**Chef server:-**

https://d.adroll.com/cm/n/out

The Chef Server acts as a central repository for cookbooks as well as for information about every node it manages. Chef cookbooks contain code that describes the desired state of your infrastructure.

**Node :-**

A node is any physical or virtual machine in your network that is managed by the Chef server. The Chef client runs on each node and securely communicates with the Chef server to get the latest configuration instructions. The Chef client uses the instructions to bring the node to its desired state.

**Chef Syntax and Examples**

The *Domain Specific Language* (DSL) used by Chef is actually just a subset of Ruby. The full power of the Ruby programming language is accessible in Chef code. This allows developers to conditionally perform actions, perform mathematical operations, and communicate with other services easily from within Chef code. Before diving into the more advanced features of Chef’s DSL, we will explore the basic syntax first.

Here is an example of the Chef DSL in action, demonstrating how a user account can be created with a Chef resource. In Chef, resources are the building blocks used to define specific parts of your infrastructure. For example, the following statement manages a user account named alice with a user identifier (UID) of 503:

user 'alice' do

uid '503'

end

The following code sample demonstrates the more abstract syntax for invoking a DSL method in Chef code, on which the previous example is based:

resource 'NAME' do

parameter1 value1

parameter2 value2

end

The first part is the name of the resource (such as template, package, or service). The next part is the name\_attribute for that resource. The interpretation of this value changes from resource to resource. For example, in the package resource, the name\_attribute is the name of the package you wish to install. In the template resource, the name\_attribute is the path on the target node where the compiled file should reside. Next comes the Ruby keyword do. In Ruby, the do must always be accompanied by a closing end. Everything that resides between the do and end is called the block. Inside the block, resource parameters and their values are declared. This varies from resource to resource. A valid parameter for the package provider is version, whereas a valid parameter for the template provider is source.

It might help to think about the code in a more object-oriented approach as follows. The Chef DSL creates a new resource object for you, sets the correct attributes and parameters, and then executes that resource when Chef evaluates the code:

resource = Resource.new('NAME')

resource.parameter1 = value1

resource.parameter2 = value2

resource.run!

template, package, and service are just three of the many types of resources built into the Chef DSL. The following code demonstrates using the template, package, and service resources in Chef code:

template '/etc/resolv.conf' do

source 'my\_resolv.conf.erb'

owner 'root'

group 'root'

mode '0644'

end

package 'ntp' do

action :upgrade

end

service 'apache2' do

restart\_command '/etc/init.d/apache2 restart'

end

This declares a template resource in Chef’s DSL. The template will be compiled from the local file *my\_resolv.conf.erb*, be owned by root:root, have 0644 permissions, and be placed at */etc/resolv.conf* on the target machine (where Chef evaluates the code).

This declares a package resource in Chef’s DSL. The “ntp” package will be upgraded.

This declares a service resource in Chef’s DSL. The “apache2” service will be accessible and manageable by Chef.

If you specify an invalid parameter (either one that does not exist or is misspelled), Chef will raise an exception:

NoMethodError

-------------

undefined method `not\_a\_real\_parameter' for Chef::Resource

Chef uses a *multiphase* execution model, which lets you include logic switches or loops inside Chef code. For example, if you wanted to execute a resource on an array of objects, you could do so using the following code. The file resource is used to manage a file. content is a Chef DSL expression used to specify a string that is written to the file:

So far, you have seen the file resource, template resource, service resource, and package resource. These are all resources packaged into the core of Chef. You can find a complete listing of all resources on [the resources page](http://docs.getchef.com/chef/resources.html). Here are some of the most commonly used Chef resources, followed by examples of their basic usage:

-----------------------------------------------------------------------------------------------

bash

Execute multi-line scripts written in the Bourne-again shell (bash) scripting language using the bash shell interpreter:

# Output 'hello' to the console

bash 'echo "hello"'

chef\_gem

Install a gem inside of Chef, for use inside Chef; useful when a Chef code requires a gem to perform a function:

*# Install the HTTParty gem to make RESTful requests*

chef\_gem 'httparty'

cron

Create or manage a cron entry that schedules commands to run periodically at specified intervals:

*# Restart the computer every week*

cron 'weekly\_restart' do

weekday '1'

minute '0'

hour '0'

command 'sudo reboot'

end

deploy\_revision

Control and manage a deployment of code from source control (such as a Rails application):

*# Clone and sync an application from revision control*

deploy\_revision '/opt/my\_app' do

repo 'git://github.com/username/app.git'

end

directory

Manage a directory or directory tree, handling permissions and ownership:

*# Recursively ensure a directory exists*

directory '/opt/my/deep/directory' do

owner 'root'

group 'root'

mode '0644'

recursive true

end

execute

Execute an arbitrary one-line command (as if it were entered on the command line):

*# Write contents to a file*

execute 'write status' do

command 'echo "delicious" > /tmp/bacon'

end

file

Manage a file already present (but not already managed by Chef):

*# Delete the /tmp/bacon file*

file '/tmp/bacon' do

action :delete

end

gem\_package

Install a gem for use outside of Chef, such as an application or utility:

*# Install bundler to manage dependencies*

gem\_package 'bundler'

group

Create or manage a local group definition with local user accounts as members:

*# Create the bacon group*

group 'bacon'

link

Create and manage symlinks and hard links:

*# Link /tmp/bacon to /tmp/delicious*

link '/tmp/bacon' do

to '/tmp/delicious'

end

mount

Mount or unmount a file system:

*# Mount /dev/sda8*

mount '/dev/sda8'

package

Install a package using the operating system’s underlying package manager:

*# Install the apache2 package (on Debian-based systems)*

package 'apache2'

remote\_file

Transfer a file from a remote location (such as a website):

*# Download a remote file to /tmp/bacon*

remote\_file '/tmp/bacon' do

source 'http://bacon.org/bits.tar.gz'

end

service

Start, stop, or restart a service:

*# Restart the apache2 service*

service 'apache2' do

action :restart

end

template

Manage plain-text file contents parsed as an Embedded Ruby template:

*# Write the /tmp/bacon template using the bits.erb source*

template '/tmp/bacon' do

source 'bits.erb'

end

user

Create or manage a local user account:

# Create the bacon user

user 'bacon'

These examples illustrate Chef’s DSL, as well as showcase some of the common resources used when working with Chef. Although our list is not comprehensive, it does include some of the most common Chef resources you will encounter. The full list of Chef’s built-in resources can be found in the [online resource documentation](http://docs.opscode.com/resource.html).

------------------------------------------------------------------------------------------------------------------------------------------

**Learning Chef:-**

**Write Your First Chef Recipe**

To show you the basics, let’s write the simplest form of Chef code to make a “Hello World” *recipe*. A *recipe* is a file that contains Chef code.

Using your favorite text editor, create the recipe file *hello.rb* to match [Example 4-1](https://www.safaribooksonline.com/library/view/learning-chef/9781491945087/ch04.html#hello.rb). This file can be anywhere—no specific directory structure is required. By convention, files that contain Chef code have the extension *.rb* to show they are written in Ruby.

The Chef coding language is a Ruby Domain Specific Language (DSL). It contains additional Ruby-like statements specialized for expressing Chef system administration concepts.

*Example 4-1. hello.rb*

file 'hello.txt' do

content 'Welcome to Chef'

end

Note

It’s not necessary to place *hello.rb* or any of the other \**.rb* example files in this chapter in a special directory. To find the *hello.rb* file containing the code from the preceding example, look among the [source code examples for the book](http://learningchef.com/) in the *chap04/* directory. Other examples in this and subsequent chapters can be found in similarly titled chapter directories.

We’ll go over what all the statements in this file mean in more detail in [Examine hello.rb](https://www.safaribooksonline.com/library/view/learning-chef/9781491945087/ch04.html#examining_hello_rb). Enter the code using a text editor, making sure you match the capitalization, spacing, and syntax exactly.

The file statement code you entered in *hello.rb*. is a *resource*. Resources are the building blocks for assembling Chef code. A *resource* is a statement within a recipe that helps define actions for Chef to perform. This particular file resource in *hello.rb* tells Chef to:

* Create the file *hello.txt*.
* Write the content Welcome to Chef to *hello.txt*.

Use the chef-apply command to get Chef to perform the actions indicated in your newly created *hello.rb* file.

**chef-apply**

The chef-apply tool is a wrapper built on top of Chef Solo (which will be discussed in more detail in [Chapter 9](https://www.safaribooksonline.com/library/view/learning-chef/9781491945087/ch09.html)). Chef Solo allows you to run Chef code locally without needing a Chef Server. As you’ll see in [Chapter 9](https://www.safaribooksonline.com/library/view/learning-chef/9781491945087/ch09.html), it is not convenient to execute Chef code in a *.rb* file using Chef Solo. The chef-apply tool was designed to provide an easy-to-use wrapper on top of Chef Solo.

When you run chef-apply hello.rb, the output should resemble, for Linux/Mac OS X:

$ **sudo chef-apply hello.rb**

Recipe: (chef-apply cookbook)::(chef-apply recipe)

\* file[hello.txt] action create

- create new file hello.txt

- update content in file hello.txt from none to 40a30c

--- hello.txt 2014-08-10 22:27:44.000000000 -0700

+++ /tmp/.hello.txt20140810-14225-6e7qc7 2014-08-10 22:27:44.000000000

-0700 @@ -1 +1,2 @@

+Welcome to Chef

To close out this chapter, let’s write a recipe to clean up the *stone.txt* file we just created. Create *cleanup.rb* following [Example 4-3](https://www.safaribooksonline.com/library/view/learning-chef/9781491945087/ch04.html#cleanup.rb).

*Example 4-3. cleanup.rb*

file "#{ENV['HOME']}/stone.txt" do

action :delete

end

Now let’s perform a Chef run using the *cleanup.rb* recipe on Linux/Mac OS X:

$ **sudo chef-apply cleanup.rb**

Recipe: (chef-apply cookbook)::(chef-apply recipe)

\* file[/Users/misheska/stone.txt] action delete

- delete file /Users/misheska/stone.txt

***recipe***

A set of instructions written in a Ruby DSL that indicate the *desired configuration* to Chef.

***resource***

A cross-platform abstraction for something managed by Chef (such as a file). Resources are the building blocks from which you compose Chef code.

***attribute***

Parameters passed to a resource.

You created recipe files with Chef code, and ran chef-apply to perform the actions specified in the recipe. You learned that in Chef code, you need only tell Chef the *desired configuration* using *resources* as building blocks. We showed you how to use the file resource to create a file, and how to use the action :delete attribute to delete a file.

**Chapter 7. Cookbook Authoring and Use**

**Your First Cookbook: Message of the Day (Chef Development Kit)**

You’ll be using a tool called chef to generate an initial directory structure for a message of the day cookbook (motd). chef is a new common utility command that debuted with the Chef Development Kit. On a command line, run the chef generate cookbook motd command to create the cookbook directory scaffolding. chef generate will create a main directory for your cookbook called *motd* as part of the process:

$ **chef generate cookbook motd**

Compiling Cookbooks...

Recipe: code\_generator::cookbook

\* directory[/Users/misheska/learningchef/motd] action create

- create new directory /Users/misheska/learningchef/motd

...

\* template[/Users/misheska/learningchef/motd/recipes/default.rb] action create

- create new file /Users/misheska/learningchef/motd/recipes/default.rb

- update content in file /Users/misheska/learningchef/motd/recipes/default

.rb from none to 9cc885

(diff output suppressed by config)

Make the *motd* directory you just created the current directory:

$ **cd motd**

As demonstrated in Example 7-1, modify the *.kitchen.yml* file to use the CentOS 6 image we’ve tailored for the book.

*Example 7-1. chefdk/motd/.kitchen.yml*

---

driver:

name: vagrant

provisioner:

name: chef\_zero

platforms:

- name: centos65

driver:

box: learningchef/centos65

box\_url: learningchef/centos65

suites:

- name: default

run\_list:

- recipe[motd::default]

attributes:

Check to make sure there are no syntax errors in your *kitchen.yml* file by running kitchen list. If you see a stack trace error instead of the following output, you likely made a typo, inadvertently used tabs instead of spaces, or didn’t line up entries correctly:

$ **kitchen list**

Instance Driver Provisioner Last Action

default-centos65 Vagrant ChefZero <Not Created>

Use the chef generate file motd command to generate the directory structure required for the *motd* file we will be creating on the node. We need only use the name of the file we want to create, not the path:

$ **chef generate file motd**

Compiling Cookbooks...

Recipe: code\_generator::cookbook\_file

\* directory[/Users/misheska/learningchef/motd/files/default] action create

- create new directory /Users/misheska/learningchef/motd/files/default

\* template[/Users/misheska/learningchef/motd/files/default/motd] action create

- create new file /Users/misheska/learningchef/motd/files/default/motd

- update content in file /Users/misheska/learningchef/motd/files/default

/motd from none to e3b0c4

(diff output suppressed by config)

*Example 7-3. chefdk/motd/recipes/default.rb*

*#*

*# Cookbook Name:: motd*

*# Recipe:: default*

*#*

*# Copyright (C) 2014*

*#*

*#*

*#*

cookbook\_file "/etc/motd" do

source "motd"

mode "0644"

end

Here’s an explanation of what each line of code in [Example 7-3](https://www.safaribooksonline.com/library/view/learning-chef/9781491945087/ch07.html#chefdk_motd_default.rb) does:

* cookbook\_file is a Chef *resource*. The cookbook\_file resource is used to transfer files from the *files/* subdirectory in a cookbook to the node.
* do/end clauses note that the Chef resource definition spans multiple lines.
* The "/etc/motd" string passed to cookbook\_file is the name. name that defines the path the file should be copied to on the node.
* source defines the name of the file in the *files/* subdirectory.
* mode defines the octal permissions to set on the file after it is copied. In this case it is octal 644, “world readable.” If you don’t set the file mode appropriately, other users might not be able to read the contents of this file.

**Your First Cookbook: Message of the Day (Chef Client)**

* You’ll be using a tool called *knife* to generate an initial cookbook directory structure for the message of the day cookbook (motd). knife is a basic utility command for working with Chef that you installed with Chef Client. On a command line, run the knife cookbook create subcommand to create the cookbook directory scaffolding. *knife* will create a main directory for your cookbook called *motd* as part of the process:
* $ **knife cookbook create motd --cookbook-path .**
* WARNING: No knife configuration file found
* \*\* Creating cookbook motd
* \*\* Creating README for cookbook: motd
* \*\* Creating CHANGELOG for cookbook: motd
* \*\* Creating metadata for cookbook: motd
* Next, overlay all the files needed for your cookbook to enable Test Kitchen support, just like you did in

$ **cd motd**

$ **kitchen init --create-gemfile**

create .kitchen.yml

create test/integration/default

create Gemfile

append Gemfile

append Gemfile

You must run `bundle install' to fetch any new gems.

Run bundle install to handle the extra Ruby dependencies:

$ **bundle install**

Fetching gem metadata from https://rubygems.org/..........

Resolving dependencies...

Using mixlib-shellout (1.4.0)

Using net-ssh (2.9.1)

Using net-scp (1.2.1)

Using safe\_yaml (1.0.3)

Using thor (0.19.1)

Using test-kitchen (1.2.1)

Installing kitchen-vagrant (0.15.0)

Using bundler (1.5.3)

Your bundle is complete!

Use `bundle show [gemname]` to see where a bundled gem is installed.

Modify the *.kitchen.yml* file to use the CentOS 6 image we’ve tailored for the book as seen in [Example 7-4](https://www.safaribooksonline.com/library/view/learning-chef/9781491945087/ch07.html#knife_motd_kitchen.yml).

*Example 7-4. knife/motd/.kitchen.yml*

---

driver:

name: vagrant

provisioner:

name: chef\_solo

platforms:

- name: centos65

driver:

box: learningchef/centos65

box\_url: learningchef/centos65

suites:

- name: default

run\_list:

- recipe[motd::default]

attributes:

A basic cookbook directory structure contains the following files:

cookbook

├── .kitchen.yml

├── README.md

├── attributes

│ └── default/

├── chefignore

├── files/

│ └── default/

├── metadata.rb

├── recipes/

│ └── default.rb

└── templates/

└── default/

First, you need to create the cookbook skeleton with Test Kitchen support. We’re going to go quickly through the cookbook creation commands this time without showing you the tool output. Refer back to [Your First Cookbook: Message of the Day (Chef Development Kit)](https://www.safaribooksonline.com/library/view/learning-chef/9781491945087/ch07.html#chef_dk_first_cookbook) if you’re using the Chef Development Kit, or [Your First Cookbook: Message of the Day (Chef Client)](https://www.safaribooksonline.com/library/view/learning-chef/9781491945087/ch07.html#chef_client_first_cookbook) if you are using Chef Client for a refresher on the steps involved.

Chef Development Kit:

$ **chef generate cookbook apache**

$ **cd apache**

Chef Client:

$ **knife cookbook create apache --cookbook-path .**

$ **cd apache**

$ **kitchen init --create-gemfile**

$ **bundle install**

*Example 7-7. chefdk/apache/.kitchen.yml*

---

driver:

name: vagrant

provisioner:

name: chef\_zero

platforms:

- name: centos65

driver:

box: learningchef/centos65

box\_url: learningchef/centos65

suites:

- name: default

run\_list:

- recipe[apache::default]

attributes:

*Example 7-13. chefdk/apache/recipes/default.rb*

*#*

*# Cookbook Name:: apache*

*# Recipe:: default*

*#*

*# Copyright (C) 2014*

*#*

*#*

*#*

package "httpd"

service "httpd" do

action [ :enable, :start ]

end

template "/var/www/html/index.html" do

source 'index.html.erb'

mode '0644'

end

Chef Client - Linux/Mac OS X:

$ **touch templates/default/index.html.erb**

**Basic Steps :-**

**1. Create New Chef Cookbook**

To create a cookbook, use “knife cookbook create” command

knife cookbook create stuff

For the above command, knife command creates a separate directory called “stuff” under ~/chef-repo/cookbooks as shown below.

### 2. Create New Cookbook with Custom Options

The metadata.rb file under the cookbook directory will have the following default values.

While creating a cookbook, the best practice is to pass the following options

In the above command:

* -C is the copyright holder name. You can also use –copyright option.
* -m is the email address. You can also use –email option.
* -I is the license type. This will automatically add the appropriate license notice in the files that you create under this cookbook. The following are the valid license types you can use. In this example, we are using gplv3, which will add the GPL v3 license notice in the header. You can also use –license option.
  + apachev2 – This will use Apache v2.0
  + gplv2 – This option is for GPL v2
  + gplv3 – For GPL v3
  + mit – For MIT
* none – This option will add this notice to the files: “Proprietary – All Rights Reserved”
* -r option will define the format for the README files in this cookbook. Use “rdoc” option for Ruby docs, and “md” for markdown. Instead of -r you can also use –readme-format
* -o (or –cookbook-path) – This option is used when you want to specify a different directory path where you want the cookbooks to be created. In the above example, we are not using this option.

### 3. Upload Cookbook to Chef Server

The following “knife cookbook upload” command will upload the cookbook that we created above to the Chef server

# knife cookbook upload stuff

Uploading stuff [0.1.0]

Uploaded 1 cookbook.

### 4. Lock a Cookbook from Future Edits

When you don’t want anybody else to be modifying a particular cookbook version, use the –freeze option

# knife cookbook upload stuff –freeze

# knife cookbook upload stuff –force

### 5. Get a List of ALL Cookbooks

The following “knife cookbook list” command will display all the cookbooks that are available on your Chef server.

# knife cookbook list

dev-cluster 1.3.0

stuff 2.1.0

### 6. Delete a Single Cookbook

Use the following “knife cookbook delete” command, which will delete the cookbook. In this example, we are deleting “stuff” cookbook.

# knife cookbook delete stuff

### 7. Delete One (or All) Versions of a Cookbook

When a cookbook has multiple versions, the delete command will display all the versions and prompt the user to choose either one of the version.

# knife cookbook delete stuff

### 9. Download Cookbook from Chef Server

When you have chef-client installed on multiple machine, and when you want to download a cookbook that someone has modified on your client, then use the “knife cookbook download” command as shown below

# knife cookbook download stuff

### 10. View Cookbook Details

You can view the details of a cookbook using “knife cookbook show” command.

When you don’t specify a version number for a cookbook, the show command will display list of all the version numbers available for the given cookbook.

# knife cookbook show stuff

stuff 2.1.0 2.0.0 1.0.0 0.1.0

**Installation steps:--**

Prerequisite:-

* 2 CentOS Linux release 7 machine
* chef-server-core-12.11.1-1.el7.x86\_64 server setup and chef-12.11.18-1.el7.x86\_64client setup

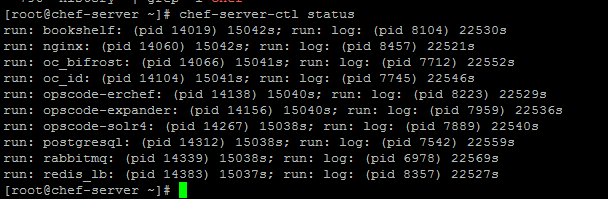
Steps:-

* Install chef server package on the 1 server with the below command.

# chef-server-ctl reconfigure

* Check the installed packages with

#chef-server-ctl status



* Create the user for user commmuncation,

# chef-server-ctl user-create sagar sagar raut xyz@gmail.com 'abc123' --filename /etc/chef-server/sagar.pem

Also create the organization with respective user

#chef-server-ctl org-create digital 'digital river' --association\_user sagar --filename /etc/chef-server/digital-validator.pem

* Once installation completed chef ,chef-server directories are created under the /etc path and its supportive pem files.Like admin.pem , cap-validator.pem, user.pem
* After that install the client package on workstation as well as client node.
* Create the chef directory under ~/.chef on all node servers
* Copy all .pem and validator.pem files under the chef directory of client.
* Configure the knife properties with ,

#knife reconfigure –i

* Fetch the SSL certificate from server to workstation.
* # knife ssl fetch and # knife ssl check
* Once done install same client package on client node also
* Copy the server validator.pem file on client server under the /etc/chef-client directory
* Create the client.rb files under /etc/chef/



* Also run the bootstrap command in the chef-server to check all setting from respective server ends.

# knife bootstrap 192.168.114.130 -x root -N chef-client

* Check the server, client connectivity and execute the program as per the need.