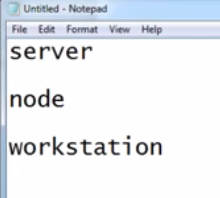


for practice.

it is just fb a/c creation

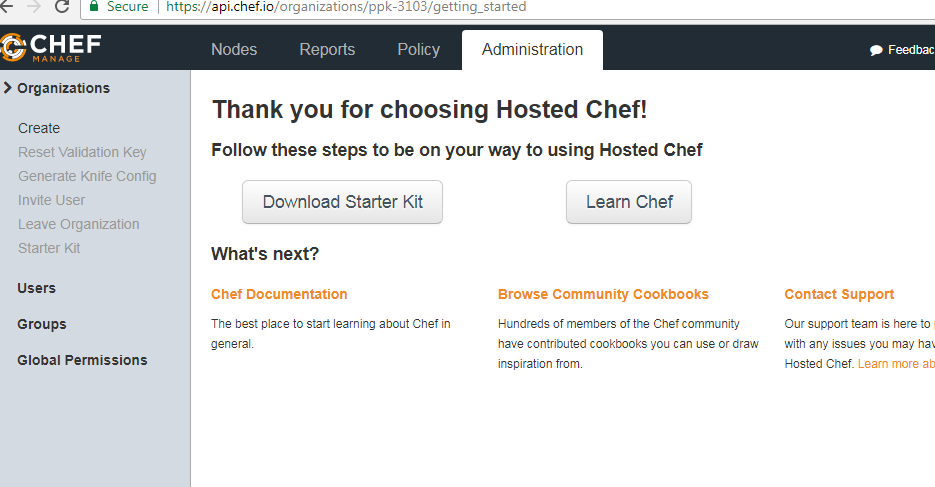
4gb and 2core

chef itself hosted chef server, you've to create [register]login and get server. free till 5 nodes

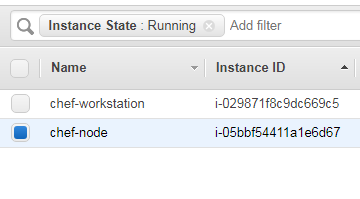


aws/any/windows

aws/any



chef server



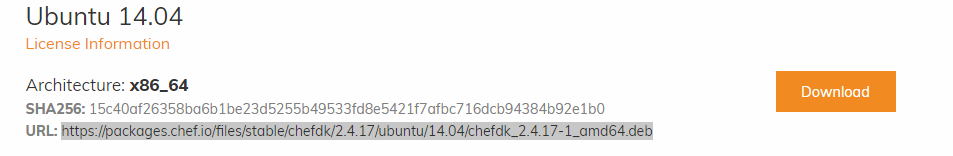
aws chef node

aws workstation



agent installation:- chef dk

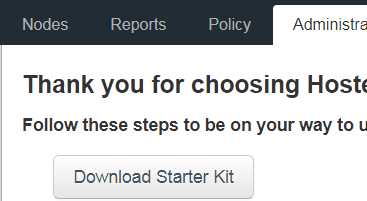
google : chef dk



wget https://packages.chef.io/files/stable/chefdk/2.4.17/ubuntu/14.04/chefdk\_2.4.17-1\_amd64.deb

chef agent installation

knife -v



starter kit : tells who you are.

bootstrap : establishing connection b/w chef server and node.

EXECUTE THIS on node after placed starter kit

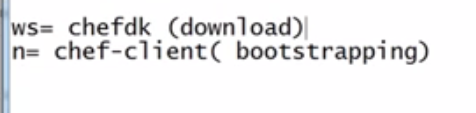
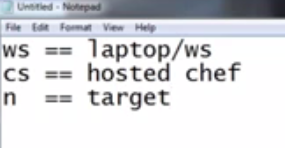
ubuntu

**knife bootstrap 172.31.25.44 --ssh-user ubuntu --sudo --identity-file /home/ubuntu/new-key.pem -N ip-172-31-25-44.us-west-2.compute.internal**

**windows:- chef-dk [redhat]**

**knife bootstrap 52.25.5.43 --ssh-user ec2-user --identity-file "C:\Users\P P K\Downloads\new-key.pem" --sudo -N ec2-52-25-5-43.us-west-2.compute.amazonaws.com**

**Note : in boot strap we use only public and public dns**



powershell

Set-Location " C:\Users\P P K\Downloads\chef-repo"

google:

chef generate cookbook helloworld

google:-

chef resources

chef resource package

**cookbook upload process**

can be upload in 2 ways.

to upload we should present in chef-repo(starte kit)

* knife cookbook upload helloworld
* berk upload (this is based on version(update metadata.rb file whenever there is a change in configuration)

**chef run-list:-**

* list of recipes that gets executed during convergence
* this run list determines what has to be executed during convergence
* every node has this run list.
* to the run list we have to add our recipes.

**Workflow:-**

we write cookbook (ws)

we upload it to chef server (whether it is right or wrong check we upload)

we add run list to apply recipes on server

google:-

knife node run\_list

**Syntax**

This argument has the following syntax:

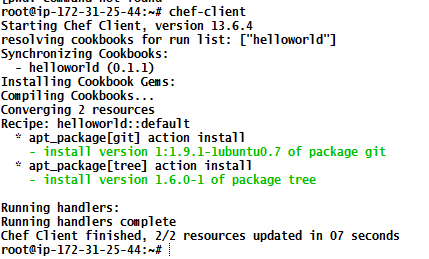
$ knife node run\_list add NODE\_NAME RUN\_LIST\_ITEM (options)

* knife node run\_list add ec2-52-25-5-43.us-west-2.compute.amazonaws.com 'recipe[helloworld]'
* knife node run\_list add ip-172-31-25-44.us-west-2.compute.internal 'recipe[helloworld]'

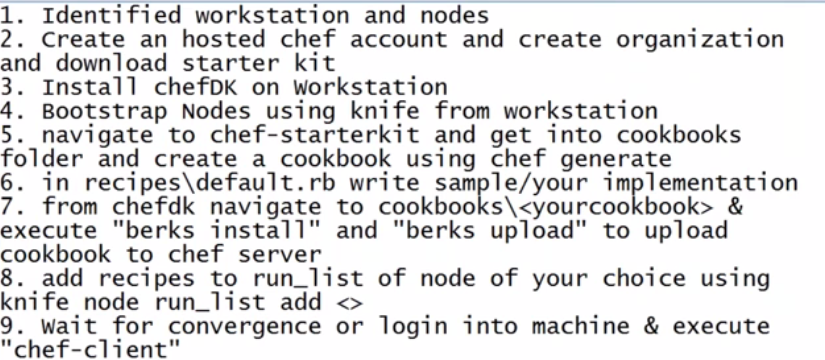
**Convergence skip:-**

* login into machine
* run chef-client command

**o/p:-**

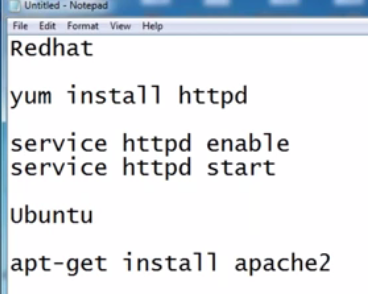


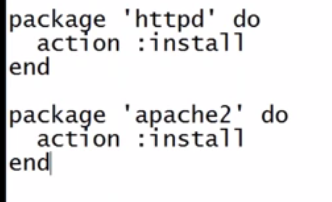
Example#1 summary



Example#2

installtions vary os to os ( **create apache.rb file in recipes of helloworld cookbook**)





package 'apache2' do

action :install

end

service 'apache2' do

action :start

end

knife node run\_list add ip-172-31-25-44.us-west-2.compute.internal 'recipe[helloworld::apache]'

Q# why do i need to write 2 different syntaxes bcoz only package name differs and rest same.

using node object



this is same as facter in puppet.

<https://github.com/jhotta/chef-fundamentals-ja/blob/master/slides/just-enough-ruby-for-chef/01_slide.md>

**Variables/alias:-**

where we can store data and/or the data might change is called variable.

**Points:-**

* so, whenever we allocate data in programming ,where it will allocate/created, it will allocate/created in memory which is ram, not the hard disk, HDD is just for saving.
* Ram is very fast , it has habit of forgetting the things, it is volatile memory, the moment you stop voltage supply, it will forget everything.
* Below is available memory allocation,generally addresses are hexa decimal,is it readable,the other point using the addresses directly is ,if i use same program in my system or in your system both might not give same address,this might be different hence i cannot go by address mechanism directly, i need to have something which make this simple.

total memory[ram]

memory address [ hexa decimal 00x012]

[

* it is alias for memory reference

package\_name='apache2'

package package\_name do

action :install

end

service package\_name do

action :start

end

**Note:-**

* if i change my code/value at variable place, my problem will be solved, i dont have to change it in 2 places.
* today you've 2 packages to install, you will use 2 variables, tomorrow you've 100 packages to install,do you still use variable for all those 100, no it is not good practice to use, for that we've concept called **"ARRAY".**
* **Array:- collection of different values.**
* Basically array start position is "0"

Example:-

packages= ['git', 'tree', 'wget']

normal position of packages is git is '1',tree is '2' and wget is '3'

but in array it is different, it starts from '0' for git,tree for '1',and 2 for wget

* Arrays are good when we're dealing with same datatypes.

packages= ['git', 'tree', 'wget']

* array is not good when we're dealing with different datatypes.

packages= ['git', '1.0']

access:-

package[0]

package[1]

this can be fixed using hashes.

**Hashes# collection of key value pairs. where you can create what's it keys and values**

i am creating structure hashes which contains keys and values.

so, it starts with curly braces and ends with curly braces.

{

name => "git"

version => "1.0"

}

which is more readable array or hash. Hash.

**Example#**

**{**

**name => test1**

**address => {**

**first lane => "dsjfa"**

**secondname => "dada"**

**city => "hyd"**

**},**

**phone = > 9177780647**

**}**

Here if you notice, address is multiple keys and values, thats beauty of hashes(key/value),now let us see how we can access them,

lets whole above data i have assigned it to variable called "student"

**student = {**

**name => test1**

**address => {**

**first lane => "dsjfa"**

**secondname => "dada"**

**city => "hyd"**

**},**

**phone = > 9177780647**

**}**

so, to access name ,i would use this syntax.

# i want get to his name

i use, student['name']

# i want to get his city

student['address'] ['city']

this hash we would be using more in chef/puppet

**Node objects = Facters**

**Conditional statements :-**

if code to work in cross platform operating system, we need to write conditional statements.

**Basic flow:-**

pkg\_name = 'apache2'

if node['platform'] == 'ubuntu'

package\_name = pkg\_name

end

package pkg\_name do

action :purge

end