CHEF COOKBOOK DEVELOPMENT

Contents

1. Chef Infrastructure

Introduction

Let's talk about some important terms used in the Chef universe.

A **cookbook** is a collection of all the components needed to change something on a server. Things such as installing MySQL or configuring SSH can be done by cookbooks. The most important parts of cookbooks are recipes, which tell Chef which resources you want to configure on your host.

You need to deploy cookbooks to the nodes that you want to change. Chef offers multiple methods for this task. Most probably, you'll use a central **Chef server**. You can either run your own server or sign up for **hosted Chef**.

The Chef server is the central registry, where each node needs to be registered. The Chef server distributes the cookbooks you uploaded to it, to your nodes.

Knife is Chef's command-line tool to interact with the Chef server. You run it on your local workstation and use it to upload cookbooks and manage other aspects of Chef.

On your nodes, you need to install **Chef Client—**the program that runs on your nodes, retrieving cookbooks from the Chef server and executing them on the node.

Using version control

A **version control system** (**VCS**) helps you stay sane when dealing with important files and collaborating on them.

Using version control is a fundamental part of any infrastructure automation. There are multiple solutions to manage source version control, including Git, SVN, Mercurial, and Perforce. Due to its popularity among the Chef community, we will be using Git. However, you could easily use any other version control system with Chef.

Note

Don't even think about building your infrastructure as code without using a version control system to manage it!

Getting ready

You'll need Git installed on your local workstation. Either use your operating system's package manager (such as Apt on Ubuntu or Homebrew on OS X, or simply download the installer from www.git-scm.org.

Git is a distributed version control system. This means that you don't necessarily need a central host to store your repositories. However, in practice, using GitHub as your central repository has proven to be very helpful. In this book, I'll assume that you're using GitHub. Therefore, you need to go to www.github.com and create an account (which is free) to follow the instructions given in this book. Make sure that you upload your Secure Shell (SSH) key by following the instructions at https://help.github.com/articles/generating-ssh-keys, so that you're able to use the SSH protocol to interact with your GitHub account.

As soon as you have created your GitHub account, you should create your repository by visiting https://github.com and using chef-repo as the repository name.

Walk through the basic steps using GitHub at https://help.github.com/categories/bootcamp

Installing the Chef Development Kit on your workstation

If you want to use Chef, you'll need to install the **Chef Development Kit** (**DK**) on your local workstation first. You'll have to develop your configurations locally and use Chef to distribute them to your Chef server.

Chef provides a fully packaged version, which does not have any external prerequisites. This fully packaged Chef is called the**omnibus installer**. We'll see how to use it in this section.

How to do it…

To install the Chef development kit:

1. Visit this page: https://downloads.chef.io/chefdk/. The Chef development kit supports macOS, Red Hat Enterprise Linux, Ubuntu, and Microsoft Windows.

2. Select a platform, and then a package. (chef-docs uses the macOS setup within the documentation.)
3. Click the download button.
4. Follow the steps in the installer and install the Chef development kit to your machine. The Chef development kit is installed to /opt/chefdk/ on UNIX and Linux systems.
5. When finished, open a command window and enter the following:

```
$ chef verify
```

Using the hosted Chef platform

If you want to get started with Chef right away (without the need to install your own Chef server) or want a third party to give you a **Service Level Agreement** (**SLA**) for your Chef server, you can sign up for hosted Chef by Chef Software, Inc. Chef Software, Inc. operates Chef as a cloud service. It's quick to set up and gives you full control, using users and groups to control access permissions to your Chef setup. We'll configure **knife**, Chef's command-line tool, to interact with hosted Chef, so that you can start managing your nodes.

Getting ready

Before being able to use hosted Chef, you need to sign up for the service. There is a free account for up to five nodes.

Visit http://manage.chef.io/signup and register for a free account.

After registering your account, it is time to prepare your organization to be used with your chef-repo repository.

How to do it…

Download Chef-starter kit from hosted chef or Carry out the following steps to interact with the hosted Chef:

1. Create the configuration directory for your Chef client on your local workstation:

```
mma@laptop:~ $ cd ~/chef-repo
mkdir .chef
```

2. Generate the knife config and put the downloaded knife.rb into the .chef directory inside your chef-repo directory. Make sure you have your user's private key saved as .chef/<YOUR USERNAME>.pem, (in my case it is .chef/webops.pem). If needed, you can reset it at https://id.chef.io/id/profile. Replace webops with the username you chose for hosted Chef, and awo with the short name you chose for your organization in your knife.rb file:

```
current_dir = File.dirname(__FILE__)
log_level:info
log_location            STDOUT
node_name                "webops"
client_key              "#{current_dir}/webops.pem"
chef_server_url          "https://api.chef.io/organizations/awo"
cache_type              'BasicFile'
cache_options( :path => "#{ENV['HOME']}/.chef/checksums" )
cookbook_path            ["#{current_dir}/../cookbooks"]Note
```

Note:

You should add the following code to your .gitingore file inside chef-repo to avoid your credentials ending up in your Git repository:

```
.chef/*.pem
```

3. Use knife to verify that you can connect to your hosted Chef organization. It should not have any clients, so far:

Copy

```
mma@laptop:~/chef-repo $ knife client list
```

How it works…

The following line of code in your knife.rb file tells knife where to find your user's private key. It is used to authenticate you with the Chef server:

```
client_key               "#{current_dir}/webops.pem"
```

Also, the following line of code in your knife.rb file tells knife that you are using hosted Chef. You will find your organization name as the last part of the URL:

```
chef_server_url          "https://api.chef.io/organizations/awo"
```

Using the knife.rb file and your user's key, you can now connect to your organization hosted by Chef Software, Inc.

There's more…

This setup is good for you if you do not want to worry about running, scaling, and updating your own Chef server and if you're happy with saving all your configuration data in the Cloud (under the control of Chef Software, Inc.).

Note

If you need to have all your configuration data within your own network boundaries, you can install Chef server on premises by choosing **ON PREMISES CHEF** at https://www.chef.io/chef/choose-your-version/ or install the Open Source version of Chef server directly from GitHub at https://github.com/chef/chef.

Managing virtual machines with Vagrant

Vagrant is a command-line tool that provides you with a configurable, reproducible, and portable development environment using VMs. It lets you define and use preconfigured disk images to create new VMs from. Also, you can configure Vagrant to use provisioners such as Shell scripts, Puppet, or Chef to bring your VM into the desired state.

Tip

Chef comes with Test Kitchen, which enables you to test your cookbooks on Vagrant without you needing to setup anything manually.

You only need to follow this section, if you want to learn how to use Vagrant and Chef for more advanced cases.

In this section, we will see how to use Vagrant to manage VMs using VirtualBox and Chef client as the provisioner.

Getting ready

1. Download and install VirtualBox at https://www.virtualbox.org/wiki/Downloads.
2. Download and install Vagrant at https://www.vagrantup.com/downloads.html.
3. Install the Omnibus Vagrant plugin to enable Vagrant to install the Chef client on your VM by running the following command:

```
mma@laptop:~/chef-repo $ vagrant plugin install vagrant-omnibus
Installing the 'vagrant-omnibus' plugin. This can take a few minutes...
Installed the plugin 'vagrant-omnibus (1.5.0)'!
```

How to do it…

Let's create and boot a virtual node by using Vagrant:

1. Visit https://github.com/chef/bento and choose a Vagrant box to base your VMs on. We'll use the amd64 image of ubuntu-16.04 in this example.
2. The URL of that box is http://opscode-vm-bento.s3.amazonaws.com/vagrant/virtualbox/opscode_ubuntu-16.04_chef-provisionerless.box.

3. Create a new Vagrantfile. Make sure that you replace <YOUR-ORG> with the name of your organization on the Chef server. Use the name and URL of the box file you noted down in the first step as config.vm.boxand config.vm.box_url:

   Copy

```
mma@laptop:~/chef-repo $ sublVagrantfile
Vagrant.configure("2") do |config|
config.vm.box = "opscode-ubuntu-16.04"
config.vm.box_url = "http://opscode-vm-bento.s3.amazonaws.com/vagrant/virtualbox/opscode_ubuntu-16.04_chef-provisionerless.box"
config.omnibus.chef_version= :latest

config.vm.provision :chef_client do |chef|
chef.provisioning_path = "/etc/chef"
chef.chef_server_url = "https://api.chef.io/organizations/<YOUR_ORG>"
chef.validation_key_path = ".chef/<YOUR_USER>.pem"
chef.validation_client_name = "<YOUR_USER> "
chef.node_name = "server"
end
end
```

4. Create your virtual node using Vagrant:

   Copy

```
mma@laptop:~/chef-repo $ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==>default: Box 'opscode-ubuntu-16.04' could not be found. Attempting to find and install...
...TRUNCATED OUTPUT...
==>default: Importing base box 'opscode-ubuntu-16.04'...
...TRUNCATED OUTPUT...
==>default: Installing Chef latest Omnibus package...
...TRUNCATED OUTPUT...
==>default: Running chef-client...
==>default: Starting Chef Client, version 12.14.89
...TRUNCATED OUTPUT...
```

5. Log in to your virtual node using SSH:

   Copy

```
mma@laptop:~/chef-repo $ vagrant ssh
Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-31-generic x86_64)
...TRUNCATED OUTPUT...
vagrant@server:~$
```

6. Log out of your virtual node:

   Copy

```
vagrant@server:~$ exit
logout
Connection to 127.0.0.1 closed.
mma@laptop:~/chef-repo $
```

7. Validate that the Chef server knows your new virtual machine as a client called **server**:

   Copy

```
mma@laptop:~/chef-repo $ knife client list
awo-validator
server
```

8. Go to https://manage.chef.io/organizations/<YOUR ORGANIZATION>/nodes and validate that your new virtual machine shows up as a registered node:

How it works…

The Vagrantfile is written in a Ruby **Domain Specific Language** (**DSL**) to configure the Vagrant virtual machines. We want to boot a simple Ubuntu VM. Let's go through the Vagrantfile step by step.

First, we create a config object. Vagrant will use this config object to configure the VM:

```
Vagrant.configure("2") do |config|
...
end
```

Inside the config block, we tell Vagrant which VM image to use, in order to boot the node:

```
config.vm.box = "opscode-ubuntu-16.04"
config.vm.box_url = "http://opscode-vm-bento.s3.amazonaws.com/vagrant/virtualbox/opscode_ubuntu-16.04_chef-provisionerless.box"
```

We want to boot our VM using a so-called Bento Box, provided by Chef. We use Ubuntu Version 16.04 here.

As we want our VM to have the Chef client installed, we tell the omnibus vagrant plugin to use the latest version of the Chef client:

```
config.omnibus.chef_version= :latest
```

After selecting the VM image to boot, we configure how to provision the box by using Chef. The Chef configuration happens in a nested Ruby block:

```
config.vm.provision :chef_client do |chef|
  ...
end
```

Inside this chef block, we need to instruct Vagrant on how to hook up our virtual node to the Chef server. First, we need to tell Vagrant where to store all the Chef stuff on your node:

```
chef.provisioning_path = "/etc/chef"
```

Vagrant needs to know the API endpoint of your Chef server. If you use hosted Chef, it is https://api.chef.io/organizations/<YOUR_ORG>. You need to replace <YOUR_ORG> with the name of the organization that you created in your account on hosted Chef. If you are using your own Chef server, change the URL accordingly:

```
chef.chef_server_url = "https://api.chef.io/organizations/<YOUR_ORG>"
```

While creating your user on hosted Chef, you must have downloaded your private key. Tell Vagrant where to find this file:

```
chef.validation_key_path = ".chef/<YOUR_USER>.pem"
```

Also, you need to tell Vagrant which client it should use to validate itself against in the Chef server:

```
chef.validation_client_name = "<YOUR_USER>"
```

Finally, you should tell Vagrant how to name your node:

```
chef.node_name = "server"
```

After configuring your Vagrantfile, all you need to do is run the basic Vagrant commands such as vagrant up, vagrant provision, and vagrant ssh. To stop your VM, just run the vagrant halt command.

There's more…

If you want to start from scratch again, you will have to destroy your VM and delete both the client and the node from your Chef server by running the following command:

```
mma@laptop:~/chef-repo $ vagrant destroy
mma@laptop:~/chef-repo $ knife node delete server -y && knife client delete server -y
```

Creating and using cookbooks

Cookbooks are an essential part of Chef. Basically, you describe the configurations you want to apply to your nodes in cookbooks. You can create them using the Chef executable installed by the Chef DK.

In this section, we'll create and apply a simple cookbook using the chef and knife command-line tools.

Getting ready

Make sure you have Chef DK installed and a node available for testing. Check out the installation instructions at http://learn.chef.io if you need help here.

Edit your knife.rb file (usually found in the hidden .chef directory) and add the following three lines to it, filling in your own values:

```
cookbook_copyright "your company"
cookbook_license "apachev2"
cookbook_email "your email address"
```

Chef will use the preceding values as the defaults whenever you create a new cookbook. We assume that you have a node called server registered with your Chef server, as described in the **Managing virtual machines with Vagrant** section in this chapter.

How to do it…

Carry out the following steps to create and use cookbooks:

1. Create a cookbook named my_cookbook by running the following command:

```
mma@laptop:~/chef-repo $ chef generate cookbook cookbooks/my_cookbook
Generating cookbook my_cookbook
- Ensuring correct cookbook file content
- Ensuring delivery configuration
- Ensuring correct delivery build cookbook content

Your cookbook is ready. Type `cd cookbooks/my_cookbook` to enter it.
...TRUNCATED OUTPUT...
```

2. Upload your new cookbook on the Chef server:

```
mma@laptop:~/chef-repo $ knife cookbook upload my_cookbook
Uploading my_cookbook    [0.1.0]
Uploaded 1 cookbook.
```

3. Add the cookbook to your node's run list. In this example, the name of the node is server:

```
mma@laptop:~/chef-repo $ knife node run_list add server 'recipe[my_cookbook]'
server:
run_list: recipe[my_cookbook]
```

4. Run the Chef client on your node:

```
user@server:~$ sudo chef-client
```

How it works…

The chef executable helps you to manage your local Chef Development environment. We used it here to generate the cookbook.

Knife is the command-line interface for the Chef server. It uses the RESTful API exposed by the Chef server to do its work and helps you to interact with the Chef server.

The knife command supports a host of commands structured as follows:

```
knife<subject><command>
```

The <subject> used in this section is either cookbook or node. The commands we use are upload for the cookbook, and run_list add for the node.

There's more…

Before uploading your cookbook to the Chef server, it's a good idea to run it in Test Kitchen first. Test Kitchen will spin up a virtual machine, execute your cookbook, and destroy the virtual machine again. That way you can evaluate what your cookbook does before you upload it to the Chef server and run it on real nodes.

To run your cookbook with Test Kitchen on an Ubuntu 16.04 virtual machine, execute the following steps:

1. Create a configuration file for Test Kitchen for executing the default recipe of my_cookbook:

```
mma@laptop:~/chef-repo $ subl .kitchen.yml
---
driver:
name: vagrant
```

```
provisioner:
name: chef_zero

platforms:
  - name: ubuntu-16.04

suites:
  - name: default
run_list:
      - recipe[my_cookbook::default]
attributes:
```

2. Run kitchen test to execute the default recipe of my_cookbook:

```
mma@laptop:~/chef-repo $ kitchen test
-----> Starting Kitchen (v1.13.2)
...TRUNCATED OUTPUT...
-----> Kitchen is finished. (0m45.42s)
```

Inspecting files on your Chef server with knife

Sometimes, you may want to peek into the files stored on your Chef server. You might not be sure about an implementation detail of the specific cookbook version currently installed on your Chef server, and need to look it up. Knife can help you out by letting you show various aspects of the files stored on your Chef server.

Getting ready

Install the iptables community cookbook by executing the following command:

```
mma@laptop:~/chef-repo $ knife cookbook site install iptables
Installing iptables to /Users/mma/work/chef-repo/cookbooks
...TRUNCATED OUTPUT...
```

Take a look at the following error:

```
ERROR: IOError: Cannot open or read ../chef-repo/cookbooks/iptables/metadata.rb!
```

If you get the preceding error, your cookbook only has a metadata.json file. Make sure that you delete it and create a valid metadata.rb, file instead.

Upload the iptables cookbook on your Chef server by executing the following command:

```
mma@laptop:~/chef-repo $ knife cookbook upload iptables --include-dependencies
Uploading iptables      [3.0.0]
Uploading compat_resource [12.14.7]
Uploaded 2 cookbooks.
```

How to do it…

Let's find out how knife can help you to look into a cookbook stored in your Chef server:

1. First, you want to find out the current version of the cookbook you're interested in. In our case, we're interested in the iptables cookbook:

```
mma@laptop:~/work/chef_helpster $ knife cookbook show iptables
iptables   3.0.0 0.14.1
```

2. Then, you can look up the definitions of the iptables cookbook, using the version number that you found in the previous step:

```
mma@laptop:~/chef-repo $ knife cookbook show iptables 0.14.1 definitions
checksum:     45c0b77ff10d7177627694827ce47340
name:         iptables_rule.rb
path:         definitions/iptables_rule.rb
specificity:  default
url:          https://s3-external-1.amazonaws.com:443/opscode-platform...
```

3. Now, you can even show the contents of the iptables_rule.rb definition file, as stored on the server:

```
mma@laptop:~/chef-repo $ knife cookbook show iptables 0.14.1 definitions
iptables_rule.rb
#
# Cookbook Name::iptables
# Definition::iptables_rule
#
#
define :iptables_rule, :enable => true, :source => nil, :variables => {}, :cookbook
=> nil do
```

```
...TRUNCATED OUTPUT...
end
```

How it works…

The knife cookbook show subcommand helps you understand what exactly is stored on the Chef server. It lets you drill down into specific sections of your cookbooks and see the exact content of the files stored in your Chef server.

There's more…

You can pass patterns to the knife show command to tell it exactly what you want to see. Showing the attributes defined by the cookbook can be done as follows:

```
mma@laptop:~/work/chef_helpster $ knife show cookbooks/iptables/attributes/*
cookbooks/iptables/attributes/default.rb:
#
# Cookbook Name::iptables
# Attribute:: default
...TRUNCATED OUTPUT...
```

To find some more examples on knife show, visit https://docs.chef.io/knife_show.html

Defining cookbook dependencies

Quite often, you might want to use features of other cookbooks in your own cookbooks. For example, if you want to make sure that all packages required for compiling software written in C are installed, you might want to include the build-essential cookbook, which does just that. The Chef server needs to know about such dependencies in your cookbooks. You declare them in a cookbook's metadata.

Getting ready

Make sure you have a cookbook named my_cookbook, and the run_list of your node includes my_cookbook, as described in the **Creating and using cookbooks** in this chapter.

How to do it…

Edit the metadata of your cookbook in the file cookbooks/my_cookbook/metadata.rb to add a dependency to the build-essential cookbook:

```
mma@laptop:~/chef-repo $ subl cookbooks/my_cookbook/metadata.rb
...
depends 'build-essential', '>= 7.0.3'
```

How it works…

If you want to use a feature of another cookbook inside your cookbook, you will need to include the other cookbook in your recipe using the include_recipe directive:

```
include_recipe 'build-essential'
```

To tell the Chef server that your cookbook requires the build-essential cookbook, you need to declare that dependency in the metadata.rb file. If you uploaded all the dependencies to your Chef server either using knife cookbook upload my_cookbook --include-dependencies or berks install and berks upload, as described in the **Managing cookbook dependencies with Berkshelf** recipe in this chapter, the Chef server will then send all the required cookbooks to the node.

The depends function call tells the Chef server that your cookbook depends on a version greater than or equal to 7.0.3 of the build-essential cookbook.

You may use any of these version constraints with depends calls:

- < (less than)
- <= (less than or equal to)
- = (equal to)
- >= (greater than or equal to)
- ~> (approximately greater than)
- > (greater than)

There's more…

If you include another recipe inside your recipe, without declaring the cookbook dependency in your metadata.rb file, Foodcritic will warn you:

```
mma@laptop:~/chef-repo $ foodcritic cookbooks/my_cookbook
FC007: Ensure recipe dependencies are reflected in cookbook metadata: cookbooks/my_cookbook/recipes/default.rb:9
```

Tip

Foodcritic will just return an empty line, if it doesn't find any issues.

Managing cookbook dependencies with Berkshelf

It's a pain to manually ensure that you have installed all the cookbooks that another cookbook depends on. You must download each and every one of them manually only to find out that, with each downloaded cookbook, you inherit another set of dependent cookbooks.

And even if you use knife cookbook site install, which installs all the dependencies locally for you, your cookbook directory and your repository get cluttered with all those cookbooks. Usually, you don't really care about all those cookbooks and don't want to see or manage them.

This is where Berkshelf comes into play. It works like Bundler for Ruby gems, managing cookbook dependencies for you. Berkshelf downloads all the dependencies you defined recursively and helps you to upload all cookbooks to your Chef server.

Instead of polluting your Chef repository, it stores all the cookbooks in a central location. You just commit your Berkshelf dependency file (called **Berksfile**) to your repository, and every colleague or build server can download and install all those dependent cookbooks based on it.

Let's see how to use Berkshelf to manage the dependencies of your cookbook.

Getting ready

Make sure you have a cookbook named my_cookbook and the run_list of your node includes my_cookbook, as described in the **Creating and using cookbooks**

How to do it…

Berkshelf helps you to keep those utility cookbooks out of your Chef repository. This makes it much easier to maintain the important cookbooks.

Let's see how to write a cookbook by running a bunch of utility recipes and manage the required cookbooks with Berkshelf:

1. Edit your cookbook's metadata:

```
mma@laptop:~/chef-repo $ subl cookbooks/my_cookbook/metadata.rb
...
depends "chef-client"
depends "apt"
depends "ntp"
```

2. Edit your cookbook's default recipe:

```
mma@laptop:~/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb
...
include_recipe "chef-client"
include_recipe "apt"
```

```
include_recipe "ntp"
```

3. Run Berkshelf to install all the required cookbooks:

```
mma@laptop:~/chef-repo $ cd cookbooks/my_cookbook
mma@laptop:~/chef-repo/cookbooks/my_cookbook $ berks install
Resolving cookbook dependencies...
Fetching 'my_cookbook' from source at .
Fetching cookbook index from https://supermarket.chef.io...
Installing apt (4.0.2)
...TRUNCATED OUTPUT...
```

4. Upload all the cookbooks on the Chef server:

```
mma@laptop:~/chef-repo/cookbooks/my_cookbook $ berks upload
Using my_cookbook (0.1.0)
...TRUNCATED OUTPUT...
Uploaded windows (2.0.2) to: 'https://api.opscode.com:443/organizations/awo'
```

How it works...

Berkshelf comes with the Chef DK.

We edit our cookbook and tell it to use a few basic cookbooks.

Instead of making us manually install all the cookbooks using `knife cookbook site install`, Chef generates a Berksfile, besides the `metadata.rb` file.

The Berksfile is simple. It tells Berkshelf to use the Chef supermarket as the default source for all cookbooks:

```
source "https://supermarket.chef.io"
```

And the Berksfile tells Berkshelf to read the `metadata.rb` file to find all the required cookbooks. This is the simplest way when working inside a single cookbook. Please see the following **There's more…** section to find an example of a more advanced usage of the Berksfile.

After telling Berkshelf where to find all the required cookbook names, we use it to install all those cookbooks:

berks install

Berkshelf stores cookbooks in ~/.berkshelf/cookbooks, by default. This keeps your Chef repository clutter-free. Instead of having to manage all the required cookbooks inside your own Chef repository, Berkshelf takes care of them. You simply need to check in Berksfile with your cookbook, and everyone using your cookbook can download all the required cookbooks by using Berkshelf.

To make sure that there's no mix-up with different cookbook versions when sharing your cookbook, Berkshelf creates a file called Berksfile.lock alongside Berksfile.

Note

Don't commit the Berksfile.lock to version control. If you use berks generate it will auto populate the .gitignorefor you. Otherwise, you need to add Berksfile.lock to your .gitignore manually.

Here, you'll find the exact versions of all the cookbooks that Berkshelf installed:

```
DEPENDENCIES
my_cookbook
path: .
metadata: true

GRAPH
apt (4.0.2)
compat_resource (>= 12.10)
chef-client (6.0.0)
cron (>= 1.7.0)
logrotate (>= 1.9.0)
windows (>= 1.42.0)
compat_resource (12.14.7)
cron (2.0.0)
logrotate (2.1.0)
compat_resource (>= 0.0.0)
```

```
my_cookbook (0.1.1)
apt (>= 0.0.0)
chef-client (>= 0.0.0)
ntp (>= 0.0.0)
ntp (3.2.0)
windows (2.0.2)
```

Berkshelf will only use the exact versions specified in the `Berksfile.lock` file, if it finds this file.

Finally, we use Berkshelf to upload all the required cookbooks to the Chef server:

```
berks upload
```

There's more...

Berkshelf integrates tightly with Vagrant via the `vagrant-berkshelf` plugin. You can set up Berkshelf and Vagrant in such a way that Berkshelf installs and uploads all the required cookbooks on your Chef server whenever you execute `vagrant up` or `vagrant provision`. You'll save all the work of running `berks install` and `berks upload` manually before creating your node with Vagrant.

Let's see how you can integrate Berkshelf and Vagrant:

1. First, you need to install the Berkshelf plugin for Vagrant:

```
mma@laptop:~/work/chef-repo $ vagrant plugin install vagrant-berkshelf
Installing the 'vagrant-berkshelf' plugin. This can take a few minutes...
Installed the plugin 'vagrant-berkshelf (5.0.0)'!
```

2. Then, you need to tell Vagrant that you want to use the plugin. You do this by enabling the plugin in `Vagrantfile`:

```
mma@laptop:~/work/chef-repo $ sublVagrantfile
config.berkshelf.enabled = true
```

3. Then, you need a Berksfile in the root directory of your Chef repository to tell Berkshelf which cookbooks to install on each Vagrant run:

```
mma@laptop:~/work/chef-repo $ sublBerksfile
source 'https://supermarket.chef.io'

cookbook 'my_cookbook', path: 'cookbooks/my_cookbook'
```

4. Eventually, you can start your VM using Vagrant. Berkshelf will first download and then install all the required cookbooks in the Berkshelf, and upload them to the Chef server. Only after all the cookbooks are made available on the Chef server by Berkshelf will Vagrant go on:

```
mma@mma-mbp:~/work/chef-repo $ vagrant up
Bringing machine 'server' up with 'virtualbox' provider...

==>default: Updating Vagrant's Berkshelf...
==>default: Resolving cookbook dependencies...
==>default: Fetching 'my_cookbook' from source at cookbooks/my_cookbook
==>default: Fetching cookbook index from https://supermarket.chef.io...
...TRUNCATED OUTPUT...
```

5. This way, using Berkshelf together with Vagrant saves a lot of manual steps and gets faster cycle times for your cookbook development. if you're using your manual Vagrant setup instead of Test Kitchen.

See also

- For the full documentation for Berkshelf, please visit http://berkshelf.com/
- Please find the Berkshelf source code at https://github.com/berkshelf/berkshelf
- Please find the Vagrant Berkshelf plugin source code at https://github.com/berkshelf/vagrant-berkshelf

Using custom knife plugins

Knife comes with a set of commands out-of-the-box. The built-in commands deal with the basic elements of Chef-like cookbooks, roles, data bags, and so on. However, it would be nice to use knife for more than just the basic stuff. Fortunately, knife comes with a plugin API and

there are already a host of useful knife plugins built by the makers of Chef and the Chef community.

Getting ready

Make sure you have an account at **Amazon Web Services** (**AWS**) if you want to follow along and try out the knife-ec2plugin. There are knife plugins available for most Cloud providers. Go through the **There's more…** section of this recipe for a list.

How to do it…

Let's see which knife plugins are available, and try to use one to manage Amazon EC2 instances:

1. List the knife plugins that are shipped as Ruby gems using the chef command-line tool:

```
mma@laptop:~/chef-repo $ chef gem search -r knife-
*** REMOTE GEMS ***
...TRUNCATED OUTPUT...

knife-azure (1.6.0)
...TRUNCATED OUTPUT...
knife-ec2 (0.13.0)
...TRUNCATED OUTPUT...
```

2. Install the EC2 plugin to manage servers in the Amazon AWS Cloud:

```
mma@laptop:~/chef-repo $ chef gem install knife-ec2
Building native extensions. This could take a while...
...TRUNCATED OUTPUT...
Fetching: knife-ec2-0.13.0.gem (100%)
Successfully installed knife-ec2-0.13.0
...TRUNCATED OUTPUT...

6 gems installed
```

3. List all the available instance types in AWS using the knife ec2 plugin. Please use your own AWS credentials instead of XXX and YYYYY:

```
mma@laptop:~/chef-repo $ knife ec2 flavor list --aws-access-key-id XXX --aws-secret-access-key YYYYY
ID          Name                        Arch   RAM     Disk     Cores
c1.medium   High-CPU Medium                    32-bit  1740.8  350 GB   5
…TRUNCATED OUTPUT…
m2.xlarge   High-Memory Extra Large            64-bit  17510.  420 GB   6.5
t1.micro    Micro Instance                     0-bit   613     0 GB     2
```

How it works…

Knife looks for plugins in various places.

First, it looks into the .chef directory, which is located inside your current Chef repository, to find plugins specific to this repository:

```
./.chef/plugins/knife/
```

Then, it looks into the .chef directory, which is located in your home directory, to find plugins that you want to use in all your Chef repositories:

```
~/.chef/plugins/knife/
```

Finally, it looks for installed gems. Knife will load all the code from any chef/knife/ directory found in your installed Ruby gems. This is the most common way of using plugins developed by Chef or the Chef community.

There's more...

There are hundreds of knife plugins, including plugins for most of the major Cloud providers, as well as the major virtualization technologies, such as VMware, vSphere, and OpenStack, among others.

Deleting a node from the Chef server

Every node managed by a Chef server has a corresponding client object on the Chef server. Running the Chef client on your node uses the client object to authenticate itself against the Chef server on each run.

Additionally, to register a client, a node object is created on the Chef server. The node object is the main data structure, which you can use to query node data inside your recipes.

Getting ready

Make sure you have at least one node registered on your Chef server that is safe to remove.

How to do it…

Let's delete the node and client object to completely remove a node from the Chef server.

1. Delete the node object:

```
mma@laptop:~/chef-repo $ knife node delete my_node
Do you really want to delete my_node? (Y/N) y
Deleted node[my_node]
```

2. Delete the client object:

```
mma@laptop:~/chef-repo $ knife client delete my_node
Do you really want to delete my_node? (Y/N) y
Deleted client[my_node]
```

How it works...

To keep your Chef server clean, it's important to not only manage your node objects but to also take care of your client objects, as well.

Knife connects to the Chef server and deletes the node object with a given name, using the Chef server RESTful API.

The same happens while deleting the client object on the Chef server.

After deleting both objects, your node is totally removed from the Chef server. Now you can reuse the same node name with a new box or virtual machine.

There's more…

Having to issue two commands is a bit tedious and error-prone. To simplify things, you can use a knife plugin called playground.

1. Run the chef command-line tool to install the knife plugin:

```
mma@laptop:~/chef-repo $ chef gem install knife-playground
...TRUNCATED OUTPUT...
Installing knife-playground (0.2.2)
```

2. Run the knife pgclientnode delete subcommand:

```
mma@laptop:~/chef-repo $ knife pgclientnode delete my_node
Deleting CLIENT my_node...
Do you really want to delete my_node? (Y/N) y
Deleted client[my_node]
Deleting NODE my_node...
Do you really want to delete my_node? (Y/N) y
Deleted node[my_node]
```

Developing recipes with local mode

If running your own Chef server seems like overkill and you're not comfortable with using the hosted Chef, you can use local mode to execute cookbooks.

Getting ready

1. Create a cookbook named my_cookbook by running the following command:

```
mma@laptop:~/chef-repo $ chef generate cookbook cookbooks/my_cookbook
Compiling Cookbooks...
Recipe: code_generator::cookbook
...TRUNCATED OUTPUT...
```

2. Edit the default recipe of my_cookbook so that it creates a temporary file:

```
mma@laptop:~/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb
file "/tmp/local_mode.txt" do
content "created by chef client local mode"
action :create
end
```

How to do it…

Let's run my_cookbook on your local workstation using the Chef client's local mode:

1. Run the Chef client locally with my_cookbook in the run list:

```
mma@laptop:~/chef-repo $ chef-client --local-mode -o my_cookbook
[2016-10-03T20:37:02+02:00] INFO: Started chef-zero at chefzero://localhost:8889
with repository at /Users/matthias.marschall/chef-repo
  One version per cookbook

[2016-10-03T20:37:02+02:00] INFO: Forking chef instance to converge...
Starting Chef Client, version 12.14.89
[2016-10-03T20:37:02+02:00] INFO: *** Chef 12.14.89 ***
[2016-10-03T20:37:02+02:00] INFO: Platform: x86_64-darwin13
...TRUNCATED OUTPUT...
Chef Client finished, 1/1 resources updated in 04 seconds
```

2. Validate that the Chef client run creates the desired temporary file on your local workstation:

```
mma@laptop:~/chef-repo $ cat /tmp/local_mode.txt
created by chef client local mode
```

How it works…

The --local-mode (short form: -z) parameter switches the Chef client into local mode. Local mode uses chef-zero—a simple, in-memory version of the Chef server provided by Chef DK—when converging the local workstation.

By providing the -o parameter, you override the run list of your local node so that the Chef client executes the default recipe from my_cookbook.

There's more…

Chef-zero saves all modifications made by your recipes to the local filesystem. It creates a JSON file containing all node attributes for your local workstation in the nodes directory. This way, the next time you run the Chef client in local mode, it will be aware of any changes your recipes made to the node.

Running knife in local mode

You can use knife in local mode, too. To set the run list of a node named laptop (instead of having to override it with -o), you can run the following command:

```
mma@laptop:~/chef-repo $ knife node run_list add -z laptop 'recipe[my_cookbook]'
```

Moving to hosted Chef or your own Chef server

When you're done editing and testing your cookbooks on your local workstation with chef-zero, you can seamlessly upload them to hosted Chef or your own Chef server:

Note: Make sure you bump the version number of modified cookbooks in their metadata.rb file and commit them to your version control system before uploading to the Chef Server.

```
mma@laptop:~/chef-repo $ berks upload
Uploaded ...
```

Using roles

Roles group nodes with similar configurations. Typical cases are using roles for web servers, database servers, and so on.

You can set custom run lists for all the nodes in your roles and override attribute values from within your roles.

Let's see how to create a simple role.

Getting ready

For the following examples, I assume that you have a node named `server` and that you have at least one cookbook (I'll use the `ntp` cookbook) registered with your Chef server.

How to do it…

Let's create a role and see what we can do with it:

1. Create a role:

```
mma@laptop:~/chef-repo $ subl roles/web_servers.rb
name "web_servers"
description "This role contains nodes, which act as web servers"
run_list "recipe[ntp]"
default_attributes 'ntp' => {
  'ntpdate' => {
    'disable' => true
  }
}
```

2. Upload the role on the Chef server:

```
mma@laptop:~/chef-repo $ knife role from file web_servers.rb
Updated Role web_servers
```

3. Assign the role to a node called `server`:

```
mma@laptop:~/chef-repo $ knife node run_list add server 'role[web_servers]'
server:
run_list: role[web_servers]
```

4. Log in to your node and run the Chef client:

```
user@server:~$ sudo chef-client
...TRUNCATED OUTPUT...
[2016-10-03T18:52:10+00:00] INFO: Run List is [role[web_servers]]
[2016-10-03T18:52:10+00:00] INFO: Run List expands to [ntp]
[2016-10-03T18:52:10+00:00] INFO: Starting Chef Run for server
...TRUNCATED OUTPUT...
```

How it works...

You define a role in a Ruby (or a JSON) file inside the roles folder of your Chef repository. A role consists of a nameattribute and a description attribute. Additionally, a role usually contains a role-specific run list and role-specific attribute settings.

Every node with a role in its run list will have the role's run list expanded into its own. This means that all the recipes (and roles) that are in the role's run list will be executed on your nodes.

You need to upload your role to your Chef server by using the knife role from file command.

Only then should you add the role to your node's run list.

Running the Chef client on a node having your role in its run list will execute all the recipes listed in the role.

The attributes you define in your role will be merged with attributes from environments and cookbooks, according to the precedence rules described at https://docs.chef.io/roles.html#attribute-precedence.

Using environments

Having separate environments for development, testing, and production is a good way to be able to develop and test cookbook updates, and other configuration changes in isolation. Chef enables you to group your nodes into separate environments so as to support an ordered development flow.

Getting ready

For the following examples, let's assume that you have a node named server in the _default environment and that you have at least one cookbook (I'll use the ntp cookbook) registered with your Chef server.

How to do it…

Let's see how to manipulate environments using knife:

Note:

This is only a good idea if you want to play around. For serious work, please create files describing your environments and put them under version control as described in the **There's more…** section of this recipe.

Make sure you've set your EDITOR environment variable to your preferred one.

1. Create your environment on-the-fly using knife. The following command will open your shell's default editor so that you can modify the environment definition:

```
mma@laptop:~/chef-repo $ knife environment create dev
{
  "name": "dev",
  "description": "",
```

```
  "cookbook_versions": {
  },
  "json_class": "Chef::Environment",
  "chef_type": "environment",
  "default_attributes": {
  },
  "override_attributes": {
  }
}
Created dev
```

2. List the available environments:

```
mma@laptop:~/chef-repo $ knife environment list
_default
dev
```

3. List the nodes for all the environments:

```
mma@laptop:~/chef-repo $ knife node list
server
```

4. Verify that the node server is not in the dev environment yet by listing nodes in the devenvironment only:

```
mma@laptop:~/chef-repo $ knife node list -E dev
mma@laptop:~/chef-repo $
```

5. Change the environment of the server to dev using knife:

```
mma@laptop:~/chef-repo $ knife node environment set server book
server:
chef_environment: dev
```

6. List the nodes in the dev environment again:

```
mma@laptop:~/chef-repo $ knife node list -E dev
server
```

7. Use specific cookbook versions and override certain attributes for the environment:

```
mma@laptop:~/chef-repo $ knife environment edit dev
{
  "name": "dev",
  "description": "",
  "cookbook_versions": {
    "ntp": "1.6.8"
  },
  "json_class": "Chef::Environment",
  "chef_type": "environment",
  "default_attributes": {
  },
  "override_attributes": {
    "ntp": {
      "servers": ["0.europe.pool.ntp.org", "1.europe.pool.ntp.org",
"2.europe.pool.ntp.org", "3.europe.pool.ntp.org"]
    }
  }
}
Saved dev
```

How it works…

A common use of environments is to promote cookbook updates from development to staging and then into production. Additionally, they enable you to use different cookbook versions on separate sets of nodes and environment-specific attributes. You might have nodes with less memory in your staging environment as in your production environment. By using environment-specific default attributes, you can, for example, configure your MySQL service to consume less memory on staging than on production.

Note:

The Chef server always has an environment called _default, which cannot be edited or deleted. All the nodes go in there if you don't specify any other environment. Make sure you've set

Be aware that roles are not environment-specific. You may use environment-specific run lists, though.

The node's environment can be queried using the node.chef_environment method inside your cookbooks.

There's more…

If you want your environments to be under version control (and you should), a better way to create a new environment is to create a new Ruby file in the environments directory inside your Chef repository:

```
mma@laptop:~/chef-repo $ cd environments
mma@laptop:~/chef-repo $ subldev.rb
name "dev"
```

You should add, commit, and push your new environment file to GitHub:

```
ma@laptop:~/chef-repo $ git add environments/dev.rb
mma@laptop:~/chef-repo $ git commit -a -m "the dev environment"
mma@laptop:~/chef-repo $ git push
```

Now, you can create the environment on the Chef server from the newly created file using knife:

```
mma@laptop:~/chef-repo $ knife environment from file dev.rb
Created environment dev
```

There is a way to migrate all the nodes from one environment to another by using knife exec:

```
mma@laptop:~/chef-repo $ knife exec -E
'nodes.transform("chef_environment:_default") { |n| n.chef_environment("dev")
```

You can limit your search for nodes in a specific environment:

```
mma@laptop:~/chef-repo $ knife search node "chef_environment:dev"
1 item found
```

Freezing cookbooks

Uploading broken cookbooks that override your working ones is a major pain and can result in widespread outages throughout your infrastructure. If you have a cookbook version, you tested successfully with Test Kitchen, it's a good idea to freeze this version so that no one can overwrite the same version with broken code. When used together with version constraints that are specified in your environment manifests, freezing cookbooks can keep your production servers safe from accidental changes.

Note

Berkshelf takes care of freezing cookbooks automatically.

Getting ready

Make sure you have at least one cookbook (I'll use the ntp cookbook) registered with your Chef server.

How to do it…

Let's see what happens if we freeze a cookbook.

1. Upload a cookbook and freeze it:

```
mma@laptop:~/chef-repo $ knife cookbook upload ntp --freeze
Uploading ntp          [3.2.0]
Uploaded 1 cookbook.
```

2. Try to upload the same cookbook version again:

```
mma@laptop:~/chef-repo $ knife cookbook upload ntp
Uploading ntp          [3.2.0]
ERROR: Version 3.2.0 of cookbook ntp is frozen. Use --force to override.
WARNING: Not updating version constraints for ntp in the environment as the cookbook is frozen.
ERROR: Failed to upload 1 cookbook.
```

3. Change the cookbook version:

```
mma@laptop:~/chef-repo $ subl cookbooks/ntp/metadata.rb
```

```
…
version          "3.2.1"
```

4. Upload the cookbook again:

```
mma@laptop:~/chef-repo $ knife cookbook upload ntp
Uploading ntp          [3.2.1]
Uploaded 1 cookbook.
```

How it works…

By using the --freeze option when uploading a cookbook, you tell the Chef server that it should not accept any changes to the same version of the cookbook anymore. This is important if you're using environments and want to make sure that your production environment cannot be broken by uploading a corrupted cookbook.

By changing the version number of your cookbook, you can upload the new version. Then you can make, for example, your staging environment use that new cookbook version.

There's more…

To support a more elaborate workflow, you can use the knife-spork knife plugin, which comes pre-installed with the Chef DK. It helps multiple developers work on the same Chef server and repository without treading on each other's toes. You can find more information about it at https://docs.chef.io/plugin_knife_spork.html.

Running the Chef client as a daemon

While you can run the Chef client on your nodes manually whenever you change something in your Chef repository, it's sometimes preferable to have the Chef client run automatically every so often. Letting the Chef client run automatically makes sure that no node misses out any updates.

Getting ready

You need to have a node registered with your Chef server. It needs to be able to run chef-client without any errors.

How to do it…

Let's see how to start the Chef client in daemon mode so that it runs automatically:

1. Start the Chef client in daemon mode, running every 30 minutes:

```
user@server:~$ sudo chef-client -i 1800
```

2. Validate that the Chef client runs as a daemon:

```
user@server:~$ psauxw | grep chef-client
```

How it works…

The -i parameter will start the Chef client as a daemon. The given number is the seconds between each Chef client run. In the previous example, we specified 1,800 seconds, which results in the Chef client running every 30 minutes.

You can use the same command in a service startup script.

Tip

You can use the chef-client cookbook to install the Chef client as a service. See: https://supermarket.chef.io/cookbooks/chef-client for details.

There's more…

Instead of running the Chef client as a daemon, you can use a Cronjob to run it every so often:

```
user@server:~$ subl /etc/cron.d/chef_client
PATH=/usr/local/bin:/usr/bin:/bin
```

```
# m h dommondow user command
*/15 * * * * root chef-client -l warn | grep -v 'retrying [1234]/5 in'
```

This cronjob will run the Chef client every 15 minutes and swallow the first four retrying warning messages. This is important to avoid Cron sending out e-mails if the connection to the Chef server is a little slow and the Chef client needs a few retries.

Note

It is possible to initiate a Chef client run at any time by sending the SIGUSR1 signal to the Chef client daemon:

```
user@server:~$ sudokillall -USR1 chef-client
```

2. Evaluating and Troubleshooting Cookbooks and Chef Runs

Introduction

Developing cookbooks and making sure your nodes converge to the desired state is a complex endeavor. You need transparency about what is happening. This chapter will cover a lot of ways to see what's going on and make sure that everything is working as it should. From running basic checks on your cookbooks to a fully test-driven development approach, we'll see what the Chef ecosystem has to offer.

Testing your Chef cookbooks with cookstyle and Rubocop

You know how annoying this is: you tweak a cookbook, run Test Kitchen, and, boom! it fails. What's even more annoying is that it fails only because you missed a mundane comma in the default recipe of the cookbook you just tweaked. Fortunately, there's a very quick and easy way to find such simple glitches before you go all in and try to run your cookbooks on Test Kitchen.

Getting ready

Install the ntp cookbook by running the following command:

```
mma@laptop:~/chef-repo $ knife cookbook site install ntp
Installing ntp to /Users/mma/work/chef-repo/cookbooks
…TRUNCATED OUTPUT…
```

```
Cookbook ntp version 3.2.0 successfully installed
```

How to do it…

Carry out the following steps to test your cookbook; run cookstyle on the ntp cookbook:

```
mma@laptop:~/chef-repo $ cookstyle cookbooks/ntp
Inspecting 5 files
...C.

Offenses:

cookbooks/ntp/recipes/default.rb:25:1: C: Extra blank line detected.

5 files inspected, 1 offense detected
```

How it works…

Cookstyle is a wrapper around Rubocop and executes a Ruby syntax check on all Ruby files within the cookbook. Rubocop is a linting and style-checking tool built for Ruby. cookstyle defines some sane rules for Chef cookbooks.

There's more…

There exists a whole ecosystem of additional tools such as ChefSpec (behavior-driven testing for Chef), and Test Kitchen (an integration testing tool to run cookbooks on virtual servers), and then some.

See also

- Read more about Rubocop at https://docs.chef.io/rubocop.html
- Find the source code of Rubocop at GitHub: https://github.com/bbatsov/rubocop

- Read more about Cookstyle at: https://github.com/chef/cookstyle

Flagging problems in your Chef cookbooks with Foodcritic

You might wonder what the proven ways to write cookbooks are. Foodcritic tries to identify possible issues with the logic and style of your cookbooks.

In this section, you'll learn how to use Foodcritic on some existing cookbooks.

Getting ready

Install version 6.0.0 of the `mysql` cookbook by running the following code:

```
mma@laptop:~/chef-repo $ knife cookbook site install mysql 6.0.0
Installing mysql to /Users/mma/work/chef-repo/cookbooks
…TRUNCATED OUTPUT…
Cookbook mysql version 6.0.0 successfully installed
```

How to do it…

Let's see how Foodcritic reports findings:

1. Run `foodcritic` on your cookbook:

```
mma@laptop:~/chef-repo $ foodcritic ./cookbooks/mysql
...TRUNCATED OUTPUT...
FC001: Use strings in preference to symbols to access node attributes: ./cookbooks/mysql/libraries/helpers.rb:273
FC005: Avoid repetition of resource declarations: ./cookbooks/mysql/libraries/provider_mysql_service.rb:77
...TRUNCATED OUTPUT...
```

2. Get a detailed list of the reported sections inside the `mysql` cookbook by using the `-C` flag:

```
mma@laptop:~/chef-repo $ foodcritic -C ./cookbooks/mysql
...TRUNCATED OUTPUT...
FC001: Use strings in preference to symbols to access node attributes
```

```
273|       @pkginfo.set[:suse]['11.3']['5.5'][:server_package] = 'mysql'
 274|
 275|       @pkginfo
 276|     end
cookbooks/mysql/libraries/provider_mysql_service.rb
FC005: Avoid repetition of resource declarations
  74|       end
  75|
  76|       # Support directories
  77|       directory "#{new_resource.name} :create #{etc_dir}" do
  78|         path etc_dir
  79|         owner new_resource.run_user
  80|         group new_resource.run_group
```

How it works…

Foodcritic defines a set of rules and checks your recipes against each of them. It comes with rules concerning various areas: style, correctness, attributes, strings, portability, search, services, files, metadata, and so on. Running Foodcritic against a cookbook tells you which of its rules matched a certain part of your cookbook. By default, it gives you a short explanation of what you should do along the concerned file and line number.

If you run `foodcritic -C`, it displays excerpts of the places where it found the rules to match.

In the preceding example, it didn't like it that the `mysql` cookbook version 6.0.0 uses symbols to access node attributes instead of strings:

```
@pkginfo.set[:suse]['11.3']['5.5'][:server_package] = 'mysql'
```

This could be rewritten as follows:

```
@pkginfo.set['suse']['11.3']['5.5']['server_package'] = 'mysql'
```

There's more…

Some of the rules, especially those from the styles section, are opinionated. You're able to exclude certain rules or complete sets of rules, such as style, when running Foodcritic:

```
mma@laptop:~/chef-repo $ foodcritic -t '~style' ./cookbooks/mysql
mma@laptop:~/chef-repo $
```

In this case, the tilde negates the tag selection to exclude all rules with the style tag. Running without the tilde would run the style rules exclusively:

```
mma@laptop:~/chef-repo $ foodcritic -t style ./cookbooks/mysql
```

If you want to run foodcritic in a **continuous integration** (**CI**) environment, you can use the -f parameter to indicate which rules should fail the build:

```
mma@laptop:~/chef-repo $ foodcritic -f style ./cookbooks/mysql
…TRUNCATED OUTPUT…
FC001: Use strings in preference to symbols to access node attributes: ./cookbooks/mysql/libraries/helpers.rb:273
FC005: Avoid repetition of resource declarations: ./cookbooks/mysql/libraries/provider_mysql_service.rb:77
mma@laptop:~/chef-repo $ echo $?
```

In this example, we tell Foodcritic to fail if any rule in the style group fails. In our case, it returns a non-zero exit code instead of zero, as it would if either no rule matches or we omit the -f parameter. That non-zero exit code would fail your build on your continuous integration server. You can use –f any if you want Foodcritic to fail on all warnings.

See also

- ✓ Find out more about Foodcritic and its rules at http://www.foodcritic.io
- ✓ Learn how to make sure that your cookbooks compile in the **Testing your Chef cookbooks with cookstyle and Rubocop** recipe in this chapter
- ✓ Check out strainer, a tool to simultaneously test multiple things, such as Foodcritic, knife test, and other stuff, at http://github.com/customink/strainer