“ Dynamic Load Balancing of Internet Traffic Between Apache Servers Developed and Deployed Using Chef ”

CSE4011 - VIRTUALIZATION

PROJECT REVIEW III

Submitted By

Amrit Krishna O **(17BCE1128)**

Rohit Subramanian **(17BCE1291)**

Under the Guidance of

Dr. Shyamala L

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

INTRODUCTION

Chef is a configuration management tool for dealing with machine setup on physical servers, virtual machines and in the cloud. Many companies use Chef software to control and manage their infrastructure including Facebook, Etsy, Cheezburger, and Indiegogo.

Configuration management is all about trying to ensure that the files and software you are expecting to be on a machine are present, configured correctly, and working as intended.

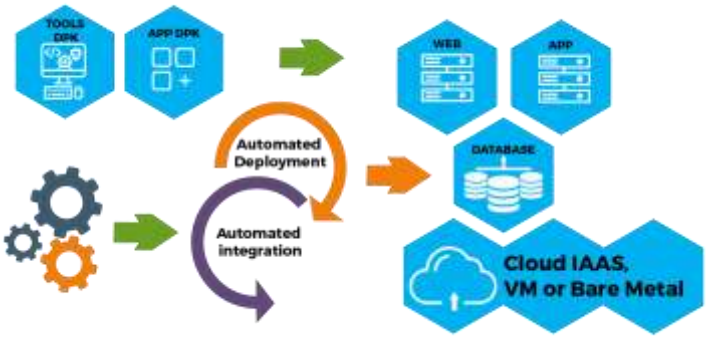
When you have only a single machine this is fairly simple. When you have five or ten servers, it is still possible to do this manually, but it may take all day. However, when your infrastructure scales up into the thousands we need a better way of doing things.

Chef helps solve this problem by treating infrastructure as code. Rather than manually changing anything, the machine setup is described in a Chef recipe.

Collections of recipes are stored in a cookbook. One cookbook should relate to a single task, but can have a number of different server configurations involved (for example a web application with a database, will have two recipes, one for each part, stored together in a cookbook).

There is a Chef server which stores each of these cookbooks and as a new chef client node checks in with the server, recipes are sent to tell the node how to configure itself.

The client will then check in every now and again to make sure that no changes have occurred, and nothing needs to change. If it does, then the client deals with it. Patches and updates can be rolled out over your entire infrastructure by changing the recipe. No need to interact with each machine individually.



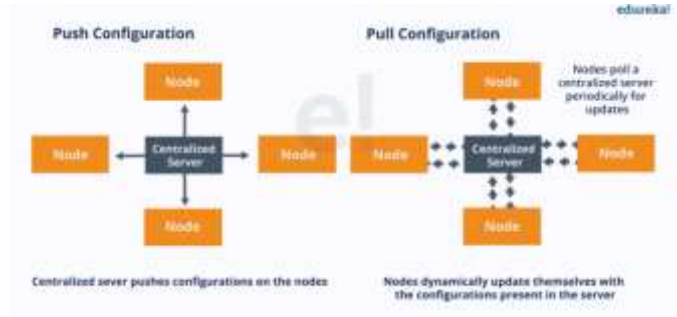
Recipes and cookbooks are the heart of the configuration management. They are written using the Ruby programming language, however, the domain specific language used by Chef is designed to be able to be understood by everyone. As the configuration is just code it can be tested and it can be version controlled. This means that there is less downtime, more reliable services and less stressed people on both the dev and ops sides.

Chef allows you to dynamically provision and de-provision your infrastructure on demand to keep up with peaks in usage and traffic. It enables new services and features to be deployed and updated more frequently, with little risk of downtime. With Chef, you can take advantage of all the flexibility and cost savings that cloud offers.

Fig:Chef Configuration

There are broadly two ways to manage your configurations namely Push and Pull configurations.

- **Pull Configuration:** In this type of Configuration Management, the nodes poll a centralized server periodically for updates. These nodes are dynamically configured so basically they are pulling configurations from the centralized server. Pull configuration is used by tools like Chef, Puppet etc.
- **Push Configuration:** In this type of Configuration Management, the centralized Server pushes the configurations to the nodes. Unlike Pull Configuration, there are certain commands that have to be executed in the centralized server in order to configure the nodes. Push Configuration is used by tools like Ansible.



Chef works with three core components, the Chef server, workstations, and nodes:

- **Chef server:** As the center of operations, the Chef server stores, manages, and provides configuration data to all other Chef components.
- **Workstations:** Workstations are personal computers or virtual servers where all configuration code is created, tested, and changed. There can be as many workstations as needed, whether this be one per person or otherwise.
- **Nodes:** Nodes are the servers that are managed by Chef – these are the machines that changes are being pushed to, generally a fleet of multiple machines that require the benefits of automation. Chef can manage nodes that are virtual servers, containers, network devices, and storage devices. A Chef client is installed on every node that is under management by Chef.

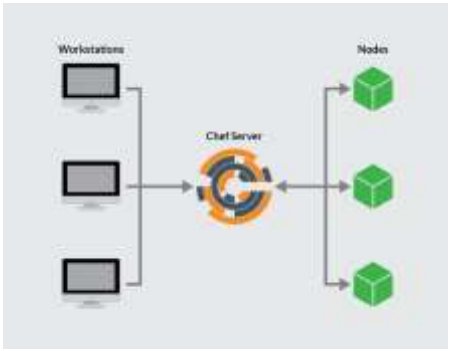
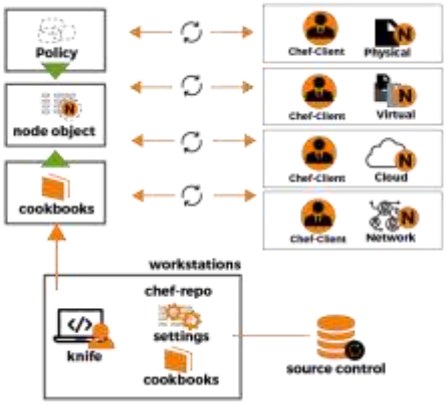
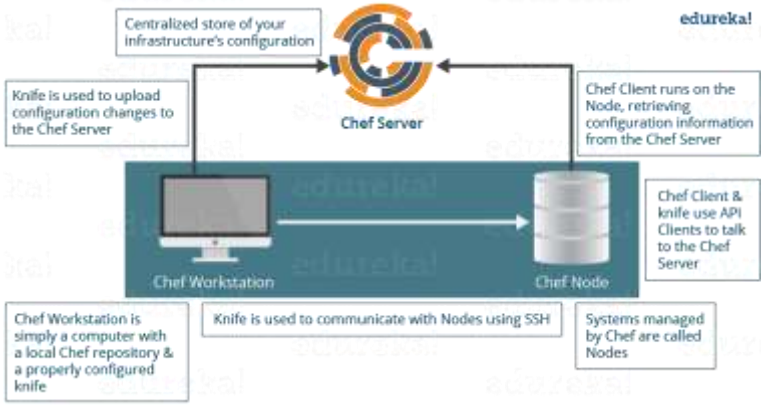


Fig: Chef Components

These three components communicate in a mostly linear fashion, with any changes being pushed from workstations to the Chef server, and then pulled from the server to the nodes and implemented on each node via their Chef client. In turn, information about the node passes to the server to determine which files are different from the current settings and need to be updated.



The user can interact with chef and chef server through Chef Workstation. Knife and Chef command line tools are used for interacting with Chef Server. Chef node is a virtual or a cloud machine managed by chef and each node is configured by Chef-Client installed on it. Chef server stores all part of the configuration. It ensures all the elements are in right place and are working as expected.



Now, let us look at reasons behind the popularity of Chef.

- Chef supports multiple platforms like AIX, RHEL/CentOS, FreeBSD, OS X, Solaris, Microsoft Windows and Ubuntu. Additional client platforms include Arch Linux, Debian and Fedora.
- Chef can be integrated with cloud-based platforms such as Internap, Amazon EC2, Google Cloud Platform, OpenStack, SoftLayer, Microsoft Azure and Rackspace to automatically provision and configure new machines.
- Chef has an active, smart and fast growing community support.
- Because of Chef's maturity and flexibility, it is being used by giants like Mozilla, Expedia, Facebook, HP Public Cloud, Prezi, Xero, Ancestry.com, Rackspace, Get Satisfaction, IGN, Marshall University, Socrata, University of Minnesota, Wharton School of the University of Pennsylvania, Bonobos, Splunk, Citi, DueDil, Disney, and Cheezburger.

Chef offers the following advantages −

- **Lower barrier for entry** − As Chef uses native Ruby language for configuration, a standard configuration language it can be easily picked up by anyone having some development experience.
- **Excellent integration with cloud** − Using the knife utility, it can be easily integrated with any of the cloud technologies. It is the best tool for an organization that wishes to distribute its infrastructure on multi-cloud environment.

Some of the major drawbacks of Chef are as follows −

- One of the huge disadvantages of Chef is the way cookbooks are controlled. It needs constant babying so that people who are working should not mess up with others cookbooks.
- Only Chef solo is available.
- In the current situation, it is only a good fit for AWS cloud.
- It is not very easy to learn if the person is not familiar with Ruby.
- Documentation is still lacking.

Key Building Blocks of Chef

Recipe

It can be defined as a collection of attributes which are used to manage the infrastructure. These attributes which are present in the recipe are used to change the existing state or setting a particular infrastructure node. They are loaded during Chef client run and comparted with the existing attribute of the node (machine). It then gets to the status which is defined in the node resource of the recipe. It is the main workhorse of the cookbook.

Cookbook

A cookbook is a collection of recipes. They are the basic building blocks which get uploaded to Chef server. When Chef run takes place, it ensures that the recipes present inside it gets a given infrastructure to the desired state as listed in the recipe.

Resource

It is the basic component of a recipe used to manage the infrastructure with different kind of states. There can be multiple resources in a recipe, which will help in configuring and managing the infrastructure. For example −

- **package** − Manages the packages on a node
- **service** − Manages the services on a node
- **user** − Manages the users on the node
- **group** − Manages groups
- **template** − Manages the files with embedded Ruby template
- **cookbook_file** − Transfers the files from the files subdirectory in the cookbook to a location on the node
- **file** − Manages the contents of a file on the node
- **directory** − Manages the directories on the node
- **execute** − Executes a command on the node
- **cron** − Edits an existing cron file on the node

Attribute

They are basically settings. They can be thought of as a key value pair of anything which one wants to use in the cookbook. There are several different kinds of attributes that can be applied, with a different level of precedence over the final settings that the node operates under.

File

It's a subdirectory within the cookbook that contains any static file which will be placed on the nodes that uses the cookbooks. A recipe then can be declared as a resource that moves the files from that directory to the final node.

Templates

They are similar to files, but they are not static. Template files end with the .ebr extension, which means they contain embedded Ruby. They are mainly used to substitute an attribute value into the files to create the final file version that will be placed on the node.

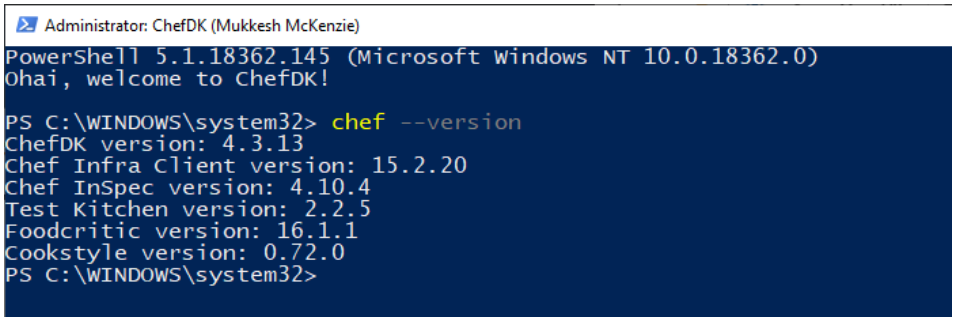
Metadata.rb

It is used to manage the metadata about the package. This includes details like the name and details of the package. It also includes things such as dependency information that tells which cookbooks this cookbook needs to operate. This allows the Chef server to build the run-list of the node correctly and ensures that all of the pieces are transferred correctly.

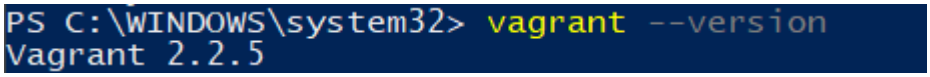
IMPLEMENTATION

Chef Installation

1. Go to https://downloads.chef.io/chef-dk/ and download the ChefDK for windows.
2. Just proceed to install the software.
3. To test it, type “chef –version” in the chefDK PowerShell



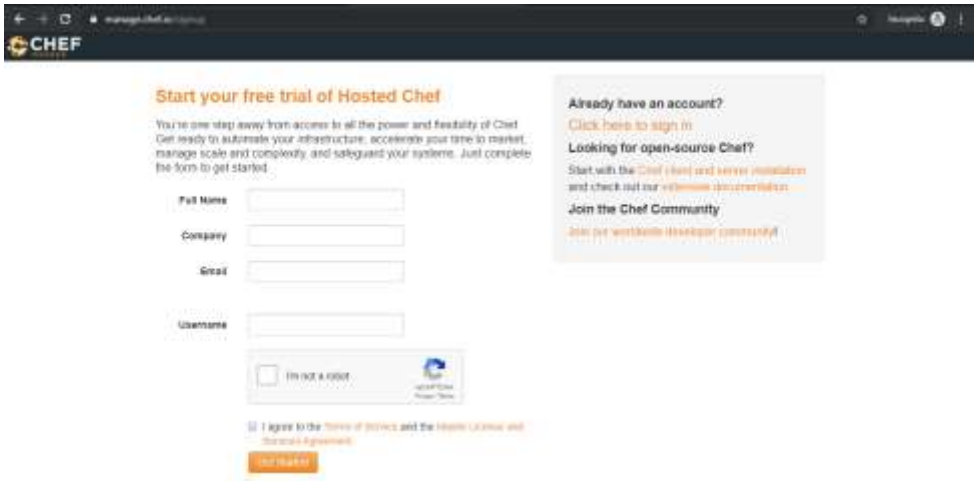
Also, download and install both Vagrant and VirtualBox.



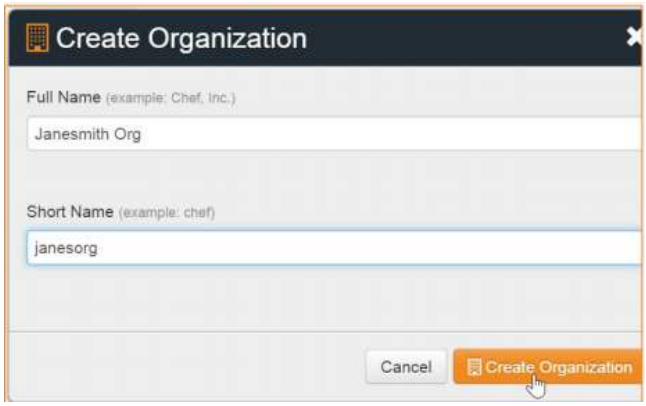
We will be using Vagrant and Chef Server to create 3 VMs (1 Load Balancer and 2 Web Servers).

Creating Chef – Server:

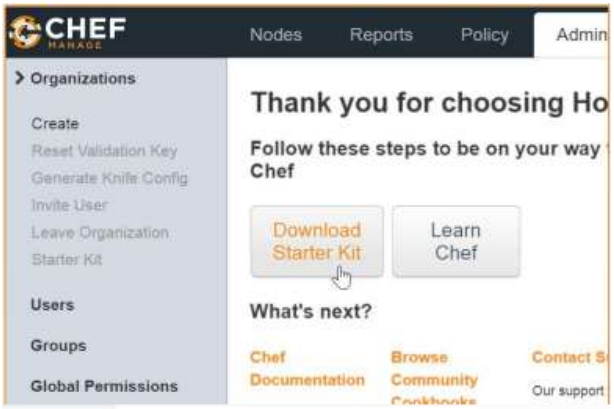
You can go to manage.chef.io/signup to create your own hosted chef server.



Create your own unique organization



Then proceed to download the Starter Kit



Unzip the contents into a chef-repo folder.

Why Chef Server?

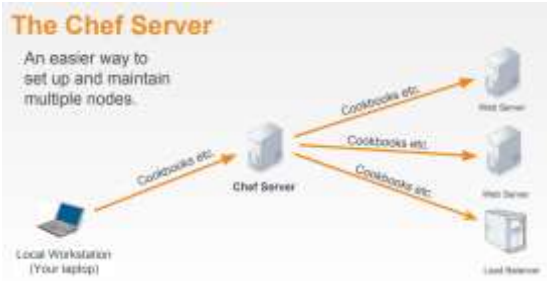
To manage another system, you would need to:

1. Provision a new node within your company or appropriate cloud provider with the appropriate access to login to administrate the system.

2. Install the Chef tools.

3. Transfer the apache cookbook.

4. Run chef-client on the new node to apply the apache cookbook's default recipe.



After this, use the following script to create and configure 3 VMs (1 load balancer and 2 web servers)

```
# -*- mode: ruby -*-

# vi: set ft=ruby :
```

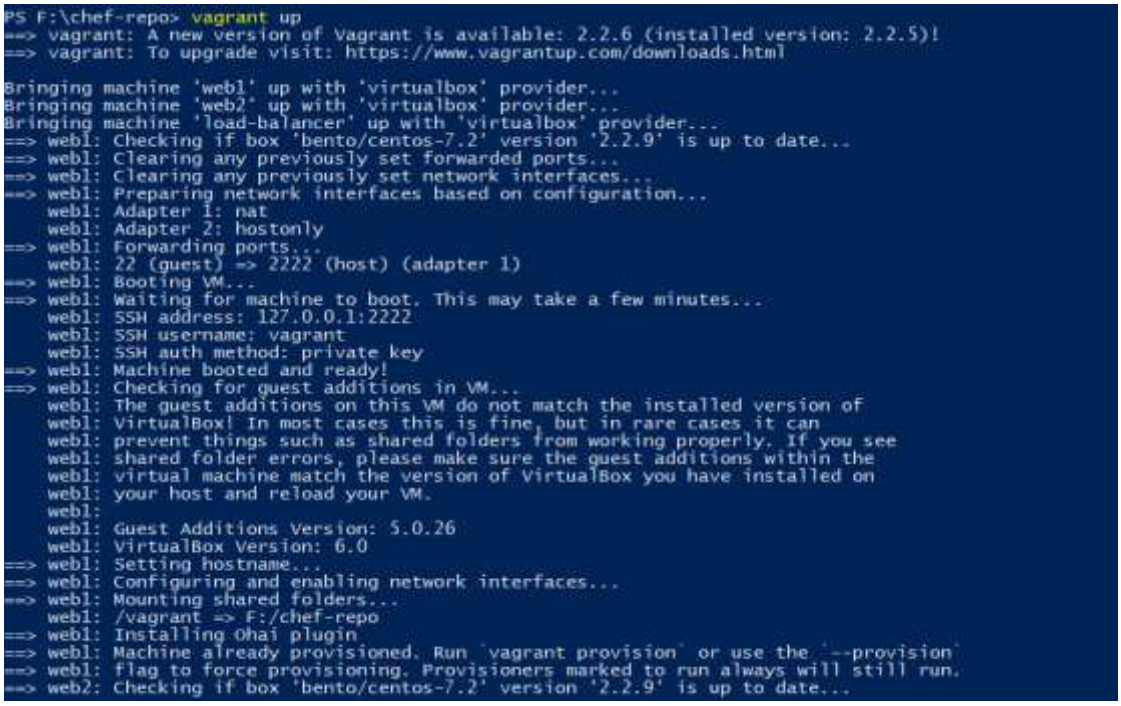
```
unless Vagrant.has_plugin?("vagrant-ohai")
  raise "vagrant-ohai plugin is not installed! Install with 'vagrant plugin install vagrant-ohai'"
end

NODE_SCRIPT = <<EOF.freeze
  echo "Preparing node..."
  # ensure the time is up to date
  yum -y install ntp
  systemctl start ntpd
  systemctl enable ntpd
  curl -L https://omnitruck.chef.io/install.sh | sudo bash -s -- -v 14
EOF

def set_hostname(server)
  server.vm.provision 'shell', inline: "hostname #{server.vm.hostname}"
end

Vagrant.configure(2) do |config|

 config.vm.define 'web1' do |n|
    n.vm.box = 'bento/centos-7.2'
    n.vm.box_version = '2.2.9'
    n.vm.hostname = 'web1'
    n.vm.network :private_network, ip: '192.168.10.43', nic_type: "virtio"
    n.vm.provision :shell, inline: NODE_SCRIPT.dup
   set_hostname(n)
  end

 config.vm.define 'web2' do |n|
    n.vm.box = 'bento/centos-7.2'
    n.vm.box_version = '2.2.9'
```

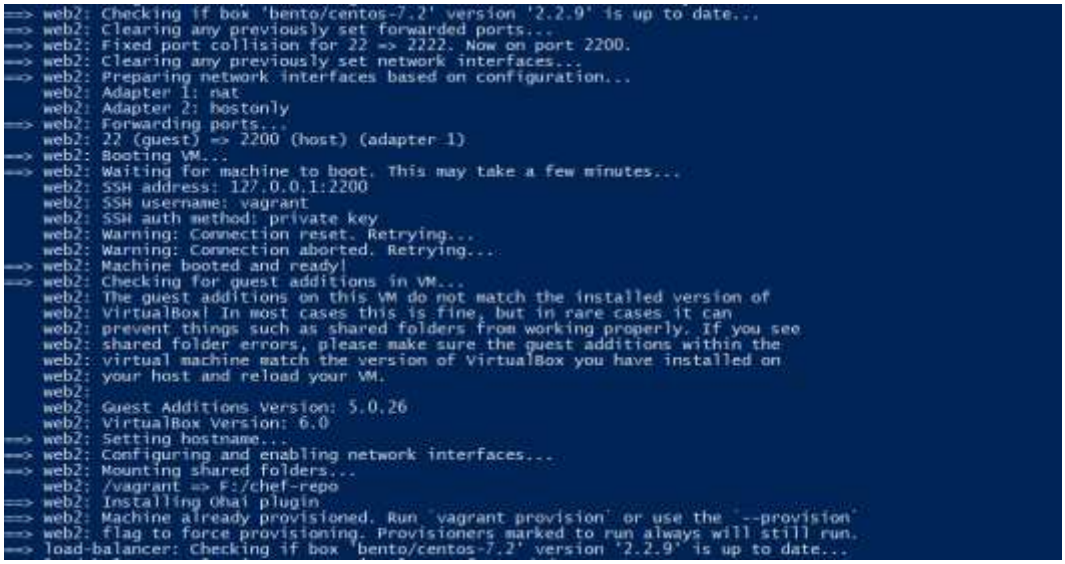
```
    n.vm.hostname = 'web2'
    n.vm.network :private_network, ip: '192.168.10.44', nic_type: "virtio"
    n.vm.provision :shell, inline: NODE_SCRIPT.dup
   set_hostname(n)
  end
 config.vm.define 'load-balancer' do |n|
    n.vm.box = 'bento/centos-7.2'
    n.vm.box_version = '2.2.9'
    n.vm.hostname = 'load-balancer'
    n.vm.network "forwarded_port", guest: 80, host: 9000
    n.vm.provision :shell, inline: NODE_SCRIPT.dup
   set_hostname(n)
  end
end
```

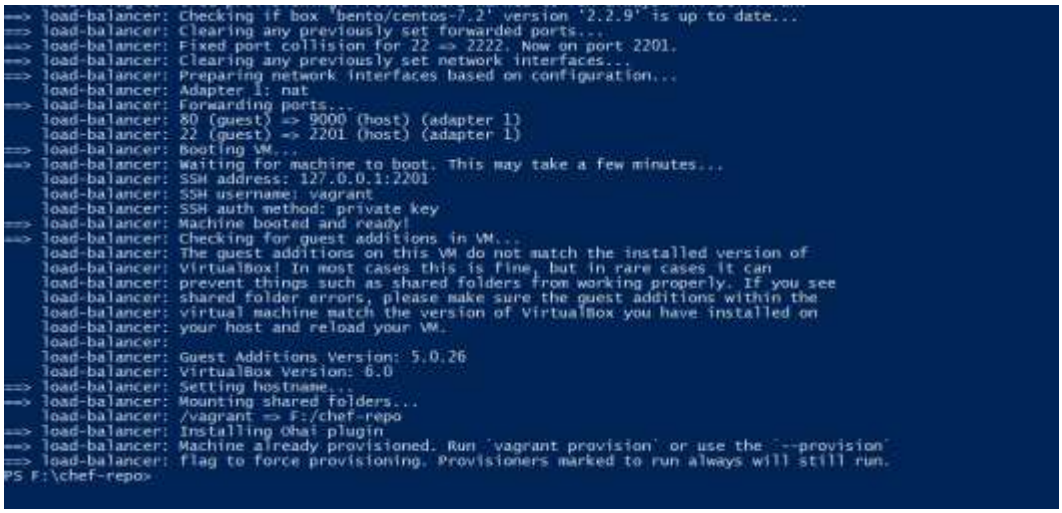
Commands:

vagrant init

vagrant up







Creating the CookBooks:

Use the command ‘chef generate cookbook “mycookbookname” ‘ to create your cookbook.

Apache CookBook:

The Apache Cookbook will set up and instantiate the Apache Webserver. The HTML page that it has to serve will be dynamically generated using ohai environment variables from an ERB file.

Server Recipe:

```
package 'httpd'

template '/var/www/html/index.html' do
  source 'index.html.erb'
end

service 'httpd' do
  action [:enable, :start]
end
```

Default Recipe:

```
include_recipe 'apache::server'
```

Template file:

```
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: <%= node["ipaddress"] %></h2>
    <h2>hostname: <%= node["hostname"] %></h2>
</body>
</html>
```

Using the OHAI node attributes, the html webpage is dynamically generated on each node.

Workstation Cookbook:

This cookbook contains the recipes for the default workstations (both webserver and loadbalancer).

Vagrant Recipe:

This contains the vagrant recipe, which will match the IP Address to the one defined by the vagrant file.

```
# the vagrant-ohai plugin modifes the node['ipaddress'] attribute gathered by ohai
# to match what we define in the Vagrant file. Otherwise ohai will detect the internal
# ipaddress assigned by vagrant. This allows our load balancer to work with the
# virtual instances we manage later in the class.
#
# @see https://github.com/avishai-ish-shalom/vagrant-ohai
# @see https://docs.chef.io/ohai_custom.html
#

# execute sudo /etc/init.d/network restart to reload the networking interfaces
# this problem seems confined to Vagrant 1.9.1
# see vagrant issue https://github.com/mitchellh/vagrant/issues/8115

execute 'reload network interfaces' do
  command 'sudo /etc/init.d/network restart'
  not_if "ifconfig | grep 'eth0'"
end
```

```
# set plugin path for ohai if not already defined in /etc/chef/client.rb
# this is always set on a subsequent chef-client run because of ruby_block[configure ohai plugin path]
if ::Ohai::Config.ohai.nil?
  ::Ohai::Config['plugin_path'] << '/etc/chef/ohai_plugins' # old format
else
  ::Ohai::Config.ohai['plugin_path'] << '/etc/chef/ohai_plugins' # new format
end

# add vagrant-ohai plugin to the resource collection
#
# @see https://docs.chef.io/resource_ohai.html
#
ohai 'vagrant-ipaddress' do
  plugin 'ipaddress'
  action :nothing
end

# modify the client.rb file to include the plugin path for the vagrant-ohai plugin
# At the beginning of every chef-client run, the client.rb file is loaded.
#
# @see https://docs.chef.io/config_rb_client.html
#
# This ruby_block resource modifies the client.rb to include the '/etc/chef/ohai_plugins'
# directory, with a guard to check if the file has already been modified.
#
# @see https://docs.chef.io/resource_common.html#guards
#
# When the client.rb file is modified, the notification adds 'ohai[vagrant-ohai]' to the
# resource collection, re-running ohai with the new plugin loaded. This ensures we have
# access to the node['ipaddress'] attribute during this chef-client run.
#
```

```
ruby_block "configure ohai plugin path" do
  block do
    file = Chef::Util::FileEdit.new("/etc/chef/client.rb")
    file.insert_line_if_no_match("ohai.plugin_path << '/etc/chef/ohai_plugins'","ohai.plugin_path <<
'/etc/chef/ohai_plugins'")
    file.write_file
  end
  notifies :reload, 'ohai[vagrant-ipaddress]', :immediately
end
```

Setup Recipe:

This cookbook will printout the OHAI stats of the node while booting up (message of the day). It reads them from a template file (ERB).

```
template '/etc/motd' do
  source 'motd.erb'
end
```

Template File:

Property of ...

```
  IPADDRESS: <%= node["ipaddress"] %>
  HOSTNAME : <%= node["hostname"] %>
  MEMORY   : <%= node["memory"]["total"] %>
  CPU      : <%= node["cpu"]["0"]["mhz"] %>
```

Default Recipe:

Used to include both the recipes (calling a recipe from another recipe, and the chef-client runs the default recipe by default).

```
include_recipe 'workstation::vagrant'
include_recipe 'workstation::setup'
```

myhaproxy Cookbook

This cookbook will be run only by the load-balancer. It contains the necessary recipes to set up a load balancing server to direct traffic between the web servers. It is configured in such a way that it can discover new nodes and run dynamically add them to the list.

Default Recipe:

```
# frozen_string_literal: true

apt_update

haproxy_install 'package'

haproxy_frontend 'http-in' do
  bind '*:80'
  default_backend 'servers'
end

all_web_nodes = search('node',"role:web")

#"role:web AND chef_environment:#{node.chef_environment}"

servers = []

all_web_nodes.each do |web_node|
        server = "#{web_node['hostname']} #{web_node['ipaddress']}:80 maxconn 32"
        servers.push(server)
end

haproxy_backend 'servers' do
        server servers
end

haproxy_service 'haproxy' do
        subscribes :reload, 'template[/etc/haproxy/haproxy.cfg]', :immediately
end
```

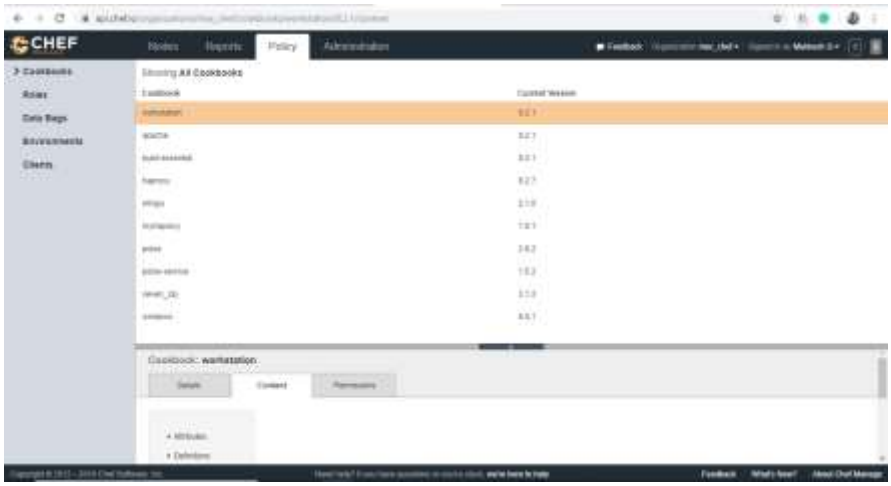
Here, you can see that we are dynamically reading the ohai stats and adding the nodes to the list. This sets up the load balancing VM.

We will now have to upload the cookbooks to the chef server using Knife or Berksfile.

Uploading the cookbooks:

Cd into the required cookbooks and use “berks upload” command to upload the cookbooks to the hosted chef server.

```
PS F:\chef-repo\cookbooks\apache> berks upload
Uploaded apache (0.2.1) to: 'https://api.chef.io/organizations/mex_chef'
PS F:\chef-repo\cookbooks\apache>
```



Once you upload all the cookbooks, you'll see this on your chef server.

Bootstrapping the nodes and adding them to your server:

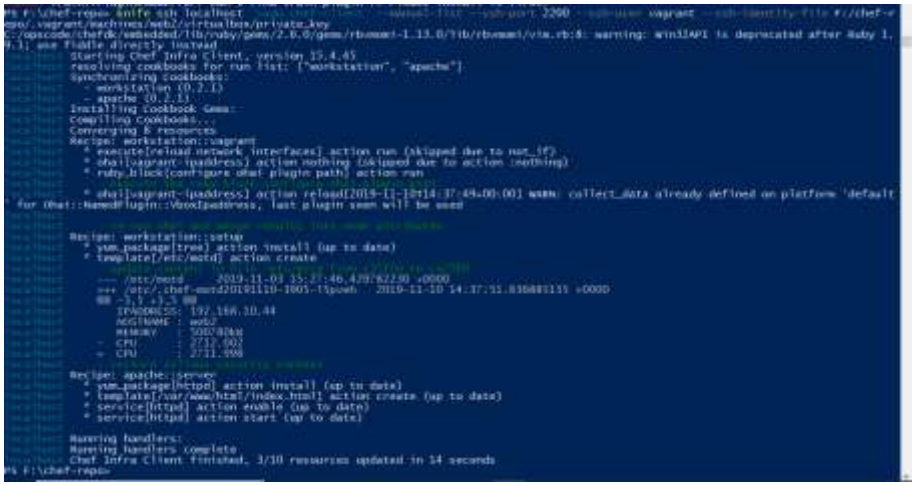
You can use the following command to bootstrap your nodes.

First use vagrant ssh-config to find out the ports and keys for your VMs. And then use the following command to bootstrap them:

knife bootstrap localhost --ssh-port WEB1_PORT --ssh-user vagrant --sudo –ssh-identity-file PATH_TO_KEY -N web1 --run-list "recipe[workstation],recipe[apache]"

knife bootstrap localhost --ssh-port WEB2_PORT --ssh-user vagrant --sudo –ssh-identity-file PATH_TO_KEY -N web2 --run-list "recipe[workstation],recipe[apache]"

knife bootstrap localhost --ssh-port LOADBALANCER_PORT --ssh-user vagrant --sudo –ssh-identity-file PATH_TO_KEY -N load-balancer --run-list "recipe[myhaproxy]"



Do this for all the nodes.

You can then use knife to assign roles and change the environments.

cd ~/chef-repo

knife role --help

knife role list

ls roles/

cat roles/starter.rb

nano roles/web.rb

knife role from file roles/web.rb

knife role list

knife role show web

knife node show web1

knife node run_list set web1 "role[web]"

knife node show web1

knife node run_list set web2 "role[web]"

knife node show web2

vagrant ssh web1

knife node show web1

vagrant ssh web2

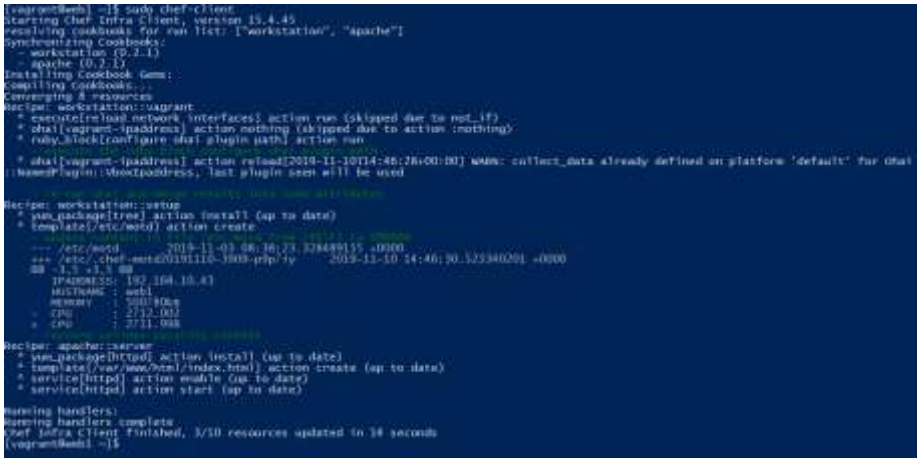
knife node show web2

You can then log into the nodes and run “sudo chef-client” to check whether these works.

Web 1:

```
PS F:\chef-repo> vagrant ssh web1
Last login: Thu Nov  7 03:58:27 2019 from 10.0.2.2
Property of ...

  IPADDRESS: 192.168.10.43
  HOSTNAME : web1
  MEMORY   : 500780kB
  CPU      : 2712.002
```

MOTD displayed.



Chef-Client updates the files.

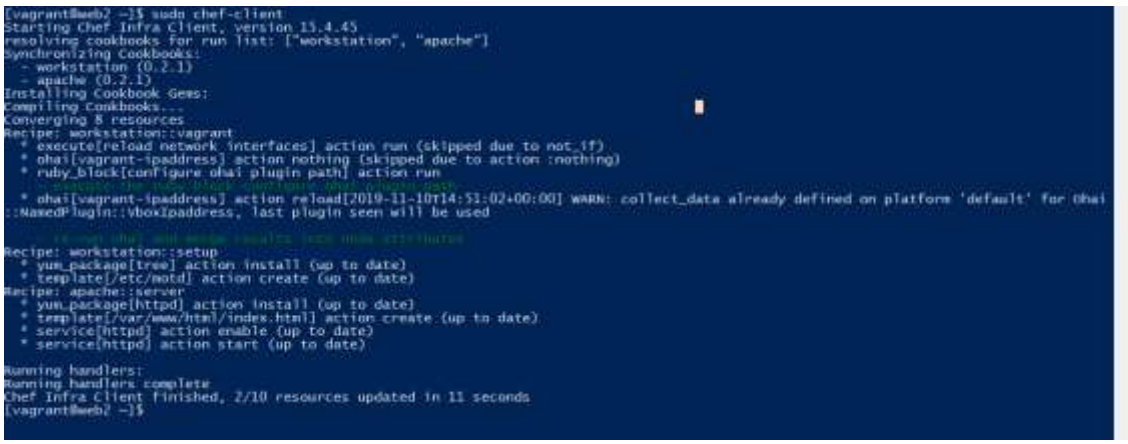
```
[vagrant@web1 ~]$ curl localhost
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: 192.168.10.43</h2>
    <h2>hostname: web1</h2>
</body>
</html>
[vagrant@web1 ~]$ curl localhost
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: 192.168.10.43</h2>
    <h2>hostname: web1</h2>
</body>
</html>
[vagrant@web1 ~]$
```

Curl localhost always returns the same IP address (since it is not running on the load balancer).

Web 1:

```
PS F:\chef-repo> vagrant ssh web2
Last login: Sun Nov 10 14:37:30 2019 from 10.0.2.2
Property of ...

  IPADDRESS: 192.168.10.44
  HOSTNAME  : web2
  MEMORY    : 500780kB
  CPU       : 2711.998
```

MOTD displayed.



Chef-Client updates the files.

```
[vagrant@web2 ~]$ curl localhost
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: 192.168.10.44</h2>
    <h2>hostname: web2</h2>
</body>
</html>
[vagrant@web2 ~]$ curl localhost
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: 192.168.10.44</h2>
    <h2>hostname: web2</h2>
</body>
</html>
[vagrant@web2 ~]$
```

Curl localhost always returns the same IP address (since it is not running on the load balancer).

Load-Balancer:

```
PS F:\chef-repo> vagrant ssh load-balancer
Last login: Sun Nov 10 14:52:54 2019 from 10.0.2.2
[vagrant@load-balancer ~]$
```

No MOTD since the runlist doesn't contain workstation. It only contains myhaproxy.

```
[vagrant@load-balancer ~]$ sudo chef-client
Starting Chef Infra Client, version 15.4.45
resolving cookbooks for run list: ["myhaproxy"]
Synchronizing Cookbooks:
  - haproxy (6.2.7)
  - myhaproxy (1.0.1)
  - build-essential (8.2.1)
  - seven_zip (3.1.2)
  - windows (6.0.1)
  - mingw (2.1.0)
  - poise-service (1.5.2)
  - poise (2.8.2)
Installing Cookbook Gems:
Compiling Cookbooks...
Converging 5 resources
Recipe: myhaproxy::default
  * apt_update[] action periodic (up to date)
  * haproxy_install[package] action create
    * yum_package[haproxy] action install (up to date)
    (up to date)
  * poise_service_user[haproxy] action create
    * group[haproxy] action create (up to date)
    * linux_user[haproxy] action create (up to date)
    (up to date)
  * directory[/etc/haproxy] action create (up to date)
  * template[/etc/haproxy/haproxy.cfg] action nothing (skipped due to action :nothing)
  * haproxy_frontend[http-in] action create (up to date)
  * haproxy_backend[servers] action create (up to date)
  * haproxy_service[haproxy] action create (up to date)
  * cookbook_file[/etc/default/haproxy] action create (up to date)
  * poise_service[haproxy] action nothing (skipped due to action :nothing)
  * template[/etc/haproxy/haproxy.cfg] action nothing (skipped due to action :nothing)
  * template[/etc/haproxy/haproxy.cfg] action create (up to date)
  * poise_service[haproxy] action enable
    * execute[systemctl daemon-reload] action nothing (skipped due to action :nothing)
    * template[/etc/systemd/system/haproxy.service] action create (up to date)
    * service[haproxy] action enable (up to date)
    * service[haproxy] action start (up to date)
    (up to date)

Running handlers:
Running handlers complete
Chef Infra Client finished, 0/20 resources updated in 14 seconds
[vagrant@load-balancer ~]$
```

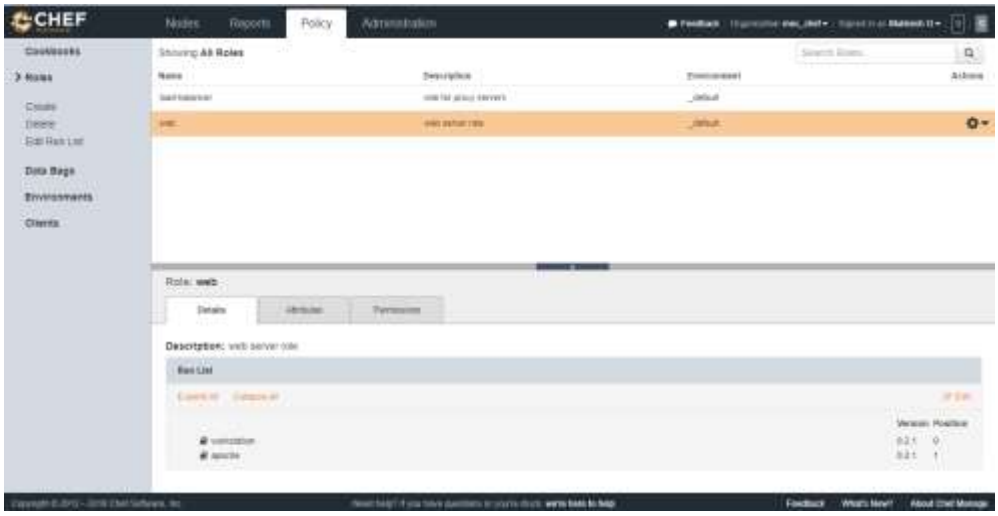
We can confirm this from the runlist of the chef-client.

```
[vagrant@load-balancer ~]$ curl localhost
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: 192.168.10.44</h2>
    <h2>hostname: web2</h2>
</body>
</html>
[vagrant@load-balancer ~]$ curl localhost
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: 192.168.10.43</h2>
    <h2>hostname: web1</h2>
</body>
</html>
[vagrant@load-balancer ~]$ curl localhost
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: 192.168.10.44</h2>
    <h2>hostname: web2</h2>
</body>
</html>
[vagrant@load-balancer ~]$ curl localhost
<html>
  <body>
    <h1>Hello, world!</h1>
    <h2>ipaddress: 192.168.10.43</h2>
    <h2>hostname: web1</h2>
</body>
</html>
[vagrant@load-balancer ~]$
```

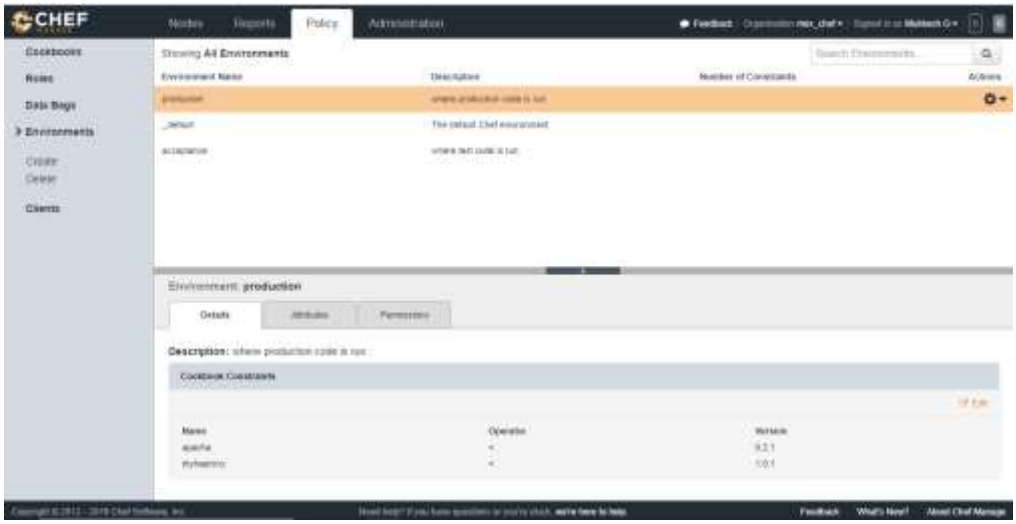
Curl Localhost keeps alternating between different servers as expected.

Nodes on the server:

Roles:



Environments:



Hence, we have successfully set up a load balancer between multiple apache web servers (which can be scaled up) using chef cookbooks, recipes, roles and runlists.