Chef Infrastructure Automation Cookbook
Second Edition

Over 80 recipes to automate your cloud and server infrastructure with Chef and its associated toolset

Matthias Marschall
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 5 'Working with Files and Packages'
- A synopsis of the book’s content
- More information on Chef Infrastructure Automation Cookbook Second Edition

About the Author

**Matthias Marschall** is a software engineer "Made in Germany" and the author of the Chef Infrastructure Automation Cookbook by Packt Publishing. His four children make sure that he feels comfortable and stays in control of chaotic situations. A lean and Agile engineering lead, he’s passionate about continuous delivery, infrastructure automation, and all things DevOps.

In recent years, Matthias has helped build several web-based businesses, first with Java and then with Ruby on Rails. He quickly moved into system administration, writing his own configuration management tool before moving his whole infrastructure to Chef in its early days.

In 2008, he started a blog (http://www.agileweboperations.com) with Dan Ackerson. There, they shared their ideas about DevOps since the early days of the continually emerging movement. You can find him on Twitter at @mmarschall.

Matthias is the CTO of www.gutefrage.net GmbH that helps run Germany's biggest Q&A site among other high traffic sites. He holds a master's degree in computer science [Dipl.-Inf. (FH)] and teaches courses on Agile software development at the University of Augsburg.

When not writing or coding, Matthias enjoys drawing cartoons and playing Go. He lives near Munich, Germany.
Irrespective of whether you're a systems administrator or developer, if you're sick and tired of repetitive manual work and don't know whether you may dare to reboot your server, it's time for you to get your infrastructure automated.

This book has all the required recipes to configure, deploy, and scale your servers and applications, irrespective of whether you manage five servers, 5,000 servers, or 500,000 servers.

It is a collection of easy-to-follow, step-by-step recipes showing you how to solve real-world automation challenges. Learn techniques from the pros and make sure you get your infrastructure automation project right the first time.

This book takes you on a journey through the many facets of Chef. It teaches you simple techniques as well as full-fl edged real-world solutions. By looking at easily digestible examples, you'll be able to grasp the main concepts of Chef, which you'll need to automate your own infrastructure. Instead of wasting time trying to get the existing community cookbooks running in your environment, you'll get ready-made code examples to get you started.

After describing how to use the basic Chef tools, the book shows you how to troubleshoot your work and explains the Chef language. Then, it shows you how to manage users, applications, and your whole Cloud infrastructure. The book concludes by providing you with additional, indispensable tools, and giving you an in-depth look into the Chef ecosystem.

Learn the techniques of the pros by walking through a host of step-by-step guides to solve your real-world infrastructure automation challenges.

What This Book Covers

Chapter 1, Chef Infrastructure, helps you to get started with Chef. It explains some key concepts, such as cookbooks, roles, and environments, and shows you how to use some basic tools like the Chef development kit (ChefDK), such as Git, knife, chef shell, Vagrant, and Berkshelf.

Chapter 2, Evaluating and Troubleshooting Cookbooks and Chef Runs, is all about getting your cookbooks right. It covers logging and debugging as well as the why run mode, and shows you how to develop your cookbooks totally test driven.
Chapter 3, Chef Language and Style, covers additional Chef concepts, such as attributes, templates, libraries, and even Light Weight Resource Providers. It shows you how to use plain old Ruby inside your recipes and ends with writing your own Ohai and knife plugins.

Chapter 4, Writing Better Cookbooks, shows you how to make your cookbooks more flexible. It covers ways to override attributes, use data bags and search, and to make your cookbooks idempotent. Writing cross-platform cookbooks is covered as well.

Chapter 5, Working with Files and Packages, covers powerful techniques to manage configuration files, and install and manage software packages. It shows you how to install software from source and how to manage whole directory trees.

Chapter 6, Users and Applications, shows you how to manage user accounts, securing SSH and configuring sudo. Then, it walks you through installing complete applications, such as nginx, MySQL, WordPress, Ruby on Rails, and Varnish. It ends by showing you how to manage your own OS X workstation with Chef.

Chapter 7, Servers and Cloud Infrastructure, deals with networking and applications spanning multiple servers. You'll learn how to create your whole infrastructure using Chef provisioning. Then it shows you how to set up high-availability services and load-balancers, and how to monitor your whole infrastructure with Nagios. Finally, it'll show you how to manage your Amazon EC2 Cloud with Chef.
Working with Files and Packages

“The file is a gzipped tar file. Your browser is playing tricks with you and trying to be smart.”

Rasmus Lerdorf

In this chapter, we will cover the following recipes:

- Creating configuration files using templates
- Using pure Ruby in templates for conditionals and iterations
- Installing packages from a third-party repository
- Installing software from source
- Running a command when a file is updated
- Distributing directory trees
- Cleaning up old files
- Distributing different files based on the target platform

Introduction

Moving files around and installing software are the most common tasks undertaken when setting up your nodes. In this chapter, we'll take a look at the various ways in which Chef supports you in dealing with files and software packages.
Creating configuration files using templates

The term Configuration Management already says it loud and clear: your recipes manage the configuration of your nodes. In most cases, the system configuration is held in local files, on disk. Chef uses templates to dynamically create configuration files from given values. It takes such values from data bags or attributes, or even calculates them on the fly before passing them into a template.

Let’s see how we can create configuration files by using templates.

Getting ready

Make sure that you have a cookbook named my_cookbook and that the run_list of your node includes my_cookbook, as described in the Creating and using cookbooks recipe in Chapter 1, Chef Infrastructure.

How to do it...

Let’s use a template resource to create a configuration file:

1. Edit your cookbook’s default recipe:

   mma@laptop:-/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb

   template "/etc/logrotate.conf" do
     source "logrotate.conf.erb"
     variables(
       how_often: "daily",
       keep: "31"
     )
   end

2. Add an ERB template file to your recipe in its default folder:

   mma@laptop:-/chef-repo $ mkdir -p cookbooks/my_cookbook/templates/default
   mma@laptop:-/chef-repo $ subl cookbooks/my_cookbook/templates/default/logrotate.conf.erb

   <%= @how_often %>
   rotate <%= @keep %>
   create
3. Upload the modified cookbook to the Chef server:

   mma@laptop:-/chef-repo $ knife cookbook upload my_cookbook

   Uploading my_cookbook [0.1.0]

4. Run the Chef client on your node:

   user@server:~$ sudo chef-client

   ...TRUNCATED OUTPUT...

   [2015-01-09T10:33:23+01:00] INFO: template[/etc/logrotate.conf]
   updated file contents /etc/logrotate.conf
   - update content in file /etc/logrotate.conf from b44f70 to
     c5c92d
   --- /etc/logrotate.conf   2015-01-08 22:20:17.000000000
     +0100
     +++ /var/folders/fz/dcb5y3qs4m5glhk8zrxd948m0000g
     n/T/chef-rendered-template20150109-63309-ly6vmk 2015-01-09
     10:33:23.000000000 +0100
     @@ -1,2 +1,4 @@
     -dailyrotate 31create
     +daily
     +rotate 31
     +create
     ...TRUNCATED OUTPUT...

5. Validate the content of the generated file:

   user@server:~$ cat /etc/logrotate.conf

   daily
   rotate 31
   create

**How it works...**

If you want to manage any configuration file by using Chef, you have to follow the given steps:

1. Copy the desired configuration file from your node to your cookbook's default directory under the templates folder.
2. Add the extension .erb to this copy.
3. Replace any configuration value that you want to manage with your cookbook with an ERB statement printing out a variable. Chef will create variables for every parameter that you define in the variables call in your template resource. You can use it in your template, like this:

   `<%= @variable_name %>`
4. Create a template resource in your recipe by using the newly created template as the source, and pass all the variables you introduced in your ERB file to it.

5. Running your recipe on the node will back up the original configuration file to the backup_path that you configured in your client.rb file (default is /var/chef/backup) and replace it with the dynamically generated version.

Whenever possible, try using attributes instead of hardcoding values in your recipes.

There's more...

Be careful when a package update makes changes to the default configuration files. You need to be aware of those changes and merge them manually into your handcrafted configuration file template; otherwise, you'll lose all the configuration settings you changed using Chef.

To avoid accidental changes, it's usually a good idea to add a comment at the top of your configuration file to say that it is managed by Chef.

See also

- Read everything about templates at https://docs.chef.io/templates.html
- Learn more about templates in the Using templates recipe in Chapter 3, Chef Language and Style

Using pure Ruby in templates for conditionals and iterations

Switching options on and off in a configuration file is a pretty common thing. Since Chef uses ERB as its template language, you can use pure Ruby to control the flow in your templates. You can use conditionals or even loops in your templates.

Getting ready

Make sure that you have a cookbook called my_cookbook and that the run_list of your node includes my_cookbook, as described in the Creating and using cookbooks recipe in Chapter 1, Chef Infrastructure.
How to do it...

Let's create a hypothetical configuration file listing the IP addresses of a given set of backend servers. We only want to print that list and set a flag called enabled to true:

1. Edit your cookbook's default recipe:

```
mma@laptop:/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb

template "/tmp/backends.conf" do
  mode "0444"
  owner "root"
  group "root"
  variables({
    :enabled => true,
    :backends => ["10.0.0.10", "10.0.0.11", "10.0.0.12"]
  })
end
```

2. Create your template:

```
mma@laptop:/chef-repo $ subl cookbooks/my_cookbook/templates/default/backends.conf.erb

<%- if @enabled %>
  <%- @backends.each do |backend| %>
    <%= backend %>
  <%- end %>
<%- else %>
  No backends defined!
<%- end %>
```

3. Upload the modified cookbook to the Chef server:

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

4. Run the Chef client on your node:

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

...TRUNCATED OUTPUT...
[2015-01-09T10:37:45+01:00] INFO: template[/tmp/backends.conf] created file /tmp/backends.conf
- create new file /tmp/backends.conf[2015-01-09T10:37:45+01:00] WARN: Could not set gid = 0 on /var/folders/fz/dcb5y3qs4m5g1hk8zrxd948m0000gn/T/chef-rendered-template20150109-63512-1y8uas4, file modes not preserved
```
Working with Files and Packages

[2015-01-09T10:37:45+01:00] INFO: template[/tmp/backends.conf]
updated file contents /tmp/backends.conf
- update content in file /tmp/backends.conf from none to
  68b086
--- /tmp/backends.conf 2015-01-09 10:37:45.000000000 +0100
+++ /var/folders/fz/dcb5y3qs4m5g1hk8zrxd948m0000gn
-/T/chef-rendered-template20150109-63512-1y8uas4 2015-01-09
10:37:45.000000000 +0100
@@ -1 +1,4 @@
+    10.0.0.10
+    10.0.0.11
+    10.0.0.12

...TRUNCATED OUTPUT...

5. Validate the content of the generated file:

    user@server:~$ cat /tmp/backends.conf

    10.0.0.10
    10.0.0.11
    10.0.0.12

How it works...

You can use plain Ruby in your templates. We will mix two concepts in our example. First, we use an if-else block to decide whether we should print a list of IP addresses or just a message. If we are going to print the list of IP addresses, we will use a loop to go through all of them.

Let's have a look at the conditional:

    <% if @enabled %>
    ...
    <% else %>
    No backends defined!
    <% end %>

We either pass true or false as the value of the variable called enabled. You can access the given variables directly in your template. If we pass true, the first block of Ruby code will be executed while rendering the template. If we pass false, Chef will render the string "No backends defined!" as the content of the file.

You can use <%- %> if you want to embed Ruby logic into your template file.
Now, let's see how we loop through the list of IPs:

```erb
<%- @backends.each do |backend| %>
  <%= backend %>
<%- end %>
```

We pass an array of strings as the value of the `backend` variable. In the template, we use the `each` iterator to loop through the array. While looping, Ruby assigns each value to the variable that we define as the looping variable between the `|` characters. Inside the loop, we simply print the value of each array element.

While it is possible to use the full power of Ruby inside your templates, it is a good idea to keep them as simple as possible. It is better to put more involved logic into your recipes and pass pre-calculated values to the template. You should limit yourself to simple conditionals and loops to keep templates simple.

**There's more...**

You can use conditionals to print strings, as shown in the following code:

```erb
<%= "Hello world!" if @enabled %>
```

If you use this in your template, the string `Hello world!` will be printed only if the variable `enabled` is set to `true`.

**See also**

- Read more about templates in the Using templates recipe in Chapter 3, Chef Language and Style
- Find more explanations and examples of templates at https://docs.chef.io/templates.html

**Installing packages from a third-party repository**

Even though the Ubuntu package repository contains many up-to-date packages, you might bump into situations in which either the package you need is missing or outdated. In such cases, you can either use third-party repositories or your own repositories (containing self-made packages). Chef makes it simple to use additional APT repositories with the `apt` cookbook.
Getting ready

Make sure that you have a cookbook called my_cookbook and that the run_list of your node includes my_cookbook, as described in the Creating and using cookbooks recipe in Chapter 1, Chef Infrastructure.

Let's retrieve the required apt cookbook:

1. Add it to Berksfile:
   ```bash
   mma@laptop:/chef-repo $ subl Berksfile
   source 'https://supermarket.getchef.com'
   cookbook 'apt'
   ```

2. Install it to your local workstation:
   ```bash
   mma@laptop:/chef-repo $ berks install
   Resolving cookbook dependencies...
   Fetching cookbook index from https://supermarket.getchef.com...
   Installing apt (2.6.1)
   ```

3. Upload it to your Chef server:
   ```bash
   mma@laptop:/chef-repo $ berks upload
   Uploaded apt (2.6.1) to: 'https://api.opscode.com:443/organizations/awo'
   ```

   Remember that if you're using Vagrant and have installed the Berkshelf plugin, all you need to run is vagrant provision to get the apt cookbook installed on your node.

How to do it...

Let's take a look at how you can install the s3cmd tool from the repository available at www.s3tools.org on a Ubuntu Lucid 12.04 LTS node:

Ubuntu Utopic 14.04 LTS already comes with s3cmd version 1.5.0-rc1-2 installed, and therefore, the following recipe isn't necessary to get the latest version of s3cmd installed.

1. Edit your cookbook's default recipe:
   ```bash
   mma@laptop:/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb
   ```
include_recipe "apt"
apt_repository "s3tools" do
  uri "http://s3tools.org/repo/deb-all"
  components ["stable/"
  key "http://s3tools.org/repo/deb-all/stable/s3tools.key"
  action :add
end
package "s3cmd"

2. Edit your cookbook’s metadata to add a dependency on the apt cookbook:

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

3. Upload the modified my_cookbook to the Chef server:

   mma@laptop:~/chef-repo $ knife cookbook upload my_cookbook

   Uploading my_cookbook  [0.1.0]

4. Validate that the s3cmd package is not yet installed:

   user@server:~$ dpkg -l s3cmd

   No packages found matching s3cmd.

5. Validate that the default repository will install an older version of s3cmd (1.0.0-1):

   user@server:~$ apt-cache showpkg s3cmd

   Package: s3cmd
   Versions:
   1.0.0-1 (/var/lib/apt/lists/us.archive.ubuntu.com_ubuntu_dists_
   precise_universe_binary-amd64_Packages)

6. Run the Chef client on your node:

   user@server:~$ sudo chef-client

   ...TRUNCATED OUTPUT...
   [2015-01-12T19:59:04+00:00] INFO: execute[apt-get update] ran successf
Working with Files and Packages

[2015-01-12T19:59:07+00:00] INFO: execute[apt-cache gencaches] ran successfully

- execute apt-cache gencaches

* apt_package[s3cmd] action install
  - install version 1.0.0-4 of package s3cmd
  ...TRUNCATED OUTPUT...

7. Ensure that the s3tools repository will install a newer version (1.0.0-4 instead of 1.0.0-1):

   user@server:~$ apt-cache showpkg s3cmd

   Package: s3cmd
   Versions:
   1.0.0-4 (/var/lib/apt/lists/s3tools.org_repo_deb-all_stable_Packages) (/var/lib/dpkg/status)

8. Ensure that the s3cmd package is installed:

   user@server:~$ dpkg -l
   ...TRUNCATED OUTPUT...
   ii  s3cmd                                   1.0.0-4  The ultimate Amazon S3 and CloudFront command line client

How it works...

The apt cookbook provides an easy way to deal with additional APT repositories.

If you don’t use Berkshelf, as described in the Managing cookbook dependencies with Berkshelf recipe in Chapter 1, Chef Infrastructure, you need to use knife cookbook site install to download the apt cookbook to your workstation and knife cookbook upload apt to install it on your Chef server.

We need to tell Chef that we want to use the apt cookbook by adding the depends call to our cookbook’s metadata.rb file.

The apt cookbook defines the apt_repository resource. To be able to use it, we need to include the apt recipe in our default recipe:

   include_recipe "apt"
As soon as the apt cookbook is available, we can add the third-party repository by using the `apt_repository` resource:

```ruby
apt_repository "s3tools" do
  uri "http://s3tools.org/repo/deb-all"
  components [*"stable/"
  key "http://s3tools.org/repo/deb-all/stable/s3tools.key"
  action :add
end
```

After adding the third-party repository, we can install the desired package from there:

```ruby
package "s3cmd"
```

See also

- Learn more about the s3cmd package at http://s3tools.org/debian-ubuntu-repository-for-s3cmd

**Installing software from source**

If you need to install a piece of software that is not available as a package for your platform, you will need to compile it yourself.

In Chef, you can easily do this by using the `script` resource. What is more challenging is to make such a `script` resource idempotent.

In the following recipe, we will see how to do both.

**Getting ready**

Make sure that you have a cookbook called `my_cookbook` and that the `run_list` of your node includes `my_cookbook`, as described in the `Creating and using cookbooks` recipe in Chapter 1, Chef Infrastructure.

Retrieve the required cookbooks:

1. Add them to your Berksfile:
   ```bash
   mma@laptop:~/chef-repo $ subl Berksfile
   source 'https://supermarket.getchef.com'
cookbook 'apt'
cookbook 'build-essential'
   ```
Working with Files and Packages

2. Install it on your local workstation:

```bash
mma@laptop:$ berks install
```

Resolving cookbook dependencies...
Fetching cookbook index from https://supermarket.getchef.com...
Installing build-essential (2.1.3)
Installing apt (2.6.1)

3. Upload it to your Chef server:

```bash
mma@laptop:$ berks upload
```

Uploaded apt (2.6.1) to: 'https://api.opscode.com:443/organizations/awo'
Uploaded build-essential (2.1.3) to: 'https://api.opscode.com:443/organizations/awo'

Remember that if you’re using Vagrant and have installed the Berkshelf plugin, all you need to run is the Vagrant provision to get the required cookbooks installed on your node.

How to do it...

Let’s take nginx as a well-known example for installing it from source:

1. Edit your cookbook’s default recipe:

```bash
mma@laptop:$ subl cookbooks/my_cookbook/recipes/default.rb
```

```ruby
include_recipe "apt"
include_recipe "build-essential"
version = "1.7.9"
bash "install_nginx_from_source" do
  cwd Chef::Config['file_cache_path']
  code <<-EOH
    wget http://nginx.org/download/nginx-#{version}.tar.gz
    tar zxf nginx-#{version}.tar.gz &&
    cd nginx-#{version} &&
    ./configure --without-http_rewrite_module && make && make install
  EOH
  not_if "test -f /usr/local/nginx/sbin/nginx"
end
```
2. Edit your cookbook's metadata to add a dependency on the `apt` cookbook:

```bash
mma@laptop:/chef-repo $ subl cookbooks/my_cookbook/metadata.rb
...
depends "apt"
depends "build-essential"
```

3. Upload the modified cookbook to the Chef server:

```bash
mma@laptop:/chef-repo $ knife cookbook upload my_cookbook
Uploading my_cookbook [0.1.0]
```

4. Run the Chef client on your node:

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

...TRUNCATED OUTPUT...

```bash
make[1]: Leaving directory '/var/chef/cache/nginx-1.7.9'
[2015-01-12T20:45:20+00:00] INFO: bash[install_nginx_from_source] ran successfully
  - execute "bash" "/tmp/chef-script20150112-13681-1m0wt4u"
...TRUNCATED OUTPUT...
```

5. Validate that `nginx` is installed:

```bash
user@server:~$ /usr/local/nginx/sbin/nginx -v
nginx version: nginx/1.7.9
```

The `nginx` community cookbook has a recipe to install `nginx` from source. The following example only illustrates how you can install any software from source.

**How it works...**

The `bash` resource executes only if the `nginx` executable is not yet there. Our `not_if` block tests for this.

To be able to compile code on your node, you'll need to have the build essentials installed. That's why you need to include the `build-essential` cookbook before you run your script to make sure you have a compiler installed.

Before Chef runs the script given as `code`, it changes into the working directory that is given as `cwd`. We use Chef's file cache directory instead of `/tmp` because the contents of `/tmp` might get deleted during reboot. In order to avoid downloading the source tarball again, we need to keep it at a permanent location.
Working with Files and Packages

Usually, you would retrieve the value for the version variable from an attribute defined in my_cookbook/attributes/default.rb.

The script itself simply unpacks the tarball, configures, prepares, and installs nginx. We chain the commands using && to avoid running the following commands if an earlier one fails.

```ruby
<<-EOH
...
EOH
The preceding code is a Ruby construct that denotes multiline strings.
```

There's more...

Right now, this recipe will download the source tarball repeatedly, even if it is already there (at least as long as the nginx binary is not found). You can use the remote_file resource instead of calling wget in your bash script. The remote_file resource is idempotent—it will only download the file if it needs to.

Change your default recipe in the following way to use the remote_file resource:

```ruby
include_recipe 'apt'
include_recipe 'build-essential'

version = "1.7.9"

remote_file "fetch_nginx_source" do
  source "http://nginx.org/download/nginx-#{version}.tar.gz"
  path "#{Chef::Config['file_cache_path']}/nginx-#{version}.tar.gz"
end

bash "install_nginx_from_source" do
  cwd Chef::Config['file_cache_path']
  code <<-EOH
    tar zxf nginx-#{version}.tar.gz &&
    cd nginx-#{version} &&
    ./configure --without-http_rewrite_module &&
    make && make install
  EOH
  not_if "test -f /usr/local/nginx/sbin/nginx"
end
```
Running a command when a file is updated

If your node is not under complete Chef control, it might be necessary to trigger commands when Chef changes a file. For example, you might want to restart a service that is not managed by Chef when its configuration file (which is managed by Chef) changes. Let's see how you can achieve this with Chef.

Getting ready

Make sure that you have a cookbook called my_cookbook and that the run_list of your node includes my_cookbook, as described in the Creating and using cookbooks recipe in Chapter 1, Chef Infrastructure.

How to do it...

Let's create an empty file as a trigger and run a bash command, if that file changes:

1. Edit your cookbook's default recipe:

   ```bash
   mma@laptop:~/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb
   template "/tmp/trigger" do
     notifies :run, "bash[run_on_trigger]", :immediately
   end

   bash "run_on_trigger" do
     user "root"
     cwd "/tmp"
     code "echo 'Triggered'"
     action :nothing
   end
   ```

2. Create an empty template:

   ```bash
   mma@laptop:~/chef-repo $ touch cookbooks/my_cookbook/templates/default/trigger.erb
   ```
3. Upload the modified cookbook to the Chef server:

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

4. Run the Chef client on your node:

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


  - update content in file /tmp/trigger from none to e3b0c4 (no diff)

[2015-01-12T20:52:19+00:00] INFO: template[/tmp/trigger] sending run action to bash[run_on_trigger] (immediate)

* bash[run_on_trigger] action runTriggered

[2015-01-12T20:52:19+00:00] INFO: bash[run_on_trigger] ran successfully

- execute "bash" "/tmp/chef-script20150112-16221-1q4r38y"

...
```

5. Run the Chef client again to verify that the run_on_trigger script does not get executed again:

```
user@server:~$ sudo chef-client
...
Recipe: my_cookbook::default

* template[/tmp/trigger] action create (up to date)
```

---

**How it works...**

We define a template resource and tell it to notify our bash resource immediately. Chef will notify the bash resource only if the template resource changes the file. To make sure that the bash script runs only when notified, we define its action as nothing.
We see in the output of the first Chef client run (which created the trigger file) that the bash script was executed:

```
bash[run_on_trigger] ran successfully
```

We see in the output of the second Chef client run that this message is missing. Chef did not execute the script because it did not modify the trigger file.

There's more...

Instead of a template, you can let a file or remote_file resource trigger a bash script. When compiling programs from source, you will download the source tarball using a remote_file resource. This resource will trigger a bash resource to extract and compile the program.

See also

- The *Installing software from source* recipe in this chapter

Distributing directory trees

You need to seed a directory tree on your nodes. It might be a static website or some backup data, which is needed on your nodes. You want Chef to make sure that all the files and directories are there on your nodes. Chef offers the remote_directory resource to handle this case. Let's see how you can use it.

Getting ready

Make sure you have a cookbook called `my_cookbook`, and that the run_list of your node includes `my_cookbook`, as described in the *Creating and using cookbooks* recipe in Chapter 1, *Chef Infrastructure*.

How to do it...

Let's upload a directory with some files to our node:

1. Edit your cookbook's default recipe:

```
mma@laptop:/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb
remote_directory "/tmp/chef.github.com" do
  files_backup 10
  files_owner "root"
```
files_group "root"
files_mode 00644
owner "root"
group "root"
mode 00755
end

2. Create a directory structure on your workstation with files that you want to upload to your node. In this example, I am using a plain GitHub pages directory, which contains a static website. To follow along, you can use whatever directory structure you want—just be careful that it doesn't get too big so that it doesn't take hours to upload. Just move the directory to the files/default directory inside your cookbook:

```
mma@laptop:~/chef-repo $ mv chef.github.com cookbooks/my_cookbook/files/default
```

Chef will not upload empty directories.

3. Upload the modified cookbook on the Chef server:

```
mma@laptop:~/chef-repo $ knife cookbook upload my_cookbook
```

Uploading my_cookbook [0.1.0]

4. Run the Chef client on your node:

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

...TRUNCATED OUTPUT...

```

- create new directory /tmp/chef.github.com
Recipe: <Dynamically Defined Resource>
  * directory[/tmp/chef.github.com/images] action create

- create new directory /tmp/chef.github.com/images
```
5. Validate that the directory and its files are there on the node:

```
user@server:~$ ls -l /tmp/chef.github.com
```

```
total 16
4 drwxr-xr-x 2 root root 4096 Mar 22 08:36 images
4 -rw-r--r-- 1 root root 3383 Mar 22 08:36 index.html
4 drwxr-xr-x 2 root root 4096 Mar 22 08:36 javascripts
4 drwxr-xr-x 2 root root 4096 Mar 22 08:36 stylesheets
```

**How it works...**

You need to put the directory that you want to distribute to your nodes into your cookbook under the default folder of files. The `remote_directory` resource picks it up from there and uploads it to your nodes. By default, the name of the resource (in our example, `/tmp/chef.github.com`) will act as the target directory.

Be careful not to put very heavy directory structures into your cookbooks. You will not only need to distribute them to every node but also to your Chef server.

**There's more...**

While you could use the `remote_directory` resource to deploy your applications, there are better ways to do the same. Either you could use any of Chef’s application cookbooks that are available, for example, for Ruby (`application_ruby`) or PHP (`application_php`) applications, or you could use tools such as Capistrano or Mina for deployment.

**See also**

- The Distributing different files based on the target platform recipe in this chapter
- Find out more about GitHub Pages at [http://pages.github.com/](http://pages.github.com/)
- The documentation for the `remote_directory` resource can be found at [https://docs.chef.io/chef/resources.html#remote-directory](https://docs.chef.io/chef/resources.html#remote-directory)
Working with Files and Packages

- Find the application_ruby cookbook at https://supermarket.chef.io/cookbooks/application_ruby
- Find the application_php cookbook at https://supermarket.chef.io/cookbooks/application_php
- Find more about Capistrano at http://www.capistranorb.com/
- Find more about Mina at http://nadarei.co/mina/

Cleaning up old files

What happens if you want to remove a software package from your node? You should be aware that Chef does not undo its changes. Removing a resource from your cookbook does not mean that Chef will remove the resource from your nodes. You need to do this by yourself.

In today's infrastructure, it's far better to replace a node than try to clean things up with Chef.

Getting ready

Make sure that you have a cookbook called my_cookbook and that the run_list of your node includes my_cookbook, as described in the Creating and using cookbooks recipe in Chapter 1, Chef Infrastructure.

Make sure that you have a remote_directory resource in my_cookbook, as described in the Distributing directory trees recipe.

How to do it...

Let's remove the remote_directory resource from my_cookbook and see what happens:

1. Edit your cookbook's default recipe and remove the remote_directory resource:

   mma@laptop:-/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb
   # there used to be the remote_directory resource

2. Upload the modified cookbook to the Chef server:

   mma@laptop:-/chef-repo $ knife cookbook upload my_cookbook
   Uploading my_cookbook    [0.1.0]
3. Run the Chef client on your node:

```bash
user@server:~$ sudo chef-client
...TRUNCATED OUTPUT...

...TRUNCATED OUTPUT...
```

4. Validate that the directory and its files are still there on the node:

```bash
user@server:~$ ls -l /tmp/chef.github.com
```

```
total 16
4 drwxr-xr-x 2 root root 4096 Mar 22 08:36 images
4 -rw-r--r-- 1 root root 3383 Mar 22 08:36 index.html
4 drwxr-xr-x 2 root root 4096 Mar 22 08:36 javascripts
4 drwxr-xr-x 2 root root 4096 Mar 22 08:36 stylesheets
```

Now, let's explicitly remove the directory structure:

1. Edit your cookbook's default recipe:

   ```bash
   mma@laptop:~/chef-repo $ subl cookbooks/my_cookbook/recipes/default.rb
   directory "/tmp/chef.github.com" do
     action :delete
     recursive true
   end
   ```

2. Upload the modified cookbook to the Chef server:

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

3. Run the Chef client on your node:

   ```bash
   user@server:~$ sudo chef-client
   ...TRUNCATED OUTPUT...
   ...TRUNCATED OUTPUT...
   ```

4. Validate that the directory and its files are deleted from the node:

   ```bash
   user@server:~$ ls -l /tmp/chef.github.com
   ls: cannot access /tmp/chef.github.com: No such file or directory
   ```
How it works...

Removing a resource from your cookbook will lead to Chef not knowing anything about it anymore. Chef does not touch things that are not defined in cookbooks, even if it might have created them once.

To clean up stuff you created using Chef, you need to put the reverse actions into your cookbooks. If you created a directory using Chef, you need to explicitly delete it by using the directory resource with action :delete in your cookbook.

The directory resource is idempotent. Even if the directory is already deleted, it will run fine and simply do nothing.

There’s more...

If you upload a directory structure by using the remote_directory resource, you can use the purge parameter to delete files within that directory structure if they are no longer in your cookbook. In this case, you do not need to delete each file by using a file resource with the delete action:

```ruby
remote_directory "~/tmp/chef.github.com" do
  ...
  purge true
end
```

See also

- The Distributing directory trees recipe in this chapter
- Learn more about the directory resource at https://docs.chef.io/resource_directory.html
- Learn more about the remote_directory resource at https://docs.chef.io/chef/resources.html#remote-directory

Distributing different files based on the target platform

If you have nodes with different operating systems, such as Ubuntu and CentOS, you might want to deliver different files to each of them. There might be differences in the necessary configuration options and the like. Chef offers a way for files and templates to differentiate which version to use, based on a node’s platform.
Getting ready

Make sure that you have a cookbook called my_cookbook and that the run_list of your node includes my_cookbook, as described in the Creating and using cookbooks recipe in Chapter 1, Chef Infrastructure.

How to do it...

Let’s add two templates to our cookbook and see which one gets used:

1. Edit your cookbook’s default recipe:
   ```sh
   mma@laptop:-/chef-repo $ subl cookbooks/my_cookbook/recipes/
   default.rb
   template "/tmp/message" do
     source "message.erb"
   end
   ```

2. Create a template as a default:
   ```sh
   mma@laptop:-/chef-repo $ subl cookbooks/my_cookbook/templates/
   default/message.erb
   ```
   Hello from default template!

3. Create a template only for Ubuntu 14.04 nodes:
   ```sh
   mma@laptop:-/chef-repo $ subl cookbooks/my_cookbook/templates/
   ubuntu-14.04/message.erb
   ```
   Hello from Ubuntu 14.04!

4. Upload the modified cookbook to the Chef server:
   ```sh
   mma@laptop:-/chef-repo $ knife cookbook upload my_cookbook
   ```
   Uploading my_cookbook [0.1.0]

5. Run the Chef client on your node:
   ```sh
   user@server:$ sudo chef-client
   ```
   ...TRUNCATED OUTPUT...
   [2015-01-16T18:19:16+01:00] INFO: template[/tmp/message] created
   file /tmp/message
   - create new file /tmp/message
   [2015-01-16T18:19:16+01:00] WARN: Could not set gid = 0 on /
   var/folders/fz/dcb5y3qs4m5g1hk8zrxd948m0000gn/T/chef-rendered-
   template20150115-74876-coftw0, file modes not preserved
6. Validate that the Ubuntu specific template has been used:

   user@server:~$ sudo cat /tmp/message

   Hello from Ubuntu 14.04!

**How it works...**

Chef tries to use the most specific template for a given platform by looking for templates in the following order, if the given platform is Ubuntu 14.04:

- my_cookbook/templates/my_node.example.com/message.erb
- my_cookbook/templates/ubuntu-14.04/message.erb
- my_cookbook/templates/ubuntu-14/message.erb
- my_cookbook/templates/ubuntu/message.erb
- my_cookbook/templates/default/message.erb

Chef takes the first hit. If there is a file in a directory with the same name as the **fully qualified domain name (FQDN)** of the node, it will take that one.

If not, it will look through the other directories (if they exist), such as ubuntu-14.04 or ubuntu-14, and so on.

The only directory, which is mandatory, is the default directory.

**See also**

- Learn more about this in the *Using templates* recipe in *Chapter 4, Writing Better Cookbooks*
- Find more details about file specificity at [https://docs.chef.io/resource_template.html#file-specificity](https://docs.chef.io/resource_template.html#file-specificity)
Where to buy this book

You can buy Chef Infrastructure Automation Cookbook Second Edition from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.