Chef:

How automations tools can work.

Imperative vs declarative

Imperative 🡪 bash scripting

Declarative🡪 ansible

Push vs pull

Understanding chef:

OPscode -🡪 chef

Chef is ruby based tool.

Adam Jacob

Chef has 2 models:

Chef-zero, standalone-🡪 development, poc, testing

Server-client model🡪 production.

Maintain the state of the node.

Application deployments

Maintain infra.

Web server package installation:

Package ‘httpd’ do

Action: install

End

Chef server🡪 manges chef nodes, It stores all cook books.

Chef client-🡪 node-🡪 client pull updates from master node.

Chef architecture:

Chef workstation: local dev planform, create, test, apply and upload chef codes. Desktop.

Chef dk installed on it.

Chef server: node, chef server s/w installed and configured. This machine is accessible.

Chef workstation pushes cookbooks to the chef server.

Chef client:

Chef client communicate with the chef server for the configuration details.

Chef Architecture:

Chef workstation:

Upload cookbooks or recipes in the form of cookbooks to chef server.

Chef workstation is communicating with the chef server is in secure manner, RSA keypair.

Chef ws uses utility called chef knife to communicate with the chef server.

Knife configuration file must be updated with the information file. Keys used for authentication.

Node is any machine, chef client s/w installed on it.

Knife bootstrap

Ohia is used to gather information from the chef node to server.

Chef terminology:

Resources are the built in functions that are executed backed to perform various operations such as user, file and package.

Ex: user, file, package

Recipe: combination multiple diff resouces or operations called recipe.

web-server.rb

cookbook: Multiple recipes together form a cookbook.

File, dir , user, package

Reusable unit🡪 mysql cookbook, docker cookbook.

Run list:

It is used to find the order of the execution.

What recipes to apply and what order to apply.

**Chef resources:**

Users,

Execute scripts,

Firewalls

Package managers

Backups

Built in resources:

Custom resources:

Chef DSL:

Domain specific language

<Resource Type> ‘<NAME>’ do

<Attribute> ‘<value>’

<Attribute> ‘<value>’

<Attribute> ‘<value>’

<Action> :<Value>

End

Example

file /var/www/html/index.html‘do

content “This is contents in file \n”

action : create

end

**Chef Recipe:**

Configuring the web server, db server

Creating custom landing page.

Code creatino

Create .rb file

Check the code with. **Cookstyle**

Perform smoke test.

Chef-client –local-mode –why-run receipe

Run the chef resource

Chef-client –local-mode <receipe-file>

chef

Chef -zero

Check syntax in file

Cookstyle recupe.rb

**Chef server setup :**

It can be install on on-premise or cloud

Chef server should have persistent hostname -🡪 FQDN

Must meet the hardware requirements

The file systems should have /var and /opt minimum of 5 gigs of space.

Firewall settings.

Time Sync

Chef server cannot be installed on the windows os.

Inaccurate time settings b/w chef client and server will get unexpected results.

Make sure all the servers configured with NTP.

Chef -v

Give the verions

Knife -v 🡪 knife version

Its aso. Know an chef infra client.

Establish connection b/w chef ws and server

Rpm -qa | grep -i chef

Setup the chef client.

Check network connectivity. Ping test

Persistent hostname

File system , /opt/chef

/var/chef should have sufficient space for the chef client.

Ensure time sync b/w them.

These machines are. Managed by the chef server.

Yum install chef

Otherway:

Boostrap is used to install chef client on. Remote system.

Knife bootstrap <IP> -N. <hostname

Chef workstation install clinet remotely

Chef architecture:

Chef workstation :

Install Chef DK

Create , test , apply , validate upload chef codes in the form of recipes or cookbooks.

Chef ws communicate with chef server secure manner

Chef workstation use utlity called knife to communicate with the chef server.

Knife is used to push developed chef code to serer .

Chef node:

Node is any machine

Chef client is installed on it.

Server. And client communicate through rsa key pair.

Everytime chef client runs it. Gather info about the node, that utility called OHAI, establish connection between the chef server and client.

Chef server get the copy of the node object. Such as node attrtibute.

Chef server compare with. Old config with. New config .

Chef server run required cookbooks and recipes to node to come in. desired state.

Then current state of the chef server updated to the chef server.

**Chef resources:**

Resources is single unit.

Multiple resources are combine together called recipes.

All recipes are written in .rb,,

Multiple recipes called cookbooks. Collection of dirs., templates and etc.

Webserver.rb. (recipe)

User (resource)

File (resource)

Package

Webserver cookbook

Db cookbook

Runlist:

Run list defines order of execution. You. Have multiple recipes.

That need to be applied to the node.

Chef resources:

Files

Users

Firewall

Packagemanger

Certificates

Dns

And etc

Chef resource is ready made tool.

Chef DSL:

<resource type>. ‘name’ do

<attribute> ‘<value>’

<attribute> ‘<value>’

<action>: <value>

End

Chef developemetn process:

Create , check, test, and run.

Create:

.rb file

Test cookbook:

Linting tool

Cookstyle file.rb

Test:

Perform the smoke test

Return expected result

Chef-client --local-mode –why-run recipe

Run the code:

Chef-client --local-mode <recipe file>

Chef has utility called chef client which is used to run chef codes for both self-contained and server client models.

Chef client is agent run locally on. Every node that is managed by chef.

When chef client is run , all the nodes comes into the desired state.

Chef client attempt contact the chef server and ask for the recipes to run for the given node.

Chef -zero ,

Chef-client --local-mode --why-run receipe

Chef-client --local-mode recipe

Chef-zero quickly testing and validating the behavior of the chef client.

Cookbooks and recipe’s

Run lists before before uploading to the chef server.

The default path for cookbooks directory can be found under the .chef directory located in chef repo path

Cat /home/centos/chef-repo/.chef/knife.rb

You change cook books dir name by using env variable cookbook\_path variable inside knife.rb file.

Chef –help

Chef-generate –help

Generate cookbook template

Chef generate cookbook webapp

It creates cookbook skelton structure

Once sketon is created update default.rb file with

Chef run list:

Order of the delivery

Order of execution.

Install ssh service

Start ssh service

Install httpd

Configure. Httpd

Start httpd

Run list. Specify. What recipes should be run on each node and what order.

In order to execute multiple recipes amend new paramer runlist

Syntax:

Chef-client --runlist “recipe [cookbook-name::Recipe-Name]”

Note: receipe name is without .rb extension.

If you do not specify a receipe name chef picks the default file with the name default.

Chef-client --runlist “recipe[Cookbook-Name]”

Example:

Webserver cookbook

Chef generate cookbook webserver

Cookstyle cookbook\_dir

Generate cookbook

Cook style

Dry run

Run

Chef-client –local-mode --runlist “receipe[webserver::apache]”

**Chef\_runlists:**

Runlist is an order collection of receipe which gets executed on the system.

**Client srever mode:**

Once the code is on the chef srever , crate runist to define what code runs on which nodes.

You define different cookbooks.

Once the code is ready HOW

KNIFE:

Knife utility is used to manage node, upload chef code, install new clients on nodes.

Knife utility must be communicating with the chef server.

Knife.rb file located at. .chef directory.

In our case we use knife utility to upload chef to the server.

Use **knife cookbook upload**

**Create runlist :**

Use knife utility again to add runlists for each. Node.

Knife node run\_list add NODE\_NAME RUN\_LIST\_ITEM

Knife node run\_list add node1 ‘receipe[web]’

**Apply code:**

Login into each node and run chef client pull the configs from the server. This is no no

Knife ssh ‘name:\*’ ‘chef-client’

Kick off ssh on all nodes

Knife ssl check -🡪 check chef-server connectivity form workstation

Knife cookbook upload <cookbook name>

Knife cookbook list 🡪 to check all uploaded cookbooks.

Chef generate cookbook <cookbook name>. 🡪 to generate cookbook structure

For receipe syntax checks -- cookstyle <cookbook name>

For local. Smoke test 🡪 chef-client --local-mode --why-run --runlist “receipe[cookbook::receipe]”

Chef-client --local-mode --runlist “receipe[cookbook::receipe]”

Chef-client --why-run

To connect work station. To chef add chef url in knife.rb

.chef/knife.rb

Chefserver\_url=””

Knife ssl fetch , fetch trusted certificate from the trusted srever.

Very certificate

Knife node list -🡪 fetch node info from the chef server.

Knife cookbook list

Knife cookbook upload starter

Knifecookbook list

**Chef client:**

Check the connection b/w client and workstation.

Ping private ip of client from ws.

Ping private ip of ws from client 🡪 both are in the same n/w.

Set persistent domain name

f

practical commands:

create chef server

download chef dk from server

configure knife.rb

chef-repo/cookbooks

create cookbook

upload cookbook 🡪 knife

join node:

knife bootstrap 3.95.194.45 -U ubuntu --sudo -i ../../pem/mychefserver.pem --node-name api-server --run-list 'recipe[default]'

**Chef setup:**

**Download kit**

**Then it will automatically connect to chef server**

**Bootstap node :**

* knife bootstrap 123.45.67.89 -x root -P password --node-name nodename

knife bootstrap x.x.x.x

--ssh-user ubuntu

--sudo

--identity-file ./.chef/mykey.pem

--run-list webserver

**Knife boostrap** 18.212.107.173. –ssh-user ubuntu --sudo --identify-file ./.chef/ chef\_practical.pem --node-name node1

**On node side:**

**Install chef client:**

**Create dir -> mkdir /etc/chef**

**Touch client.rb**

**Touch role**

**Touch client-validator.pem**

buntu@ip-172-31-47-34:/etc/chef$ cat client.rb

**log\_level :info**

**log\_location STDOUT**

**ssl\_verify\_mode :verify\_none**

**verify\_api\_cert false**

**chef\_server\_url "https://manage.chef.io/organizations/intelliblue"**

**validation\_client\_name "intelliblue-validator"**

**validation\_key "/etc/chef/intelliblue-validator.pem"**

**node\_name "node1"**

<https://gist.github.com/anthonygrees/3a9e4aa438558d1782b2a55695038007>

powershell- > cookbooks

joing windows node:

c:\chef\

create client.rb

create validator.pem

addd license to client.rb if asks.

**Chef resources vs Arbitrary commands:**

**Execute resource** allows for the execution of a single command. It is generally not an idempotent resource, but when used with only\_if and not\_if, it can be guarded to ensure idempodent.

Idempotence: execute same operation again and again.

**Chef command:**

Chef generate generator my\_cook

Make custom generator for cookbook

**Chef gem:**

The chef command or gem is a package manager for Ruby.

gem list local

gem install docker

Foodcritic:

Foodcritic is part of ChefDK and is invoked with the foodcritic command.

It’s a linting tool.

Foodcritic look DSL.

Using foodcritic:

Foodcritic tags are rules which have been grouped together and classified based on what they examine. A Tag can be checked by using running foodcritic.

Foodcritic <path> <tagname> , where tag name is one of the available tags ot individual rule numbers. Path is path of the cookbook or cookbooks. All of the tags and rules are. Documented at foodcritic.

Foodcritic ./ -t correctness

Running multiple rules by using comma

Foodcritic ./ -t ~tagname

It will execute all the rules except this tagname

You can place .foodcritic file in cookbook to check errors.

Cd /cookbook

Foodcritic .

Berks:

Managing cookbook dependencies by manually downloading required cookbooks can [resent a no of challenges.

Using Berkshelf makes handling dependencies much easier and simpler to keep run.

Berkshelf keeps the dependencies if resolves in a central location .

~/.berkshelf/cookbooks/,cookbook- version>

Which keeps the cookbooks repository free of clutter.

Which include private and public supermarket.

Metadata.rb defines the dependencies.

Berksfile configuration:

The default Berksfile is created when you generate a cookbook and configured to reference the public chef supermarket via source keyword. A source could be combination of public and private repos.

The **cookbook** keyword in the Berksfile allows you to define where you want to cookbook to installed from or to. Set additional version constraints

Cookbook ‘lcd\_web’, path: ‘../lcd\_web’

Cookbook ‘tar’, path: ‘../tar’

\*\*\*To evaluate and fetch dependencies run berks install once your metadata has been configured.

berks install

then berks.lock is created ,which acts like a snapshot of the versions and their dependencies.

Berks update:

It will delete berks.lock file and rerun the berks install

Berks apply

The version locking mechanism can be applied to entire environemnts as well, by specifying an environment’s JSON file.

Given a /tmp/development.json file with following error.

Berks package:

If you want to package up. All the cookbooks and their dependencies , you could go into the berkshelf directory and grab the files you. Want , or you could use the **berks package command**

Berks package is like a tar. And bundles up the dependencies of a Berksfile.

Berks outdated

The berks outdated command will show what dependencies have updated available.

Berks help

**Attribute:**

Attribute directory can hold collection of files. These contain attributes or settings can be used to configure infrastructure. Attributes function as a way to abstract configuration setting data from the recipe code.

For each cookbook loaded, the default.rb attributes file gets loaded first. Any additional attribute files are loaded in lexical sort order.

In the list shown, the default.rb was generated on purpose, and was not created automatically.

Chef generate attributes

The chef obtains attributes in the following locations.

Node information from Ohai

Attribute files from within cookbooks

Recipes within cookbooks.

Environments.

Roles

Attribute behavior on chef run:

The chef client will obtain the code object from the chef server, which has all the attribute data from the last chef client run. After that is complete, all attributes except normal are reset. This implies a rebuild process in which attributes from. Their various sources are consolidated and evaluated for precedence.

Attributes support multiple levels of precedence. An attribute may have multiple values internally but will only return the higher precedence value when read.

If an attribute is uniquely identified, it is consolidated into node object. If this is singke instance there would be no precedence .

There are 6 types of attributes

node. <type>[‘attribute\_name’]= ‘value’ Attributes

default: default has the lowest precedence and should be used as often as possible.

Force\_default: when defined in a cookbook, this takes precedence over default attribute set in a role or environment.

s

**Custom resources:**

Custom resoures are used to customize the resource.

It’s the extension of resource.

Create custom resource;

Property :homepage, string

action :create do //creat action has 3 resources we are customized

package ‘httpd’

service ‘httpd’ do

action [:enable, :start]

end

file ‘/var/www/html/index.html’

content new\_resource.homepage

end

end