes.
- **Run list**: A "run list" is a list of recipes to be applied to a given node in a certain order. A run list can be composed of zero or more roles or recipes, and the order is important as the run list's items are executed in the order specified. Therefore, if one recipe is dependent upon the execution of another, you need to ensure that they run in the correct order.
- **Resource**: Resources are the way of describing *what* a recipe is processing. Some examples of resources would include files, directories, printers, users, packages, and so on. A resource is an abstraction of something that is concretely implemented in a provider.
- **Provider**: A provider is a concrete implementation of a resource. For example, a user is a generic resource, but LDAP users or Active Directory users are concrete implementations of a user resource. The type of provider being selected will depend on factors such as the platform.
- **Data bags**: Data bags contain shared data about your infrastructure. Information that is not specific to a role or a resource such as firewall rules, user accounts, and so on would be stored in data bags. This is a good place to store system-wide configuration data.
- **Environments**: Environments provide a level of encapsulation for resources. For example, you may have two identical environments, one for testing and one for production. Each of these may have similar setups but different configurations such as IP addresses and users.

An overview of Chef's architecture

Chef has three main players in its overall architecture: the nodes that are being managed (servers, desktop clients, routers, and so on), the workstation that a system administrator uses to run the `knife` command, and the Chef service that is responsible for storing and managing all of the roles, recipes, and configuration data to be applied to the end hosts. The following diagram represents the Chef architecture:



The nodes communicate with the Chef service over HTTP (preferably HTTPS) using the `chef-client` script provided as part of the Chef client installation. This is a Ruby script that is responsible for connecting to the configured Chef service (self-hosted or using hosted Chef) and downloading the run list that is configured for that node along with any cookbooks and configuration data it needs. Once it has done that, `chef-client` will evaluate the run list in order to execute the recipes in the order in which they were specified.

The workstation also communicates with the Chef service using HTTP(S), but its job is a bit different. The workstation is where a system administrator will use the command-line utilities to interact with the data stored in the Chef service. From there, the administrator can read and modify any of the system data as JSON, perform searches, and interact with the nodes through the `knife` command-line utility.

In addition to the command-line utility, Chef also presents a web-based interface for modifying the system data. Anything that can be performed by the web interface can also be achieved using `knife`; however, there are a number of advanced operations that cannot be performed without `knife` such as executing remote commands on a group of hosts and searching through data.

Installing the Chef client on Windows

In order to install the Chef client on Windows, there are three basic options to be performed, as follows:

1. Use the `knife-windows` plugin to bootstrap the host as described previously.
2. Download and manually install the client using the MSI installer.
3. Deploy the software via an in-place update tool such as WSUS (this mechanism will not be discussed because it is outside the scope of this book).

Preparing to bootstrap Windows hosts

As discussed in the previous chapter, bootstrapping a host is the process of installing any components required to initially incorporate a host into your infrastructure. Typically, this will involve installing Ruby along with the Chef client and any certificates required for authentication, as well as registering the host with your Chef server or hosted Chef account. In order to do this, you will need to have a workstation configured with the `knife-windows` gem installed. You can install the gem easily with the following command:

```
gem install knife-windows
```

Tip

You may need to perform this action as an administrator on Windows or via `sudo` on a Linux system if you are using the system Ruby installation. Alternatively, you may wish to install Chef and any subsequent gems with a Ruby version manager such as RVM, rbenv, or pik.

The `knife-windows` gem may have an issue with character encoding on versions of Ruby greater than 1.9. If you encounter issues such as the one outlined at https://tickets.opscode.com/browse/KNIFE-410, try installing the `chef` and `knife-windows` gems using Ruby 1.9. (This is where a Ruby version management tool comes in handy.)

Before proceeding to the bootstrap phase, you need to ensure that the following things are configured on the Windows host:

- Windows Remote Management is enabled

- Any firewalls in between you and the Windows host permit WinRM traffic
- You are able to authenticate with WinRM

Enabling Windows Remote Management

Most modern Windows server OS installations enable Windows RM by default, but if it is not already enabled (or if you are unsure), you can quickly configure it by running the following command from a command prompt (as an Administrator):

```
winrm quickconfig
```

The preceding command performs the following tasks:

- Starts the Windows Remote Management service and sets it to autostart on boot-up
- Defines firewall rules for the Windows Remote Management service and opens the ports for HTTP and HTTPS

Configuring firewall ports

In order to use WinRM, the firewall on the end host will need to be configured to permit traffic to the ports that WinRM uses. There are two versions of WinRM, 1.0 and 2.0; each version has a different set of default ports. For version 2.0 of Windows Remote Management, the default ports are as follows:

- HTTP: 5985
- HTTPS: 5986

For version 1.0, the default ports are as follows:

- HTTP: 80
- HTTPS: 443

Enabling basic authentication

To allow the authentication to WinRM from the outside and via basic, non-encrypted HTTP authentication, you will need to run the following commands from the command prompt (not via the PowerShell prompt):

```
winrm set winrm/config/client/auth
@{Basic="true"}
winrm set winrm/config/service/auth
@{Basic="true"}
```

```
winrm set winrm/config/service
@{AllowUnencrypted="true"}
```

> **Tip**
>
> As this permits unencrypted basic HTTP authentication, do not do this for public hosts, as it could present a significant security risk. Use an appropriately defined policy and certificates with HTTPS for a more secure authentication mechanism.

Bootstrapping a Windows host

Once the `knife-windows` gem is installed on the workstation, you can proceed to execute the bootstrapping process, such as the following bootstrapping of an EC2 Windows host:

```
knife bootstrap windows winrm
ec2-54-204-177-250.compute-
1.amazonaws.com -x Administrator
```

In the preceding command, we are executing the `knife` utility with the `bootstrap` command, specifying that we want to use the Windows bootstrap script through the Windows Remote Management protocol on host `ec2-54-204-177-250.compute-1.amazonaws.com` as the Administrator user (via the `-x` command-line flag).

This will take some time, as it will need to download and install the Chef client and then register itself with the Chef server. While it is running, the following is an overview of what it is doing:

1. The `knife` command will render a batch file from an ERB template located on the workstation (the one initiating the bootstrapping) and write it to a file on the target node.
2. After generating the batch file from the ERB file template, the node will execute the newly created batch file.
3. The generated script will download the Chef client MSI installer onto the node and perform the installation.
4. Once the installation of the client has succeeded, `knife` will store any files needed to get the host registered including certificates and configuration data.
5. The Chef client will perform an initial pass to execute any initial run lists that were specified on the command line (in our case, none).
6. Once the Chef client has completed its initial set of work, it will print out what it did and how long it took to complete that task.

Once the target node has completed the bootstrapping, you should validate that it was provisioned with the following command:

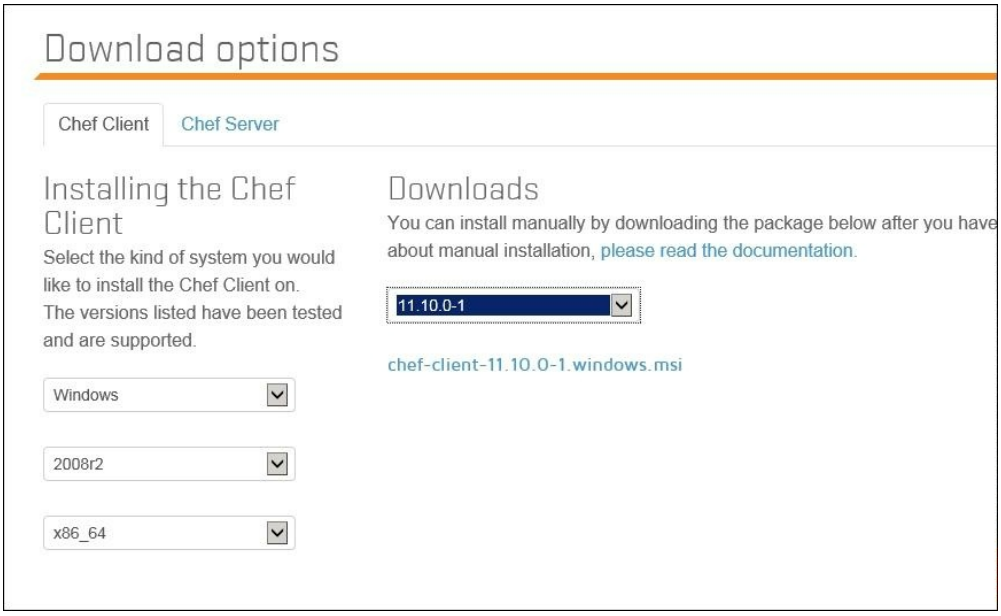
```
[user@workstation]% knife node list

WIN-
7QKOA4QQ21M
[user@workstation:~]
```

Installing via MSI

An alternative, for those who already have tools in place to deploy software using **Microsoft installer packages** (**MSIs**), is to use an MSI that is available for installing the Chef client on a Microsoft Windows machine. To download the Chef client for Windows systems, perform the following steps:

1. Go to http://www.opscode.com/chef/install.
2. Click on the **Chef Client** tab.
3. Click on **Windows** and choose an appropriate version and architecture.
4. Under **Downloads**, select the version of the Chef client to download, and then click on the link that appears below the drop-down menu to download the package.

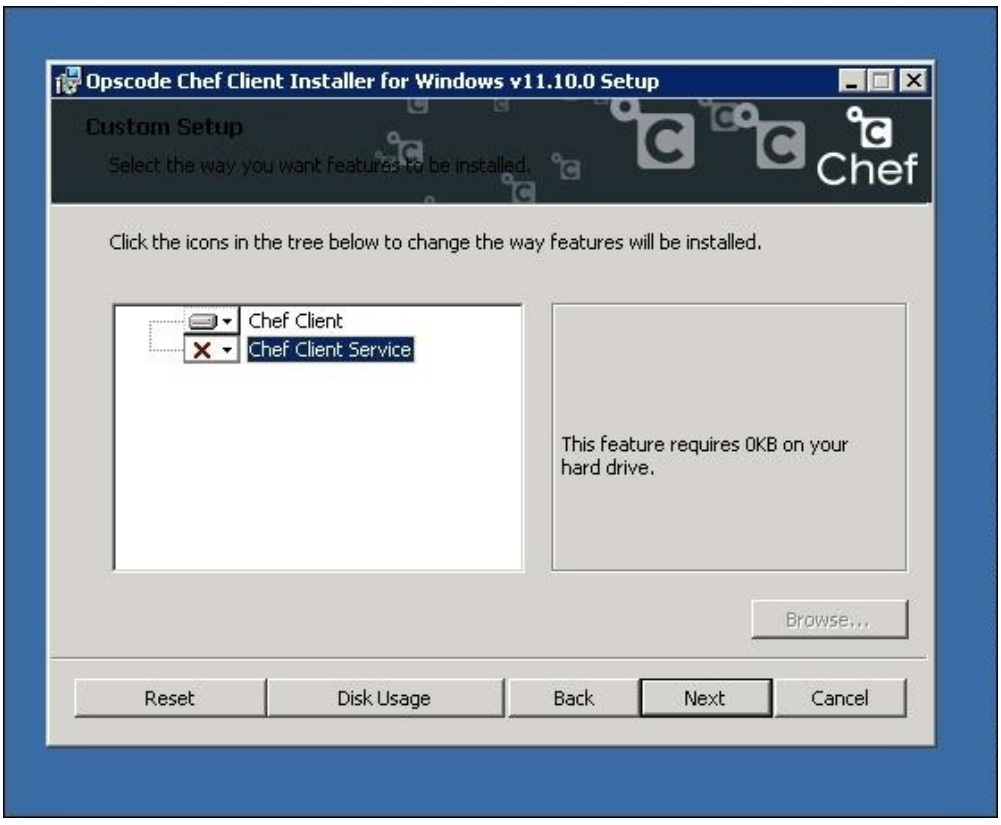
   The web page shown in the following screenshot appears when the preceding steps are performed:



5. Once the installer has downloaded successfully, launch the installer as follows:



6. Select the default options (refer to the information box after the following screenshot for information about the service):



By default, the Chef client is only run manually—someone must execute the command through a WinRM session or on the console. In order to run the client periodically in the background, you must enable the service or configure a scheduled task. If you check the **Chef Client Service** box during the installation phase, the service will be set to run automatically. If you want to enable this later, you can do so with the following command:

```
chef-service-manager -a install
```

If you choose to run the client in the background, you will not be able to see the console output. To find the execution logs, refer to the file `chef-client.log` under `c:\chef`.

> ## Note
>
> Installing the Chef client manually through the MSI download option will not automatically register the host with the Chef service. In order to do this, you will need to copy the validator key and a functional `client.rb` configuration file onto the end host.

For reference, a sample `client.rb` file for a self-managed Chef service would look like the following:

```
log_level        :info
log_location     STDOUT
chef_server_url  'https://yourchefserver.com/'
validation_client_name 'chef-validator'
validation_key   "C:/chef/validator.pem"
```

A `client.rb` file when using the hosted Chef service would look like the following:

```
log_level        :info
log_location     STDOUT
chef_server_url
'https://api.opscode.com/organizations/ORGNAME'
validation_key   "C:/chef/ORGNAME-validator.pem"
validation_client_name 'ORGNAME-validator'
```

For a self-managed Chef service, the `validator.pem` file will likely be located under `/etc/chef`. For a hosted Chef service, you can download your validator key from the management console.

Summary

By now, you should have a good feeling for the key components of the Chef architecture including important terminology and how to install the Chef client on Windows-based hosts through two different mechanisms:

- Bootstrapping the host with `knife` from a workstation
- Installing the client onto a node manually using the MSI package

In the next chapter, we will take a look at what Windows-specific resources and features are available to us through Chef.

Chapter 3. Windows-specific Resources

When managing Windows with Chef, there are some Windows-specific resources that are available to you as part of the Windows stack. This chapter covers those resources that are specific to Windows such as the Windows Registry, roles, MSIs, and so on; the ones that won't be available on Linux systems.

Working with Windows-specific resources

As might be expected by most systems administrators, managing Windows means that there are resources and configuration data that are not available to non-Windows systems. A list of those resources includes the following:

- Roles and features
- Batch scripts
- Autorun scripts
- Software packages (MSIs, installers, and so on)
- Printers
- Registry manipulation
- Paths
- Tasks (requires Windows Server 2008)
- Pagefiles
- System reboots
- ZIP files

Platforms supported by Chef

Not all versions of Windows are supported by Chef, and not all functionality is supported on all platforms. A list of Windows versions known to work with Chef is as follows:

- Windows XP
- Windows Vista
- Windows Server 2003 R2
- Windows 7
- Windows Server 2008
- Windows Server 2012

These resources are provided by the `windows` cookbook (https://github.com/opscode-cookbooks/windows) and behave like any other Chef resource apart from the fact that they are platform-specific resources and have no providers on non-Windows systems. Let's take a look at these in detail, with examples of how to use them.

In order to use the `windows` cookbook, you need to have a few dependencies installed, specifically the `chef_handler` and `powershell` recipes.

> **Tip**
>
> We cannot specifically depend on Chef's `powershell` cookbook because `powershell` depends on this cookbook—this creates a circular dependency if you are not careful. As a result, do not add a dependency, but rather ensure that `recipe[powershell]` exists in the node's expanded run list.

Managing roles and features

Similar to how Linux distributions have package management tools and a repository of packages, Windows has long had built-in packages that come with the OS. Both desktop and server releases of Windows have installable components out of the box, with servers having more than desktops.

In Windows parlance, roles are similar to Chef's notion of roles—a collection of software packages and services that work together to provide a certain set of functionality such as web application services or DNS. Multiple services can be required to provide a particular role on a Windows server. However, because these roles are managed as part of Windows, the level of control that you have over them is somewhat limited through Chef. You can enable or disable them through the `windows_feature` resource, but Windows (instead of Chef) determines what gets installed.

Features, on the other hand, are more like packages; they provide functionality that may not be critical to participating in a particular role, or may not even be related to roles in general. Chef refers to both roles and features as *features*, and the `windows_feature` resource provides a mechanism for managing both of these through Chef.

There are currently two providers for `windows_feature`: **Deployment Image Servicing and Management** (**DISM**) and `servermanagercmd` (the server manager command-line tool). The `servermanagercmd` command is deprecated in favor of DISM as the recommended mechanism for managing roles and features. Therefore, the default provider used by Chef is DISM if it is present on the system being managed, falling back to using `servermanagercmd` if it is not.

For a complete list of all roles and features that are available on a node, use one of the following mechanisms. On DISM-enabled systems, run the `dism` command as follows:

```
dism /online /Get-Features
```

The output of this command gives you a list of features and their current status. The following screenshot is an excerpt from the output obtained on running this command on a host:

```
PS C:\Users\Administrator> dism /online /Get-Features

Feature Name: FaxServiceRole
State: Disabled
Feature Name: Printing-Server-Role
State: Enabled
```

Here, the `Feature Name` key would be used to map to the `feature_name` attribute in the `windows_feature` resource. For hosts without DISM, use the `servermanagercmd` tool as follows:

```
servermanagercmd -query
```

The output of this command would look like the following screenshot:

```
PS C:\Users\Administrator> servermanagercmd -query

Starting discovery
.........
Discovery complete.

------ Roles -----
[ ] Active Directory Certificate Services [AD-Certificate]
   [ ] Certification Authority [ADCS-Cert-Authority]
[ ] Application Server [Application-Server]
   [ ] Application Server Core [AS-AppServer-Core]
   [ ] Web Server (IIS) Support [AS-Web-Support]
```

In the preceding screenshot, the string in square brackets would be the value to supply to the `feature_name` attribute to manage that component.

In order to install these, Chef provides us with the `windows_feature` resource, which is described in the following table:

windows_feature	
Action	**Description**
`install`	This installs the specified Windows role or feature.
`remove`	This uninstalls the specified Windows role or feature.
Parameter	**Description**

windows_feature	
`feature_name`	This is the name attribute—Windows' fully qualified name of the feature or role to be managed. The name may vary depending on the provider being used.

Installing roles using different mechanisms

As discussed earlier, there are multiple backends for the Windows feature resource—DISM and `servermanagercmd`. Each one has a specific Ruby class that will be used based on the determined backend as follows:

- `Chef::Provider::WindowsFeature::DISM`: This uses DISM to manage roles/features (default unless DISM is not present)
- `Chef::Provider::WindowsFeature::ServerManagerCmd`: This uses Server Manager to manage roles/features (the fallback provider when DISM is absent)

Examples of using features and roles

Enable the printer service role (`Printing-Server-Role`) as shown in the following example output from DISM:

```
windows_feature "Printing-Service-Role" do
  action :install
end
```

Since you can use arbitrary Ruby code inside your recipes, you can perform an operation on multiple components with ease. For example, to remove both the fax server and the printing server, you could use the following code:

```
to_remove = ["Printing-Service-Role",
"FaxServiceRole"]

to_remove.each do |feat|
  windows_feature feat do
    action :remove
  end
end
```

In order to enable IIS on a system where DISM is not available, you would need to use the role name as output using `servermanagercmd` as follows:

```
windows_feature "AS-Web-Support" do
  action :install
end
```

Executing batch scripts

Similar to Linux script resources for `bash`, `ruby`, and so on, Chef can execute arbitrarily-defined Windows batch scripts through the command interpreter. When these resources are used, Chef compiles the contents of the `batch` script as defined in the resource block's `code` attribute and then deposits it on the managed host and it is executed from there.

> **Note**
>
> Take caution when using script resources; they are unstructured and can easily perform actions that have unintended side effects. Similar to the way the software is built, two immediate subsequent runs of the Chef client on a node should have the same effect as only running it once in order to guarantee a reliable and consistent system configuration. Make sure that you develop scripts that are idempotent in nature (that is, it can be run multiple times and have the same effect as only running once), or use conditionals to prevent multiple executions.

Since scripts are arbitrary and of a free form, you can use them to achieve anything that you cannot model using the existing resources. However, care must be taken to prevent repeated execution that would cause negative side effects. One way to avoid performing potentially destructive actions is to use the `not_if` and `only_if` conditions to prevent multiple executions. That being said, if you find that you are performing the same type of action repeatedly, consider writing a custom resource and provider if possible.

> **Tip**
>
> Chef 11.6.0 and upwards includes a built-in `batch` resource; use `windows_batch` when implementing an earlier Chef version.

When running a batch script on a Windows host using earlier versions of Chef, the `windows_batch` resource can be used. The following table shows the available actions and parameters when using the `windows_batch` resource:

windows_batch	
Action	**Description**
`run`	This runs the batch script that is specified.
Parameter	**Description**
`code`	This is the batch script to be executed. It is an arbitrary string (refer to the example following the table).
`command`	This is a name attribute—the name of the block being executed. Typically, this should have a meaningful value.
`creates`	A file is to be used as a semaphore for this script. If the file specified exists, this script is not executed as it is expected to be created by the script.
`cwd`	Change the working directory to this before running the batch script.
`flags`	These are the command-line flags passed to the `cmd.exe` interpreter when running this batch script.
`group`	This specifies the group name/ID that this script should be executed as.
`user`	This specifies the user name/ID that this script should be executed as.

Example of batch scripts

In the following code, we will look at how we might build a command to execute `rsync` with some parameters specified using the `node` attributes:

```
windows_batch 'synchronize_files' do
  code <<-EOH
  rsync.exe -a -v -z #{node[:rsyncserver]}
#{node[:rsyncdest]}
  EOH
end
```

Alternatively, one might want to execute a Ruby script on the node as follows:

```
windows_batch 'execute_some_ruby' do
  cwd "C:/Temp"
   code "C:\Ruby210\ruby.exe C:\Temp\run_me.rb"
 end
```

Running scripts at startup

This resource allows you to create an autorun entry that will execute when the system is logged into. This is useful for anything that needs to be run when a user logs onto the system such as accounting, setting up user profiles, paths, environment variables, downloading patches or updates, making certain that specific programs are running, and so on.

In the following table, we describe the `windows_auto_run` resource along with its available actions and parameters:

windows_auto_run	
Action	**Description**
`create`	This makes a new item that executes at login.
`remove`	This removes a previously created autorun entry.
Parameter	**Description**
`args`	These are the arguments to pass to the autorun program.
`name`	This is the resource name parameter used to name the autorun script.
`program`	This is the program to be executed.

Example of creating an autorun script

Install an autorun item with the following code that executes an accounting tool at login to track various bits of accounting data, only installing the autorun item if it hasn't been installed:

```
windows_auto_run 'ACCOUNTING' do
```

```
   program 'C:/MyOrg/accountingtool.exe'
   args    '/NOTIFYTHEBOSS /PLAYAPRILFOOLSJOKE'

   action  :create
   not_if  { Registry.value_exists?(AUTO_RUN_KEY,
'ACCOUNTING') }
end
```

Installing software packages

A large number of managed systems require configuration of software that is outside the scope of the built-in Windows roles and features. Chef has a very handy resource for installing arbitrary software onto a Windows host through the `windows_package` resource, which behaves somewhat like the Linux-based package resource only for Windows-specific installations. The `windows_package` resource is capable of installing software through a variety of popular installation mechanisms. Currently, that list includes the following:

- MSI packages
- InstallShield
- Wise InstallMaster
- Inno Setup
- Nullsoft Scriptable Install System

If an installer type is not provided in the resource's attributes, then Chef will try to identify the installer by examining the package. For software that does not use one of the supported installation mechanisms, Chef provides the ability to describe a custom installer workflow by providing the custom installation type.

> **Tip**
>
> In order for Chef to manage the installation of the software, it must support some form of unattended or quiet mode that does not rely on any user input to successfully install the software package. This applies for both installation and removal.

The following table lists the available actions and parameters for the `windows_package` resource:

windows_package	
Action	**Description**
`install`	This installs a software package.

windows_package	
`remove`	This removes the specified package.
Parameter	**Description**
`package_name`	This specifies the name attribute and is the display name of the application installation package; this is the value in the `DisplayName` registry key, typically found in `CurrentVersion\Uninstall` under one of the following registry keys: • `HKLM\Software\Microsoft\Windows` • `HKCU\Software\Microsoft\Windows` • `HKLM\Software\Wow6464Node\Microsoft\Windows`
`installer_type`	The possible values for the installer type are `:msi`, `:inno`, `:nsis`, `:wise`, `:installshield`, and `:custom`. Without this value, the provider will try to guess the mechanism by examining the file.
`source`	This determines where to locate the installer; this can be a local path or a URL. In cases where the path is a URL, the installer will be downloaded and executed.
`checksum`	This is the SHA-256 checksum of the file. It is typically used in conjunction with a download URL—if the cached file matches the checksum, Chef will not re-download the file present at the URL. It also prohibits unexpected packages from being installed.
`options`	This includes any command-line options to pass to the installer.
`timeout`	This is the download timeout (the default is 10 minutes).

windows_package	
`version`	This is the version of the package being installed. This can be found in the `DisplayVersion` value in the application's registry settings. If the currently installed version (value in the registry) does not match this, the package will be installed/upgraded.
`success_codes`	This is an array of return codes that indicates that the package was successfully installed. Typically, this is used with custom installers, but there are plenty of other possible cases where this is useful. The default value is (0, 42, 127)

Examples of installing Windows packages

Using a locally provided file, perform an installation of Ruby 1.9.3-p448. This package uses the Inno Setup packager and a hint is provided, but remember that even without it, the provider will try to determine the installer type:

```
windows_package "Ruby 1.9.3-p448" do
 source File.join("C:", "temp",
"rubyinstaller-1.9.3-p448.exe")
 options '/dir="C:/Ruby193" /tasks="modpath"'
 installer_type  :inno
 action :install
end
```

You can specify arbitrary flags to pass to an installer that does not use one of the known packaging systems. One example of this is Firefox, which supports passing the `-ms` command-line arguments to the installer in order to run in silent mode:

```
ff_url =
"http://archive.mozilla.org/pub/mozilla.org/mozilla.org/firefox/releases/
US/Firefox%20Setup%2028.0b4.exe"

windows_package 'Mozilla Firefox 28.0b4 (x86
en-US)' do
  source ff_url
  action :install
  installer_type :custom
```

```
  options '-ms'
end
```

The options for Firefox can be found at https://wiki.mozilla.org/Installer:Command_Line_Arguments.

In order to remove a previously installed package, such as iTunes, make use of the `:remove` action as follows:

```
windows_package "iTunes" do
  action :remove
end
```

Manipulating printers

Chef provides two different resources for managing printers—ports and printers themselves. The `windows_printer_port` resource allows you to install TCP/IP printer ports on a Windows host to connect a printer to. The `windows_printer` resource is responsible for an actual printer installation, and they are both used in conjunction with one another.

> **Tip**
>
> The printer resources depend on PowerShell to make certain that `recipe[powershell]` is included on the node's expanded run list to ensure the `powershell` cookbook is downloaded to avoid circular dependencies.

Managing printer ports

With the `windows_printer_port` resource, you can create and delete TCP/IPv4 printer ports. This resource is useful for creating printer ports with specific settings when you need control over the port configuration. The `windows_printer` resource uses the `windows_printer_port` resource to dynamically create printer ports as needed, so this typically comes in handy only if the default settings are insufficient.

The available actions and parameters for the `windows_printer_port` resource are described in the following table:

windows_printer_port	
Action	**Description**
`create`	This creates a new TCP/IP printer port (default action).
`delete`	This deletes a named TCP/IP printer port.
Parameter	**Description**

windows_printer_port	
`ipv4_address`	This specifies the resource name attribute, that is, the IPv4 address that the port is configured for.
`port_name`	This gives the optional port name. The default value is `IP_#{ipv4_address}`
`port_number`	This gives the optional port number. The default value is 9100.
`port_description`	This gives the optional port description.
`snmp_enabled`	This gives the optional flag that marks SNMP as enabled or disabled. The default value is false.
`port_protocol`	This gives the optional port protocol. The following values are supported: • 1 (RAW) • 2 (LPR) The default value is 1.

Examples of managing printer ports

Create a TCP/IP printer port named `10.0.0.1` by only specifying the IP address (everything else is set to the default value) as shown in the following code:

```
windows_printer_port '10.0.0.1' do
end
```

Delete the printer port we just created as follows:

```
windows_printer_port '10.0.0.1' do
  action :delete
```

```
end
```

Create a port with some nondefault options such as name, port number, description, and SNMP support as shown in the following code:

```
windows_printer_port '10.0.0.1' do
  port_name 'Remote office port'
  port_number 8080
  port_description 'This is my newly created
port'
  snmp_enabled true
end
```

Delete the port that was previously created as follows:

```
windows_printer_port '10.0.0.1' do
  port_name 'Remote office port'
  action :delete
end
```

Managing printers

In order to create a new printer, you will need to have the driver already installed on the system. If the driver is not already on the end host, it could easily be installed with a `windows_package` resource and delivered from a central network location in order to distribute your printer drivers to end hosts.

> **Tip**
>
> The `windows_printer` resource will automatically create a TCP/IP printer port for you using the `ipv4_address` property. If you want more granular control over the printer port, create it using the `windows_printer_port` resource before creating the printer.

The following table shows the available actions and parameters that can be used with the `windows_printer` resource:

windows_printer	
Action	**Description**

windows_printer	
create	This creates a new printer resource (default action).
delete	This deletes a named printer.
Parameter	**Description**
device_id	This is the resource name attribute—set to the printer queue name, for example, Tokyo Office HP LaserJet.
driver_name	This is required to be the exact name of the printer driver. This needs to be installed ahead of time (as mentioned previously)—use a windows_package resource if needed to install it before creating the printer.
ipv4_address	This is the printer's IPv4 address, for example, 10.4.64.23. It need not be network-accessible at creation time. We will create the port if it does not exist.
location	This is the printer location used for the printer metadata.
shared	This is the Boolean flag indicating whether this printer needs to be shared. The default value is false.
share_name	This is the printer's share name, if it needs to be shared.
comment	This will contain the optional description of the printer.

windows_printer	
`default`	This determines whether the printer is the default printer. The default value is false.

As outlined earlier, this resource will automatically create a printer port for this printer if one does not exist. In its current form, the underlying printer provider expects that the printer port has the name `IP_#{ipv4_address}`—so if you have a custom-named port, it will not be used.

Examples of managing printers

Create a new printer that is named `HP LaserJet, Bldg. A Floor 20` addressed by the `10.0.0.5` IP. This will use the default settings when constructing a printer port, so the new port will be named `IP_10.0.0.15` and will use port `9001` and the RAW protocol with no SNMP as follows:

```
windows_printer 'HP LaserJet, Bldg. A Floor 20'
do
  driver_name 'HP LaserJet 1320'
  ipv4_address '10.0.0.5'
end
```

Create a more finely grained printer and port configuration with the following code:

```
windows_printer_port '10.0.0.10' do
  port_name 'IP_10.0.0.10'
  port_number 9001
  port_description 'Building B print server'
  snmp_enabled true
end
```

The following code would create our port, `IP_10.0.0.10`, which we could then bind a printer to:

```
windows_printer 'HP LaserJet 6L - Copy Room' do
  driver_name 'HP LaserJet 6L'
  ipv4_address '10.0.0.10'
  location 'Building B, Floor 13, Copy Room'
  comment 'Haunted floor, look out for ghosts.'
end
```

Now, when the provider constructs the printer, it will use the `IP_10.0.0.10` port, which we built with custom settings earlier.

To delete a printer, leverage the `:delete` action. Note that this does not delete the port, as it is possible there may be other printers at that location. To delete the port, you must use the `windows_printer_port` resource with the `:delete` action separately as follows:

```
windows_printer 'HP LaserJet, Bldg. A Floor 20'
do
  action :delete
end
```

Interacting with the Windows Registry

One of the most well-known differences between managing UNIX-like systems and Windows systems is the Windows Registry. Chef has resources for creating, modifying, and deleting Windows Registry keys. Beware that these operations are nonreversible (there is no implicit backup of values, so it may be worth preparing a backup before modifying values), and that they can potentially be very destructive.

Paths to registry keys must also include the registry hive. The hive can be fully specified or we could use the following abbreviations:

- `HKLM` for `HKEY_LOCAL_MACHINE`
- `HKCC` for `HKEY_CURRENT_CONFIG`
- `HKCR` for `HKEY_CLASSES_ROOT`
- `HKU` for `HKEY_USERS`
- `HKCU` for `HKEY_CURRENT_USER`

> **Tip**
>
> Chef 10.x uses a resource named `windows_registry`, which will be described here for those using an older Chef client and server. For newer install versions using 11.x, the resource is `registry_key` and is the preferred way to interact with Registry values when using a newer version of Chef. Use the one that corresponds with your Chef version.

The Chef 10.x resource

The `windows_registry` resource allows you to control registry settings, but has different actions and attributes than the `registry_key` resource. Moving from `windows_registry` to `registry_key` should be a fairly straightforward operation when it is time to migrate.

The following table provides a description of the actions and parameters used with the `windows_registry` resource:

windows_registry	
Action	**Description**
`create`	This creates a new registry key with the provided data.
`modify`	This updates an existing registry key with the desired values.
`force_modify`	This modifies an existing registry key's values but does so insistently. This action checks the value a number of times to make certain that the key contains your desired value.
`remove`	This removes a value from an existing registry key.
Parameter	**Description**
`key_name`	This is the resource name attribute specifying which registry key to create, modify, or remove.
`values`	This is a hash of the values to set under the registry key. The individual hash items will become respective *'Value name' => 'Value data'* items in the registry key.
`type`	This determines the type of key to create. The following is a list of options: • `:binary`: REG_BINARY • `:dword`: REG_DWORD • `:dword_big_endian`: REG_DWORD_BIG_ENDIAN • `:expand_string`: REG_EXPAND_SZ • `:multi_string`: REG_MULTI_SZ • `:string`: REG_SZ • `:qword`: REG_QWORD

windows_registry	
	The default value is `:string`.

Examples of managing registry keys

Manage NTP servers through the registry to allow off-site NTP synchronization with the NTP pool and update every 45 minutes until you have performed three good syncs, and then once every 8 hours with the following code:

```
regkey =
'HKLM\SYSTEM\CurrentControlSet\Services\W32Time
\Parameter'

windows_registry regkey do
  values 'AvoidTimeSyncOnWan' => 0,
         'NtpServer' => "0.pool.ntp.org",
         'Period' => 'SpecialSkew'
end
```

As shown in the following code, delete two specific key/value pairs from the registry at a given location—setting the value to blank achieves the following result:

```
windows_registry 'HKCU\Software\SomeApp\SomeKey'
do
  values 'UnwantedValueOne' => '',
         'UnwantedValueTwo' => ''
  action :remove
end
```

There are also some helper methods available for determining if keys exist and/or fetching the data from the registry. These come in handy when writing guards around other blocks or for extracting registry data for use in other resources.

Use the following code to determine if a value exists:

```
Windows::RegistryHelper.value_exists?
(path, value)
```

Use the following code to check to see if a key exists:

```
Windows::RegistryHelper.key_exists?(path)
```

Use the following code to get the value from a registry key:

```
result = Windows::RegistryHelper.get_value(path,
value)
```

The Chef 0.11.x resource

In the newer versions of Chef, the `registry_key` resource is used for setting values in the Windows Registry, as described in the following table:

registry_key	
Action	**Description**
`create`	This creates a new registry key with the provided data.
`create_if_missing`	This creates a key or value if it does not exist.
`delete`	This deletes the specified values for a specified registry key.
`delete_key`	This deletes a specific key and all subkeys.
Parameter	**Description**
`key`	This is the resource name attribute specifying which registry key to create, modify, or remove.
`architecture`	This specifies the architecture for which the keys will be created. The possible values are as follows: • `:x86`: Force 32-bit registry • `:x86_64`: Force 64-bit registry

registry_key	
	• `:machine`: Allow the client to determine The default value is `:machine`.
`provider`	This determines which provider to use for manipulating the registry. Currently, there is only one (`Chef::Provider::Windows::Registry`) whose short name is `registry_key`, but it is possible that this could change in the future. The default value is `registry_key`.
`recursive`	This determines whether this operation should be performed recursively. When creating a new registry key, create any required path components; when deleting, delete all subkeys.
`values`	This is an array of hashes that contain the values to be set under the specified registry key. Each hash is composed of three keys which are as follows: • `:name`: The value's name • `:type`: The type of value stored • `:data`: The data to store The `:type` key has the same options as the type for the `windows_registry` entries and defaults to `:string`.

Examples of managing Registry values

To set the NTP configuration the same way that would have been done with the `windows_registry` resource, a matching `registry_key` resource would look like the following code:

```
regkey =
'HKLM\SYSTEM\CurrentControlSet\Services\W32Time
\Parameter'

registry_key regkey do
```

```
 values [{
            :name => 'AvoidTimeSyncOnWan',
            :type => :reg_dword,
            :data => 0
          }, {
            :name => 'NtpServer',
            :data => "0.pool.ntp.org",
          }, {
            :name => 'Period',
            :type => :reg_string,
            :data => 'SpecialSkew'
          },]
  action :create
end
```

Managing the system path

Chef has resources to support manipulating the system path. This is useful when installing new software or configuration that needs to make changes to the system path.

The following table illustrates the available actions and parameters when using the `windows_path` resource:

windows_path	
Action	**Description**
`add`	This adds a new entry to the system path.
`remove`	This removes an entry from the system path.
Parameter	**Description**
`path`	This is the resource name attribute specifying the directory to add to the system path.

Examples of modifying the path

Add Ruby 2.1.0's binaries to the system path as follows:

```
windows_path 'C:\Ruby\2.1.0\bin do
  action :add
end
```

Remove Ruby 1.9.3 from the system path as follows:

```
windows_path 'C:\Ruby\1.9.3\bin' do
  action :remove
end
```

Scheduling tasks

The `windows_task` resource allows you to create, delete, or execute a Windows scheduled task. These are tasks that execute at regular intervals and are useful for things such as running the Chef client regularly, checking for updates, cleaning up temporary files, downloading cache data, or anything else that needs to be scheduled.

> **Tip**
>
> This resource provider requires Windows Server 2008 because of the API that it uses.

The following table lists the available actions and parameters when creating scheduled tasks through the `windows_task` resource:

windows_task	
Action	**Description**
`create`	This creates a new scheduled task.
`delete`	This deletes a task.
`run`	This runs the named task.
`change`	This updates the properties of a specified task.

windows_task	
Parameter	**Description**

windows_task	
`name`	This is the resource name attribute specifying the name of the task (arbitrary name).
`command`	This is the command to execute on an interval.
`cwd`	This specifies the directory to run the command from.
`user`	This is the user to execute the command as.
`password`	This is the user's password.
`run_level`	The level of privileges to run with are listed as follows: • `:highest` • `:limited` The default value is `:limited`.
`frequency`	This is the unit of time for which this command executes; the options are as follows: • `:minute` • `:hourly` • `:daily` • `:weekly` • `:monthly` • `:once` • `:on_login` • `:onstart` • `:on_idle` The default value is `:hourly`

windows_task	
`frequency_modifier`	This is the frequency interval such as 15, 2, 30, and so on. This is multiplied by the unit of time, so a modifier of 15 combined with `:minute` as the frequency would cause the command to repeat every 15 minutes. The default value is 1.
`start_day`	This is an optional date to start the interval, formatted as MM/DD/YYYY. This is the first date on which the command is first executed and then runs on the specified frequency interval. The command runs immediately if this parameter is not specified.
`start_time`	This determines the time of the day to perform the first execution (immediately if nil). The format is HH:MM.

Examples of managing Windows tasks

Run Chef every 15 minutes with the following code:

```
windows_task 'Chef client' do
  user 'Administrator'
  password '$ecR3t'
  cwd 'C:\chef\bin'
  command 'chef-client -L C:\tmp\'
  run_level :highest
  frequency :minute
  frequency_modifier 15
end
```

Delete the Chef client task as follows:

```
windows_task 'Chef client' do
  action :delete
end
```

Interacting with Windows pagefiles

This resource allows you to set the default (and create if it does not exist) or delete a Windows pagefile. Chef can designate the initial size, maximum size, and management pattern for the newly created pagefile.

The following table illustrates the actions and parameters that can be used with the `windows_pagefile` resource:

windows_pagefile	
Action	**Description**
`set`	This sets the default system pagefile to the specified pagefile. If the pagefile does not exist, the provider will create it automatically.
`delete`	This deletes the specified pagefile.
Parameter	**Description**
`name`	This is the resource name attribute specifying the path to the pagefile.
`system_managed`	This allows the system to manage this pagefile (initial size = 0, maximum size = 0).
`automatic_managed`	Should the size of the pagefile be automatically managed? If so, the initial size and maximum size are ignored. The default value is false.
`initial_size`	This is the initial size of the pagefile in bytes.

windows_pagefile	
`maximum_size`	This is the maximum size of the pagefile in bytes.

Examples of how to manage the pagefile

Set the default pagefile to `D:\pagefile.sys` with an initial size of 8 GB and a maximum size of 16 GB (creating it, if need be) with the following code:

```
one_gb = 1024 * 1024 * 1024
windows_pagefile 'D:\pagefile.sys' do
  initial_size (8 * one_gb)
  maximum_sze (16 * one_gb)
   system_managed false
   automatic_managed false
   action :set
end
```

Delete the pagefile at `X:\pagefile.sys` as shown in the following code:

```
windows_pagefile 'X:\pagefile.sys' do
  action :delete
end
```

ZIP files

There is not a consistent way to natively manage packing or unpacking of ZIP files on all Windows platforms. To work around this, the `windows_zipfile` resource provides a pure Ruby implementation for manipulating ZIP files on the Windows platform.

The following table lists the actions and parameters available for the `windows_zipfile` resource:

windows_zipfile	
Action	**Description**
`unzip`	This decompresses a specified ZIP file at the source into the destination.
`zip`	This creates a ZIP file at the destination from the source.
Parameter	**Description**
`path`	This is the resource name attribute specifying the path—unzipping this designates where the unzipped files go, while zipping the path specifies where the ZIP file is to be created.
`source`	This indicates the ZIP file location to uncompress when unzipping, or the directory containing files to be compressed when zipping.
`checksum`	This is the optional checksum that verifies whether the ZIP file is the expected one before decompressing it.
`overwrite`	This determines whether this operation should overwrite files.

windows_zipfile	
	The default value is false.

Examples of interacting with ZIP files

The following code compresses a ZIP file located at `D:\backups\` named as per the current date, containing the files at `C:\development\webapp\`:

```
require 'date'
today = Date.today.strftime('%Y-%m-%d')
windows_zipfile 'D:\backups\#{today}.zip' do
  source 'C:\development\webapp'
   action :zip
end
```

The following code decompresses a ZIP file located at `D:\dist\ruby-1.9.3.zip` to `C:\ruby`, overwriting files only if the checksum matches the specified one:

```
windows_zipfile 'C:\ruby' do
   source 'D:\dist\ruby-1.9.3.zip'
   checksum
'bfed985c4f0e44cd6b97f93b9440940335313384'
   overwrite true
end
```

Rebooting Windows

Often times, Windows needs to be rebooted to update system settings that have been changed during the course of an installation or reconfiguration. The `windows` cookbook provides the `windows_reboot` resource to notify `WindowsRebootHandler` that a reboot is required. If `WindowsRebootHandler` is registered as a report handler, a reboot will be requested upon a complete and successful execution of the Chef client.

Typically, this resource is notified by other resource blocks when a reboot is required, often after installing new software or changing system settings.

Your recipe will need to make sure that the reboot handler is included with the following command:

```
include_recipe 'windows::reboot_handler'
```

The following table outlines the actions and parameters that can be associated with the `windows_reboot` resource:

windows_reboot	
Action	**Description**
`request`	This requests a reboot to be scheduled once the Chef client has completed execution. This requires `WindowsRebootHandler` to be registered as a report handler.
`cancel`	This cancels any pending reboot requests by changing the `node.run_state` setting.
Parameter	**Description**
`reason`	This is a textual reason that the system is being rebooted. The default value is `Chef initiated reboot`.

windows_reboot	
`timeout`	This is the name attribute that specifies how long the command will wait in seconds until rebooting. The default value is 60.

Examples of scheduling a reboot

The following example shows you how to schedule a reboot with a three-minute timeout at the end of a Chef client run by notifying the reboot block when a package is successfully installed via `windows_package`:

```
windows_reboot 180 do
  reason "Installed a new package"
  action :nothing
end

windows_package 'sql_server' do
 action :install
 notifies :request, 'windows_reboot[180]'
end
```

If you need to cancel a reboot, you can do so from a resource block using the following code:

```
windows_reboot  do
  action :cancel
end
```

Summary

This chapter has exposed you to the various Windows-specific resources that are provided when using Chef. As you can see, these resources are very similar to those you may have encountered when managing Linux-based systems, only tailored to the Windows environment.

In the next chapter, we will take a look at how to use the resources that are available for Windows management in order to provision an application stack using IIS, the .NET framework, and a database server.

Chapter 4. Provisioning an Application Stack

Now that you have learned about the Windows-specific resources and some background information on using Chef with Windows, let's take a look at how to provision a full-stack application. In this chapter, we will dissect a cookbook that is responsible for installing an open source CMS application, Umbraco. Note that the recipe in this cookbook requires Windows Server 2012 as the target platform, and plan accordingly if you are going to run it yourself.

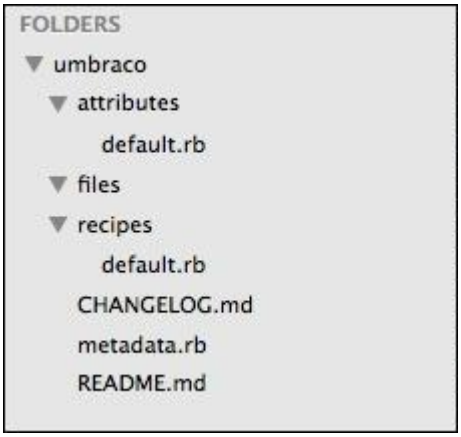
In this chapter, we will cover the following topics:

- Bootstrapping a new Windows host
- Installing the dependencies for the Umbraco CMS
- Installing the Umbraco CMS on the Windows host
- Configuring a site in the IIS to run the Umbraco CMS

Examining the cookbook

We will dive into a cookbook that has been written specifically to address the installation of the Umbraco CMS along with its requirements. The source code for this cookbook is available on GitHub at http://github.com/johnewart/umbraco-cookbook.

Chef cookbooks share the same layout and structure whether they target Windows or other platforms. The primary difference in a cookbook that supports Windows will be that its `metadata.rb` file declares that the cookbook supports Windows and the recipes contained within the cookbook will support Windows through a combination of Windows-specific resources and conditional logic that may behave differently on the Windows platform. The following screenshot shows the layout of the `umbraco` cookbook:



Here, you will notice that the cookbook looks just like any other cookbook; it can contain folders for recipes, attributes, templates, resources, providers, and any other Chef resources. In this case, the cookbook only contains some default attributes and recipes as it is designed to be a compact example for Windows.

If you look at the `metadata.rb` file for this cookbook, you will see the following contents:

```
name             'umbraco'
maintainer       'John Ewart'
maintainer_email 'john@johnewart.net'
license          'Apache 2.0'
description      'Installs and configures
umbraco, an open source CMS for ASP.NET
(http://www.umbraco.com)'
long_description 'Install and configure the
Umbraco CMS'
version          '1.0'

# Cookbook dependencies (IIS and Windows)
depends          iis
depends          windows

# Supported platforms
supports 'windows'
```

For those who are still somewhat new to Chef, the `metadata.rb` file describes who the maintainer of the cookbook is, what license it adheres to, the version number, any cookbooks that it is dependent upon, and what platforms are supported.

In this case, we can see that the cookbook supports the Windows platform and requires the `iis` and `windows` cookbooks in order to function. These are going to be required for just about any IIS web application you want to install via Chef.

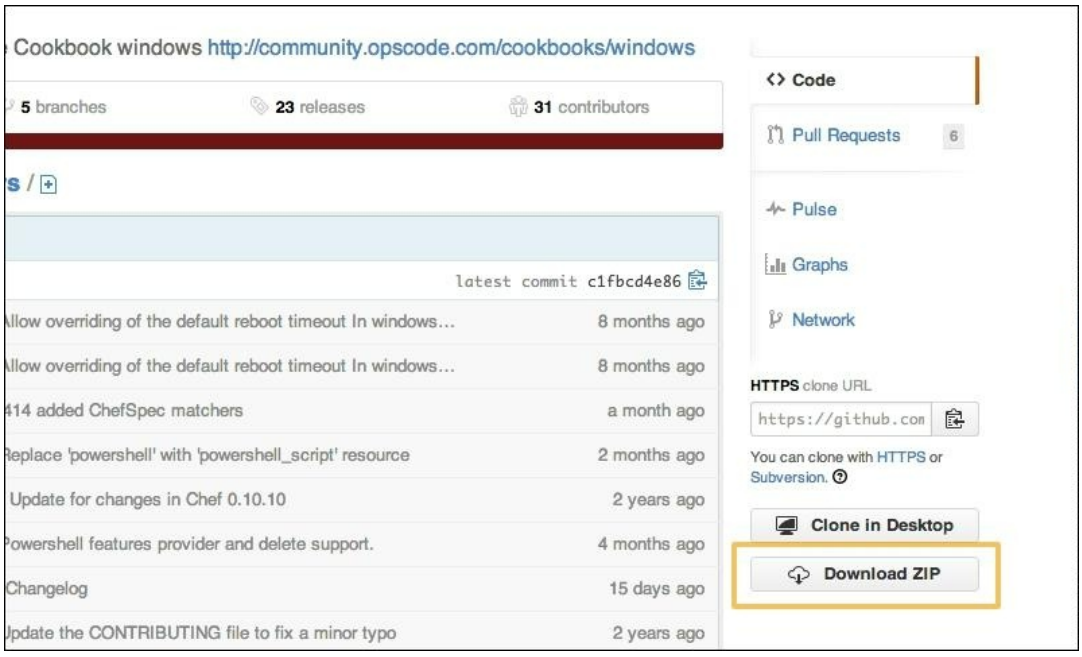
Installing the cookbook

As mentioned, this cookbook has two direct dependencies, `windows` and `iis`, which in turn are required in a third dependency, `chef_handler`. All of these cookbooks can be found at the following locations:

- https://github.com/opscode-cookbooks/windows
- https://github.com/opscode-cookbooks/iis
- https://github.com/opscode-cookbooks/chef_handler
- https://github.com/johnewart/umbraco-cookbook

Fetching from GitHub

In order to use these resources, you will want to either download them from GitHub or use Git to clone them to a local system. If you are not interested in working with them as Git repositories (that is, version controlled source code), downloading them will be the simplest way to get them. If you go to the previous URLs, you will see a link titled **Download ZIP** on the right-hand side of the screen, as shown in the following screenshot:



If you opt to download them as ZIP files rather than clone them with Git, you will want to ensure the directories are named to match the proper name of the cookbook (`chef_handler`, `windows`, `iis`, and `umbraco`). For example, if you clone the `umbraco` cookbook, it will be put in an `umbraco-cookbook` directory, whereas decompressing the ZIP files may result in an `umbraco-cookbook-master` directory. Once you have them downloaded and

renamed, install them on your Chef server by running the following command from the directory where all four cookbooks are located:

```
knife cookbook upload -o . chef_handler windows
iis umbraco
```

Chef will validate that each cookbook's dependencies are satisfied (or are in the process of being uploaded) as you upload them. The `chef_handler` cookbook must be installed in order for the `windows` cookbook to work, which is needed to support the `iis` cookbook, and then finally the `umbraco` cookbook can be installed as its dependencies have been met.

Examining the recipe

Now that you have the source for the `umbraco` cookbook downloaded to your local machine, let's take a look at what it's doing. Umbraco is a very easy-to-install ASP.net CMS application that will give you a good feeling for how to install and configure an IIS application using Chef. Umbraco was chosen for this example cookbook because it does not require a database, although it supports one, and its single ZIP file installation makes it easy to follow along.

Let's take an in-depth look at the `default.rb` recipe that is contained within the `umbraco` cookbook. The code snippets in the upcoming section are contained within `umbraco/recipes/default.rb`.

Installing the prerequisites

Before we can set up the IIS application, our recipe will need to ensure that the IIS web server Windows role is installed on the host. As we have seen in the previous chapter, Chef has a resource, `windows_feature`, that allows us to ensure that it is either already available or gets installed as part of the recipe execution. Let's look at the recipe's components piece by piece:

1. First, install the role using the `IIS-WebServerRole` role as follows:

```
windows_feature 'IIS-WebServerRole' do
  action :install
end
```

2. Once that has been installed, install the ASP.net feature inside of IIS; however, that has some other prerequisites, namely the ISAPI filter, ISAPI extensions, the .NET 3 server features, ASP.net 4.5 support, and the .NET 4.5 extensibility libraries. These are the required roles on Windows Server 2012; discovering the names of the roles can be accomplished by using command-line tools such as `dism` or `servermanagercmd`. The following code snippet shows the installation of the ASP.net feature of IIS:

```
features = %w{IIS-ISAPIFilter
IIS-
ISAPIExtensions NetFx3ServerFeatures
NetFx4Extended-
ASPNET45 IIS-NetFxExtensibility45}

features.each do |f|
  windows_feature f do
    action :install
  end
end
```

3. After installing the dependencies, install the IIS ASP.net 4.5 feature on the Windows host as follows (note that this is required to run the Umbraco application):

```
windows_feature 'IIS-ASPNET45' do
  action :install
end
```

Preparing the IIS service

By now, our recipe has completed the installation of the required roles and features in order to manage an IIS application. Let's see how it prepares the IIS service by removing the default IIS site and application pool from our server.

The `iis` cookbook has a default recipe that turns on a service named `W3SVC`. We do not want to do this, instead we want to add our own IIS service. We can accomplish this by" overriding the `action` attribute to `:nothing`. This will prevent the default `W3SVC` service from being installed, as shown in the following code snippet:

```
service "iis" do
  service_name "W3SVC"
  action :nothing
end
```

We don't want the default IIS application to be serving traffic, so we will remove it using the `remove_default_site` recipe from the `iis` cookbook as follows:

```
include_recipe "iis::remove_default_site"
```

If you dig into the `iis` cookbook, you will find that this is just a convenient way of performing the following operations, and either mechanism would achieve the same goal:

```
iis_site 'Default Web Site' do
  action [:stop, :delete]
end

iis_pool 'DefaultAppPool' do
  action [:stop, :delete]
end
```

Fetching the application

Umbraco's distribution comes as a ZIP file, which we need to download and then decompress. Here, the resource name (the argument passed to the

resource) is the directory that we are targeting for expansion of the ZIP file. The destination is found in the hash element `node['umbraco']['approot']`, which if not overridden will be defined as `"#{ENV['SYSTEMDRIVE']}\\inetpub\\apps"` as specified in the `attributes/default.rb` file. If you are using a simple server setup such as EC2 or a vanilla install, this will likely map to `C:\inetpub\apps` on your target host. The file being downloaded is specified by the `source` attribute and by default will point to http://our.umbraco.org/ReleaseDownload?id=92348. If the package has already been downloaded and uncompressed, this resource will be skipped due to the `not_if` check, as shown in the following code snippet:

```
windows_zipfile node['umbraco']['app_root'] do
  source node['umbraco']['dist']
  action :unzip
  not_if
{::File.exists?(::File.join(node['umbraco']
['app_root'], "umbraco"))}
end
```

Configuring the application

Umbraco has some directories that the IIS user will need to have access to. The following code block will make sure that we grant the IIS user permission to modify the required directories:

```
%w{App_Data Views css config media masterpages
xslt usercontrols bin umbraco scripts images
macroscripts}.each do |d|
  directory
win_friendly_path(::File.join(node['umbraco']
['app_root'], d)) do
    rights [:read,:modify], 'IIS_IUSRS'
  end
end
```

Similarly, the IIS user needs to be able to update the `web.config` file to update it with some settings when you run the web-based installation utility. We need to grant the modify permission for the `web.config` file to the IIS user using the following code snippet:

```
file
win_friendly_path(::File.join(node['umbraco']
['app_root'], "web.config")) do
  rights :modify, 'IIS_IUSRS'
end
```

Generating an IIS pool and site

Once the IIS services have been installed and the application code has been downloaded, we need to configure an IIS pool for the application to run in, specifying the .NET runtime version to use (4.0 in this case) as follows:

```
iis_pool node['umbraco']['pool_name'] do
  runtime_version "4.0"
  action :add
end
```

Once we have the pool configured and the files deployed, instruct IIS to define a site that uses the application pool with the path pointed to our site root on port 80. Once it's been added, we want to start it (as indicated by providing the `action` attribute with an array containing `:add` and `:start`). The path is pointed at the unzipped Umbraco installation directory `#{node['umbraco']['approot']}\\umbraco`, which will be expanded as `C:\inetpub\apps\umbraco by default`, as shown in the following code snippet:

```
iis_site 'umbraco' do
  protocol :http
  port 80
  path node['umbraco']['app_root']
  application_pool node['umbraco']['pool_name']
  action [:add,:start]
end
```

Performing the installation

Now that you know what the cookbook and its recipe are doing, let's go ahead and apply the recipe to our Windows host. Here, we will install the Umbraco CMS onto a Windows host through the following steps:

- Bootstrapping the Windows server with the Chef client
- Creating a role for the Umbraco CMS application
- Adding the `umbraco` recipe to the Umbraco role's `run_list`
- Applying the newly created role to the host
- Completing the configuration of the CMS through a web browser

Bootstrapping the host

In the following example, we are using a Windows Server 2012 host with a fresh installation of Windows. As usual, we will bootstrap our host using `knife` as follows:

```
knife bootstrap windows winrm HOSTIP -x
Administrator -d windows-chef-client-msi
```

This will execute the contents of the `windows-chef-client-msi.rb` bootstrapping template on the host located at `HOSTIP`, installing the Chef client and registering the host with the Chef service. Once it has completed, you will see an output similar to the following screenshot:

```
WARN: Node WIN-CJDQ9DEOJFK has an empty run list.
INFO: Chef Run complete in 1.985377 seconds
INFO: Running report handlers
INFO: Report handlers complete
INFO: Sending resource update report (run-id: 2b919690-3496-447a-a764-08aae2c705b9)
```

You can see from the preceding screenshot that the host is `WIN-CJDQ9DEOJFK`, and it had no run lists assigned to it. This makes sense, as we have not given it any Chef roles yet, so there is nothing to run. In order to assign a role for our new host, we must create a role for our new CMS application; here, we will create a role named `umbraco`.

Creating the role

Once the cookbook is uploaded (as performed earlier in the chapter), create a role from the following JSON in a file, `umbraco.json`:

```
{
  "name": "umbraco",
  "description": "Umbraco CMS",
```

```
  "json_class": "Chef::Role",
  "default_attributes": {
  },
  "override_attributes": {
  },
  "chef_type": "role",
  "run_list": [
    "recipe[umbraco]"
  ],
  "env_run_lists": {
  }
}
```

Create the role from the JSON file with the following `knife` command:

```
knife role from file umbraco.json
```

As you can see, the preceding JSON file defines a role (as specified by `"chef_type": "role"`) which declares that the role's `run_list` contains one entry, the `umbraco` recipe, as shown in the following code snippet:

```
"run_list": [
    "recipe[umbraco]"
  ],
```

Applying the role to the node

In order to apply the role to our newly bootstrapped node, we will need to get a list of the hosts and then edit its configuration and apply the new role to the host's configuration as follows:

1. First, get a list of nodes with `knife node list` as follows (in our case, we have a list of one host, the newly created host):

```
knife node list
WIN-CJDQ9DEOJFK
```

2. Edit the node configuration for the host you want to install Umbraco on with `knife node edit HOSTNAME`, for example:

```
knife node edit WIN-CJDQ9DEOJFK
```

   This will open the node's configuration JSON in your editor as follows (in our case, the default configuration for the host that was just created):

```
{
  "name": "WIN-CJDQ9DEOJFK",
  "chef_environment": "_default",
  "normal": {
    "tags": [

    ]
  },
  "run_list": [
  ]
}
```

Notice that the `run_list` property is currently empty (this may look a bit different if you are using an existing host and not a newly provisioned one). Add the `umbraco` role to the node's JSON under the `run_list` property as follows:

```
"run_list": [
  "role[umbraco]"
]
```

3. Apply the role to the desired nodes with `knife` using a simple search for all hosts with the `umbraco` role as follows:

```
knife winrm 'role:umbraco' 'chef-client' -x
Administrator
```

4. You will see the output of the Chef client pass by on the screen. Once it has finished, and assuming that everything went well, you can visit the newly installed Umbraco CMS by browsing to the `/install` URL at the default HTTP service on your host (in this example scenario, it would be `http://WIN-CJDQ9DEOJFK/install`). You will be guided through the steps required to finalize your Umbraco installation such as selecting the database, creating an administrative user, and selecting any starter kit and themes.

Summary

At this point, you should have a fully functioning installation of Umbraco on your Windows server without having to manually configure anything. As you can see, you could now repeat this across one, tens, or hundreds of hosts if needed by simply bootstrapping and configuring the roles for those hosts.

In the next chapter, let's take a look at how to automate the provisioning of hosts when they need custom configuration during the bootstrapping phase, and how we can use Chef to manage cloud hosts.

Chapter 5. Managing Cloud Services with Chef

One of the very powerful uses of Chef is leveraging it to extend your infrastructure into the cloud as seamlessly as onsite hardware. Chef has a number of mechanisms for integrating with popular cloud service providers. In this chapter, we find out how to use Chef to manage and deploy our software to a variety of popular cloud service providers.

In this chapter, we will cover the following topics:

- Microsoft Azure
- AWS EC2
- Rackspace Cloud

Working with Microsoft Azure

Azure is Microsoft's competitor to EC2 and Rackspace Cloud. Any of the three will provide you with Windows virtualization, so included in this chapter is the information on how to integrate Azure with Chef.

The `knife-azure` gem provides the functionality needed to control your Azure account via the `knife` utility. Chef uses plugins to provide the extended functionality, including managing cloud services. Support for Azure is present in a Ruby gem named `knife-azure` and is installed via the `gem` command-line utility as follows:

```
gem install knife-azure
```

This command will install all the gems that `knife-azure` is dependent upon, not just Chef alone.

> **Tip**
>
> As of this writing, the `knife-azure` gem has a dependency on an older version of `bundler`, which can cause issues and may require that you manually downgrade the version of `bundler` that is installed. Using RVM or another Ruby manager will help isolate these issues and allow you to manage your gems.

Downloading the management certificate

In order to use the Azure API, you need to have a management certificate that identifies you with the service. You can create and download a generated management certificate from https://manage.windowsazure.com/publishsettings/index?client=xplat.

Configuring knife for Azure

Now that you have your management certificate inside your `.publishsettings` file, `knife` needs to know about it. This is done, as you might expect, through the `knife.rb` configuration file. To let `knife` know where to find your newly downloaded `.publishsettings` file, first move it to somewhere (such as your home directory or `$HOME/.chef`) and then add the following line to your `knife.rb` file:

```
knife[:azure_publish_settings_file] =
"/path/to/publishsettings_file"
```

Now that `knife` has all the information it needs, we will be able to make API calls to Azure.

Creating a new Azure virtual machine

In order to create a new server, you need to know what image to use; you can view a list of available images using the `image list` subcommand of the `azure` plugin as follows:

```
knife azure image list
Windows-Server-2012-
Essentials  Windows   East Asia, Southeast Asia
Ubuntu-12_04_2-LTS-amd64-
server Linux     East Asia, Southeast Asia
Windows-Server-2008-
R2          Windows   West US, East Asia
```

As you can see, the output will list the image identifiers (shortened for print), the platform that they provide, and the regions that they are available in. The list of available images will be quite long and changes over time as Azure adds more base images to its growing list; thus, this command will come in handy later on any time you provision a new virtual machine. This list provides you with two pieces of information that you will need in order to construct a new host: the image identifier followed by the regions that the image is available in.

Using this information, let's take a look at the following commands to see how we could provision a new Windows Server 2012 Datacenter edition host with 127 GB of storage in the Western US datacenter with a DNS name of `00c0ff3300`:

```
knife azure server create --azure-dns-name
"00c0ff3300" \
     --azure-service-
location "West US" \
     --azure-source-
image
a699494373c04fc0bc8f2bb1389d6106__Windows-Server-
2012-Datacenter-201401.01-en.us-
127GB.vhd \
     --winrm-
user 'azure' --winrm-password '+0ps3kr3+' \
     --distro 'windows-chef-client-msi'
```

After issuing this command, you will see the status of the host provisioning as it progresses. Chef will wait until the virtual machine has been provisioned and is marked as ready. During this phase, you can see the status and how long it takes for Azure to reach that state using the following command:

```
Waiting for virtual machine to reach status
'provisioning'..... vm state 'provisioning'
reached after 2.67 minutes
```

Once the host is in the ready state, Chef will be able to tell you a little more about the host, including the IP addresses and WinRM port. You should expect an output similar to the following command:

```
DNS Name: 0c0ff330.cloudapp.net
VM Name: 0c0ff330
Size: Small
Azure Source Image:
a699494373c04fc0bc8f2bb1389d6106__Windows-Server-
2012-Datacenter-201401.01-en.us-
127GB.vhd
Azure Service Location: West US
Public Ip Address: 138.91.227.224
Private Ip Address: 100.82.76.57
WinRM Port: 5985
Environment: _default
```

The provisioning process may take a while, depending on the location and base image that you selected. During the provisioning process, it will create the WinRM user specified on the command line, `azure` in this case, with the specified password. However, WinRM's basic authentication will not be turned on by default; authentication via Kerberos will be enabled by default, but Kerberos support is outside the scope of this book. As a result, you will encounter errors here about being unable to bootstrap the host, which we will fix by enabling WinRM's basic HTTP authentication mechanism in the next set of steps.

Bootstrapping your Azure node

Your initial **virtual machine** (**VM**) will need to have basic authentication enabled for WinRM; once this is done, you can save your newly created image as a reusable Azure image with your administrative credentials and WinRM basic authentication enabled.

To turn on WinRM support for basic authentication, you will need to perform the following steps:

1. Enable RDP on your Azure instance.
2. Use Remote Desktop to connect to your new virtual machine and log in with the credentials specified during the provisioning phase (here, the account was `azure` and the password was `+0ps3kr3+`).
3. Allow WinRM to accept basic HTTP authentication by opening the Windows command prompt and running the following command:

```
winrm set winrm/config/client/auth
@{Basic="true"}
```

At this point, you can return to your workstation and bootstrap your new host with the `knife bootstrap` command as follows:

```
knife bootstrap windows winrm
00c0ff3300.cloudapp.net \
      --winrm-
user azure \
      --winrm-transport ssl
```

The preceding command will prompt you for your administrator password. After entering the correct credentials, you can proceed to bootstrap the Chef client on your new Azure host and register it with your Chef server. Once it has completed, you can verify that it was properly registered with the `node list` command as follows:

```
$ knife node list
00c0ff3300
i-49117415
```

Notice that among our registered nodes, our new virtual machine is now displayed.

Creating a reusable image

In order to create a reusable image from your newly created cloud server, use the following instructions:

1. Connect to the virtual machine via RDP.
2. Open the **Command Prompt** window as an administrative user.
3. Change the directory to `windows\system32\sysprep` and then run `sysprep`.
4. In **System Cleanup Action**, click on **Enter System Out-of-Box Experience (OOBE)** and make sure that **Generalize** is checked.
5. In **Shutdown Options**, select **Shutdown**.
6. Clicking on **OK** will prepare the host and shut down the virtual machine.
7. Go to the Azure management portal and click on **Virtual Machines**.
8. Select the virtual machine you want to capture.
9. On the command bar at the bottom of the page, click on **Capture**.
10. Type a name for the new image in the dialog box that is provided; as an example, we will call it `windows2012winrm`.
11. Click on the check mark to capture the image.

> **Tip**
>
> When you capture an image of a virtual machine, the original machine is deleted.

12. The new image is now available on the web control panel (under **Images**), or can be found via the `knife azure image list` command.

Now that you have saved the image, you can use it to generate new hosts that will be preconfigured with WinRM's basic authentication enabled. If you run the command after the imaging is complete, you will see something similar to the following output:

```
$ knife azure image list |grep winrm
windows2012winrm                          Windows
West US
```

Here, we have filtered out only images with the word `winrm` in them, showing us only our newly created image, `windows2012winrm`. You will also want to notice that images are locked to the region you created them in.

You can use this at a later time to provision a host using the `--azure-source-image` value of the image name, which is `windows2012winrm` in our example:

```
knife azure server create --azure-dns-name
"0c0ff330" \
     --azure-service-
location "West US" \
     --azure-source-
image windows2012winrm \
     --winrm-
user 'azure' --winrm-password '+0ps3kr3+' \
     --distro 'windows-chef-client-msi'
```

Check whether it has been created in Azure with `knife azure server list` as shown in the following code:

```
knife azure server list
DNS Name              VM Name  Status IP Address
   SSH Port WinRM Port
0c0ff330.cloudapp.net 0c0ff330 ready
138.91.227.224          5985
```

As you can see, once you have a base image with the features you need, you can use this to quickly provision new hosts for your infrastructure in a consistent manner.

Managing Amazon EC2 instances

Amazon EC2 is a very popular cloud-computing platform, and `knife` has support for managing EC2 instances from the command line through the `knife-ec2` plugin. In this section, we will demonstrate the following steps for working with EC2:

- Installing the EC2 `knife` plugin
- Configuring `knife` with your AWS credentials
- Finding the desired **Amazon Machine Image** (**AMI**), a pre-built system image
- Provisioning a new host with `knife`
- Bootstrapping the newly created host
- Configuring the new host with a role

Installing the EC2 knife plugin

As of Chef 0.10, the `ec2` subcommands have been moved from being built-in to `knife` into an external gem, `knife-ec2`. In order to use EC2 commands, you will need to install the gem, which can be done with the help of the following command:

```
gem install knife-ec2
```

This will install all of the gem dependencies that the EC2 plugin requires.

Setting up EC2 authentication

In order to manage your EC2 hosts, you will need your SSH key pair and your AWS access keys set in your `knife` configuration file. Download your SSH key pair somewhere on your host (`$HOME/.ssh` on UNIX-like systems) and add the following commands to your `knife.rb` configuration file:

```
knife[:aws_access_key_id] = "YOUR ACCESS KEY"
knife[:aws_secret_access_key] = "SECRET KEY"
```

These tell `knife` which AWS credentials to use when making API calls to perform actions such as provision new hosts and terminate instances. Without this, `knife` will be unable to make API calls to EC2. With these required changes made, let's look at how to create a new EC2 instance with `knife`.

Provisioning an EC2 instance

Initially, we will look at provisioning an instance using one of the Windows 2012 Server AMIs. With `knife`, we specify the AMI to use, which availability zone to target, and what size instance to create. For example, to create `m1.large` in the `us-east-1e` availability zone with Windows Server 2012, we would need to use the AMI with `ami-2a80bd6f` as its identifier. The AMI ID can be found through the AWS EC2 dashboard or a variety of other websites that vend this information. Remember when deciding which AMI to use that some of the EC2 instances are 32-bit and some are 64-bit; choose the appropriate AMI based on the instance type, region, and storage method you want to use.

The act of provisioning a new host can be achieved simply by providing `knife` with the AMI, region, flavor, and SSH keys, as shown in the following command:

```
[user]% knife ec2 server create -I ami-2a80bd6f \
              -
f m1.large -Z us-west-1a \
              -S jewartec2
```

When you run this command, `knife` will interactively show you the progress of provisioning a new host. The following is an example of what you can expect to see during this process:

```
Instance ID: i-3c5385d60
Flavor: m1.large
Image: ami-2a80bd6f
Region: us-west-1
Availability Zone: us-west-1a
Security Groups: default
Tags: Name: i-3c5385d60
SSH Key: jewartec2

Waiting for instance.....................
Public DNS Name:
ec2-54-219-245-213.us-west-
1.compute.amazonaws.com
Public IP Address: 54.219.245.213
Private DNS Name:
ip-10-168-109-170.us-west-
1.compute.internal
Private IP Address: 10.168.109.170

Waiting for winrm.........
```

If you were to execute this, you would find that it would sit at this point and eventually timeout trying to connect to the new EC2 instance using WinRM. Unfortunately, as with Azure instances, WinRM does not allow HTTP basic authentication by default, making it a bit tricky to bootstrap the host. However, unlike Azure, EC2 allows you to provision a system and provide an initial script that is executed during the setup of the system. We can leverage this to enable the features we need in Windows during the provisioning step so that Chef can bootstrap the host, and it can be managed as a one-step process.

Executing custom user scripts in EC2

Provisioning an EC2 instance with a custom initialization script using `knife` is a two-step process. The first thing to be done is to create the script as a text file on your workstation containing the instructions to execute. Once you have that, you can provide the script to your newly created EC2 instance by using a command-line argument when provisioning a host with `knife`.

Writing the user script

The script we will be writing for our EC2 instance will be responsible for performing the following actions:

- Turn on basic HTTP authentication for WinRM
- Enable WinRM over plaintext (you will likely want to adjust this in a production environment)
- Increase the maximum memory per shell instance
- Increase the WinRM timeout to 30 minutes
- Add firewall rules to the host to enable WinRM on port 5985 and 5986
- Restart WinRM to ensure that it picks up the changes and is set to start on boot

In the following code, we designate that we are running a `powershell` script and provide the contents inside of a `<powershell>...</powershell>` block:

```
<powershell>
winrm quickconfig -q
winrm set winrm/config/winrs
'@{MaxMemoryPerShellMB="300"}'
winrm set winrm/config '@{MaxTimeoutms="1800000"}'
winrm set winrm/config/service
'@{AllowUnencrypted="true"}'
winrm set winrm/config/service/auth
'@{Basic="true"}'
```

```
netsh advfirewall firewall add rule name="WinRM
5985" protocol=TCP dir=in localport=5985
action=allow
netsh advfirewall firewall add rule name="WinRM
5986" protocol=TCP dir=in localport=5986
action=allow

net stop winrm
sc config winrm start=auto
net start winrm
</powershell>
```

Providing a custom user script

Now that we have a custom user script, `knife` can be informed to load the script using the `--user-data` command-line option. In order to acquire the randomly generated administrator password on EC2, we also need to pass the `--identity-file` option, pointing `knife` to the downloaded copy of your SSH key pair (in this case, `$HOME/.ssh/jewartec2.pem`):

```
$ knife ec2 server create -I ami-5eccf31b -f
m1.large \
         -
Z us-west-1a -S jewartec2   \
         --user-
data ec2userdata.txt  \
         --identity-file $HOME/.ssh/jewartec2.pem
```

Tip

Make certain that your default security group allows for connections to WinRM (ports 5985 and 5986 for non-SSL and SSL respectively) or that you specify a security group that does. You will not be able to bootstrap the newly provisioned host without access to these ports.

Now, this time you will see that the instance is provisioned and bootstrapped; our custom script has enabled `knife` to connect via WinRM and install the Chef client onto the host. However, here you will see something different; WinRM will be preconfigured and working, so the bootstrapping will succeed during the provisioning stage. You will see that the bootstrap script is rendered and executed on the new virtual machine. A shortened example of the output is as follows:

```
Waiting for winrm..... done

Waiting for Windows Admin password to be
available...
```

```
Bootstrapping Chef on
ec2-54-193-161-138.us-west-
1.compute.amazonaws.com
Rendering
"C:\Users\ADMINI~1\AppData\Local\Temp\bootstrap-
293489-
123904.bat chunk 1
Rendering
"C:\Users\ADMINI~1\AppData\Local\Temp\bootstrap-
293489-
123904.bat chunk 2
Starting chef to bootstrap the node...
C:\Users\Administrator>chef-
client -c c:/chef/client.rb -j
c:/chef/first-
boot.json -E _default
[2014-03-
19T08:11:30+00:00] INFO: *** Chef 11.10.4 ***
[2014-03-
19T08:11:30+00:00] INFO: Chef-client pid: 8754
```

Once bootstrapping is complete, you will be presented with the instance details a second time, including some more information such as the type, ID, and local device name of the EC2 instance's root volume (whether it is using EBS or instance storage).

> **Tip**
>
> Occasionally, EC2 instances don't properly configure the administrator password. If this happens, the bootstrap step will fail to authenticate after WinRM is available; try discarding the instance and provisioning a new one.

Once the host is provisioned and the bootstrap step is finished, assuming that there are no errors, you can verify that your newly provisioned host is listed in your Chef service as follows:

```
$ knife node list
0c0ff3300
i-49117415
```

The output will contain your newly bootstrapped node ID as specified by you on the command line, or the default EC2 instance name if you did not specify a node name. You will also see the Azure instance name if you provisioned an Azure instance in the previous section. Congratulations, you have now provisioned a new Windows Server EC2 instance and registered it with your Chef service with only two commands!

Terminating an EC2 instance

Once you are done with testing, you may not want to leave the EC2 instance running, as it will incur costs while idle. To do this, perform the following steps:

1. List your EC2 instances.
2. Delete the server from EC2.
3. Remove the server from Chef.
4. Verify that the instance no longer exists in Chef or EC2.

To list our EC2 instances, use the `server list` subcommand of the `ec2` command, which will list all of the EC2 instances in the specified region. If you do not specify a region, `us-east-1` is the default region. As an example, executing this command after provisioning the first host will show a list of one instance. The output is too wide for printing, but you will also see in the list information such as the security groups, SSH key, AMI used, IAM profile, and current state of the virtual host as follows:

```
$ knife ec2 server list
Instance ID Name          Public  IP       Private
IP      Flavor
i-
49117415 i-49117415 54.219.201.136 10.168.93.149
m1.large
```

Deleting an instance is just as easy as creating or listing them. Here, the `server delete` subcommand is invoked with the instance identifier to be terminated. This will use the EC2 API to issue a terminate command—this is not reversible and so the command will prompt you to ensure whether you really do want to delete the instance:

```
$ knife ec2 server delete i-49117415
Instance ID: i-49117415
Flavor: m1.large
Image: ami-5eccf31b
Region: us-west-1
Availability Zone: us-west-1a
Security Groups: default
SSH Key: jewartec2
Root Device Type: ebs
Public DNS Name:
ec2-54-219-201-136.us-west-
1.compute.amazonaws.com
Public IP Address: 54.219.201.136
Private DNS Name:
ip-10-168-93-149.us-west-
1.compute.internal
Private IP Address: 10.168.93.149
```

```
Do you really want to delete this server? (Y/N) y
WARNING: Deleted server i-49117415
WARNING: Corresponding node and client for the
i-
49117415 server were not deleted and remain
registered with the Chef server
```

This command is deliberately verbose. The last thing you want to do is destroy a critical running instance because you made a typo. Make sure to double-check the attributes to ensure that the node being terminated is the one you expect.

Removing the Chef node

At this point, the EC2 instance is being terminated and removed from your account. However, it is not removed from the Chef service, which needs to be done separately with the `node delete` command. In the following code, the Chef node name is specified, not the instance identifier:

```
$ knife node delete i-49117415
Do you really want to delete i-49117415? (Y/N) y
Deleted node[i-49117415]
```

Verify that the node was removed from Chef with `node list` as follows:

```
$ knife node list
00c0ff3300
```

The output should show you that your EC2 instance is no longer registered with Chef.

Interacting with Rackspace Cloud

Rackspace Cloud is another popular cloud computing provider that is well supported by Chef. Similar to EC2, there is a `knife` plugin for Rackspace Cloud as follows:

```
gem install knife-rackspace
```

In the same way that AWS requires a set of credentials to interact with the API for creating and terminating instances, Rackspace Cloud has its own configuration. However, the Rackspace Cloud API is a little simpler; you will need to provide `knife` with your Rackspace Cloud username and API key. For those who do not already know their API key, it can be found in your Rackspace Cloud control panel. The appropriate configuration to add to your `knife.rb` file is as follows:

```
knife[:rackspace_api_username] = "Your Rackspace
API username" knife[:rackspace_api_key] = "Your
Rackspace API Key"
```

This data can be hardcoded into your configuration file; since the `knife` configuration file is just Ruby, the data could be generated by evaluating environment variables or by looking at a local file. This is useful if you are submitting your `knife.rb` file into a source repository so that credentials are not leaked.

Provisioning a Rackspace instance

Rackspace Cloud server provisioning is just as straightforward as it is with EC2. There is some variation in the command-line options passed to `knife` because of the way Rackspace provides images for systems. Instead of using the instance size and an AMI, you specify the flavor of the system to provision (the node's CPU, memory, and disk allocation) and the operating system to image the instance with. In order to determine what flavors are available, the `knife rackspace` plugin provides the `rackspace flavor list` subcommand as follows:

```
$ knife rackspace flavor list
--rackspace-region=IAD
```

Because it is possible that there are different capacities in different regions, it is a good idea to check what is available in the region you want to

provision a node. This will result in a list of flavors and their specifications; as of now, some of the current offerings in IAD are as follows:

```
$ knife rackspace flavor list
--rackspace-
region=IAD
ID  Name                        VCPUs   RAM    Disk
2   512MB Standard Instance     1       512    20 GB
3   1GB Standard Instance       1       1024   40 GB
4   2GB Standard Instance       2       2048   80 GB
5   4GB Standard Instance       2       4096   160 GB
```

In addition to knowing what flavor host to use, you need an image identifier (similar to an AMI identifier) to install onto the new host. Again, this list may vary with the region and change over time, so there is a command to list the various images, `rackspace image list`, which is given as follows:

```
$ knife rackspace image list
--rackspace-
region=IAD | grep -i windows
db7692f7-3cfa-4a9f-a072-
99b21bd126da Windows Server 2008 R2 SP1
a2139adc-df9b-98dc-aa10-
5dcdff8a982b Windows Server 2012 + SQL Server
7e106eaa-c863-49cb-bf5a-
0cb7c7fc9ba1 Windows Server 2012 + SharePoint
```

As you can see, there are a number of Windows distributions available to configure. In order to provision a new host, you will use the `server create` command, similar to the EC2 command. The following `knife` command would provision a 1 GB host running Windows Server 2012 in the IAD datacenter:

```
$ knife rackspace server create \
       -
I db7692f7-3cfa-4a9f-a072-99b21bd126da \
       --
flavor 3 --rackspace-region=IAD \
       --server-create-timeout 1800
```

Injecting configuration into the virtual machine

Similar to EC2, we need to inject some commands when the host is provisioned to open up the firewall and perform a few other tasks. Save the following to a file, `bootstrap.cmd`:

```
net start w32time
```

```
w32tm /config /manualpeerlist:"0.pool.ntp.org
1.pool.ntp.org 2.pool.ntp.org
3.pool.ntp.org" /syncfromflags:manual /reliable:yes /update
w32tm /resync
netsh advfirewall firewall set rule group="remote
administration" new enable=yes
netsh advfirewall firewall add rule name="WinRM
Port" dir=in action=allow protocol=TCP
localport=5985
```

This script will start the time service, sync the clock, and open the firewall on port 5985 to allow WinRM to connect. Once we have created this file, we need to inject it into the Rackspace virtual machine as a special file named `C:\cloud-automation\bootstrap.cmd` when it gets provisioned, and tell `knife` to use WinRM instead of SSH to bootstrap the host:

```
$ knife rackspace server create \
       -
I db7692f7-3cfa-4a9f-a072-99b21bd126da \
       --
flavor 3 --rackspace-region=IAD \
       --server-create-
timeout 1800 \
       --
file
"C:\\cloud-automation\
\bootstrap.cmd=bootstrap.cmd" \
       --bootstrap-
protocol winrm \
       -x Administrator
```

As soon as the API responds to the request to provision a new host, you will see the Rackspace metadata for the host such as the instance ID, name, flavor, and image. After that, you will see a series of dots printed to the terminal while Rackspace Cloud spins up the new instance. It is worth noting that this can take around 20 to 30 minutes depending on the image, instance size, and so on. As a result, it may make sense to alter the value of the `--server-create-timeout` argument. After it has completed, you will see an output similar to the following commands:

```
Instance ID: a1ad2318-28c4-47b7-a353-b6a99a7cc574
Name: rs-5889646228538071
Flavor: 1GB Standard Instance
Image: Windows Server 2012
Metadata: []
RackConnect Wait: no
ServiceLevel Wait: no
............
Public DNS Name: mail.drizzlelabs.com
```

```
Public IP Address: 162.209.101.233
Private IP Address: 10.176.13.30

Waiting for winrm....
Bootstrapping Chef on 162.209.101.233
```

Once the bootstrap step is complete, assuming that there are no errors, you will see that the Chef client has run, similar to previous executions, and you will be provided with the image data all over again. There will be only one slight difference; that is, the administrator password will be provided to you in the output as follows:

```
Public IP Address: 162.209.101.233
Private IP Address: 10.176.13.30
Password: P123atLaMNbd
Environment: _default
```

You can now verify that your newly provisioned host is listed in your Chef service with `knife node list`:

```
$ knife node list
0c0ff3300
rs-5889646228538071
```

The output will contain your newly bootstrapped node ID as specified by you on the command line (via `-N`) or the name generated by Rackspace (in this example, it will be `rs-5889646228538071`). Congratulations, you have provisioned a new Rackspace instance with a single command!

Terminating the instance

Once you are done with testing, you may not want to leave the Rackspace instance running, as it will incur costs while idle. To do this, perform the following steps:

1. List your Rackspace servers.
2. Delete the server from Rackspace.
3. Remove the server from Chef.
4. Verify that the instance no longer exists in Chef or Rackspace.

To list your Rackspace instances, use the `server list` subcommand of the `rackspace` command, which will list all of the Rackspace instances in the specified region. Similar to the output from the EC2 `server list` command, the `rackspace` server list output will look like the following commands:

```
$ knife rackspace server list
--rackspace-
region=IAD
Instance ID   Name                Public IP
Private IP    Flavor
a1ad2318-
28c4 rs-5889646228538071 162.209.10  10.176.13.30
 3
```

As with the other server list output, this is too wide for printing, but you will see the image ID and the state of the servers in the results. Here, our newly created virtual host should be in the *active* state.

Deleting the instance is just a single command—the `rackspace server delete` subcommand is invoked with the Rackspace instance identifier to be terminated. Remember that this is not reversible and so the command will prompt you to ensure whether you really do want to delete the instance:

```
$ knife rackspace server delete
a1ad2318-28c4-47b7-a353-
b6a99a7cc574
Instance ID: a1ad2318-28c4-47b7-a353-b6a99a7cc574
Name: rs-5889646228538071
Flavor: 1GB Standard Instance
Image: Windows Server 2012
Public IP Address: 162.209.101.233
Private IP Address: 10.176.13.30

Do you really want to delete this server? (Y/N) y
WARNING: Deleted server
a1ad2318-28c4-47b7-a353-
b6a99a7cc574
WARNING: Corresponding node and client for the
a1ad2318-28c4-47b7-a353-
b6a99a7cc574 server were not deleted and remain
registered with the Chef server
```

Removing the Chef node

At this point, the Rackspace instance is being terminated and removed from your account. However, it is not removed from the Chef service; that needs to be done separately with the `node delete` command. In the following code, the Chef node name is specified, not the instance identifier:

```
$ knife node delete rs-5889646228538071
```

Verify that the node was removed from Chef with `knife node list` using the following command:

```
$ knife node list
0c0ff3300
```

Summary

As you have seen, Chef has powerful support for managing cloud servers from the command line. You can combine cloud hosts with physical infrastructure to make your systems scalable and elastic with Chef in ways that may not have been possible before. These are just three of the most popular cloud services on the market today; Chef has a growing level of support for a variety of other infrastructure providers as well, and this list will grow as time passes.

Now that you have learned how to write recipes and expand your infrastructure using popular cloud services, let's take a look at some advanced topics including testing recipes and multiplatform support in Chef in the next chapter.

Chapter 6. Going Beyond the Basics

Now that you have seen how you can integrate Chef into your Windows environment, let's take a look at some advanced topics, which are as follows:

- Managing heterogeneous networks
- Handling multiple platforms
- Versioning and source control
- Testing recipes for a variety of platforms using `ChefSpec`

Chef's declarative language

Chef recipes are declarative, which means that it provides a high-level language for describing what to do to accomplish the task at hand without requiring that you provide a specific implementation or procedure. This means that you can focus on building recipes and modeling infrastructure using abstract resources so that it is clear what is happening without having to know how it is happening. Take, as an example, a portion of the recipes we looked at earlier for deploying an IIS application that is responsible for installing some Windows features:

```
features = %w{IIS-ISAPIFilter IIS-ISAPIExtensions
              NetFx3ServerFeatures
NetFx4Extended-ASPNET45
              IIS-NetFxExtensibility45}

features.each do |f|
  windows_feature f do
    action :install
  end
end
```

Because of Chef's declarative language, the preceding section of code reads in a natural way. We have a list of features. For each of those features, which we know to be Windows features, install them.

Because of this high-level abstraction, your recipe can describe what is going on without containing all of the logic necessary to do the actual work. If you were to look into the `windows` cookbook, you would see that there are a number of implementations using DISM, PowerShell, and ServerManagerCmd. Rather than worrying about that in the recipe itself, the logic is deferred to the provider that is selected for the given resource. The feature resource knows that if a host has DISM, it will use the DISM

provider; otherwise, it will look for the existence of `servermanagercmd.exe` and, if it is present, use that as the installation provider. This makes recipes more expressive and much less cluttered.

If Chef did not provide this high-level abstraction, your recipe would look more like the following code snippet:

```
features = %w{IIS-ISAPIFilter IIS-ISAPIExtensions
              NetFx3ServerFeatures
NetFx4Extended-ASPNET45
              IIS-NetFxExtensibility45}

features.each do |f|
  if ::File.exists?(locate_cmd('dism.exe'))
    install_via_dism(f)
  elsif ::File.exists?(locate_
cmd('servermanagercmd.exe'))
    install_via_servermgrcmd(f)
  else
    fail
  end
end

def install_via_dism(feature_name)
     ## some code here to execute DISM
end

def install_via_servermgrcmd(feature_name)
  ## some code here to execute servermgrcmd.exe
end
```

This, while understandable, significantly increases the overall complexity of your recipes and reduces readability. Now, rather than simply focusing on installing the features, the recipe contains a lot of logic about how to perform the installation. Now, imagine writing a recipe that needs to create files and set ownership on those files and be usable across multiple platforms. Without abstractions, the recipe would contain implementation details of how to determine if a platform is Windows or Linux, how to determine user or group IDs from a string representation, what file permissions look like on different platforms, and so on. However, with the level of abstraction that Chef provides, that recipe would look like the following code snippet:

```
file_names = %w{hello.txt goodbye.txt README.md
myfile.txt}

file_names.each do |file_name|
  file file_name
    action :create
    owner "someuser"
    mode 0660
```

```
  end
end
```

Behind the scenes, when the recipe is executed, the underlying providers know how to convert these declarations into system-level commands. Let's take a look at how we could build a single recipe that is capable of installing something on both Windows and Linux.

Handling multiple platforms

One of Chef's strengths is the ability to integrate Windows hosts with non-Windows hosts. It is important to not only develop recipes and cookbooks that are capable of supporting both Linux and Windows hosts, but also to be able to thoroughly test them before rolling them out into your infrastructure. Let's take a look at how you can support multiple platforms in your recipes as well as use a popular testing framework, `ChefSpec`, to write tests to test your recipes and validate platform-specific behaviors.

Declaring support in your cookbook

All Chef cookbooks have a `metadata.rb` file; this file outlines dependencies, ownership, version data, and compatibility. Compatibility in a homogeneous environment is a less important property—all the hosts run the same operating system. When you are modeling a heterogeneous environment, the ability to describe compatibility is more important; without it, you might try to apply a Windows-only recipe to a Linux host or the other way around. In order to indicate the platforms that are supported by your cookbook, you will want to add one or more `supports` stanzas to the `metadata.rb` file. For example, a cookbook that supports Debian and Windows would have two `supports` statements as follows:

```
supports "windows"
supports "debian"
```

However, if you were to support a lot of different platforms, you can always script your configuration. For example, you could use something similar to the following code snippet:

```
%w(windows debian ubuntu redhat fedora).each |os|
    supports os
end
```

Multiplatform recipes

In the following code example, we will look at how we could install Apache, a popular web server, on both a Windows and a Debian system inside of a single recipe:

```
if platform_family? 'debian'
  package 'apache2'
elsif platform_family? 'windows'
  windows_package
node['apache']['windows']['service_name'] do
   source node['apache']['windows']['msi_url']
```

```
  installer_type :msi
   # The last four options keep the service from
failing
   # before the httpd.conf file is created
   options %W[
     /quiet
     INSTALLDIR="#{node['apache']['install_dir']}"
     ALLUSERS=1
     SERVERADMIN=#{node['apache']['serveradmin']}
     SERVERDOMAIN=#{node['fqdn']}
     SERVERNAME=#{node['fqdn']}
   ].join(' ')
  end
end

template node['apache']['config_file'] do
  source "httpd.conf.erb"
  action :create
  notifies :restart, "service[apache2]"
end

# The apache service
service "apache2" do
  if platform_family? 'windows'
    service_name
"Apache#{node['apache']['version']}"
  end
  action [ :enable, :start ]
end
```

In this example, we perform a very basic installation of the Apache 2.x service on the host. There are no modules enabled, no virtual hosts, or anything else. However, it does allow us to define a recipe that will install Apache, generate an `httpd.conf` file, and then enable and start the Apache 2 service. You will notice that there is a little bit of platform-specific configuration going on here, first with how to install the package and second with how to enable the service.

Because the package resource does not support Windows, the installation of the package on Windows will use the `windows_package` resource and the package resource on a Debian host.

To make this work, we will need some configuration data to apply during installation; skimming over the recipe, we find that we would need a configuration hash similar to the following code snippet:

```
'config': {
  'apache': {
    'version': '2.2.48',
    'config_file': '/opt/apache2/conf/httpd.conf',
    'install_dir': '/opt/apache2',
    'serveradmin': 'admin@domain.com',
```

```
    'windows': {
      'service_name': 'Apache 2.x',
      'msi_url': 'http://some.url/apache2.msi'
    }
  }
}
```

This recipe allows our configuration to remain consistent across all nodes; we don't need to override any configuration values to make this work on a Linux or Windows host. You may be saying to yourself "but, `/opt/apache2` won't work on Windows". It will, but it is interpreted as `\opt\apache2` on the same drive as the Chef client's current working directory; thus, if you ran Chef from `C:`, it would become `c:\opt\apache2`. By making our configuration consistent, we do not need to construct any special roles to store our Windows configuration separate from our Linux configuration.

If you do not like installing Apache in the same directory on both Linux and Windows hosts, you could easily modify the recipe to have some conditional logic as follows:

```
apache_config_file = if platform_family? 'windows'
  node['apache']['windows']['config_file']
else
  node['apache']['linux']['config_file']
end


template apache_config_file do
  source "httpd.conf.erb"
  action :create
  notifies :restart, "service[apache2]"
end
```

Here, the recipe is made slightly more complicated in order to accommodate the two platforms, but at the benefit of one consistent cross-platform configuration specification.

Alternatively, you could use the recipe as it is defined and create two roles, one for Windows Apache servers and the other for Linux Apache servers, each with their own configuration. An `apache_windows` role may have the following override configuration:

```
'config': {
  'apache': {
    'config_file':
"C:\\Apps\\Apache2\\Config\\httpd.conf",
    'install_dir': "C:\\Apps\\Apache2"
  }
}
```

In contrast, an `apache_linux` role might have a configuration that looks like the following code snippet:

```
'config': {
  'apache': {
    'config_file':
"/usr/local/apache2/conf/httpd.conf",
    'install_dir': "/usr/local/apache2"
  }
}
```

The impact of this approach is that now you have to maintain separate platform-specific roles. When a host is provisioned or being configured (either through the control panel or via `knife`), you need to remember that it is a Windows host and therefore has a specific Chef role associated with it. This leads to potential mistakes in host configuration as a result of increased complexity.

Reducing complexity

Some of the key goals of using a tool such as Chef are as follows:

- Minimizing complexity of your configuration
- Increasing automation
- Guaranteeing repeatability

To achieve these goals, it is important to make sure that you do not introduce needless complexity. While Chef provides you with the mechanism to create an incredibly complex set of configurations by using its multilevel configuration overlays, that will likely end in trouble. As a popular quote that is attributed to Eric Allman goes:

"Unix gives you just enough rope to hang yourself -- and then a couple more feet, just to be sure."

The same can be said for any complicated system around, be it a software package, a tool, or your infrastructure configuration. It is important to see where you can simplify your components so that they are easily understood not only now, but in the future by other engineers and possibly even yourself at 3 A.M. when you are trying desperately to fix something.

Where possible, it is a good idea to keep your configuration consistent and rely on your recipes to make the distinction in how the application gets configured. Remember that keeping a mental model of disparate pieces of configuration is quite often more complicated than reading code.

Versioning and source control

Chef's job is to store cookbooks and configuration data, and then give the correct combination to hosts that it knows about. Cookbooks maintain their own notion of a version as well as their dependencies' versions. For this reason, it is important to treat them as artifacts of a source-controlled project and version them appropriately the same way you might version any other software library. Maintaining a predictable versioning scheme helps to ensure compatibility with other cookbooks and reduce conflict and errors when adding new features or refactoring code.

As with any other software project, cookbook versions are typically denoted in the form of `MAJOR.MINOR.PATCH`. Major version number changes indicate that the public API is not backwards compatible. For example, deprecation or removal of public functionality or inconsistent behavior of public APIs would be good reasons to make a major version update. In contrast, minor version numbers are incremented when new features are added but existing public APIs are backwards compatible. Implementing a new method or adding support for a different service manager would be good reasons for increasing the minor version. Patch levels that are incremented as bugs are fixed, so these should be very minor changes to public APIs, if any. For some more information on good versioning practices, take a look at http://semver.org.

Additionally, you will likely find that you are downloading cookbooks from public repositories. It would probably make sense to commit these to an internal repository, public or private, so that if you make your own changes or the public repository goes away, you have a version of your cookbook's code available to you.

Some organizations keep all of their cookbooks inside of one repository; some keep multiple repositories, one for each cookbook. Regardless of what version control system and repository strategy you choose, keeping your cookbooks in a source control system is a critical part of ensuring consistent deployment of recipes. It is very easy to make changes locally, issue a `knife cookbook upload` command, and then forget to commit your changes only to have them wiped out by someone else at a later date. Additionally, make sure that you increment version numbers as you modify your cookbooks, even if they are only for internal use, to clearly identify changes in cookbooks.

If you plan to release your recipes to the public, you will likely take a one-cookbook-per-repository approach that makes the cookbook a complete and standalone entity. Also, it is a good idea to make sure that you include some sort of license information (often in the form of a `LICENSE` file) and

some sort of `README` file to go along with it that contains useful information about the cookbook, example usage, how to contribute, and so on.

Testing recipes

There are a number of ways to test your recipes; the simplest of which is to install them onto your Chef server and run them against a test host. This is a viable approach as long as your client runs do not take exceedingly long times to complete. However, there are, perhaps unsurprisingly, better options out there. One of those options is to use a library called `ChefSpec`, which provides functionalities similar to `RSpec` for testing your cookbooks.

RSpec and ChefSpec

For those who have not used `RSpec`, it is a framework for testing Ruby code that allows you to use a domain-specific language to provide tests. This is similar in concept to Chef's domain-specific language for managing infrastructure. `RSpec` and `ChefSpec` provide a number of language primitives for expressing execution of code, expectations, and mocking of external components.

Testing basics

If you are new to testing software, and in particular testing Ruby code, this is a brief introduction to some of the concepts that we will be covering. Testing can happen at many different levels of the software lifecycle as follows:

- Single-module level (called unit tests)
- Multimodule level (known as functional tests)
- System-level testing (also referred to as integration testing)

In the **test-driven-development** (**TDD**) philosophy, tests are written and executed early and often, even before code is written. This guarantees that, assuming your tests are accurate, your code conforms to your expectations from the beginning and does not regress to a previous state of nonconformity. This section will not delve deep into the TDD philosophy and continuous testing, but rather provide you with enough knowledge to begin testing the recipes that you write and feel confident that they will do the correct thing when deployed into your production environment.

RSpec

As mentioned earlier, `RSpec` is designed to provide a more expressive testing language. What this means is that the methods and primitives provided intentionally work together to build a higher-level language that

feels more like a natural language such as English. For example, using RSpec, one could write the following code:

```
factorial(4).should eq 24
```

This, if you read it, feels like "The factorial of four should equal 24". This is the goal of RSpec's DSL, to feel natural as you read your tests. Compare this to a similar JUnit test (for Java) as follows:

```
assertEquals(24, factorial(4));
```

While this is readable by most programmers, it does not feel as natural. In addition to this, RSpec's DSL has contexts and activity blocks that make tests easier to read. For example, we can use RSpec as follows:

```
describe Array do
  it "should be empty when created" do
    Array.new.should == []
  end
end
Again, compared to a similar NUnit (.NET example):
namespace MyTest {
  using System.Collection
  using NUnit.Framework;
  [TestFixture]
  public class ArrayTest {
    [Test]
    public void NewArray() {
      ArrayList list = new ArrayList();
      Assert.AreEqual(0, list.size());
  }
  }
}
```

Clearly, the RSpec test is much more concise and easier to read, which is the goal of RSpec. The ChefSpec test brings the expressiveness of RSpec to Chef cookbooks and recipes by providing Chef-specific primitives and mechanisms on top of RSpec's simple testing language.

ChefSpec

ChefSpec, for those who have not used it, is a fantastic addition to any Chef developer's collection of tools. It is easy to use and provides a convenient way to add tests to your cookbooks and recipes.

Getting started with ChefSpec

In order to get started with ChefSpec, you will need to install a gem that contains the ChefSpec libraries and all the supporting components. Not

surprisingly, that gem is named `chefspec`, and can be installed simply by running the following command:

```
gem install chefspec
```

However, as there are a number of other dependencies being installed, here is a `Gemfile` being used to lock the gems to the versions used when writing these examples:

```
source 'https://rubygems.org'

gem 'chef',         '11.10.0'
gem 'chefspec',     '3.2.0'
gem 'colorize',     '0.6.0'
```

In order to use a `Gemfile`, you will need to have `bundler` installed; if you are using RVM, `bundler` should be installed with every gem set you create. If you are not, you will need to install it on your own by typing the following line of code:

```
gem install bundler
```

Once `bundler` is installed, and a `Gemfile` containing the preceding code lines is placed in the root directory of your cookbook, you can execute the following command from inside your cookbook's directory:

```
bundle install
```

`Bundler` will parse the `Gemfile` in order to download and install `chefspec`, `chef`, and `colorize`, along with any dependencies you do not already have. Once these are installed, you will want to create a `spec` directory inside of your cookbook and create a `default_spec.rb` file. The name of the `spec` file should match the name of the recipe files, so if you have a recipe file named `default.rb` (which most cookbooks will), you would name your `spec` file `default_spec.rb`. Let's take a look at a very simple recipe and matching `ChefSpec` test.

The recipe, as defined here, will simply create a file at `/tmp/myfile.txt` on the end host as follows:

```
file "/tmp/myfile.txt" do
  owner "root"
  group "root"
  mode "0755"
  action :create
end
```

The matching test, shown in the following code snippet, will verify that our cookbook would do what we expected it to:

```
require 'chefspec'

describe 'demo_cookbook::default' do
  let(:chef_run) {

 ChefSpec::Runner.new.converge(described_recipe)
  }

  it 'creates a file' do
    expect(chef_run).to
create_file('/tmp/myfile.txt')
  end
end
```

Executing tests

Now, in order to run it, we use the `rspec` command, which will run the test as a Ruby script using the `RSpec` language. Here, it will also use the `ChefSpec` extensions because in our spec test we have the line require `'chefspec'` at the top to include `ChefSpec`. The following example runs `RSpec` using `bundler` in order to load all the required Ruby gems before execution:

```
bundle exec rspec spec/default_spec.rb
```

This will run `RSpec` using `bundler`, which will make sure that the correct versions of the gems specified in the `Gemfile` are loaded and process the `default_spec.rb` file. Once it runs, you should see the results of your tests, and they should look something like the following output:

```
Finished in 0.17367 seconds
1 example, 0 failures
```

This is `RSpec` telling you that it completed and that you had one test with zero failures. However, the results would be quite different if we had a test that failed; `RSpec` would tell us which test failed and why.

Understanding failure

`RSpec` is very good at telling you what went wrong with your tests; it doesn't do you any good to have failing tests if it's impossible to determine what went wrong. When an expectation in your test is not met, `RSpec` will tell you which expectation was unmet and what the unexpected value was. For

example, let's take our demonstration recipe file resource from the following code:

```
file "/tmp/myfile.txt" do
```

Let's replace it with a different filename, such as `myfile2.txt` instead of `myfile.txt`, as shown in the following code:

```
file "/tmp/myfile2.txt" do
```

Now, if we re-run our spec tests, we would see that our test is now failing because Chef did something that was not expected by our spec test, as shown in the following example:

```
[user@host]$ bundle exec rspec
spec/default_spec.rb
F

Failures:

  1) demo_cookbook::default creates a file
     Failure/Error: expect(chef_run).to
create_file('/tmp/myfile.txt')
       expected "file[/tmp/myfile.txt]" with
action :create to be in Chef run. Other file
resources:

         file[/tmp/myfile2.txt]

     # ./spec/default_spec.rb:9:in `block (2
levels) in <top (required)>'

Finished in 0.18071 seconds
1 example, 1 failure
```

In the preceding example, you can see that `RSpec` (in conjunction with `ChefSpec`) is telling us that the test `'creates a file'` in the `'demo_cookbook::default'` test suite failed. It also tells us that it failed on line 9 of `default_spec.rb` (as indicated by the line containing `./spec/default_spec.rb:9`) because it expected that the file resource `/tmp/myfile.txt` be interacted with by the `:create` action, but instead the recipe interacted with a file resource, `/tmp/myfile2.txt`.

Expanding your tests

`ChefSpec` provides a comprehensive suite of tools for testing your recipes; you can stub and mock resources (replace their real behavior with some artificial behavior), simulate different platforms, and more. Let's take a look

at some more complex examples to see what other things we can do with `ChefSpec`. Let's look at a hypothetical example for simulating the installation of MySQL on Windows 2012 and some things we might want to validate during such a run. The following code example demonstrates testing our recipe when executed on a Windows 2012 host:

```
context 'when run on Windows 2012' do
  let(:chef_run) do
    # construct a 'runner' (simulate chef-client)
running
    # on a Windows 2012 host
    runner = ChefSpec::ChefRunner.new(
        'platform' => 'windows',
        'version' => '2012'
    )
    # set a configuration variable
    runner.node.set['mysql']['install_path'] =
'C:\\temp'
    runner.node.set['mysql']['service_user'] =
'SysAppUser'
    runner.converge('mysql::server')
  end

  it 'should include the correct Windows server
recipe' do
    chef_run.should include_recipe
'mysql::server_windows'
  end

  it 'should create an INI file in the right
directory' do
    ini_file = "C:\\temp\\mysql\\mysql.ini"
    expect(chef_run).to create_template ini_file
  end
end
```

By constructing the runner with the platform and version options, the test will exercise running the `mysql::server` recipe and pretend as though it were running on a Windows 2012 host. This allows us to set up expectations about the templates that would be created, recipes that are being executed, and more on that particular platform. Presuming that the `mysql::server` recipe was to delegate to the OS-specific recipe on a given platform, we could write another test as follows:

```
context 'when run on Debian' do
  let(:chef_run) do
    runner = ChefSpec::ChefRunner.new(
        'platform' => 'debian'
    )
    runner.node.set['mysql']['install_path'] =
'/usr/local'
```

```
    runner.node.set['mysql']['service_user'] =
'mysql'
    runner.converge('mysql::server')
  end

  it 'should include the correct Linux server
recipe' do
    chef_run.should include_recipe
'mysql::server_linux'
  end

  it 'should create an INI file in the right
directory' do
    ini_file = "/usr/local/mysql/mysql.ini"
    expect(chef_run).to create_template ini_file
  end

  it 'should install the Debian MySQL package' do
    expect(chef_run).to
install_package('mysql-server')
  end
end
```

In this way, we can write our tests to validate the expected behavior on platforms that we may not have direct access to in order to ensure that they will be performing the expected actions for a collection of platforms. The `RSpec` with `ChefSpec` extensions provide us with really powerful tools for testing our cookbooks and recipes, which is a critical step towards automating management.

Summary

By this point, you have seen how to manage Windows hosts, automate deployment and configuration of IIS apps, extend your infrastructure into the cloud, and test your recipes and cookbooks. From this, you can take what you have learned to begin developing advanced Windows-specific recipes and cross-platform ones that support multiple platforms, as well as test them using `RSpec` and `ChefSpec`. With Chef, you can easily manage your heterogeneous Windows and Linux environments, scaling your systems across physical hosts and cloud servers to meet your infrastructure's growth needs.

Index

A

- Amazon EC2
  - custom user scripts, executing in / Executing custom user scripts in EC2
- Amazon EC2 instances
  - managing / Managing Amazon EC2 instances
  - knife plugin, installing / Installing the EC2 knife plugin
  - authentication, setting up / Setting up EC2 authentication
  - provisioning / Provisioning an EC2 instance
  - user script, writing / Writing the user script, Providing a custom user script
  - custom user script, providing / Providing a custom user script
  - terminating / Terminating an EC2 instance
  - Chef node, removing / Removing the Chef node
- attributes
  - about / Getting to know Chef better
- autorun script
  - about / Running scripts at startup
  - example / Running scripts at startup
- Azure virtual machine
  - creating / Creating a new Azure virtual machine

B

- batch scripts
  - executing / Executing batch scripts
  - example / Executing batch scripts
- bootstrap
  - about / Getting to know Chef better
- bootstrap script

- about / Getting to know Chef better

C

- Chef architecture
  - overview / An overview of Chef's architecture
- Chef client
  - installing, on Windows / Installing the Chef client on Windows
  - installing, on Windows with MSI / Installing via MSI
- Chef client installation, Windows
  - hosts, bootstrapping / Preparing to bootstrap Windows hosts, Enabling basic authentication, Bootstrapping a Windows host
  - MSI used / Installing via MSI
- Chef components
  - node / Getting to know Chef better
  - workstation / Getting to know Chef better
  - bootstrap / Getting to know Chef better
  - recipes / Getting to know Chef better
  - cookbook / Getting to know Chef better
  - attributes / Getting to know Chef better
  - role / Getting to know Chef better
  - run list / Getting to know Chef better
  - resources / Getting to know Chef better
  - provider / Getting to know Chef better
  - data bags / Getting to know Chef better
  - environments / Getting to know Chef better
- Chef node
  - removing, from EC2 / Removing the Chef node
  - removing, from Rackspace / Removing the Chef node
- ChefSpec
  - used, for testing / RSpec and ChefSpec, ChefSpec, Getting started with ChefSpec
- cloud providers
  - scaling with / Scaling with cloud providers

- code parameter
    - about / Executing batch scripts
- command parameter
    - about / Executing batch scripts
- cookbook
    - about / Getting to know Chef better
- creates parameter
    - about / Executing batch scripts
- custom user scripts
    - executing, in EC2 / Executing custom user scripts in EC2, Providing a custom user script, Terminating an EC2 instance
- cwd parameter
    - about / Executing batch scripts

D

- data bags
    - about / Getting to know Chef better
- declarative language
    - about / Chef's declarative language
    - benefits / Chef's declarative language
- DISM
    - about / Managing roles and features
- domain-specific language (DSL) / Working with Windows
- DSL
    - about / Working with Windows

E

- EC2 authentication

    - setting up / Setting up EC2 authentication
- EC2 knife plugin
    - installing / Installing the EC2 knife plugin
- end hosts
    - interacting with / Interacting with end hosts
- environments
    - about / Getting to know Chef better

F

- features, Windows
    - managing / Managing roles and features
- feature_name parameter
    - about / Managing roles and features
- flags parameter
    - about / Executing batch scripts

G

- GitHub
    - Umbraco cookbook, downloading from / Fetching from GitHub
- group parameter
    - about / Executing batch scripts

I

- IIS pool
    - generating / Generating an IIS pool and site
- IIS service

    - preparing / Preparing the IIS service
- IIS site
    - generating / Generating an IIS pool and site

K

- knife
    - configuring, for Azure / Configuring knife for Azure

L

- Linux-based systems
    - interacting with / Integrating with Linux-based systems

M

- management certificate, Azure
    - downloading, URL / Downloading the management certificate
- Microsoft Azure
    - working with / Working with Microsoft Azure
    - management certificate, downloading / Downloading the management certificate
    - knife, configuring for / Configuring knife for Azure
    - new virtual machine, creating / Creating a new Azure virtual machine
    - node, bootstrapping / Bootstrapping your Azure node
- Microsoft Azure node
    - bootstrapping / Bootstrapping your Azure node
    - reusable image, creating / Creating a reusable image
- MSI
    - about / Installing via MSI
    - used, for installing Chef client on Windows / Installing via MSI
- multiple platform recipies

    - complexity, reducing / Reducing complexity

- multiple platforms

    - handling / Handling multiple platforms
    - support, declaring in cookbook / Declaring support in your cookbook
    - recipes / Multiplatform recipes

N

- node

    - about / Getting to know Chef better
    - role, applying to / Applying the role to the node

P

- PowerShell

    - scripting with / Scripting with PowerShell

- prerequisites, Umbraco recipe

    - installing / Installing the prerequisites

- printer ports

    - managing / Managing printer ports
    - managing, examples / Managing printer ports

- printers

    - manipulating / Manipulating printers
    - managing / Managing printers
    - actions and parameters / Managing printers
    - managing, examples / Managing printers

- provider

    - about / Getting to know Chef better

R

- Rackspace cloud

    - interacting with / Interacting with Rackspace Cloud
    - instance, provisioning / Provisioning a Rackspace instance
    - configuration, injecting into virtual machine / Injecting configuration into the virtual machine
    - Chef node, removing / Removing the Chef node

- Rackspace cloud instance

    - provisioning / Provisioning a Rackspace instance, Injecting configuration into the virtual machine
    - terminating / Terminating the instance

- recipe, Umbraco

    - prerequisites, installing / Installing the prerequisites
    - IIS service, preparing / Preparing the IIS service
    - application, fetching / Fetching the application
    - application, configuring / Configuring the application
    - IIS pool, generating / Generating an IIS pool and site
    - IIS site, generating / Generating an IIS pool and site

- recipes

    - about / Getting to know Chef better
    - testing, RSpec used / RSpec and ChefSpec
    - testing, ChefSpec used / RSpec and ChefSpec

- registry keys

    - managing, examples / The Chef 10.x resource

- registry values

    - managing, examples / The Chef 0.11.x resource

- registry_key resource

    - about / The Chef 0.11.x resource

- registry_key resource actions

    - create / The Chef 0.11.x resource
    - create_if_missing / The Chef 0.11.x resource
    - delete / The Chef 0.11.x resource
    - delete_key / The Chef 0.11.x resource

- registry_key resource parameters

    - key / The Chef 0.11.x resource
    - architecture / The Chef 0.11.x resource

    - provider / The Chef 0.11.x resource
    - recursive / The Chef 0.11.x resource
    - values / The Chef 0.11.x resource

- remove action
    - about / Managing roles and features

- resources
    - about / Getting to know Chef better

- role
    - about / Getting to know Chef better
    - creating / Creating the role
    - applying, to node / Applying the role to the node

- roles
    - managing / Managing roles and features
    - installing, different mechanisms used / Installing roles using different mechanisms

- RSpec
    - used, for testing / RSpec and ChefSpec, RSpec

- run action
    - about / Executing batch scripts

- run list
    - about / Getting to know Chef better

S

- scripts
    - running, at startup / Running scripts at startup

- software packages
    - installing / Installing software packages
    - installing, examples / Installing software packages

- source control

    - about / Versioning and source control

- system path

    - managing / Managing the system path
    - managing, examples / Managing the system path

T

- tasks

    - scheduling / Scheduling tasks
    - managing, example / Scheduling tasks

- tests

    - executing / Executing tests
    - failure, understanding / Understanding failure
    - expanding / Expanding your tests

U

- Umbraco CMS

    - downloading / Fetching the application
    - configuring / Configuring the application
    - installing / Performing the installation
    - role, creating / Creating the role
    - role, applying, to node / Applying the role to the node

- Umbraco CMS installation

    - host, bootstrapping / Bootstrapping the host

- Umbraco cookbook

    - about / Examining the cookbook
    - URL / Examining the cookbook
    - examining / Examining the cookbook
    - installing / Installing the cookbook
    - downloading, from GitHub / Fetching from GitHub
    - recipe, examining / Examining the recipe

- user parameter

    - about / Executing batch scripts

V

- versioning
    - about / Versioning and source control
    - and source control / Versioning and source control
    - URL / Versioning and source control
    - recipes, testing / Testing recipes
    - basics, testing / Testing basics
    - tests execution / Executing tests
    - test failure / Understanding failure
    - test expansion / Expanding your tests
- virtual machine (VM) / Bootstrapping your Azure node

W

- Windows
    - working with / Working with Windows
    - interacting, with end hosts / Interacting with end hosts
    - supported platforms, for Chef / Supported platforms
    - rebooting / Rebooting Windows
    - rebooting, examples / Rebooting Windows
- Windows-specific resources
    - working with / Working with Windows-specific resources, Working with Windows-specific resources
    - list / Working with Windows-specific resources
    - Chef supported platforms / Platforms supported by Chef
- Windows hosts
    - bootstrapping / Bootstrapping Windows hosts, Bootstrapping a Windows host
    - scaling, with cloud providers / Scaling with cloud providers
    - scripting, with PowerShell / Scripting with PowerShell
- Windows hosts, bootstrapping
    - WinRM, enabling / Enabling Windows Remote Management
    - firewall ports, configuring / Configuring firewall ports
    - basic authentication, enabling / Enabling basic authentication

- Windows pagefiles
    - interacting with / Interacting with Windows pagefiles
    - managing, examples / Interacting with Windows pagefiles
- Windows Registry
    - interacting with / Interacting with the Windows Registry
    - Chef 10.x resource / The Chef 10.x resource
    - Chef 0.11.x resource / The Chef 0.11.x resource
- Windows Server 2012 host
    - bootstrapping / Bootstrapping the host
- windows_auto_run resource actions
    - create / Running scripts at startup
    - remove / Running scripts at startup
- windows_auto_run resource parameters
    - args / Running scripts at startup
    - name / Running scripts at startup
    - program / Running scripts at startup
- windows_batch resource parameters
    - code / Executing batch scripts
    - command / Executing batch scripts
    - creates / Executing batch scripts
    - cwd / Executing batch scripts
    - flags / Executing batch scripts
    - group / Executing batch scripts
    - user / Executing batch scripts
- windows_feature resource actions
    - install / Managing roles and features
    - remove / Managing roles and features
- windows_feature resource parameters
    - feature_name / Managing roles and features
- windows_package resource actions
    - install / Installing software packages
    - remove / Installing software packages
- windows_package resource parameters

    - package_name / Installing software packages
    - installer_type / Installing software packages
    - source / Installing software packages
    - checksum / Installing software packages
    - options / Installing software packages
    - timeout / Installing software packages
    - version / Installing software packages
    - success_codes / Installing software packages

- windows_pagefile resource actions

    - set / Interacting with Windows pagefiles
    - delete / Interacting with Windows pagefiles

- windows_pagefile resource parameters

    - name / Interacting with Windows pagefiles
    - system_managed / Interacting with Windows pagefiles
    - automatic_managed / Interacting with Windows pagefiles
    - initial_size / Interacting with Windows pagefiles
    - maximum_size / Interacting with Windows pagefiles

- windows_path resource actions

    - add / Managing the system path
    - remove / Managing the system path

- windows_path resource parameters

    - path / Managing the system path

- windows_printer resource

    - about / Manipulating printers, Managing printers

- windows_printer resource actions

    - create / Managing printers
    - delete / Managing printers

- windows_printer resource parameters

    - device_id / Managing printers
    - driver_name / Managing printers
    - ipv4_address / Managing printers
    - location / Managing printers
    - shared / Managing printers
    - share_name / Managing printers
    - comment / Managing printers

    - default / Managing printers

- windows_printer_port resource

    - about / Managing printer ports
    - actions / Managing printer ports
    - parameters / Managing printer ports

- windows_printer_port resource actions

    - create / Managing printer ports
    - delete / Managing printer ports

- windows_printer_port resource parameters

    - ipv4_address / Managing printer ports
    - port_name / Managing printer ports
    - port_number / Managing printer ports
    - port_description / Managing printer ports
    - snmp_enabled / Managing printer ports
    - port_protocol / Managing printer ports

- windows_reboot resource actions

    - request / Rebooting Windows
    - cancel / Rebooting Windows

- windows_reboot resource parameters

    - reason / Rebooting Windows
    - timeout / Rebooting Windows

- windows_registry resource

    - about / The Chef 10.x resource

- windows_registry resource actions

    - create / The Chef 10.x resource
    - modify / The Chef 10.x resource
    - force_modify / The Chef 10.x resource
    - remove / The Chef 10.x resource

- windows_registry resource parameters

    - key_name / The Chef 10.x resource
    - values / The Chef 10.x resource
    - type / The Chef 10.x resource

- windows_task resource actions

    - create / Scheduling tasks
    - delete / Scheduling tasks
    - run / Scheduling tasks
    - change / Scheduling tasks

- windows_zipfile resource actions
    - unzip / ZIP files
    - zip / ZIP files

- windows_zipfile resource parameters
    - path / ZIP files
    - source / ZIP files
    - checksum / ZIP files
    - overwrite / ZIP files

- WinRM
    - about / Interacting with end hosts

- workstation
    - about / Getting to know Chef better

Z

- Zip files
    - managing / ZIP files
    - managing, examples / ZIP files