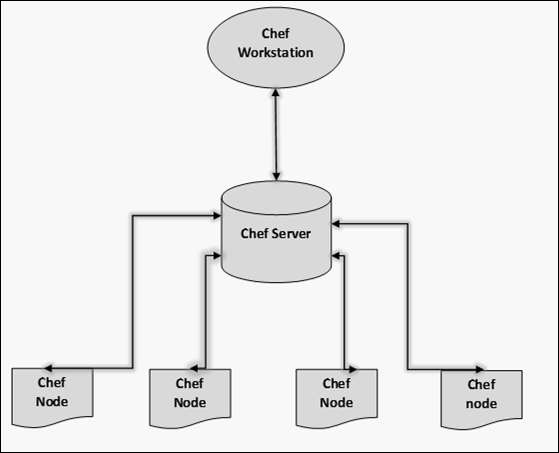
# Chef – Introduction

Key Point : if anything works manually, it will work with Automation. If any error , first check manual way is working or not.

**Chef** is one of the most popular configuration management tools. It uses Ruby and handles configuration by packing details into what it calls recipes.

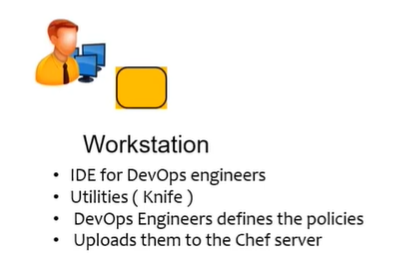
## Architecture



we have the following components.

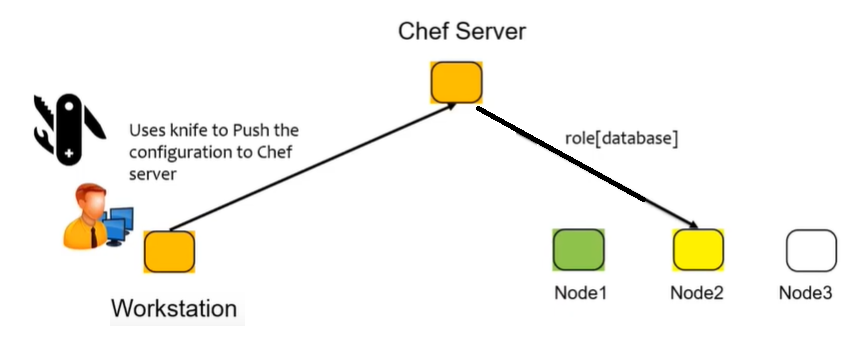
#### Chef Workstation

This is the location where all the configurations are developed. Chef workstation is installed on **the local machine**.



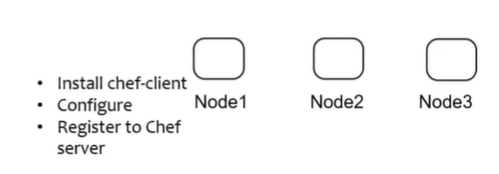
#### Chef Server

This works as a centralized working unit of Chef setup, where all the configuration files are uploaded post development. There are different kinds of Chef server, some are hosted Chef server whereas some are built-in premise.



#### Chef Nodes

They are the actual machines which are going to be managed by the Chef server. Chef client is the key component of all the nodes, which helps in setting up the communication between the Chef server and Chef node.



## Installation

### Install the Chef Development kit(ChefDK) – Workstation

The ChefDK contains everything you need to start working with Chef on a workstation. It provides all the tools that a developer needs to create and modify cookbooks and upload them to a Chef server.

* It includes the **Chef client,**an embedded version of Ruby, RubyGems, OpenSSL, and command-line utilities such as **Knife**, **Ohai**, and Chef Zero.
* It also includes community tools, such as Test Kitchen, Foodcritic, Berkshelf, Chef Vault, Rubocop, and ChefSpec

**Chef client**

The Chef client is the tool that is deployed to all nodes. If we want to run Cookbook on Nodes, Chef client must be installed on it.it is a communicator between Nodes and workstation/Chef Server.

**Ohai Chef tool**

Ohai is the tool that gathers information about a node. Information such as platform details, operating system data, and processor information etc.,

**Recipes and cookbooks**

* Recipes are the building blocks used to define the desired states. Recipes are files of ruby code that define the commands to be run on nodes. They are like blueprints used to "build" a node. Recipes are collections of Chef resources.
* **Cookbooks are collections of related recipes**, templates, files, and custom resources. They provide organization and versioning for recipes. Each unique version of a cookbook represents unique sets of functionalities, such as bug fixes or added features.

**Chef run lists**

A "run list" is, as the name would suggest, a list of, and the sequence for, the recipes, cookbooks, and policies to be applied to a node

**Chef roles**

Roles are functional groupings of recipes and cookbooks used to describe the full blueprint needed for a node to become everything it is intended to be. Roles are reusable configurations, and they can be applied to multiple nodes to make functionally identical servers, such as a farm of web servers.

**Chef resources**

Chef resources define what we want to happen to a node. Chef resources have some types: **Package, Template, Service**, and so on.

* **Package**: This contains software or applications, such as apache, ntp, and cron, and the action to be performed on that software or application, such as "install".
* **Template**: These are files with placeholders for attributes that are transformed into configuration files for package installation and execution.
* **Service**: This is the installed executable of the package and the actions that the executable can perform, such as start, stop, or restart. Service also defines whether the software or application is launched at node startup.

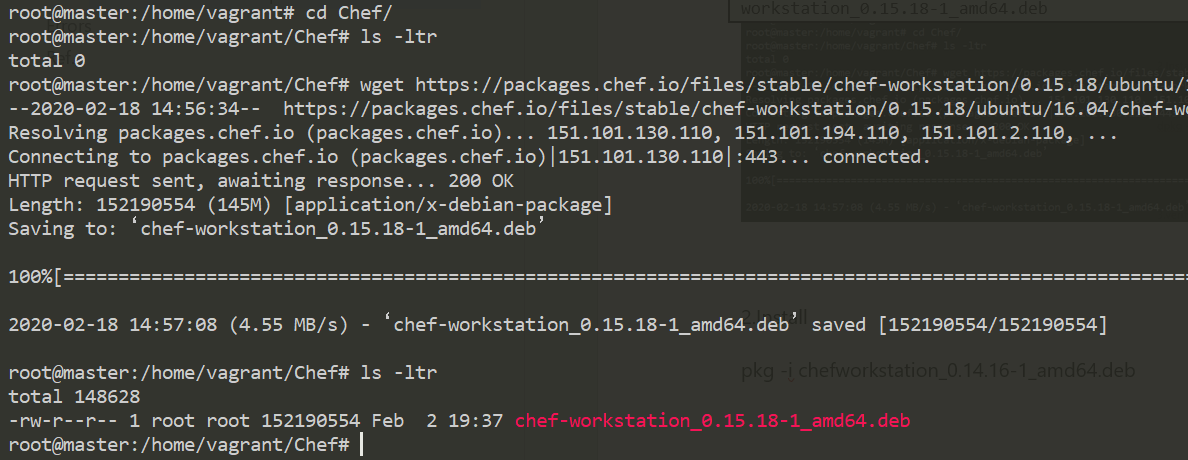
**Chef supermarket**

The supermarket is a site that provides shared cookbooks. There is a public supermarket that contains community-created and -maintained cookbooks. This site is hosted by Chef and is available at [https://supermarket.chef.io](https://supermarket.chef.io/)

Install the Chef Development kit (Chef DK) on Chef workstation

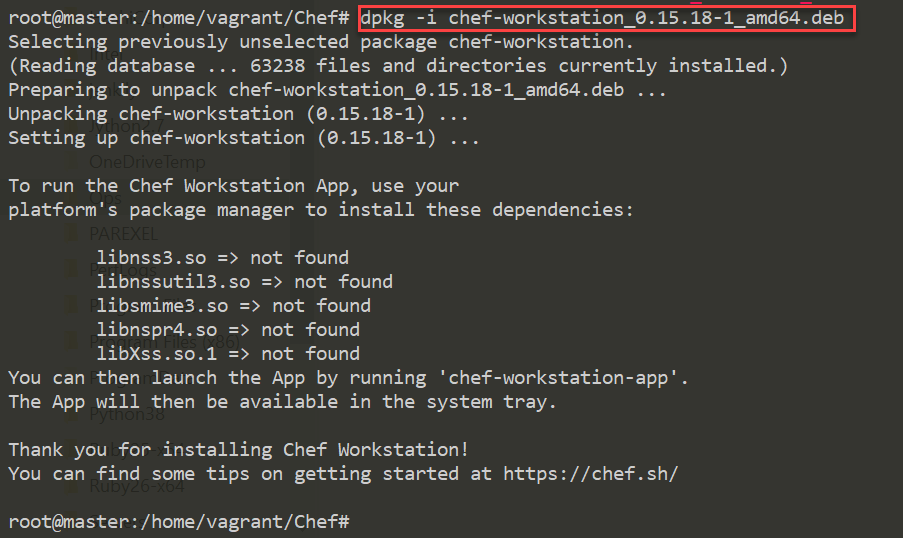
Go to  [Chef Workstation downloads page](https://downloads.chef.io/chef-workstation) and Select URL of appropriate package for your distribution.

wget https://packages.chef.io/files/stable/chef-workstation/0.15.18/ubuntu/16.04/chef-workstation\_0.15.18-1\_amd64.deb



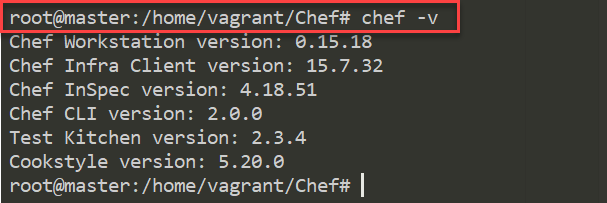
2.Install

dpkg -i chefworkstation\_0.14.16-1\_amd64.deb



3.verify the installation, run:

chef -v



# Chef – Recipe

Chef written in Ruby. Now we are going to create a recipe **“hello.rb”**, here. **rb** is Ruby extension. Collection of Recipes are called Cookbook.

**chef-apply**

* chef-apply tool is installed with the ChefDK. This is used to execute a single recipe on your local machine. There is no Chef server involved. Everything is local.
* **chef-apply** is not a tool for the deployment of production nodes. It’s just a quick and easy way to use Chef to configure your local system.

Syntax

sudo chef-apply -e "package '<pkg-name>"

sudo chef-apply <recipe>

## Chef Recipe– Hello World Recipe on Localhost

1.Create chef\_repo folder, go into it

mkdir chef\_repo

cd chef\_repo

2.create `**hello.rb**` file inside chef\_repo

file 'hello.txt' do

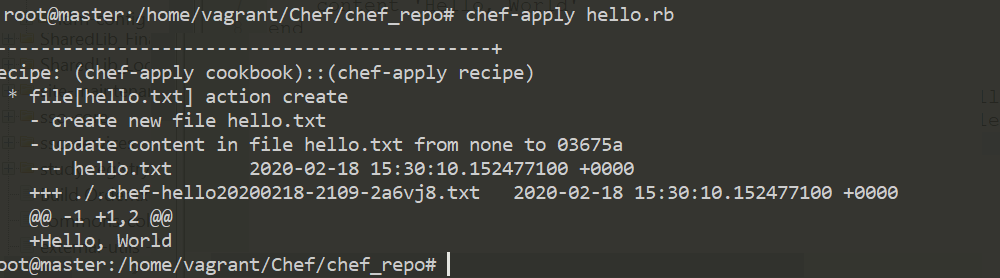
content 'Hello, World'

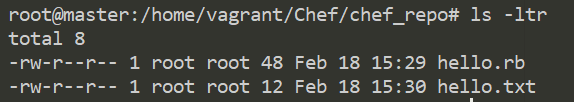
end

**3.Apply recipe to Current System.**

The above Code means, create a **resource(file)** with name `hello.txt` with content as “**hello world”.** By executing chef-apply **hello.rd,** Chef will create a new file in our current infrastructure system

chef-apply hello.rb



4.Now check, we can observe file is created with `hello.txt` 

5.Similarly, if you want to delete file, change **hello.rb** with below lines

file 'hello.txt' do

action:delete

end

## Chef Recipe– Install Apache on Localhost

In above example, we just created a file as a resource in host(local) machine. Now go to more advance install Software as a pkg in host machine

* Apache2 pkg should install in host machine
* Apache2 Should enable & Auto Start
* Create index.html, & make it as apache Homepage

**1.Create `apache.rd` to perform above 3 steps on host system**

package 'apache2'

# Here package apache2 has default action is install, so we didn't mention that action here. Space between action & [] must be there

service 'apache2' do

action [:enable, :start]

end

# Create a File with content to display Hello, Chef

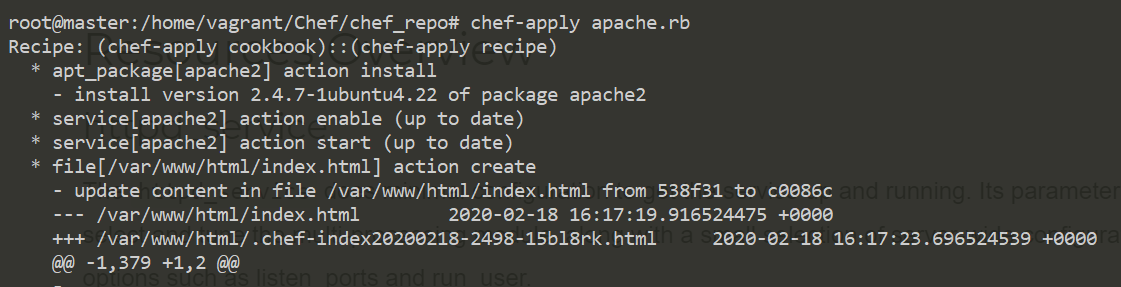
file '/var/www/html/index.html' do

content '<h1>Hello, Chef!!</h1>'

end

**2. Run `apache.rb` recipe using `chef-apply`**

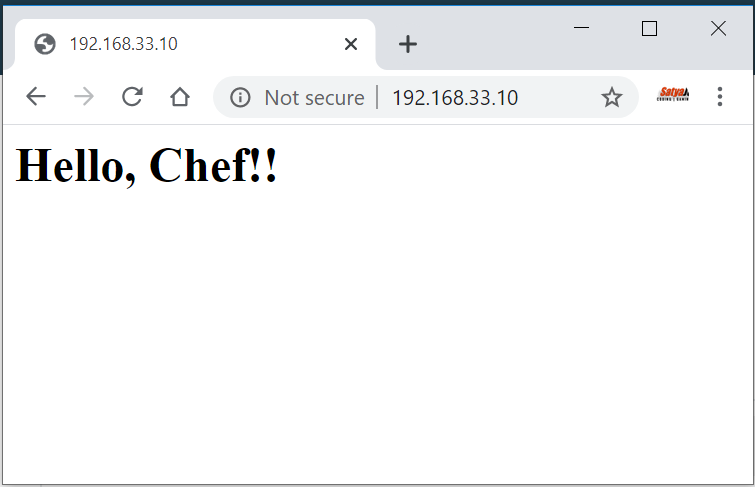
chef-apply apache.rb



3.Check Apache status

service apache2 status

\* apache2 is running

4.Check index.html by opening localhost/IP in browser 

5.we can check using curl as well

curl localhost

<h1>Hello, Chef!!</h1>

# Chef – CookBook

Cookbooks are created on the workstation and then uploaded to a Chef server. From there, recipes and policies described within the cookbook can be assigned to nodes as part of the node’s “run-list”. A run-list is a sequential list of recipes and roles that are run on a node by chef-client.

## Creating a Cookbook

There are two ways to dynamically create a cookbook.

**Using chef command**

chef generate cookbook <Cookbook Name>

**Using knife utility**

knife cookbook create <Cookbook Name>

## Chef Cookbook – Creating Sample Cookbook

We can generate a Cookbook Using Chef Tool

chef generate cookbook <Cookbook\_Name>

Here I am creating Cookbook for Apache Installation, so I’m proving cookbook name as `**apache\_cookbook** `

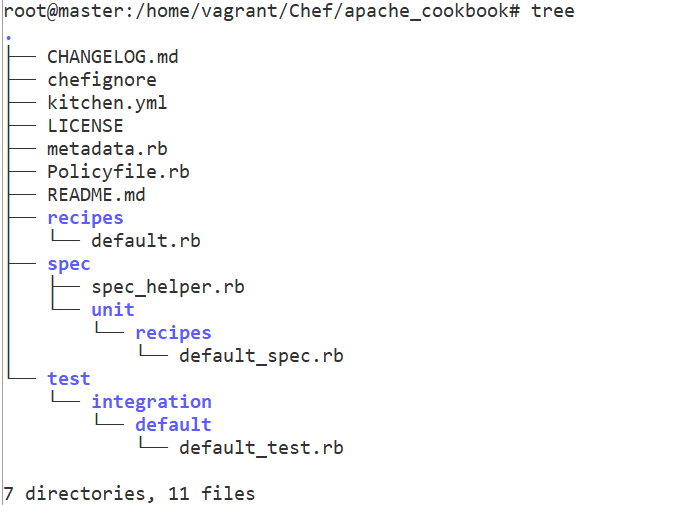
chef generate cookbook apache\_cookbook

In This cookbook we perform below tasks

* Apache2 pkg should install in host machine
* Apache2 Should enable & Auto Start
* Create index.html, & make it as apache Homepage

Cookbook Directory Structure

Cookbooks are organized in a directory structure that is completely self-contained. There are many different directories and files that are used for different purposes.

Go to **apache\_cookbook** directory & observe the File structure. 

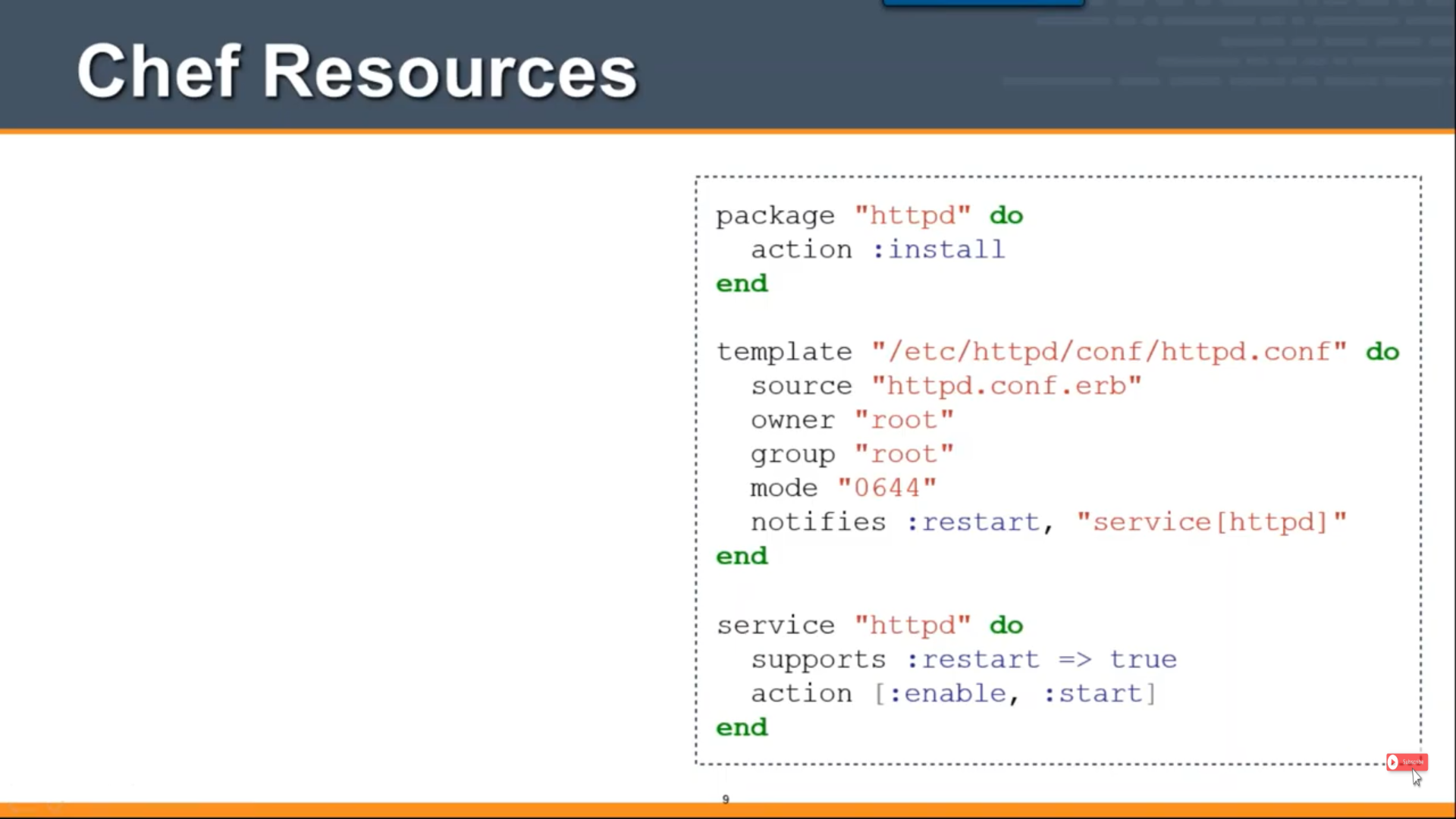
**Recipes**

* A recipe is the main workhorse of the cookbook. A cookbook can contain more than one recipe, or depend on outside recipes
* Recipes are used to declare the state of different resources.
* For instance, a resource could say “the package x should be installed”. Another resource may say “the x service should be running”.

Resources can be of many different types. Some common ones are:

* **package**: Used to manage packages on a node
* **service**: Used to manage services on a node
* **user**: Manage users on the node
* **group**: Manage groups
* **template**: Manage files with embedded ruby templates
* **cookbook\_file**: Transfer files from the files subdirectory in the cookbook to a location on the node
* **file**: Manage contents of a file on node
* **directory**: Manage directories on node
* **execute**: Execute a command on the node
* **cron**: Edit an existing cron file on the node

You can learn about the [different resource types](http://docs.opscode.com/resource.html) here.



**Templates**

Template files end with the .erb extension, meaning that they contain embedded Ruby.

These are mainly used to substitute attribute values into the file to create the final file version that will be placed on the node.

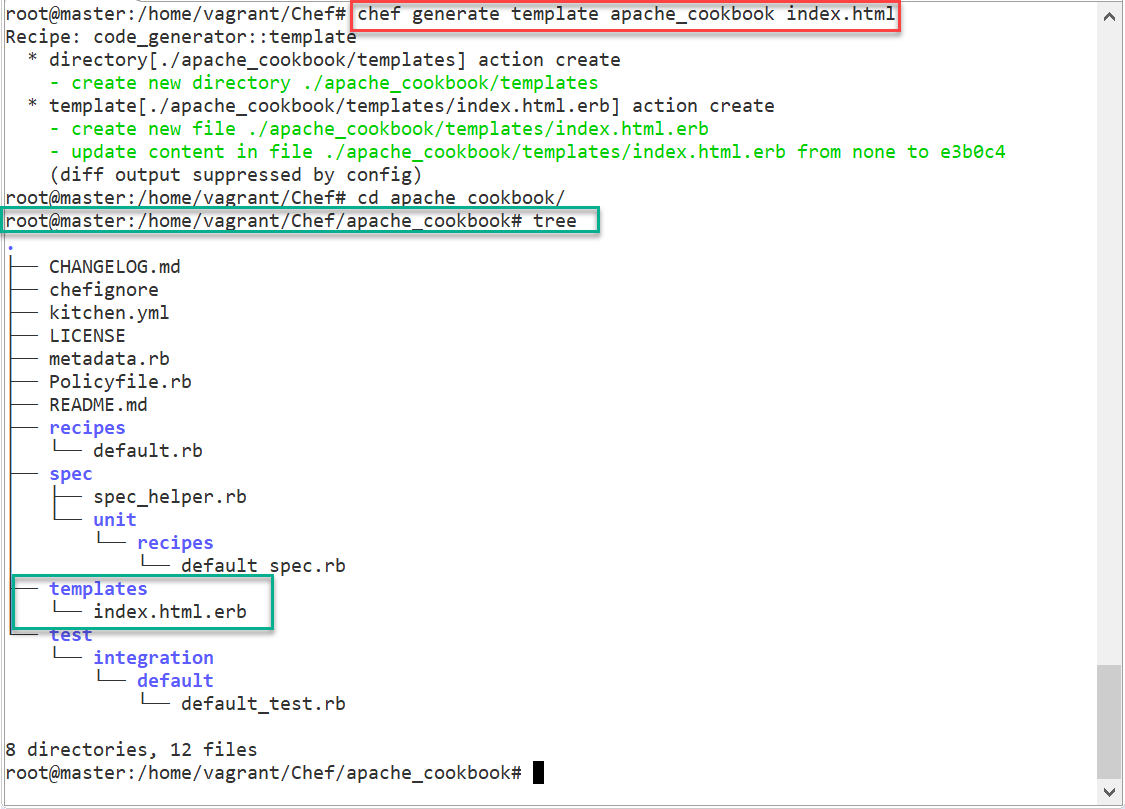
For example, if we have an attribute that defines the default port for a service, the template file can call to insert the attribute at the point in the file where the port is declared. Using this technique, you can easily create configuration files, while keeping the actual variables that you wish to change elsewhere.

**We need to create index.html to place apache2 server using Templates. But In above we don’t have Templates folder**.

**1.Create Templates Folder**

Create a Template called index.html in **apache\_cookbook(***Inside Cookbook folder/ outside apache***)**

chef generate template apache\_cookbook index.html



**2. Write “hello world” inside index.html.erb**

# vi templates/index.html.erb

<h1>

Hello, Chef - Using Cookbook

</h1>

**3.Edit Recipe file : default.rb**

Open recipe file recipes/default.rb & write steps to perform Install & Start Apache2 server and place index.html

# vi recipes/default.rb

package 'apache2'

service 'apache2' do

action [:enable, :start]

end

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

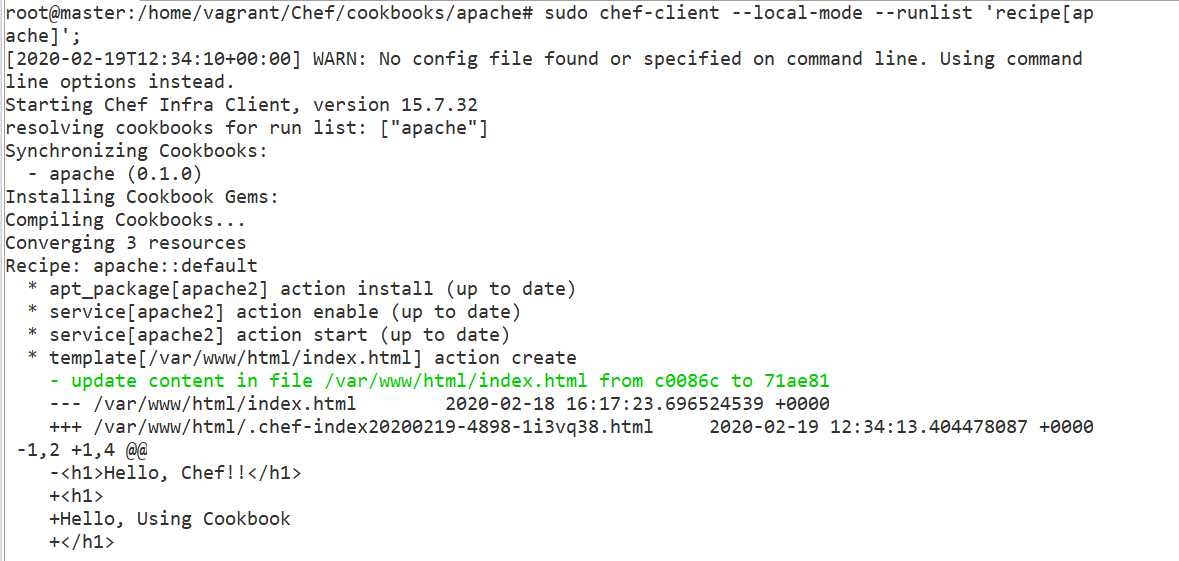
source 'index.html.erb'

end

**4.Apply complete CookBook on Local System**

sudo chef-client --local-mode --runlist ‘recipe[apache\_cookbook]’;

**Waring: should end with ;**

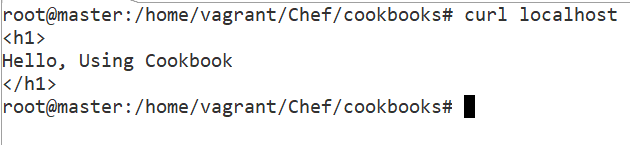


**Check Apache2 Status**

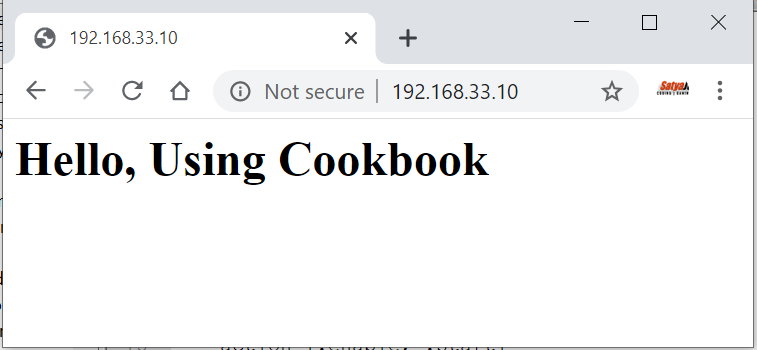
service apache2 status

\* apache2 is running

**Now open apache URL,** **Using CURL**



**Via Browser**



# Chef – Repo

Every Chef workstation has at least one Chef repository. The Chef repo is where all working copies of the cookbooks, recipes, templates, and so on live. Chef Repo contains

* **Cookbooks (***including recipes, attributes, custom resources, libraries, and templates)*
* **Data bags**
* **Policyfiles**

The chef-repo directory should be synchronized with a version control system, such as git. All of the data in the chef-repo should be treated like source code.

You’ll use the **chef** and **knife** commands to upload data to the Chef Infra Server from the chef-repo directory. Once uploaded, Chef Infra Client uses that data to manage the nodes registered with the Chef Infra Server and to ensure that it applies the right cookbooks, policyfiles, and settings to the right nodes in the right order.

### Directory Structure

| **Directory** | **Description** |
| --- | --- |
| **.chef/** | A hidden directory that is used to store validation key files and optionally a [config.rb](https://docs.chef.io/config_rb.html) file. |
| **cookbooks/** | Contains cookbooks that have been downloaded from the [Chef Supermarket](https://supermarket.chef.io/) or created locally. |
| **data\_bags/** | Stores data bags (and data bag items) in JSON (.json) format. |
| **policyfiles/** | Stores Policyfiles in Ruby (.rb) format. |

#### Create the chef-repo

Use the `chef generate repo` command to create your chef-repo directory along with the base folder structure.

chef generate repo <REPO\_NAME>

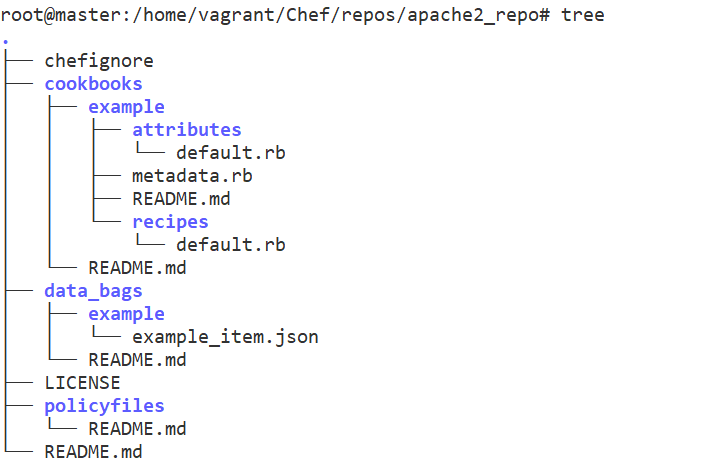
Create Sample Repo named - **apache2\_repo**

chef generate repo apache2\_repo

Generating Chef Infra repo apache2\_repo

- Ensuring correct Chef Infra repo file content

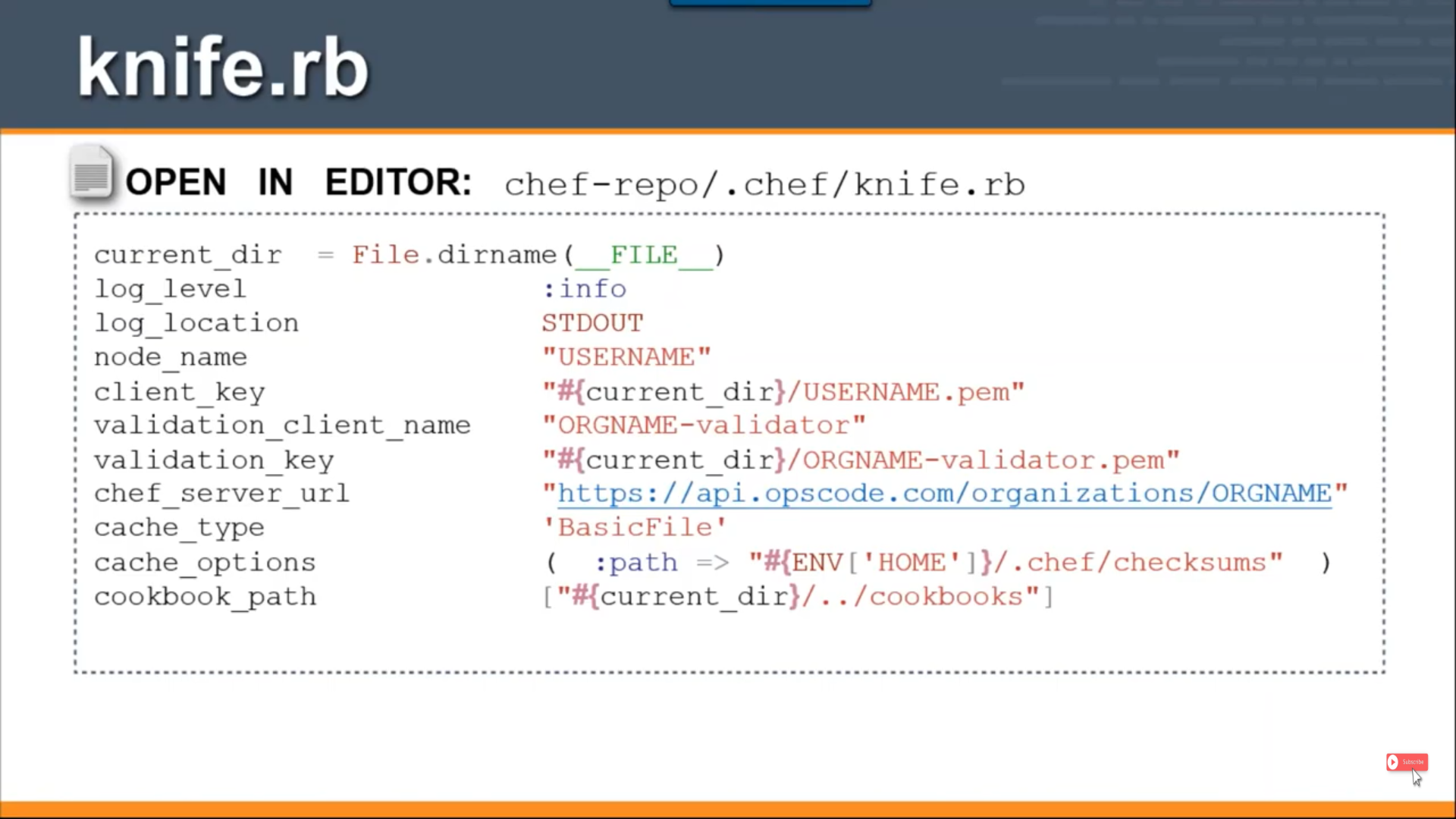
Your new Chef Infra repo is ready! Type `cd apache2\_repo` to enter it.



**.chef folder**

Really, the only files you need from the starter kit are the .chef folder files:

* The organization's validator private key file
* Your user private key file
* A knife.rb file



### Chef supermarket

The supermarket is a site that provides shared cookbooks. There is a public supermarket that contains community-created and -maintained cookbooks. This site is hosted by Chef and is available at <https://supermarket.chef.io>

There are two versions of Supermarket available today.

**Public Supermarket**

This is available at the Chef Supermarket site. This is an open source project, you can find and contribute to the repo on GitHub.

**Private Supermarket**

There is also a version of Supermarket that can be run privately in your own infrastructure.

You will need Knife to interact with the Supermarket

### Chef Knife

**Knife** is a command-line tool that supplies the interface between your **chef-repo and your Chef server**.

* It allows developers to upload their work on the workstation to the Chef server, from where it can be distributed to the nodes during their Chef client runs.
* Knife also allows you to obtain cookbooks from the supermarket, and it provides a mechanism to bootstrap nodes.

You can find more about Knife on Next Tutorial.

# Chef – Knife

knife is a command-line tool that provides an interface between a local chef-repo and the Chef Infra Server. knife helps users to manage:

* Nodes
* Cookbooks and recipes
* Roles, Environments, and Data Bags
* Resources within various cloud environments
* The installation of Chef Infra Client onto nodes
* Searching of indexed data on the Chef Infra Server

## Knife Subcommands

knife includes a collection of built in subcommands that work together to provide all of the functionality required to take specific actions against any object in an organization, including cookbooks, nodes, roles, data bags, environments, and users.

All knife subcommands have the following syntax:

knife subcommand [ARGUMENT] (options)

#### Knife Bootstrap

You use the knife bootstrap subcommand to install the chef-client on Chef Nodes. The bootstrap process also registers the Nodes with the Chef server to facilitate communication between the two.

you would use a command like knife

knife **bootstrap 192.168.10.44**.

This instructs knife to install the chef-client on the system with the IP address 192.168.10.44, register that system with the Chef server, and create a node representing the system on the Chef server.

#### Knife Node

The knife node subcommand is used to query and set the various configuration values for a node.

#### Knife Diff

Knife diff is used to compare the files in your current Workstation repo with what is stored on the Chef server.

#### Knife Exec

Exec is a knife subcommand that lets you run Ruby scripts against the Chef server in the context of a fully configured chef-client,

knife exec -E 'nodes.transform("chef\_environment:\_default") { |n| n.chef\_environment("development") }'

#### Knife SSH

The knife ssh subcommand lets you run ssh commands on all or a subset of the nodes known to your Chef Server.

knife ssh "chef\_environment:production" "uptime" -p 2222 -x earl -PMyPW99

#### Knife Search

knife search lets you do queries against all of the data stored in the Chef server.

#### Knife Windows plugin

We use the knife Windows plugin when we are working with nodes running the Windows operating system

Installing the Knife Windows plugin on Master

chef gem install knife-windows

Bootstrapping a Windows node

knife bootstrap windows winrm node03 -p 55985 -x Administrator -P MyPW99 -r 'recipe[winbase]'

#### Knife EC2 plugin

This plugin is used to integrate knife with Amazon cloud instances that allow you to create and manage instances from the command line.

Installing the Knife EC2 plugin

chef gem install knife-ec2

The knife EC2 plugin requires some specific data related to your Amazon account. to add the Amazon data to your knife.rb file, it should be added in this format:

knife[:aws\_access\_key\_id] = "Your AWS Access Key ID"

knife[:aws\_secret\_access\_key] = "Your AWS Secret Access Key"

If you already have some EC2 instances running.You can issue a command like this:

knife ec2 server list -r us-west-2

**Creating an EC2 instance**

Using the Knife EC2 command to create a new EC2 instance is somewhat complicated. A "simple" version of the command might look something like this:

knife ec2 server create --node-name ec2-node08 \

--groups=launch-wizard-1 \

--region=us-west-2 \

--availability-zone=us-west-2a \

--image=ami-5189a661 \

--server-connect-attribute=public\_ip\_address \

--flavor=t2.micro \

--ssh-user=ubuntu \

--ssh-key=my\_chef\_service \

--identity-file=/Users/earlwaud/.aws/my\_chef\_service.pem \

--run-list=recipe['base']

**Bootstrapping an existing EC2 instance**

To bootstrap an existing EC2 instance, you will issue a command much like the one used to bootstrap any other server. It might look something like this:

knife bootstrap 54.213.235.40 \

--node-name my\_old\_ec2\_node \

--ssh-user ubuntu \

--identity-file /Users/earlwaud/.aws/my\_chef\_service.pem \

--sudo

**Deleting EC2 instances**

delete --node-name <node-name> --purge

#### Knife cookbook commands

**List** - To see a list of all community cookbooks available from Supermarket.

knife cookbook site list

**Search** - To search particular cookbook

knife cookbook site search mysql

**Show** - To view more information about a particular cookbook

knife cookbook site show mysql

**Download** - To download a cookbook as a tar.gz archive and place it in the current working directory

knife cookbook site download mysql

Uploading to the Supermarket

There are a few things you’ll need in place before you can upload your cookbook to the Supermarket.

edit .chef/knife.rb.

node\_name "satyacodes" # Replace with the login name you use to login to the Supermarket.

client\_key "#{ENV['HOME']}/.chef/client.pem" # Define the path to wherever your client.pem file lives. This is the key you generated when you signed up for a Chef account.

cookbook\_path [ '/Users/nshamrell/Projects/my\_chef\_repo/cookbooks' ] # Directory where the cookbook you're uploading resides.

**Share** - to upload the cookbook to the Supermarket

knife cookbook site share "my\_apache2\_cookbook" "Web Servers"

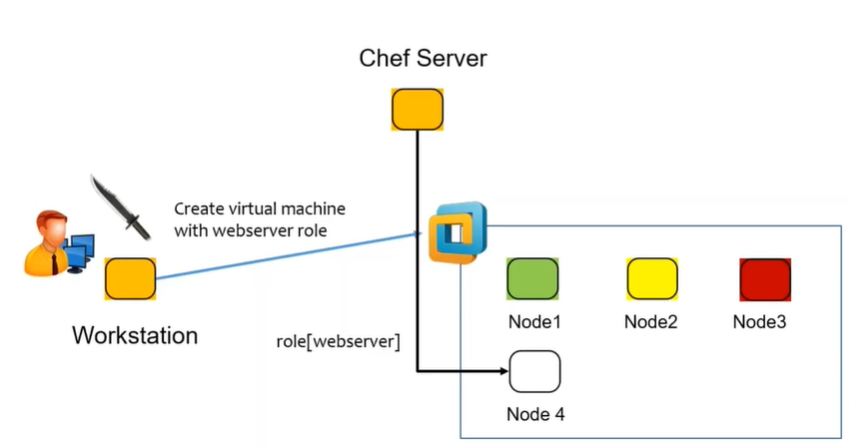
“Web Servers" - is category my cookbook

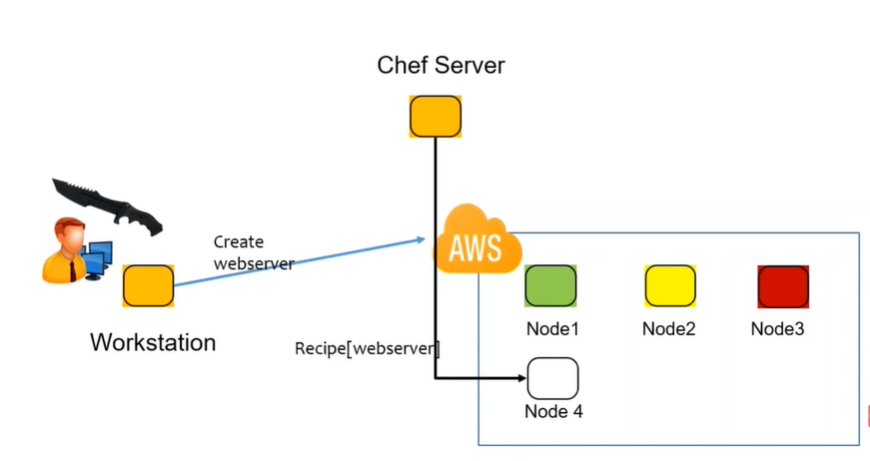
**Unshare** - to Remove your cookbook from the Supermarket.

knife cookbook site unshare my\_apache2\_cookbook

## Knife plug-ins to Create New Nodes/ Servers

Knife Plug-in : Create server with VMWare plugin



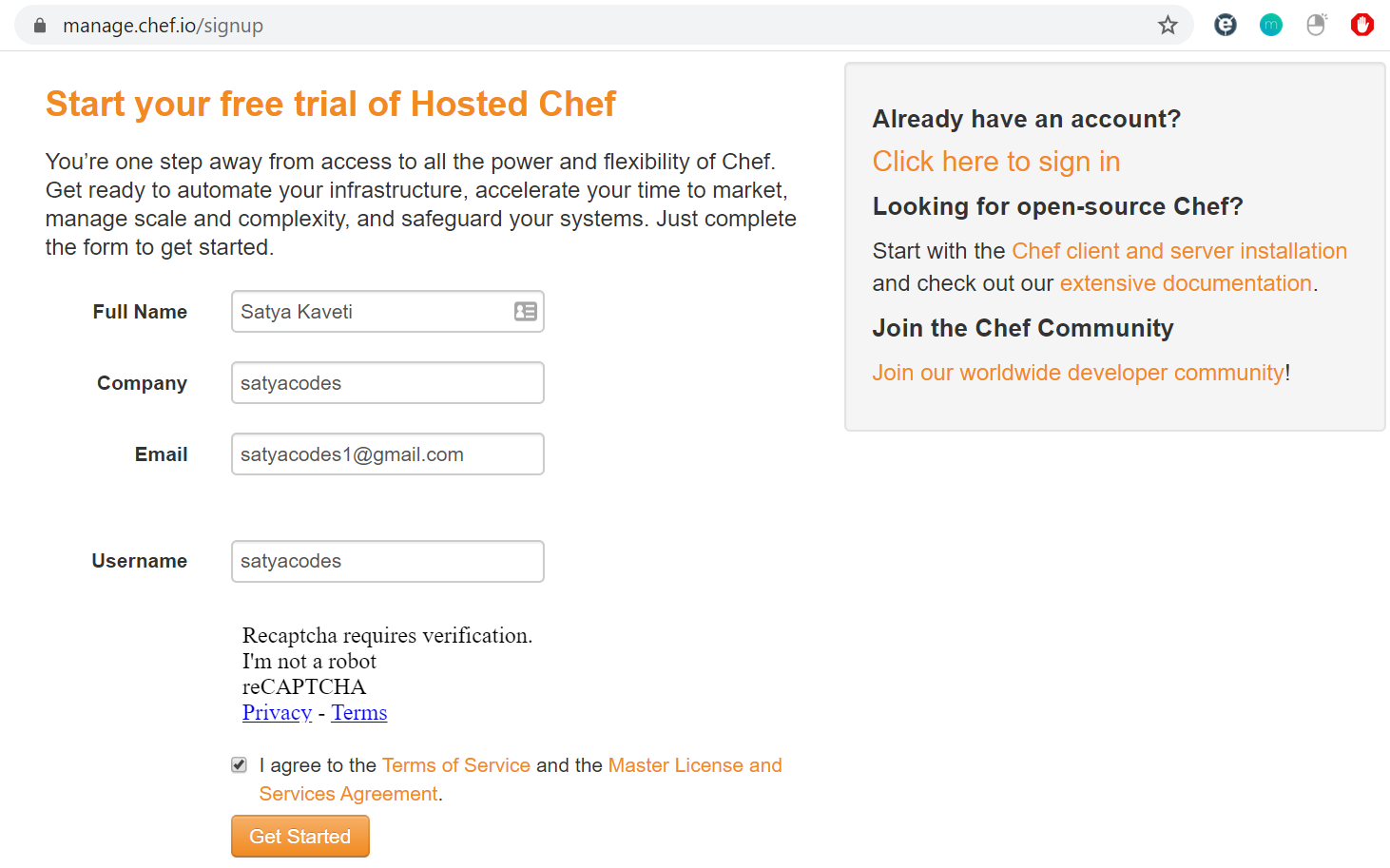
Knife Plug-in : Create server with AWS EC2 plugin 

# Chef – Server

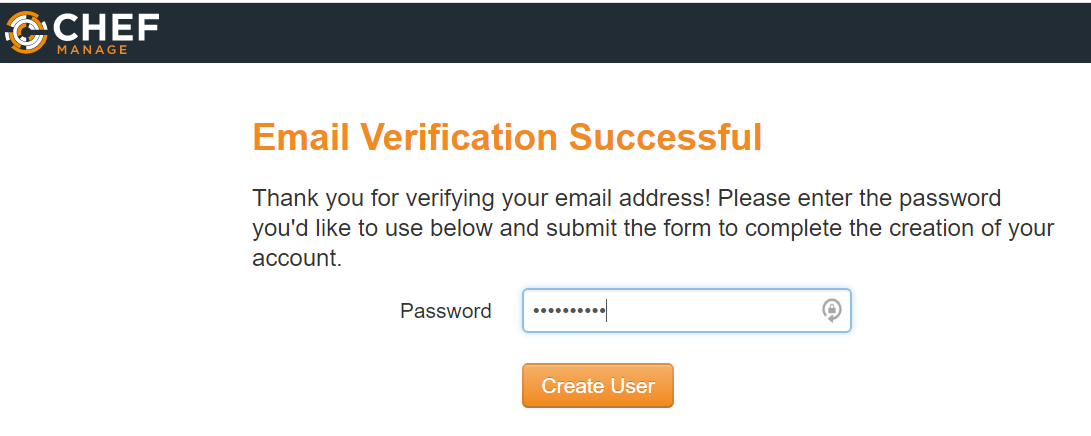
So far we did changes on local system only. Now we are going to execute our cookbooks in Hosted Nodes through **Chef server**

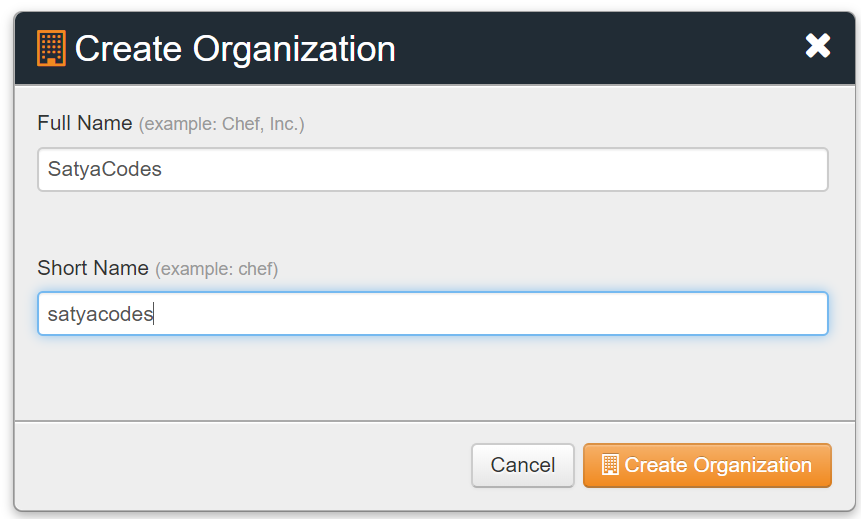
## Chef Server – Setup Online Chef server

Signup By going <https://manage.chef.io/signup>



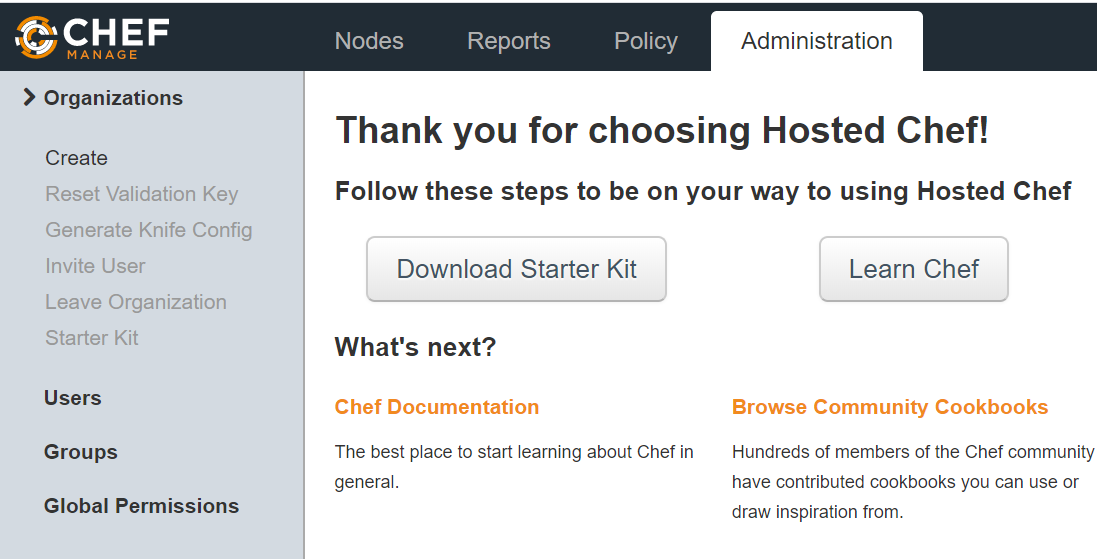
Verify Email & Setup Password



It will ask for Organization, Create one. 

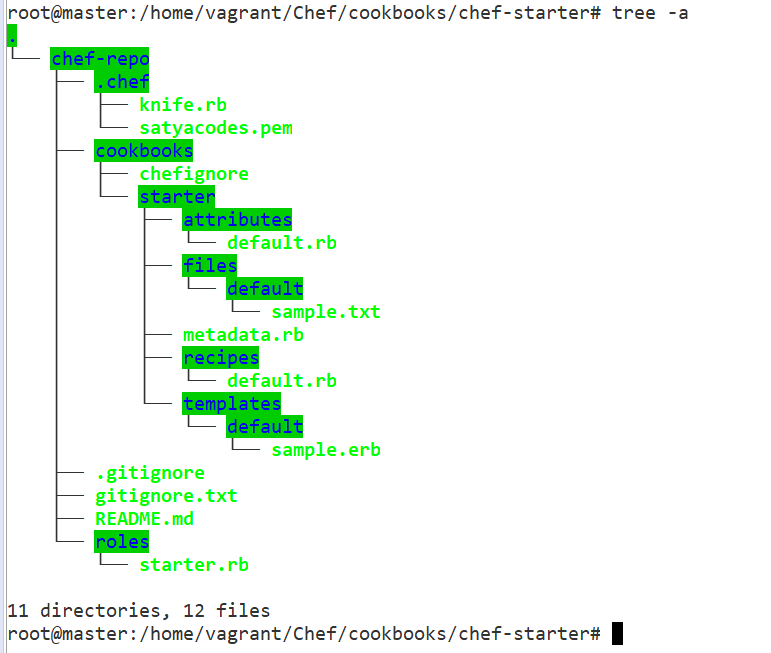
**Download Starter Kit**

Go to Administration > Organization >satyacodes > Actions > Starter Kit > Download Starter Kit



**Download Kit.zip in Local System**

* Copy to Vagrant shared folder
* `vagrant reload` to refresh Shared Folder content
* copy that to chef folder : mv /shareFolder/chef-starter/ /home/vagrant/Chef/cookbooks/
* Check the Folder Structure: tree -a

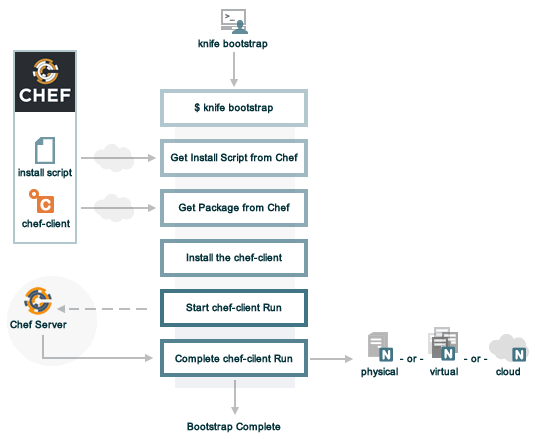


## Chef Server – Bootstrap Chef Nodes

The knife bootstrap command is a common way to install the chef-client on a node.

Once the workstation machine is configured, it can be used to install the chef-client on one (or more) nodes across the organization using a knife bootstrap operation.

* The knife bootstrap command is used to SSH into the target machine
* It will install the chef-client executable (if necessary), generate keys,
* register the node with the Chef server.

The following diagram shows the stages of the bootstrap operation, and the list below the diagram describes each of those stages in greater detail. 

| **Stages** | **Description** |
| --- | --- |
| **$ knife bootstrap** | Enter the knife bootstrap subcommand from a workstation. Include the hostname, IP address, or FQDN of the target node as part of this command. Knife will establish an SSH or WinRM connection with the target node using port 22 and assemble a shell script using the chef-full.erb file, which is the default bootstrap template. |
| **Get the install script from Chef** | The shell script that is derived from the chef-full.erb bootstrap template will make a request to the Chef website to get the most recent version of a second shell script (install.sh).  On Microsoft Windows machines: The batch file that is derived from the windows-chef-client-msi.erb bootstrap template will make a request to the Chef website to get the .msi installer. |
| **Get the Chef Infra Client package from Chef** | The second shell script (or batch file) then gathers system-specific information and determines the correct package for Chef Infra Client, and then downloads the appropriate package from omnitruck-direct.chef.io. |
| **Install Chef Infra Client** | Chef Infra Client is installed on the target node. |
| **Start a Chef Infra Client run** | On UNIX- and Linux-based machines: The second shell script executes the chef-client binary with a set of initial settings stored within first-boot.json on the node. first-boot.json is generated from the workstation as part of the initial knife bootstrap subcommand.  On Microsoft Windows machines: The batch file that is derived from the windows-chef-client-msi.erb bootstrap template executes the chef-client binary with a set of initial settings stored within first-boot.json on the node. first-boot.json is generated from the workstation as part of the initial knife bootstrap subcommand. |
| **Complete a Chef Infra Client run** | a Chef Infra Client run proceeds, using HTTPS (port 443), and registers the node with the Chef Infra Server.  The first Chef Infra Client run, by default, contains an empty run-list. A [run-list can be specified](https://docs.chef.io/knife_bootstrap.html) as part of the initial bootstrap operation using the --run-list option as part of the knife bootstrap subcommand. |

The bootstrap operation requires the **IP address** or FQDN of the target system**, the SSH credentials** and **Root access** to the node.

The general syntax of the command will be:

knife bootstrap <IP / HOSTNAME> [options]

#### Bootstrap With password authentication

knife bootstrap <IP / HOSTNAME>

--ssh-user <USERNAME>

--ssh-password <PASSWORD>

--sudo --use-sudo-password

--node-name <NODE-NAME>

--run-list 'recipe[<cookbook-name>]';

#### Bootstrap With With key authentication

knife bootstrap <IP / HOSTNAME>

-x <USERNAME>

-i <PATH\_TO\_KEY\_FILE>

--sudo --node-name <NODE-NAME>

--environment <ENVIRONMENT>

--run-list 'recipe[MY\_COOKBOOK1]’

when you are using hosted chef, you need to pass in a private key with the bootstrap and have the public key in your autherized\_keys file....

1. install the ChefSDK in Master
2. Open master Terminal, Download your starter kit from hosted Chef
3. extract the starter kit to ~/chef-repo
4. Go to chef-repo, generate a new keypair: **ssh-keygen**
5. add the public key to your authorized\_keys file: $ cat id\_rsa.pub >> authorized\_keys
6. run the knife bootstrap with the following:

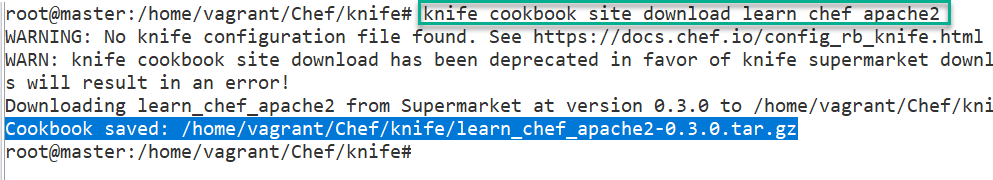
sudo knife bootstrap {{server-ip}} --ssh-user {{your-server-user}} -i ~/.ssh/id\_rsa --sudo --node-name web1

## Using Chef Server - Install Apache2 on Two Nodes machines

#### 1.Download Apahe2 Cookbook from Supermarket

we are going to use **learn\_chef\_apache2** cookbook from Supermarket in this example.To Download **learn\_chef\_apache2** , use below command

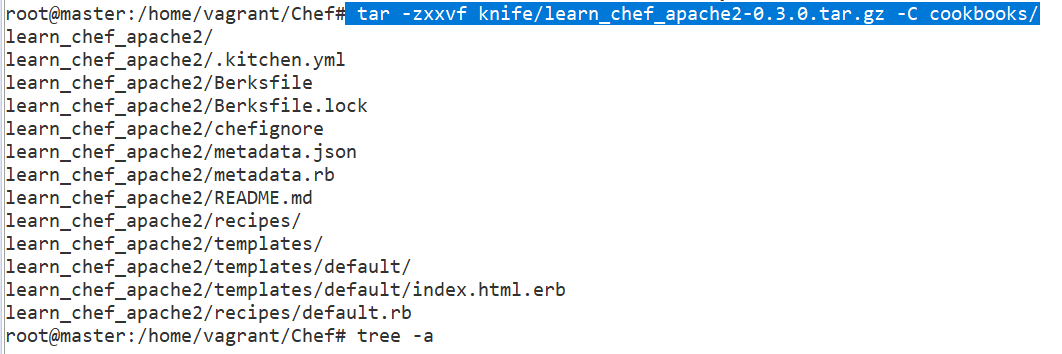
knife cookbook site download learn\_chef\_apache2

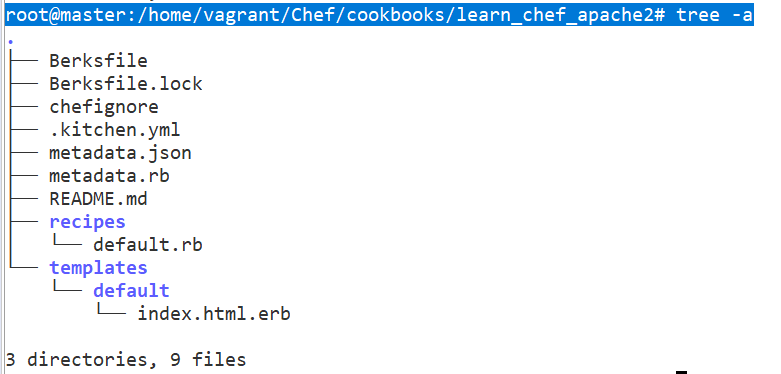


Extract tar file to cookbooks folder

tar -zxxvf knife/learn\_chef\_apache2-0.3.0.tar.gz -C cookbooks/

we can see following files were extracted





Check **learn\_chef\_apache2/recipes/default.rb** it contains same code we configured before

apt\_update 'Update the apt cache daily' do

frequency 86\_400

action :periodic

end

package 'apache2'

service 'apache2' do

supports :status => true

action [:enable, :start]

end

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

source 'index.html.erb'

end

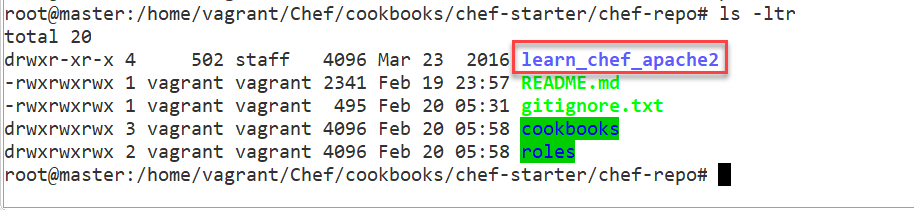
#### 2. Upload CookBook to Chef Server

We are downloaded this from Supermarket. If we want Upload this CookBook to Chef Server

**First, copy this repository to Chef-Repo/cookbooks**

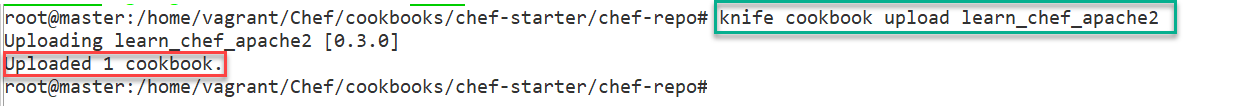
go to /chef/chef-starter/chef-repo(Extracted Starter Kit) folder & copy **learn\_chef\_apache2** here

mv cookbooks/learn\_chef\_apache2/ cookbooks/chef-starter/chef-repo/cookbooks/



**To Upload/commit changes** to Chef Server, Run below command inside **/chef-starter/chef-repo**

knife cookbook upload learn\_chef\_apache2



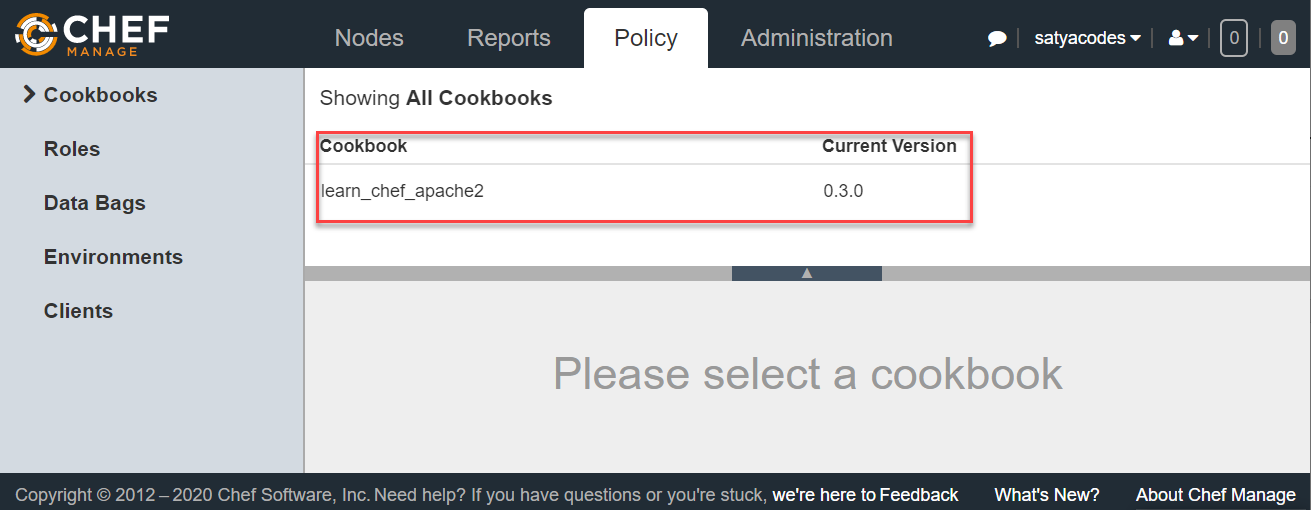
If we not copy this to /cookbooks folder it will throw below error

ERROR: Cannot find a cookbook named learn\_chef\_apache2; did you forget to add metadata to a cookbook?

Make sure you copy cookbook to chef-repo/cookbooks folder

**Check Changes in Chef Server**

* Login to Chef Server : <https://manage.chef.io/login>
* Go to Dashboard > Top Menu > Policy Tab : observe cookbook uploaded.



#### 3.Chef Nodes Details

We have Two Vagrant Nodes

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

# Node: 1

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

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

config.vm.box = "ubuntu/trusty64"

config.vm.network "private\_network", ip: "192.168.33.11"

config.vm.hostname = "web1.satyacodes.vm"

config.vm.network "forwarded\_port", guest: 80, host: 5555,

auto\_correct: true

end

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

# Node: 2

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

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

config.vm.box = "ubuntu/trusty64"

config.vm.network "private\_network", ip: "192.168.33.12"

config.vm.hostname = "web2.satyacodes.vm"

config.vm.network "forwarded\_port", guest: 80, host: 6666,

auto\_correct: true

end

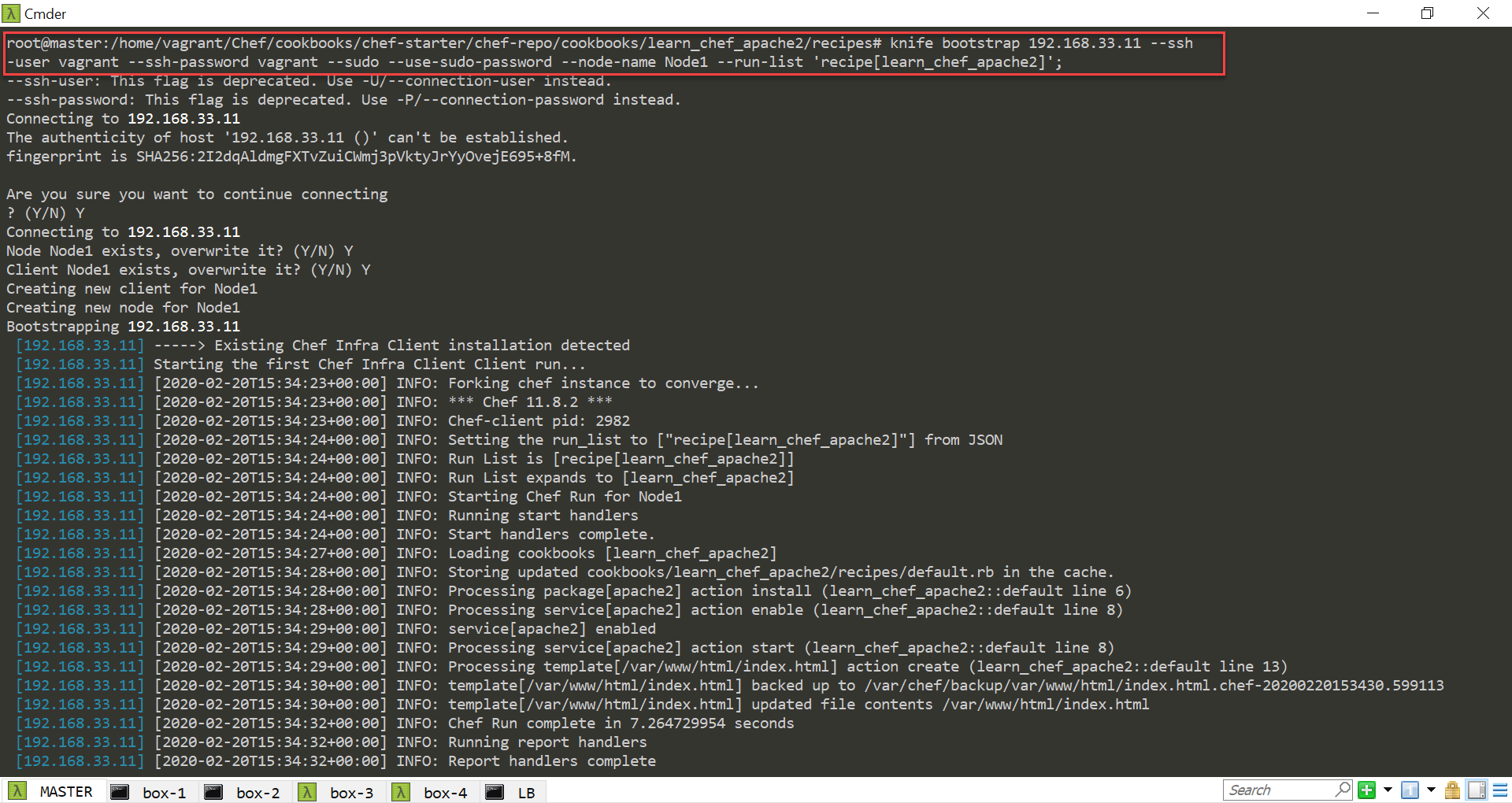
#### 4.Boostrap & Run Apache Cookbook on Nodes

Go to Chef workstation & Navigate to Chef Repo(chef/chef-starter/chef-repo), execute below commands to bootstrap & Install Apache2 on both the nodes.

Remove Apache2 if already installed using : sudo apt remove apache2.\*

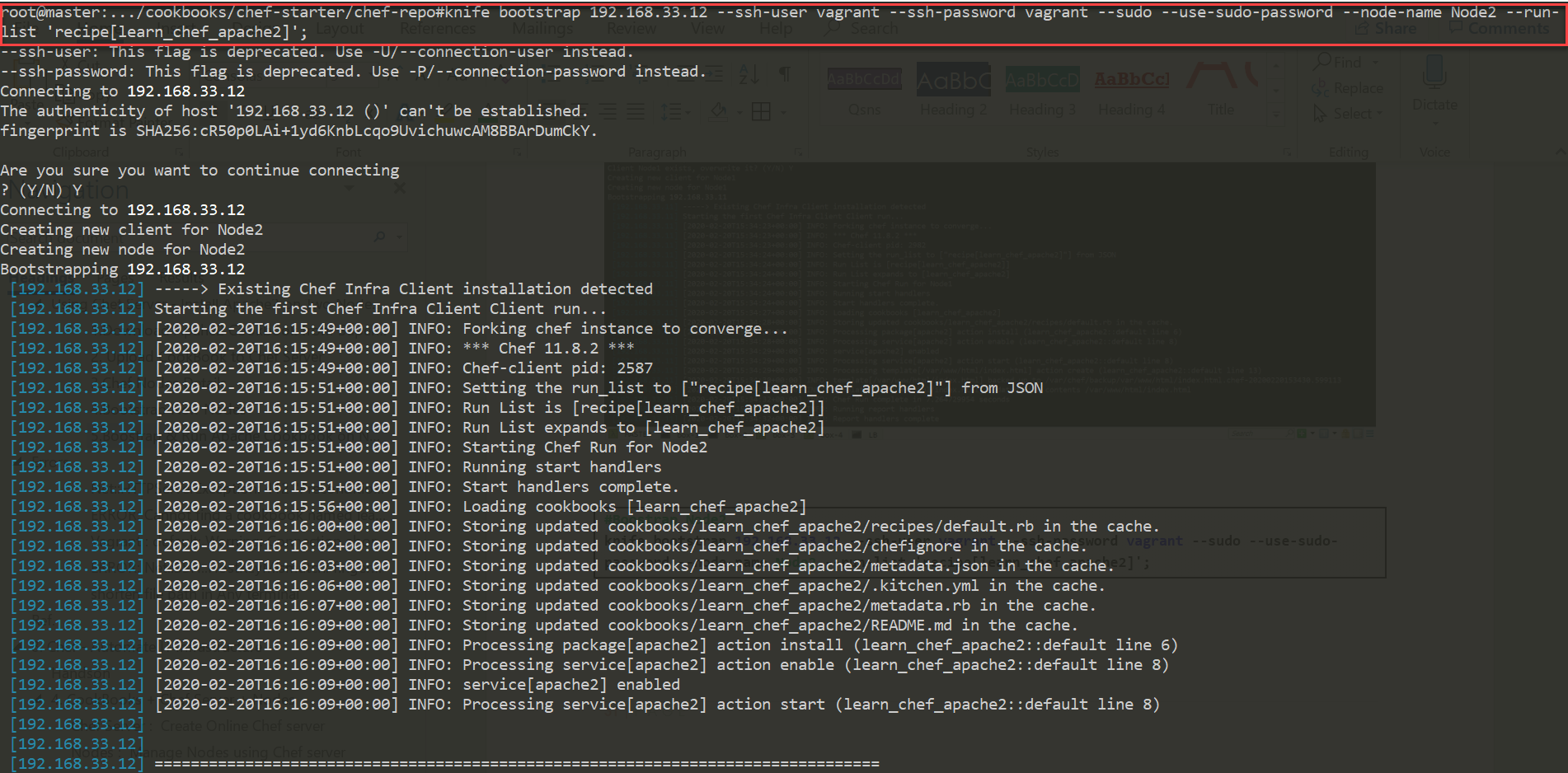
#Boostrap Node1

knife bootstrap 192.168.33.11 --ssh-user vagrant --ssh-password vagrant --sudo --use-sudo-password --node-name Node1 --run-list 'recipe[learn\_chef\_apache2]';



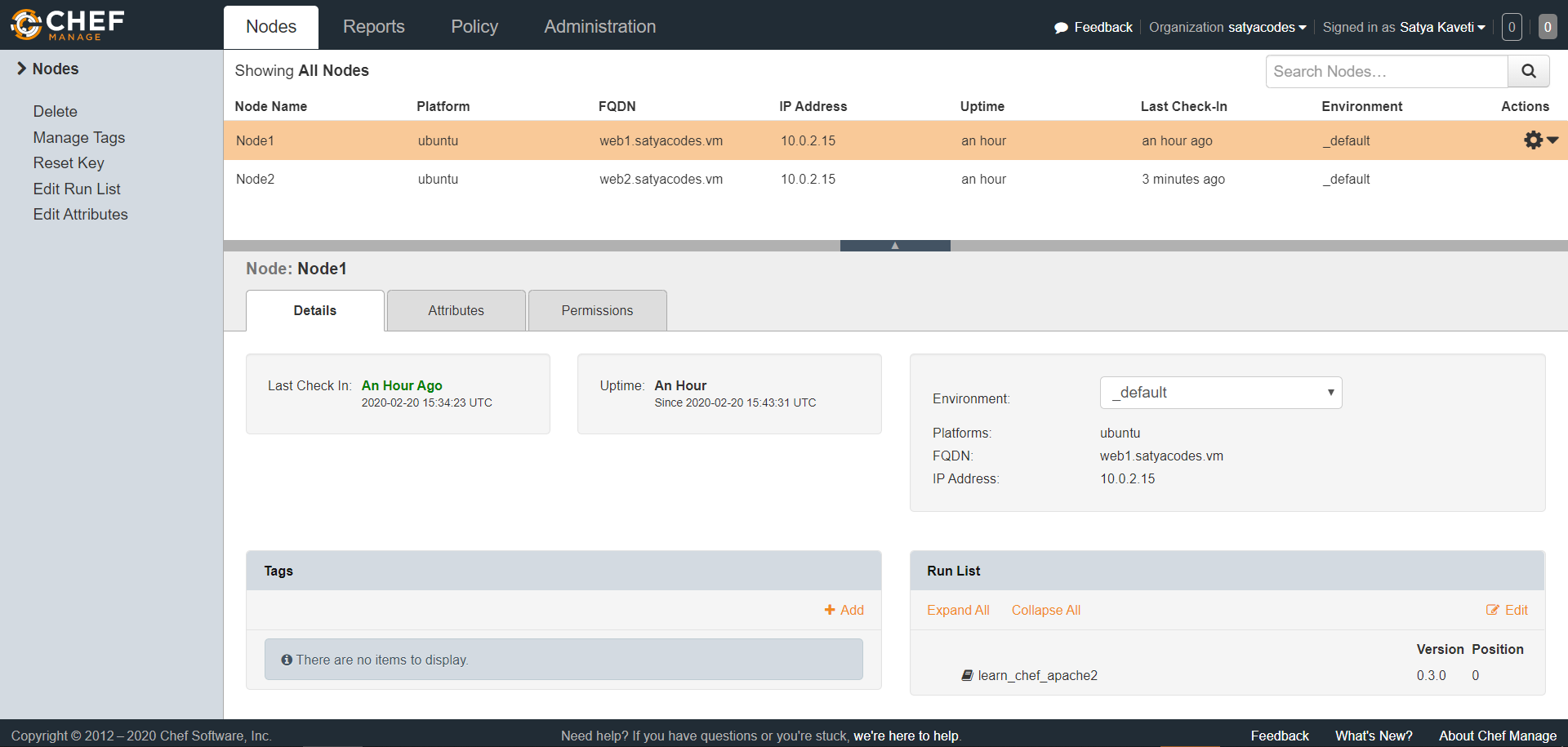
#Boostrap Node2

knife bootstrap 192.168.33.12 --ssh-user vagrant --ssh-password vagrant --sudo --use-sudo-password --node-name Node2 --run-list 'recipe[learn\_chef\_apache2]';



#### 5.Check Chef Server – Nodes Registered

Now we can check, knife automatically register Ubuntu **Node1 & Node 2** , with [**Chef Server**](https://manage.chef.io/)



#### 6.Check index.html in both Nodes

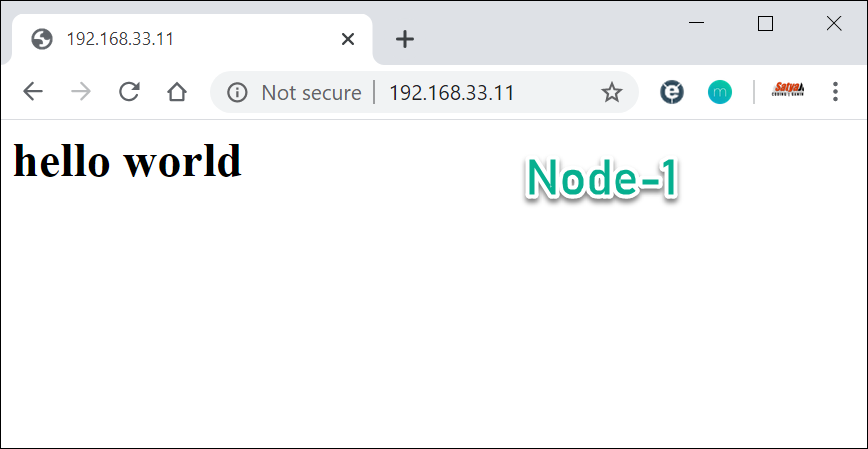
Check apache status

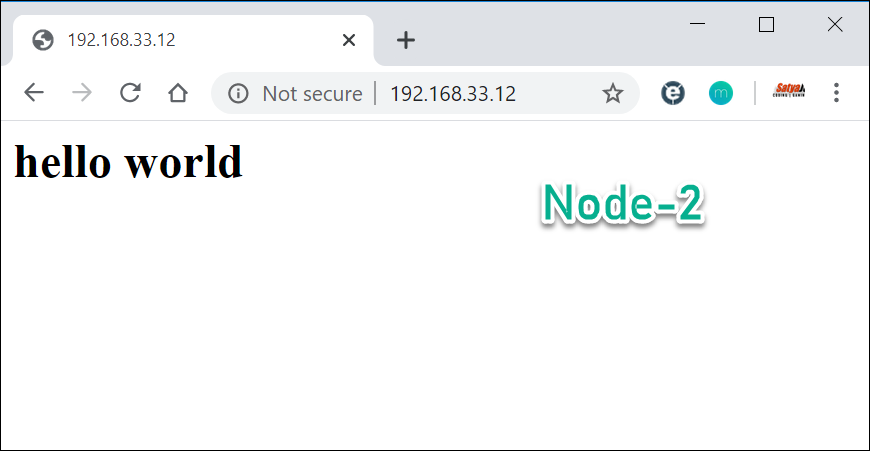
service apach2 status

Check Running apache2 Port

sudo lsof -i

Open Browser, Hit Nodes IP , check the index page





#### 7.More on Chef Nodes

To get no.of Nodes in Chef server

chef-repo> knife node list

Node1

Node2

Get Node Info

chef-repo> knife node show Node1

Node Name: cnode1

Environment: \_default

FQDN: vagrant-ubuntu-trusty-64

IP: 10.0.2.15

Run List: recipe[learn\_chef\_apache2]

Roles:

Recipes: learn\_chef\_apache2

Platform: ubuntu 14.04

Tags:

## Separate Bootstrapping

Just Bootstrap

Knife bootstrap 192.168.1.10 -x root -P vagrant

To set Cookbook to a Node

knife node run\_list add node1 “recipe[apache]”

node1:

run\_list: recipe[apache]

Run ,manually on node machine - node will pick updated one

sudo chef-clinet

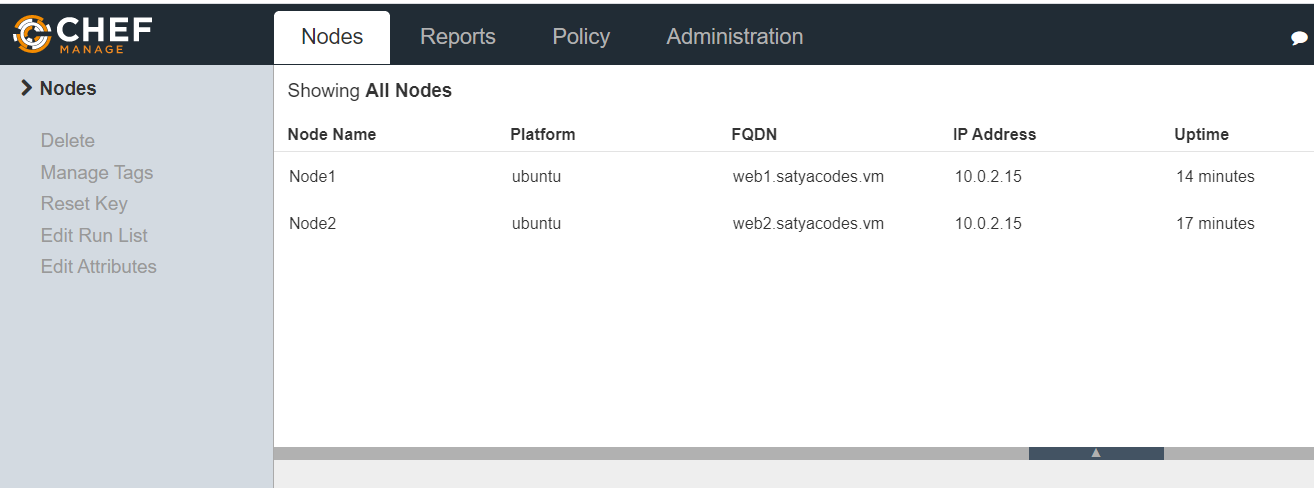
1.Boostrap Nodes – Without executing recipes

knife bootstrap 192.168.33.11 --ssh-user vagrant --ssh-password vagrant --sudo --use-sudo-password --node-name Node1

knife bootstrap 192.168.33.12 --ssh-user vagrant --ssh-password vagrant --sudo --use-sudo-password --node-name Node2

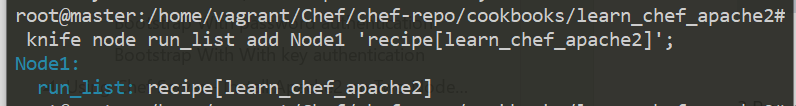


**2.Check Chef Server** – for Reregistered Nodes



3.Set Cookbook to **– Node1** if we update the Chef server with his assigned Cookbooks

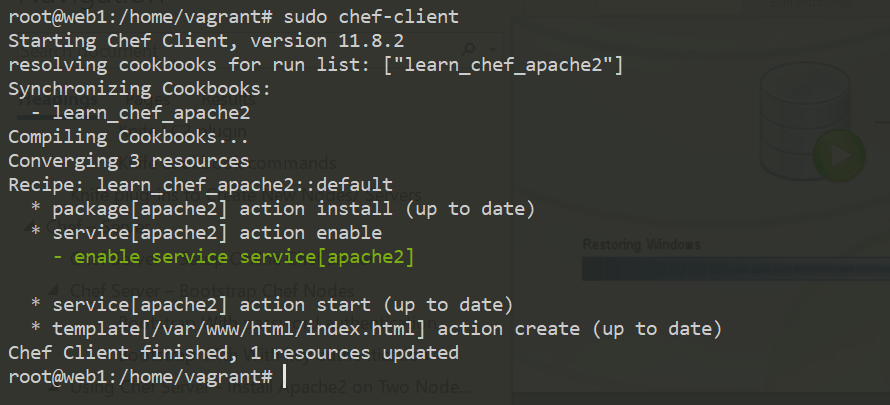
knife node run\_list add Node1 'recipe[learn\_chef\_apache2]';



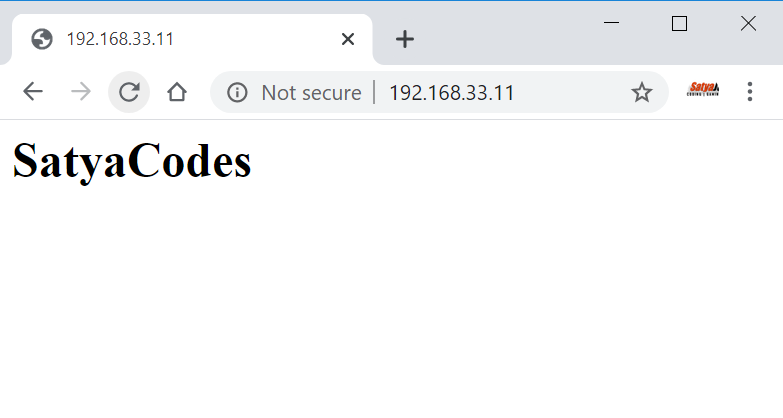
**4.Run chef-client**

If we run chef-client on Node1, it will Contact Chef-Server , it will ask for what are the **run\_list** assigned him, & it will run those tasks.

sudo chef-client



5.Test apache Server.



# Chef – Roles

So far, we've been just adding recipes directly to a single node. But that's not how your infrastructure works.

**Roles are there in Chef to group nodes with similar configuration**. Typical cases are to have roles for web servers, database servers, and so on.

Roles make it easy to configure many nodes identically without repeating yourself each time

In your organization, if your infrastructure grows to be multiple, redundant servers that all perform the same basic tasks. For instance, these might be web servers that a load balancer passes requests to. They would all have the same basic configuration and could be said to each satisfy the same “role”.

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

## Creating Roles

For creating any role we need to follow below steps

1. Create a role
2. Upload the role on the Chef server
3. Update Run list
4. Assign the role to the nodes
5. Run the Chef client

**we can create Role in two ways**

1. **Using Knife [JSON]-** Create Role in Chef Server directly
2. **Using Ruby** - In local repo under chef-repo folder

### 1.Using Knife [JSON]- Create Role in Chef Server directly

**1.Create a role**

knife role create <ROLE\_NAME>

**2.Update Run list**

{

"name":"client1",

"description":"",

"json\_class":"Chef::Role",

"default\_attributes":{

},

"override\_attributes":{

},

"chef\_type":"role",

"run\_list":[

"recipe[nginx]",

"recipe[phpapp::web]"

],

"env\_run\_lists":{

}

}

**3.Upload the role on the Chef server**

By closing Knife will auto save with Chef Server. To check

knife role show <ROLE\_NAME> -d -Fjson

**4.Assign the role to the nodes**

knife node run\_list set Node2 "role[<ROLE\_NAME>]"

**5.Run the Chef client**

sudo chef-client

### 2.Using Ruby - In local repo under chef-repo folder

**1.Create a role**

Create a `role` folder inside `chef-repo` & create webserver.rb file with runlists.

name "web\_server"

description "Role for web servers"

run\_list("role[base]", "recipe{web\_server]")

**2.Upload the role on the Chef server**

knife role from file chef-repo/roles/webserver.rb

**3.Update Runlist**

Already Did at Step#1

**4.Assign the role to the nodes**

knife node edit <node\_name>

**5.Run the Chef client**

sudo chef-client

## Create Role for Webserver

we are creating a Role; it consists of following recipes. we download these from Supermarket.

knife supermarket download apt

knife supermarket download nginx

knife supermarket download ohai

#Exatract It

tar -zxxvf -C cookbooks/

#upload Cookbooks

knife cookbook upload apt/nginx/ohai

Each role file can be written either in Ruby, or in JSON.

Create a *roles* directory in *chef-repo*:

$ mkdir roles

We’re going to create a *.json* file representing the role data. A basic role has a

* name:
* description: and
* run\_list:
* a long list of recipes

#### Create a Role for Web Server: Ruby

vi web\_server.rb

Start with Name of the Roles & Description: what this role for

name "web\_server"

description "A role to configure our front-line web servers"

Next, we can specify the runlist, contain cookbooks (recipe["<Cookbook-Name>"]), recipes (cookbook::recipe), and other roles. Remember, a run\_list is always executed sequentially, so put the dependency items before the other items.

name "web\_server"

description "A role to configure our front-line web servers"

run\_list "recipe[apt]", "recipe[nginx]"

to Upload Role to Chef Server manually, using below command

knife role from file web\_servers.rb

#### Create a Role Using JSON

To create a webserver role:

knife role create webserver

It should look something like this:

{

"name": "test",

"description": "",

"json\_class": "Chef::Role",

"default\_attributes": {

},

"override\_attributes": {

},

"chef\_type": "role",

"run\_list": [

],

"env\_run\_lists": {

}

}

When we save a JSON file created using the knife command, the role is created on the Chef server. But for Ruby.rb, we need to Upload Role to Chef Server manually

Change json file it as per Our Role

{

"name": "webserver",

"description": "A role to configure our front-line web servers",

"json\_class": "Chef::Role",

"default\_attributes": {

"nginx": {

"log\_location": "/var/log/nginx.log"

}

},

"override\_attributes": {

"nginx": {

"gzip": "on"

}

},

"chef\_type": "role",

"run\_list": [

"recipe[apt]",

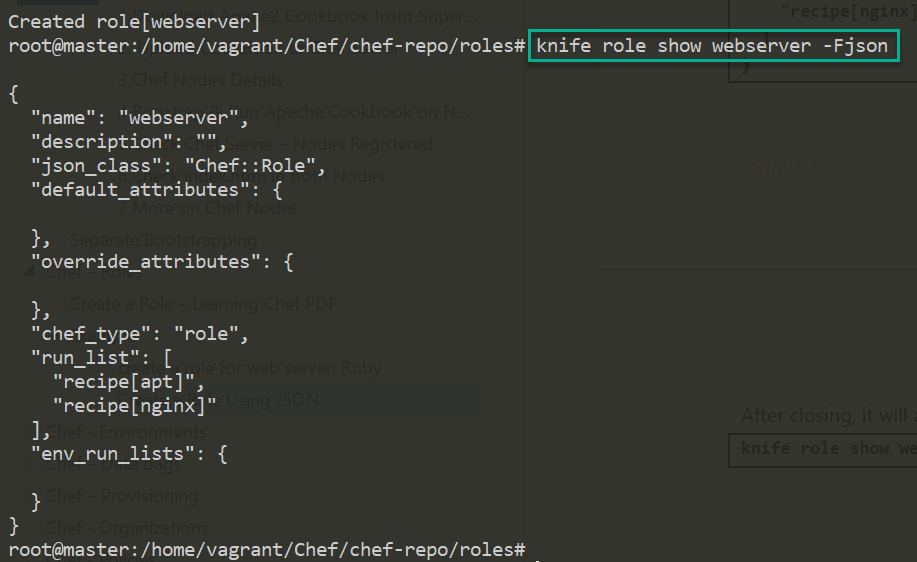
"recipe[nginx]"

]

}

After closing, it will automatically Saved in Chef Server. If we want to get our JSON file from the server.

knife role show webserver -Fjson



**Add Role to a Node**

Now we need to add our role to a node. For that first find list of nodes available

knife node list

Node1

Node2

And then we would give a command like:

knife node edit <node\_name>

knife node edit Node1

This will show the node’s definition file, which will allow us to add a role to its run\_list:

{

"name": "Node1",

"chef\_environment": "\_default",

"normal": {

"tags": [

]

},

"run\_list": [

"recipe[nginx]"

]

}

Now, we need to replace our recipe with our role in this file:

{

"name": "Node1",

"chef\_environment": "\_default",

"normal": {

"tags": [

]

},

"policy\_name": null,

"policy\_group": null,

"run\_list": [

"role[webserver]"

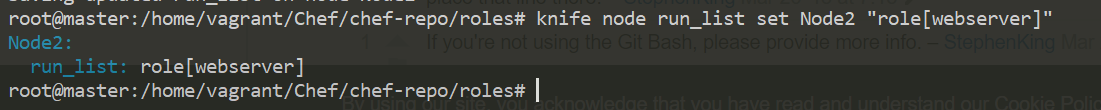
]

}

On close, it will automatically update Node1 run list with newly assigned role.

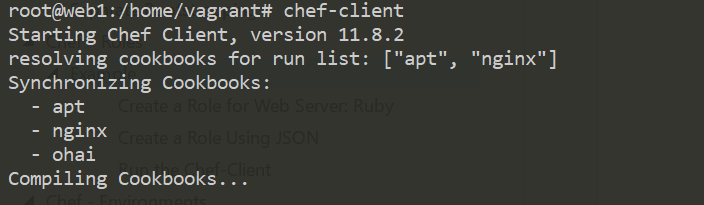
We can also do this in single command

knife node run\_list set Node2 "role[webserver]"



#### Run the Chef-Client

Login to Node1, run chef-client to



# Chef - Environments

Having separate environments for development, testing, and production are good ways to be able to develop and test cookbook updates and other configuration changes in isolation.Chef enables grouping nodes into separate environments to support an ordered development flow.

Every Chef Server starts out with a single environment, the **\_default** environment

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.

For Example, if you want one configuration for Development, one for Testing, & one for Production - we will do this by Environments.

## Creating New Environment

#### 1. Create new Environment

To Create new Environment, we need to create environments directory inside “**chef-repo**”.This is where we should put our environment files.

cd ~/chef-repo/environments

To create any environment, we need to place that environment specific configuration inside Environments folder. Here I’m creating `**development**` environment

vi development.rb

In this file you can specify **cookbooks and their version constraints**, and recipes that are deployed.so it contains

* name
* description
* cookbook\_versions
* override\_attributes

#development.rb

name "development"

description "The master development branch"

cookbook\_versions({

"nginx" => "<= 1.1.0",

"apt" => "= 0.0.1"

})

override\_attributes ({

"nginx" => {

"listen" => [ "80", "443" ]

},

"mysql" => {

"root\_pass" => "root"

}

})

We could also use the JSON format. The knife tool can generate the template of an environment file by typing. Provide default editor export EDITOR=vi

knife environment create development

This will open our editor with a preloaded environment file with the name filled in.

# Def. Generated File

{

"name": "development",

"description": "",

"cookbook\_versions": {

},

"json\_class": "Chef::Environment",

"chef\_type": "environment",

"default\_attributes": {

},

"override\_attributes": {

}

}

We need to edit with our details

{

"name": "development",

"description": "The master development branch",

"cookbook\_versions": {

"nginx": "<= 1.1.0",

"apt": "= 0.0.1"

},

"json\_class": "Cheff:Environment",

"chef\_type": "environment",

"default\_attributes": {

},

"override\_attributes": {

"nginx": {

"listen": [

"80",

"443"

]

},

"mysql": {

"root\_pass": "root"

}

}

}

#### 2.Upload Environment Files to/from the Server

We could upload our **Ruby file** to the Chef server by typing this:

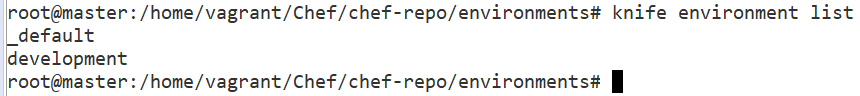
knife environment from file ~/chef-repo/environments/development.rb

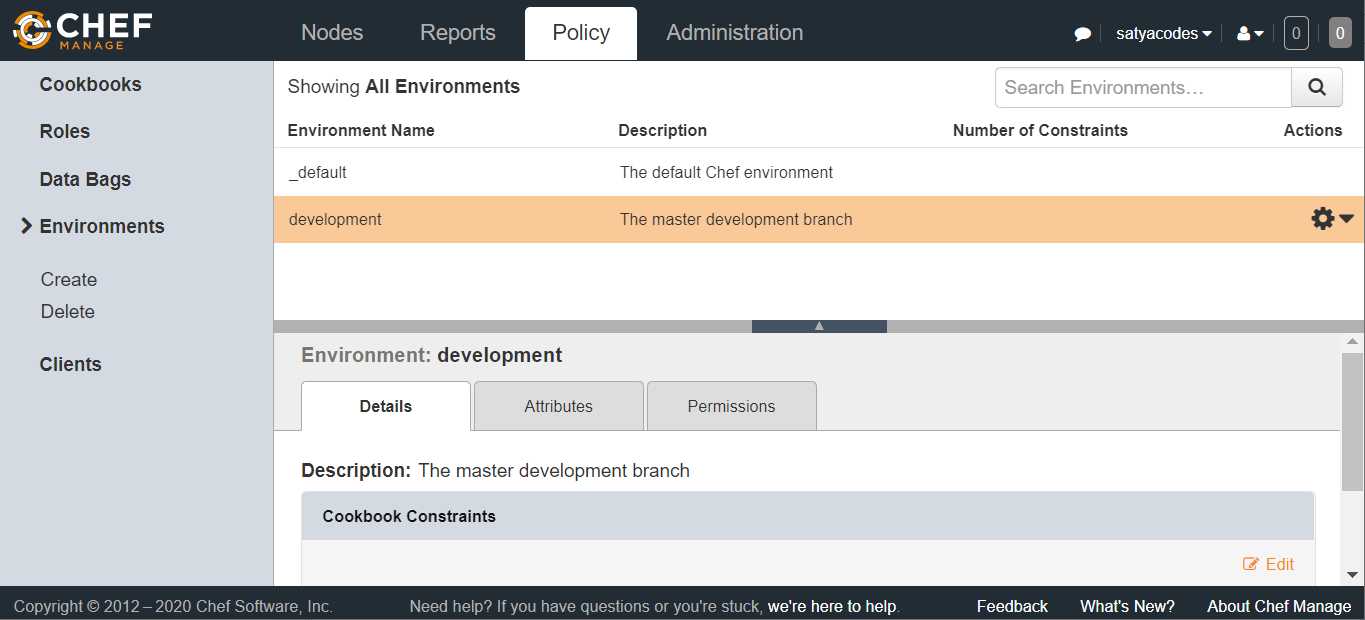
For our **JSON file,** we can get the environment file off of the server by typing something like:

knife environment show development -Fjson > ~/chef-repo/environments/development.json

Now check all the environments Configured in Chef Server

knife environment list



You can also check in Chef Server 

#### 3.Setting Environments in Nodes

Each node can be in exactly one environment. we need to edit its node environment information by using node edit <node>.

knife node edit <Node1>

This will open up a JSON formatted file with the current node parameters:

{

{

"name": "Node1",

"chef\_environment": "\_default",

"normal": {

"tags": [

]

},

"policy\_name": null,

"policy\_group": null,

"run\_list": [

"role[webserver]"

]

}

As you can see, the chef\_environment is set to \_default originally. We can simply modify that value to put the node into a new environment.

"chef\_environment": "development"

#### 4.Run chef-client

run chef-client on the node, it will pick up the new attributes and version constraints and modify itself to align with the new policy

# Chef – Data bags

A data bag is a global variable that is stored as JSON data and is accessible from a Chef server.Using Databags, we can secure/encrypt sensitive data such as passwords, keys, etc.

A data bag is a container for items that represent information about your infrastructure that is not tied to a single node. Data bags contain information that needs to be shared among more than one node. For example:

* Shared passwords
* License keys for software installs
* Shared lists of users and groups

Elements of Chef Databags

* DATA BAG
* DATA BAG ITEM
* DATA BAG ITEM ELEMENTS

We would need to create a data-bag to encrypt the sensitive data.

knife data bag create <my\_databag> <my\_databag\_item>

This would create a directory named “my\_databag” which is our data-bag and this directory has created a file named “my\_databag\_item.json” which is a data-bag item.

we have “plain-text” content to encrypt:

{

"id": "my\_databag",

"password1": “abc123”,

"password2": “123abc”

}

And after encryption this file would look like:

{

"id": "my\_databag",

"password1": {

"encrypted\_data": "XXXXXXXXXXXXXXXXXXXXXXXXX",

"iv": "XYXYXYXYXYYXYXYXYXYXYXY",

"version": 1,

"cipher": "aes-256-cbc"

},

"password2": {

"encrypted\_data": "YYYYYYYYYYYYYYYYYYYYYYYYYY",

"iv": "YXYXYXYXYXYXYXYXYXYXYXYX",

"version": 1,

"cipher": "aes-256-cbc"

}

}

## Data Bags Example

In this example we create a data bag, which will hold the Employees endpoint URL details(<http://dummy.restapiexample.com/>) and use it in our recipe.

Create data bag directory inside /chef-repo/.

mkdir data\_bags/employees

Create a data bag item for storing URL

vi data\_bags/employees/get.json {

"id": "get\_url",

"url": "http://dummy.restapiexample.com/api/v1/employees"

}

Create a data bag on the Chef server

knife data bag create employees

Created data\_bag[employees]

Upload the data bag to the Chef server

knife data bag from file employees get.json

Updated data\_bag\_item[employees::get\_url]

Update the default recipe of the cookbook to receive the required cookbook from a data bag.

vi cookbooks/learn\_chef\_apache2/recipes/default.rb

package 'apache2'

service 'apache2' do

supports :status => true

action [:enable, :start]

end

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

source 'index.html.erb'

end

employees = data\_bag\_item('employees', 'get\_url')

http\_request 'callback' do

url employees['get\_url']

end

Upload the modified cookbook to the Chef server.

knife cookbook upload learn\_chef\_apache2

Uploading learn\_chef\_apache2 [0.3.0]

Uploaded 1 cookbook.

Run the Chef client on the node to check if the http request bin gets executed.

sudo chef-client

root@web1:/home/vagrant# sudo chef-client

Starting Chef Client, version 11.8.2

resolving cookbooks for run list: ["learn\_chef\_apache2"]

Synchronizing Cookbooks:

- learn\_chef\_apache2

Compiling Cookbooks...

Converging 4 resources

Recipe: learn\_chef\_apache2::default

\* package[apache2] action install (up to date)

\* service[apache2] action enable

- enable service service[apache2]

\* service[apache2] action start (up to date)

\* template[/var/www/html/index.html] action create (up to date)

\* http\_request[callback] action get

- http\_request[callback] GET to http://dummy.restapiexample.com/api/v1/employees

Chef Client finished, 2 resources updated

# Chef – Provisioning

Chef Provisioning helps you to use the power of Chef to create your whole infrastructure for you.

No matter whether you want to create a cluster of Vagrant boxes, [Docker](https://subscription.packtpub.com/tech/docker) instances, or Cloud servers, Chef Provisioning lets you define your infrastructure in a simple recipe and run it idempotently.

## how “to create a Vagrant machine and install nginx “on it

Chef Provisioning comes with a selection of drivers for all kinds of infrastructures, including Fog (supporting Amazon EC2, [OpenStack](https://subscription.packtpub.com/search?released=Available&tool=OpenStack), and others), VMware VSphere, Vagrant (supporting Virtualbox and VMware Fusion), various Containers, such as LXC [Docker](https://subscription.packtpub.com/tech/docker) “and Secure Shell (SSH).

In this recipe, we make sure that we can use the directives provided by Chef Provisioning by requiring *chef/provisioning* library.

Then, we configure the driver that we want to use. We use Vagrant and tell Chef to use the *opscode-ubuntu-14.04* Vagrant box to spin up our machine.

Using the machine resource, we ask Chef to spin up a Vagrant machine and configure it using Chef by applying the role *web\_server*.

The *web\_server* role uses the cookbook *my\_cookbook* to configure the newly created Vagrant machine. To make sure that all the required cookbooks are available to Chef, we use *berks install* and *berks vendor cookbooks*. The *berks vendor cookbooks* installs all the required cookbooks in the local cookbooks directory. The Chef client can access the cookbooks here, without the need for a Chef server.

Finally, we use the Chef client to execute our Chef Provisioning recipe. It will spin up the defined Vagrant machine and execute a Chef client run on it.

Chef Provisioning will put the Vagrant Virtual Machine (VM) definition into the directory *~/.chef/vms*. To manage the Vagrant VM, you need to change to this directory.

**Describe your Vagrant machine in a recipe called mycluster.rb:**

mma@laptop:~/chef-repo $ subl mycluster.rb

require 'chef/provisioning'

with\_driver 'vagrant'

with\_machine\_options :vagrant\_options => { 'vm.box' => 'opscode-ubuntu-14.04' }

machine 'web01' do

role 'web\_server'

end

**Install all required cookbooks in your local chef-repo:**

mma@laptop:~/chef-repo $ berks install

mma@laptop:~/chef-repo $ berks vendor cookbooks

Resolving cookbook dependencies...

Using apt (2.6.1)

...TRUNCATED OUTPUT...

Vendoring yum-epel (0.6.0) to cookbooks/yum-epel

Run the Chef client in local mode to bring up the Vagrant machine and execute a Chef run on it:

mma@laptop:~/chef-repo $ chef-client -z mycluster.rb

[2015-03-08T21:09:39+01:00] INFO: Starting chef-zero on host localhost, port 8889 with repository at repository at /Users/mma/work/chef-repo

...TRUNCATED OUTPUT...

Recipe: @recipe\_files::/Users/mma/work/chef-repo/mycluster.rb

\* machine[webserver] action converge[2015-03-08T21:09:43+01:00] INFO: Processing machine[web01] action converge (@recipe\_files::/Users/mma/work/chef-repo/mycluster.rb line 6)

...TRUNCATED OUTPUT...

[2015-03-08T21:09:47+01:00] INFO: Executing sudo chef-client -l info on vagrant@127.0.0.1

[web01] [2015-03-08T20:09:21+00:00] INFO: Forking chef instance to converge...

Starting Chef Client, version 12.1.0

...TRUNCATED OUTPUT...

Chef Client finished, 18/25 resources updated in 73.839065458 seconds

...TRUNCATED OUTPUT...

[2015-03-08T21:11:05+01:00] INFO: Completed chef-client -l info on vagrant@127.0.0.1: exit status 0

- run 'chef-client -l info' on web01

[2015-03-08T21:11:05+01:00] INFO: Chef Run complete in 82.948293 seconds

...TRUNCATED OUTPUT...

Chef Client finished, 1/1 resources updated in 85.914979 seconds

Change” into the directory where Chef put the Vagrant configuration:

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

Validate that there is a Vagrant machine named web01 running:

mma@laptop:~/.chef/vms $ vagrant status

Current machine states:

web01 running (virtualbox)

Validate that nginx is installed and running on the Vagrant machine:

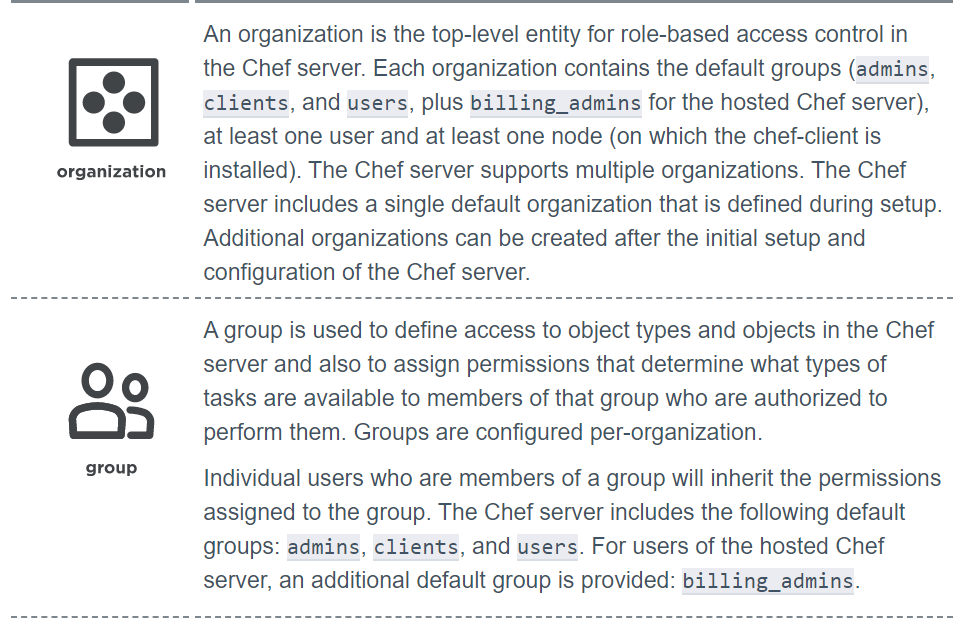
mma@laptop:~/.chef/vms $ vagrant ssh

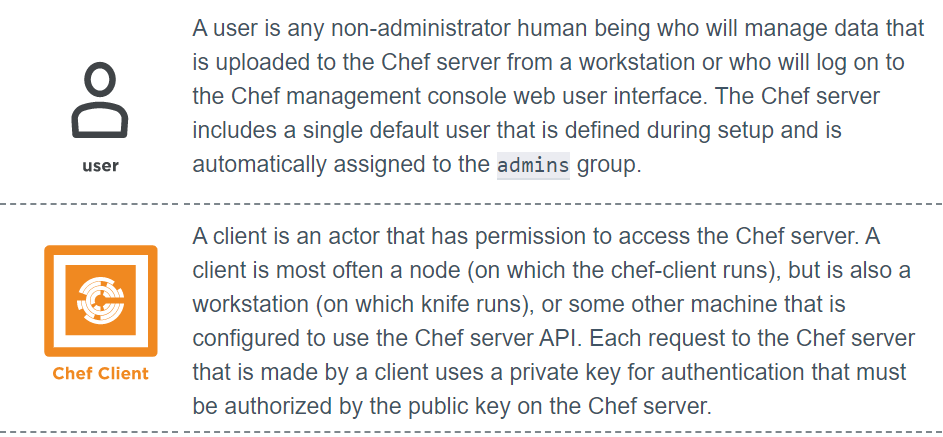
vagrant@web01:~$ wget localhost:80

...TRUNCATED OUTPUT...

2015-03-08 22:14:45 (2.80 MB/s) - 'index.html' saved [21/21]

# Chef - Organizations





# Chef - Policies

Policy maps business and operational requirements, process, and workflow to settings and objects stored on the Chef Infra Server:

* Roles define server types, such as “web server” or “database server”
* Environments define process, such as “dev”, “staging”, or “production”
* Certain types of data—passwords, user account data, and other sensitive items—can be placed in data bags, which are located in a secure sub-area on the Chef Infra Server that can only be accessed by nodes that authenticate to the Chef Infra Server with the correct SSL certificates
* The cookbooks (and cookbook versions) in which organization-specific configuration policies are maintained

#### Cookbook Versions

A cookbook version represents a set of functionalities that is different from the cookbook on which it is based. A version may exist for many reasons, such as ensuring the correct use of a third-party component, updating a bug fix, or adding an improvement. A cookbook version is defined using syntax and operators, may be associated with environments, cookbook metadata, and/or run-lists, and may be frozen (to prevent unwanted updates from being made).

A cookbook version is maintained just like a cookbook, with regard to source control, uploading it to the Chef Infra Server, and how Chef Infra Client applies that cookbook when configuring nodes.

#### Data Bags (Secrets)

Data bags store global variables as JSON data. Data bags are indexed for searching and can be loaded by a cookbook or accessed during a search.

#### Environments

An environment is a way to map an organization’s real-life workflow to what can be configured and managed when using Chef Infra. This mapping is accomplished by setting attributes and pinning cookbooks at the environment level. With environments, you can change cookbook configurations depending on the system’s designation. For example, by designating different staging and production environments, you can then define the correct URL of a database server for each environment. Environments also allow organizations to move new cookbook releases from staging to production with confidence by stepping releases through testing environments before entering production.

#### Roles

A role is a way to define certain patterns and processes that exist across nodes in an organization as belonging to a single job function. Each role consists of zero (or more) attributes and a run-list. Each node can have zero (or more) roles assigned to it. When a role is run against a node, the configuration details of that node are compared against the attributes of the role, and then the contents of that role’s run-list are applied to the node’s configuration details. When a Chef Infra Client runs, it merges its own attributes and run-lists with those contained within each assigned role.

#### Policyfile

A Policyfile is an optional way to manage role, environment, and community cookbook data with a single document that is uploaded to the Chef Infra Server. The file is associated with a group of nodes, cookbooks, and settings. When these nodes perform a Chef Infra Client run, they utilize recipes specified in the Policyfile run-list

A Policyfile file allows you to specify in a single document the cookbook revisions and recipes that Chef Infra Client will apply. A Policyfile file is uploaded to the Chef Infra Server, where it is associated with a group of nodes. When these nodes are configured during a Chef Infra Client run, Chef Infra Client will make decisions based on your Policyfile settings and will build a run-list based on that information. A Policyfile file may be versioned, and then promoted through deployment stages to safely and reliably deploy new configuration.

**Syntax**[**¶**](https://docs.chef.io/policyfile.html#syntax)

A Policyfile.rb is a Ruby file in which run-list and cookbook locations are specified. The syntax is as follows:

name "name"

run\_list "ITEM", "ITEM", ...

default\_source :SOURCE\_TYPE, \*args

cookbook "NAME" [, "VERSION\_CONSTRAINT"] [, SOURCE\_OPTIONS]

# Chef – Solo

**Chef-solo** behaves as a standalone system, it is in itself a client & a server.

# Errors

#### Net::HTTPServerException: 412 "Precondition Failed" :No such cookbook: apache\_cookbook

May be we need to create cookbooks under cookbooks folder.

#### ERROR: RuntimeError: Please set EDITOR environment variable.

You have to define EDITOR as environment variable.

Try

export EDITOR=$(which vi)

To persist this, add it to ~/.bashrc or ~/.bash\_profile.

#### ERROR: Cannot find a cookbook named learn\_chef\_apache2; did you forget to add metadata to a cookbook?

Make sure you copy cookbook to chef-repo/cookbooks folder

#### Vagrant : default: Warning: Connection aborted. Retrying..

inserting ' config.ssh.insert\_key = false' which resolved my issue.

<https://www.thisprogrammingthing.com/2016/fixing-vagrant-connection-error/>

#### ERROR: Net::SSH::HostKeyMismatch: fingerprint … does not match for “…”

If the known\_hosts file already contains an entry for a different server with the same IP address, we get the error message

ERROR: Net::SSH::HostKeyMismatch: fingerprint … does not match for “…”

Open the **~/.ssh/known\_hosts** file and delete the line that contains the IP address of the server.

#### [shorten file path in Any terminal](https://askubuntu.com/questions/302667/shorten-file-path-in-terminal)

**PS1='\W> '**

**Or**

PS1='\u@\h:\W:\$'

PS1='\u@\h:\w:\$'

# Ref.

Installation.

<https://docs.chef.io/install_workstation.html>

Best

<https://www.youtube.com/watch?v=4aaK2rzzOLc&list=PLsgnv1SN76ILtD3TnVtXpX1hmwjyY9OuT&index=1>

<https://www.youtube.com/watch?v=LTIjUJEehDA>

part1

<https://youtu.be/LTIjUJEehDA?list=PL9ooVrP1hQOFDz4cy_X-oeDIrpfkqIIZw>

Best:<https://www.digitalocean.com/community/tags/chef?subtype=tutorial_series>