CHEF

Content:

Chef Introduction:

Chef is one of the open source infrastructure management/configuration management tool/ powerful automation platform used to automate infrastructure.

Chef has 3 main components

Chef server: it is a hub for configuration data, It stores cookbooks recipes policies

Chef workstation: Where we write recipes cookbooks

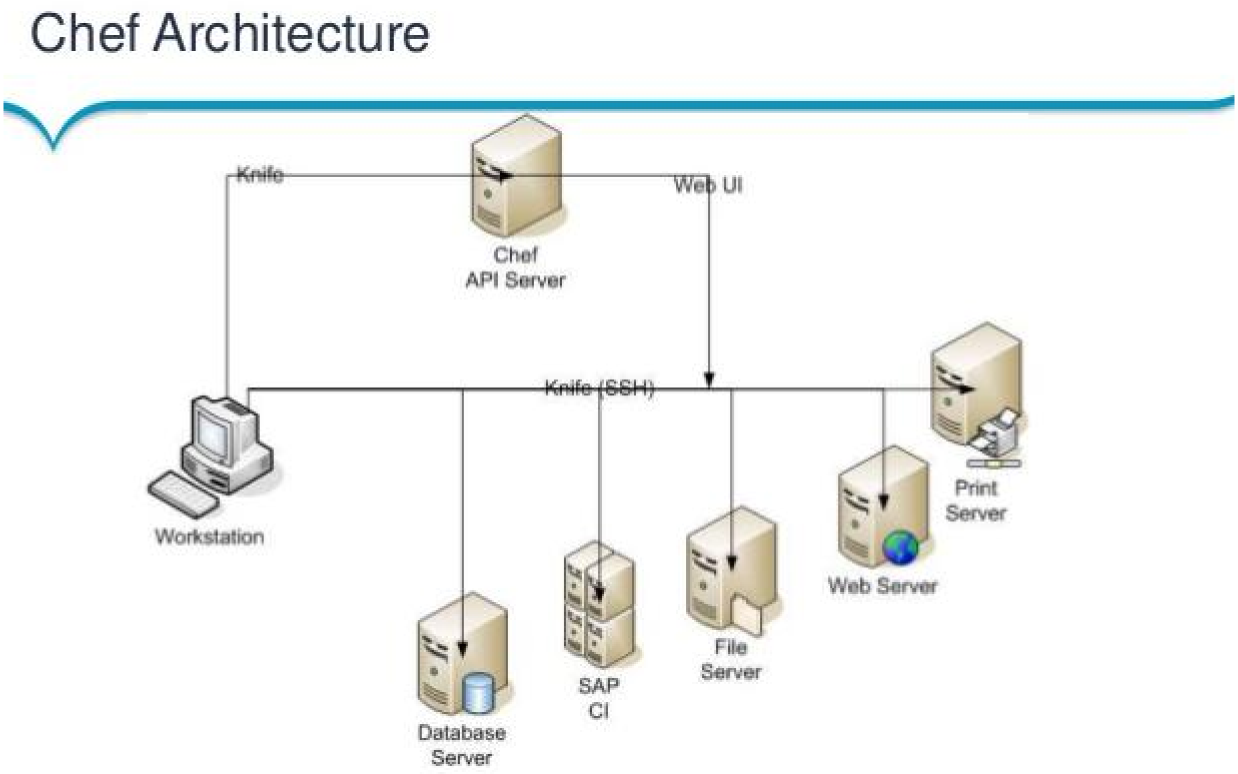
Check nodes - nodes can be cloud nodes, physical machines, VMs

The workstation is the location from which all of Chef is managed, including installing the Chef DK, authoring cookbooks, and using tools like Kitchen, chef-zero (a command-line tool that runs locally as if it were connected to a real Chef server), command-line tools like Knife (for interacting with the Chef server) and chef (for interacting with your local chef-repo), and resources like core Chef resources (for building recipes) and InSpec (for building security and compliance checks into your workflow).

Nodes are the machines—physical, virtual, cloud, and so on—that are under management by Chef. The chef-client is installed on each node and is what performs the automation on that machine.

Use the Chef server as your foundation to create and manage flexible, dynamic infrastructure whether you manage 50 or 500,000 nodes, across multiple datacenters, public and private clouds, and in heterogeneous environments.

The Chef server acts as a hub for configuration data. The Chef server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by the chef-client. Nodes use the chef-client to ask the Chef server for configuration details, such as recipes, templates, and file distributions. The chef-client then does as much of the configuration work as possible on the nodes themselves (and not on the Chef server). This scalable approach distributes the configuration effort throughout the organization.



CHEF CONFIGURATION

CHEF Server setup:

1. set selinux to be permissive or disabled on all nodes

Command:

Execute setenforce Permissive command on the server, once its execute check the status of command setenforce, it should be Permissive**.**

```
setenforce Permissive

[root@ctx3p12 ~]# getenforce

Permissive
```

2. stop iptables/firewall

disable it

3. open ports 80 and 443

```
iptables -I INPUT 1 -p tcp -m tcp --dport 80 -j ACCEPT

iptables -I INPUT 1 -p tcp -m tcp --dport 443 -j ACCEPT

service iptables save
```

hostname has to be FQDN , set the FULLY QUALIFIED HOSTNAME,

Note: before installing the server,client, workstation make sure you have added all 3 machines ipaddress and hostname in all 3 servers as below

```
cat /etc/hosts

192.168.100.1 chefserver

192.168.100.2 chefclient

192.168.100.3 chefworkstation
```

4. install chef server

```
[root@ctx3p12 MISC]# rpm -ivh chef-server-core-12.9.1-1.el7.x86_64.rpm

warning: chef-server-core-12.9.1-1.el7.x86_64.rpm: Header V4 DSA/SHA1 Signature, key ID 83ef826a: NOKEY

Preparing...                          ################################# [100%]

Updating / installing...

   1:chef-server-core-12.9.1-1.el7    ################################# [100%]
```

5. Right after installation execute reconfigure:

```
chef-server-ctl reconfigure
```

creates default data and configuration files.

Note: The reconfigure subcommand is used when changes are made to the chef-server.rb file to reconfigure the server. When changes are made to the chef-server.rb file, they will not be applied to the Chef server configuration until after this command is run. This subcommand will also restart any services for which the service_name['enabled'] setting is set to true.

```
mkdir /root/.chef
```

6. create users:

We need to create the user and organization to access the webGUI

For accessing the webGUI, we need chef-mange rpm has to be installed on the server

Syntax: chef-server-ctl user-create stevedanno Steve Danno steved@chef.io 'abc123' --filename /path/to/stevedanno.pem

example:

```
chef-server-ctl user-create santosh santosh kumar santosh@gmail.com 'abc123' --filename /root/.chef/santosh.pem
```

Note:

Here user name is santosh

Password is abc123

Note: execute the command as is to avoid syntax errors

7. create organization:

syntax: chef-server-ctl org-create short_name 'full_organization_name' --association_user user_name --filename ORGANIZATION-validator.pem

example:

chef-server-ctl org-create org 'organization' --association_user santosh --filename /root/.chef/org-validator.pem

Note: execute the command as is to avoid syntax errors

8. Install chef-manage rpm with option -- accept the license as below

chef-manage-ctl reconfigure --accept-license

9. test Chef server by running

chef-server-ctl test

Note: the above command should not give any errors

10. Access web GUI

https://FQDN-OR-IP-OF-CHEF-SERVER

example:

https://ctx1p21.company.in.com

chef-server-ctl status to make sure all the processes are running

(**Note:** none should say down. If one is down, try running chef-server-ctl restart).

Packages for chef:

chef server - chef-server-core-12.9.1-1.el7.x86_64.rpm

chef workstation - chefdk-0.19.6-1.el7.x86_64.rpm

chef client - chef-12.15.19-1.el6.x86_64.rpm

SETUP WORK STATION:

1. Install Development kit rpm

```
rpm –ivh <chef-developmentkit.rpm>
```

After rpm is installed check chef client version as below

```
    chef-client -v
 Chef: 11.6.0
```

2. Create the “.chef” directory

```
    mkdir .chef        under /root
```

copy the keys from server to this directory, (.pem files)

Modify the knife.rb file as below or create knife.rb with below content

From the server copy the .pem files to work station

```
[root@reviewb .chef]# pwd
/root/.chef
[root@reviewb .chef]# ls -lrt
total 8
-rw-r--r--. 1 root root 1674 Dec 12 21:31 santosh.pem
-rw-r--r--. 1 root root 1674 Dec 12 21:32 org-validator.pem
[root@reviewb .chef]#
```

Create a directory on the work station /root/chef-repo/.chef

Under the directory /root/chef-repo/.chef

Create another file knife.rb

```
[root@ctx3p10 .chef]# cat knife.rb

log_level                :info

log_location             STDOUT

node_name                'santosh'

client_key               '~/chef-repo/.chef/santosh.pem'

validation_client_name   'org-validator'

validation_key           '/chef-repo/.chef/org-validator.pem'

chef_server_url          'https://ctx1p05/organizations/org'

syntax_check_cache_path  '~/chef-repo/.chef/syntax_check_cache'

cookbook_path [ '~/chef-repo/cookbooks' ]
```

```
    knife ssl fetch     to fetch ssh keys

    knife user list
```

```
[root@ctx3p10 chef-repo]# knife ssl fetch

WARNING: Certificates from ctx1p05  will be fetched and placed in your trusted_cert

directory (/root/chef-repo/.chef/trusted_certs).


Knife has no means to verify these are the correct certificates. You should

verify the authenticity of these certificates after downloading.

Adding certificate for ctx1p05 in /root/chef-repo/.chef/trusted_certs/ctx1p05_in_company_com.crt
```

```
   [root@ctx3p10 chef-repo]# knife user list
```

santosh

knife node list – lists nothing as the no new nodes are added.

```
[root@ctx3p10 chef-repo]# ls -lrt
total 12
-rw-r--r--. 1 root root   70 Oct 26 07:54 LICENSE
-rw-r--r--. 1 root root 1499 Oct 26 07:54 README.md
-rw-r--r--. 1 root root 1133 Oct 26 07:54 chefignore
drwxr-xr-x. 3 root root   36 Oct 26 07:54 data_bags
drwxr-xr-x. 2 root root   41 Oct 26 07:54 roles
drwxr-xr-x. 2 root root   41 Oct 26 07:54 environments
drwxr-xr-x. 3 root root   36 Oct 26 07:54 cookbooks
[root@ctx3p10 chef-repo]# pwd
/root/chef-repo
[root@ctx3p10 chef-repo]#
```

```
[root@ctx3p10 .chef]# ls -lrt
total 12
-rw-r--r--. 1 root root 1678 Oct 26 07:55 org-validator.pem
-rw-r--r--. 1 root root 1678 Oct 26 07:55 santosh.pem
drwxr-xr-x. 2 root root   35 Oct 26 08:21 trusted_certs
-rw-r--r--. 1 root root  436 Oct 26 08:22 knife.rb
[root@ctx3p10 .chef]# pwd
```

```
/root/chef-repo/.chef

[root@ctx3p10 .chef]#
```

NODE/Client SETUP:

Download and install chef-<>.rpm

```
[root@ctx3p04 MISC]# rpm -ivh chef-12.15.19-1.el7.x86_64.rpm

warning: chef-12.15.19-1.el7.x86_64.rpm: Header V4 DSA/SHA1 Signature,
key ID 83ef826a: NOKEY

Preparing...
################################# [100%]

Updating / installing...

   1:chef-12.15.19-1.el7
################################# [100%]

Thank you for installing Chef!
```

Note:

chef-12.15.19-1.**el7**.x86_64.rpm   el7 is for RHEL 7

el6 is for RHEL 6

Make sure you use the right package for the linux flavor installed on the machine.

```
[root@ctx3p04 MISC]# chef-client  -v

Chef: 12.15.19

[root@ctx3p04 MISC]#
```

mkdir /etc/chef

make sure you have ssh keys available on the clients (under ~/.ssh)

if not execute ssh-keygen on the command line to generate the keys (on server, workstation and clients)

copy the files (.pem keys) from server

Chef Bootstrap Method:

Boot strap is method of adding the nodes to the server

from work station execute this command

syntax:

```
knife bootstrap <node ip> --ssh-user root --sudo --identity-file ~/.ssh/id_rsa --node-name <FQDN name of node>
```

```
knife bootstrap 9.182.76.58 --ssh-user root --sudo --identity-file ~/.ssh/id_rsa --node-name ctx3p04.x.com
```

```
[root@ctx3p10 chef-repo]# knife bootstrap 9.182.76.58 --ssh-user root --sudo --identity-file ~/.ssh/id_rsa --node-name ctxp04

Creating new client for ctxp04

Creating new node for ctxp04

Connecting to 9.182.76.58

root@9.182.76.58's password:

9.182.76.58 -----> Existing Chef installation detected

9.182.76.58 Starting the first Chef Client run...

9.182.76.58 Starting Chef Client, version 12.15.19

9.182.76.58 resolving cookbooks for run list: []

9.182.76.58 Synchronizing Cookbooks:

9.182.76.58 Installing Cookbook Gems:

9.182.76.58 Compiling Cookbooks...
```

```
9.182.76.58 [2016-10-26T18:48:58+05:30] WARN: Node ctxp04  has an empty run list.

9.182.76.58 Converging 0 resources

9.182.76.58

9.182.76.58 Running handlers:

9.182.76.58 Running handlers complete

9.182.76.58 Chef Client finished, 0/0 resources updated in 03 seconds

   [root@ctx3p10 chef-repo]# echo $?

0
```

If the chef client is installed on the node, it adds to server

If not, it will try to download and install on the node.

Once the node is added, nodelist will show the node as below

```
   [root@ctx3p10 chef-repo]# knife node list

ctxp04
```

https://learn.chef.io/tutorials/manage-a-node/rhel/hosted/bootstrap-your-node/

we can add as many nodes as possible to chef-server

Chef Cookbooks

Writing cookbooks/recipes

Sample cook books:

1. create a file /tmp/xhellp.txt

```
file '/tmp/xhello.txt' do

  content 'Welcome to Chef - Devopsss '
```

```
end
```

```
export EDITOR=vi
cd ~/chef-repo
```

Creating the cookbooks:

Cookbook syntax is based on JSON – java script object Notation format

```
knife cookbook create sample
```

Use this command instead of above

```
chef generate cookbook
```

```
cd cookbooks/sample/
cd recipes/
default.rb
```

upload the cookbook to server

```
knife cookbook upload  sample
```

```
knife node edit ctx3p12
 knife node edit ctx1p13
knife node edit ctx3p12
```

Cook book examples:

2. create a file /tmp/xhellp.txt

   ```
   file '/tmp/xhello.txt' do
   ```

```
  content 'Welcome to Chef - Devopsss '
end
```

3. Start and enable httpd service

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

4. Install Apache package

```
package "httpd" do
action :"install"
end
```

5. Create a user on a machine

```
user 'user11' do
  comment 'A random user'
  uid '12334'
  gid '513'
  home '/home/random'
  shell '/bin/bash'
  password '$1$JJsvHslasdfjVEroftprNn4JHtDi'
end
```

6. Execute a command on the client

```
execute ‘name' do
  command 'command'
end
```

7. Using **templates** to copy our own configuration file from workstation to client

```
template '/etc/proxy' do
source 'proxy.p'
owner "root"
group "root"
mode "644"
end
```

Here /etc/proxy is file on the client that needs to be replaced with the configuration file proxy.p which is present on the work station in the path cook cookbooks/<cookbookname>/templates/default/

To execute cookbook on the client, we need to upload it first using below commands

Create cookbook:

```
knife cookbook create createuser
```

Create run list:

```
knife node run_list add ctx1p13  createuser
```

```
[root@ctx1p10 chef-repo]# knife node run_list add ctx1p11.in.company.com
createuser
ctx1p11.in.company.com:
  run_list: recipe[createuser]
[root@ctx1p10 chef-repo]# knife node list
ctx1p11.in.company.com
```

Note: run list can be modified using GUI as well

Run list – is for associating the cookbook with the node

Upload it to server:

```
knife upload cookbooks –a
```

To execute it on a client:

execute chef-client on the nodes

To delete all cook books at once

```
knife cookbook bulk delete '.*' –p
```

to upload all cookbooks

```
knife cookbook upload –a
```

if you don’t want any one else to upload a new version, use freeze option:

```
knife cookbook upload thegeekstuff –freeze
```

upload cookbook forcibly

```
knife cookbook upload thegeekstuff –force
```

to list all cookbooks

```
knife cookbook list –a
```

```
[root@ctx3p10 chef-repo]# knife cookbook list -aw
createuser:
  0.1.0: https://ctx1p05/organizations/org/cookbooks/createuser/0.1.0
createuser1:
  0.1.0: https://ctx1p05/organizations/org/cookbooks/createuser1/0.1.0
createuser2:
  0.1.0: https://ctx1p05/organizations/org/cookbooks/createuser2/0.1.0
sample:
  0.1.0: https://ctx1p05/organizations/org/cookbooks/sample/0.1.0
user234:
  0.1.0: https://ctx1p05/organizations/org/cookbooks/user234/0.1.0
[root@ctx3p10 chef-repo]#
```

To delete cookbooks permanently from server:

The -p option will delete the cookbook, and permanently purge the cookbook from the Chef server. Use this option with caution.

```
knife cookbook delete createuser  –p
```

bulk delete

```
knife cookbook bulk delete create* -p
```

to download a cookbook from chef server

```
knife cookbook download
```

To generate metadata for cookbook

```
knife cookbook metadata –a
```

Some more examples:

Resources:

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

Cook book examples:

To install apache package on the node, start the service and show custom message on the screen

```
package "httpd" do
```

```
        action :install
end

service "httpd" do
        action [:enable, :start]
end

cookbook_file "/var/www/html/index.html" do
        source "index.html"
        mode "0644"
end
```

create index.html file under files

```
<head>
    <title>Company Chef Demo</title>
</head>
<body>
    <h1>
        Welcome to Chef .
    </h1>
</body>
</html>
```

Chef template

```
package "httpd" do
        action :install
end

service "httpd" do
        action [:enable, :start]
end

template "/var/www/html/index.html" do
        source "index.html"
        mode "0644"
end
```

To execute a specific recipe for a node

```
[root@ctx1p11 ~]# chef-client -o "recipe[templatecb]
> ^C
[root@ctx1p11 ~]# chef-client -o "recipe[templatecb]"
Starting Chef Client, version 12.15.19
[2017-02-20T09:02:31-05:00] WARN: Run List override has been provided.
[2017-02-20T09:02:31-05:00] WARN: Original Run List: [recipe[servicecb], recipe[templatecb]]
[2017-02-20T09:02:31-05:00] WARN: Overridden Run List: [recipe[templatecb]]
resolving cookbooks for run list: ["templatecb"]
```

```
Synchronizing Cookbooks:
  - templatecb (0.1.0)
Installing Cookbook Gems:
Compiling Cookbooks...
Converging 1 resources
Recipe: templatecb::default
  * template[/var/www/html/index.html] action create (up to date)
[2017-02-20T09:02:31-05:00] WARN: Skipping final node save because override_runlist was given

Running handlers:
Running handlers complete
Chef Client finished, 0/1 resources updated in 07 seconds
[root@ctx1p11 ~]#
```

To install a package

```
[root@ctx1p11 ~]# chef-apply -e "package 'vim'"
Recipe: (chef-apply cookbook)::(chef-apply recipe)
  * yum_package[vim] action install (up to date)
[root@ctx1p11 ~]#
```

To execute a single recipe:

```
chef-workstation##chef-apply file/recipes/default.rb
Recipe: (chef-apply cookbook)::(chef-apply recipe)
```

```
  * file[/tmp/log1.txt] action create
    - create new file /tmp/log1.txt
    - update content in file /tmp/log1.txt from none to d8f469
    --- /tmp/log1.txt   2017-02-20 16:25:36.356677978 -0500
    +++ /tmp/.chef-log120170220-12807-188jrlj.txt       2017-02-20 16:25:36.356677978 -0500
    @@ -1 +1,2 @@
    +Hello world1
    - restore selinux security context
```

To debug chef-client

Chef-client -l debug

Starts the chef-client which will poll the chef-server every 3600 sec for changes

Chef-client -i 3600   chef client will be executed every 3600 seconds, prompt will not come back till control+c is entered.

Include_recipe

```
[root@ctx3p07 cookbooks]# cat apt/recipes/default.rb
execute 'touch' do
  command 'touch /file900'
end
```

```
[root@ctx3p07 cookbooks]# cat sshcb/recipes/default.rb
```

```
include_recipe "apt"

package 'openssh' do
  action :install
end

service 'sshd' do
  action [ :enable, :start ]
end

cookbook_file "/etc/ssh/sshd_config" do
  source "sshd_config"
  mode "0644"
  notifies :restart, 'service[sshd]' , :immediately
end
```

Using attributes in cookbooks

```
cat syslog/recipes/default.rb
package "rsyslog" do
action [ :install ]
end
```

```
cookbook_file "/etc/rsyslog.conf" do
owner node[:syslog][:user]
group node[:syslog][:group]
mode "640"
source "rsyslog.conf"
end

service "rsyslog" do
action [ :enable, :start ]
end
```

```
[root@ctx3p07 cookbooks]# cat syslog/attributes/default.rb
default[:syslog][:user]          = "user1234"
default[:syslog][:group]         = "user1234"
[root@ctx3p07 cookbooks]#
```

Create user with defined user and group in attributes.rb

```
[root@ctx3p06 cookbooks]# cat file/attributes/default.rb
default['user'] = 'user1234'
default['group'] = 'group1234'
[root@ctx3p06 cookbooks]# cat file/recipes/default.rb
file '/tmp/helloworld1' do
```

```
content 'Hello world'

user node['user']

group node['group']

end
```

Ohai profiler in Chef:

Ohai profiler is used to get the values of system parameters in chef.

It comes by default with chef-client

Execute ohai on command line.

Search option in Chef:

```
knife search node "*:*" -a kernel

knife search node "*:*" -a os

knife search node "*:*" -a fqdn
```

get values from ohai and use in search criterion:

```
[root@ctx3p06 cookbooks]# knife search node "*:*" -a platform

2 items found


ctx2p06.in.company.com:

  platform: redhat


ctx2p03.in.company.com:

  platform: redhat
```

```
knife node show <>
```

in JSON (Java Script Object Notation) format:

```
knife node show <> –Fj     shows in key value pair format
```

Chef Runlists:

Runlist can be altered from GUI aswell

By dragging the cookbook to node box

Chef internally takes care of the package management (to use yum/yast/apt-get) based on the OS/flavor.

Data bags:

A data bag is a global variable that is stored as JSON data and is accessible from a Chef server. A data bag is indexed for searching and can be loaded by a recipe or accessed during a search

Details like user’s data are stored in a container called data bags

Data bags puts encryption so password can be protected

Create databags under chef-repo locally first

```
mkdir –p data_bags/users
```

```
mkdir –p data_bags/groups
```

Then upload it to server

```
knife data_bag create users
```

```
knife data_bag create groups
```

Create 2 files for 2 users in JSON format

```
[root@ctx3p10 users]# cat user0.json
{
"id": "user0",
"comment": "user0 user",
"uid": "3012",
"gid": "0",
"home": "/home/user0",
"shell": "/bin/bash"

}
```

```
[root@ctx3p10 users]# cat user1.json

{

"id": "user1",

"comment": "user1 user",

"uid": "3022",

"gid": "0",

"home": "/home/user1",

"shell": "/bin/bash"


}
```

To push the users in to the server

knife data bag from file users user0.json

Users – name of the data bag on the server

To search the data in databag

```
[root@ctx3p10 users]# knife search users "id:user0"

1 items found


chef_type: data_bag_item

comment:   user0 user

data_bag:  users

gid:       0

home:      /home/user0

id:        user0
```

```
shell:     bin/bash
uid:       3012

[root@ctx3p10 users]# knife search users "id:user*" -a shell
2 items found

:
  shell: bin/bash

:
  shell: bin/bash

[root@ctx3p10 users]# knife search users "id:user0" -a shell
1 items found

:
  shell: bin/bash
```

create groups directory

```
mkdir groups
```

create group1 json

```
[root@ctx3p10 groups]# cat group1.json
{
"id": "group0",
"gid": 3000,
"members": ["user0", "user1"]
}
[root@ctx3p10 groups]#
```

knife data bag from file groups group1.json

```
[root@ctx3p10 groups]# knife search groups "*:*"
1 items found

chef_type: data_bag_item
data_bag:  groups
gid:       3000
id:        group0
members:
  user0
  user1

[root@ctx3p10 groups]#
```

Create a cookbook

Knife create cookbook users

Modify the default.rb as below

```
search("users", "*:*").each do |user_data|
        user user_data["id"] do
```

```
    comment user_data["comment"]

    uid user_data["uid"]

    gid user_data["gid"]

    home user_data["home"]

    shell user_data["shell"]

    end

end

include_recipe "users::groups"
```

the above step, searches all the users from JSON data and creates users

include_recipe “users::groups”   calls another recipe groups from users recipe

The below file has to be created with .rb extension ie groups.rb in the same directory as default.rb.

```
search("groups", "*:*").each do |group_data|

    group group_data["id"] do

    gid group_data["gid"]

    members group_data["members"]

    end

end
```

the above step, searches all the groups from JSON data and creates groups

so users recipe executes groups recipe automatically upload it to server

```
[root@ctx3p10 cookbooks]# knife cookbook upload users
Uploading users          [0.1.0]
Uploaded 1 cookbook.
[root@ctx3p10 cookbooks]#
```

Execute chef-client on the client it creates new users and groups

Roles:

Clubbing servers together

Ex: webserver

Database server

Monitoring server

Application server

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

role maintains a runlist for itself

helps you manage servers which are identical.

In addition to above roles, it is common practice to group any functionality that goes together in a role

Ex: base role

Where we include all recipes that should be run on every node

To install basic software on bunch of machines

Best practice is to have a role for a specific purpose.

Create a file/ role under role

Webserver.rb

Upload to server

```
knife role from file webserver.rb
```

if there are 1000 webservers, only for the first time we need to assign role to those 1000 servers.

After that we do not have to touch them

After that we can just edit run list for webserver i.e adding recipe to webserver.

```
[root@ctx3p10 roles]# cat webserver.rb
name "webserver"
description "webserver"
run_list "recipe[apache]"
```

```
knife role from file webserver.rb
[root@ctx3p10 roles]# knife role show webserver
chef_type:           role
default_attributes:
description:         webserver
env_run_lists:
json_class:          Chef::Role
name:                webserver
override_attributes:
run_list:            recipe[apache]
```

How to add role to node:

Go to Chef GUI and edit the node runlist and add webserver to the role from available roles

so if we just add recipe to role, chef client atomically runs that recipe when chef-client is started.

search based on the role:

```
knife search node “role:webserver” –a
```

```
[root@ctx3p10 chef-repo]# knife search node 'role:w*'

1 items found

Node Name:   ctxp04.in.company.com

Environment: _default

FQDN:        ctx3p04.in.company.com

IP:          9.182.76.58

Run List:    role[weserver]

Roles:       weserver

Recipes:     apache, apache::default

Platform:    redhat 7.2

Tags:
```

Example:

```
[root@ctx1p10 roles]# knife search node 'role:w*'
```

```
2 items found


Node Name:   ctx2p10.in.company.com
Environment: _default
FQDN:        ctx2p10.in.company.com
IP:          9.182.76.52
Run List:    recipe[file123], role[webserver]
Roles:       webserver
Recipes:     file123, file123::default, file, file::default, users, users::default,
users::groups
Platform:    redhat 6.8
Tags:


Node Name:   ctx1p13.in.company.com
Environment: _default
FQDN:        ctx1p13
IP:          9.182.76.103
Run List:    recipe[ntp], role[webserver]
Roles:
Recipes:
Platform:    redhat 7.3
Tags:


[root@ctx1p10 roles]#
```

How to upload cookbook with a specific version:

While creating the cookbook using knife cookbook create <> change the metadata.rb as below

```
[root@ctx3p06 cookbooks]# cat file/metadata.rb

name             'file'

maintainer       'YOUR_COMPANY_NAME'

maintainer_email 'YOUR_EMAIL'

license          'All rights reserved'

description      'Installs/Configures file'

long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))

version          '0.2.0'
```

It updates with different version 0.2.0

And both contents of 0.1.0 and 0.2.0 can be checked in chef UI, under Policy cookbook content

Environment:

Segregation in chef server like dev test staging production

Every organization starts with a single environment

Environment reflects patterns and work flow

Different envs in one organization:

Development

Test

Staging

Production

Every node in organization has to belong to one org and one env at given point of time

The default env is read only and sets no policy

Knife environment list

_default

Env has names

Env has description

Env can have one or more cookbook constraints

Example:

dev.rb

name dev

Description: “For development”

Cookbook “apache”, “=0.2.0”

Apache cookbook can have multiple versions but it will pick 0.2.0 only

If any cookbook is modified and the new version is 0.3.0

Generally, nodes take up new version

With this restriction, we can restrict the nodes not to use latest until it is passed (change)

So change dev cookbook and select 0.3.0 if its successful then push to staging and production

We can have :

=0.2.0

Or

>0.2.0

< 0.2.0

Create another env called production.rb

It can be created from GUI aswell

By changing the attribute, we can modify the node without touching cookbook recipe.

Cookbooks can be exchanged between envs

They cannot be exchanged between organizations

Org ex: customer 1

Customer 2

Their code nodes are separate

In some ways, environments are fairly similar to roles. They are also used to differentiate different servers, but instead of differentiating by the function of the server, environments differentiate by the phase of development that a machine belongs to.

A cookbook which is under testing cannot be part of the production

So we need to control the different phases using the version of the cookbook

Create Environments:

```
mkdir  ~/chef-repo/environments
```

create a file development.rb

```
chef-workstation##cat development.rb

name "development"
```

```
description "The master development branch"

cookbook_versions({

   "apache" => "<= 0.1.0"

})
```

```
knife environment create development
```

this command will open a vi editor provide below thing in the version section and close

```
"apache": "<= 0.1.0"
```

So here we are creating a restriction for the cookbook called apache.

So development env can run only the cookbook with version 0.1.0 and not 0.2.0

Upload it to Chef server

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

Updated Environment development

Adding nodes to env:

```
knife node edit ctx2p06.in.company.com
```

This command will open a vi editor as below

```
{

  "name": "ctx2p06.in.company.com",

  "chef_environment": "_default",     change _default to development

  "normal": {

    "tags": [



    ]

  },
  "policy_name": null,
  "policy_group": null,
  "run_list": [
  "recipe[users]"
]


}
```

Now If chef-client is executed on the node, it shoud use 0.1.0 version

```
[root@ctx2p06 ~]# chef-client
Starting Chef Client, version 12.15.19
resolving cookbooks for run list: ["apache"]
Synchronizing Cookbooks:
  - apache (0.1.0)
```

Chef Super market:

Website: supermarket.chef.io

We can find preview and download cookbooks from chef supermarket community site

Use knife to work with the chef supermarket API

Download extract and examine implement cookbooks from supermarket

Hundreds of cookbooks are already available

Example my sql

Go thr readme for platform support/desc about cookbooks/authors/change log

View source   to view source code

We can request for changes/contribute to cookbooks

Download cookbook in zip format and extract.

Example: chef-client

Is the cookbook is used to configure system as chef-client

This cookbook can be used to configure interval in which chife-client executes. Default is 30 min 1800 sec

Knife has a plugin to

Search

Show

Download etc super maket cookbooks

> Chef server authentication is time sensitive, If the time sync is more than 15 min, authentication fails.

Cookbook: ntp

Knife cookbook site   connects to supermarket and can search/download etc

Download ntp in zip format

Copy it to workstation

In cookbook directory

Then upload it

```
[root@ctx3p10 ntp]# pwd
/root/chef-repo/cookbooks/ntp
[root@ctx3p10 ntp]# cd ..
[root@ctx3p10 cookbooks]# knife cookbook upload ntp
Uploading ntp          [3.2.0]
Uploaded 1 cookbook.
[root@ctx3p10 cookbooks]#
```

Berkshelf   A tool to manage dependencies between cookbooks, pulls down dependencies.

1) Install git on the work station
2) Execute

   git config --global user.email santoshdevops1@gmail.com

   git config --global user.name Santosh

3) Knife cookbook generate mydb
4) Change recipe, metadata.rb to point to 7.1.1version
5) Berks install   installs all cookbook dependancies
6) Berks upload   no-ssl-verify - to upload it to server

This ntp cookbook can be added to base role, so that ntp config is set.

Installs ntp packages and configures ntp.

How to execute a command on 1000 servers in data center without using cookbook

knife ssh <search criterion> <command>

ex:

```
[root@ctx1p10 chef-repo]# knife ssh 'name:ctx2p10.in.company.com' 'ls'

root@ctx2p10.in.company.com's password:

ctx2p10.in.company.com anaconda-ks.cfg  ant  install.log  install.log.syslog

[root@ctx1p10 chef-repo]#
```

```
[root@ctx1p10 chef-repo]# knife ssh "role:webserver" "ls -lrt "

root@ctx2p10.in.company.com's password:

ctx2p10.in.company.com total 52

ctx2p10.in.company.com -rw-r--r--. 1 root root  7572 May 23  2016 install.log.syslog

ctx2p10.in.company.com -rw-r--r--. 1 root root 29317 May 23  2016 install.log
```

```
ctx2p10.in.company.com -rw-------. 1 root root  1500 May 23  2016 anaconda-ks.cfg
ctx2p10.in.company.com drwxr-xr-x  2 root root  4096 Dec 21 09:38 ant
[root@ctx1p10 chef-repo]#
```

knife ssh "role:webserver" "chef-client "

Executes chef-client on all nodes in role webserver

to check status of client

```
knife status
```

Note: We need to download the chefbook from chef super maket, entire cookbook has to be downloaded not the source code.

Download copy to sever and add it to role/node and execute chef-client

Ntp cook book:

```
[root@ctx3p10 ntp]# cat recipes/default.rb
#
# Cookbook Name:: ntp
# Recipe:: default
# Author:: Joshua Timberman (<joshua@chef.io>)
# Author:: Tim Smith (<tsmith@chef.io>)
#
# Copyright 2009-2016, Chef Software, Inc.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

```
::Chef::Resource.send(:include, Opscode::Ntp::Helper)

if platform_family?('windows')
  include_recipe 'ntp::windows_client'
else

  node['ntp']['packages'].each do |ntppkg|
    package ntppkg
  end

  package 'ntpdate' do
    action :remove
    only_if { node['platform_family'] == 'debian' &&
node['platform_version'].to_i >= 16 }
  end

  [node['ntp']['varlibdir'], node['ntp']['statsdir']].each do |ntpdir|
    directory ntpdir do
      owner node['ntp']['var_owner']
      group node['ntp']['var_group']
      mode '0755'
    end
  end

  cookbook_file node['ntp']['leapfile'] do
```

```
    owner node['ntp']['conf_owner']
    group node['ntp']['conf_group']
    mode '0644'
    source 'ntp.leapseconds'
    notifies :restart, "service[#{node['ntp']['service']}]"
  end

  include_recipe 'ntp::apparmor' if node['ntp']['apparmor_enabled']
end

if node['ntp']['servers'].empty?
  node.default['ntp']['servers'] = [
    '0.pool.ntp.org',
    '1.pool.ntp.org',
    '2.pool.ntp.org',
    '3.pool.ntp.org'
  ]
  Chef::Log.debug 'No NTP servers specified, using default ntp.org server pools'
end

if node['ntp']['listen'].nil? && !node['ntp']['listen_network'].nil?
  if node['ntp']['listen_network'] == 'primary'
    node.normal['ntp']['listen'] = node['ipaddress']
  else
```

```
    require 'ipaddr'
    net = IPAddr.new(node['ntp']['listen_network'])

    node['network']['interfaces'].each do |_iface, addrs|
      addrs['addresses'].each do |ip, params|
        addr = IPAddr.new(ip) if params['family'].eql?('inet') ||
params['family'].eql?('inet6')
        node.normal['ntp']['listen'] = addr if net.include?(addr)
      end
    end
  end
end

node.default['ntp']['tinker']['panic'] = 0 if node['virtualization'] &&
                                              node['virtualization']['role'] == 'guest' &&

node['ntp']['disable_tinker_panic_on_virtualization_guest']

template node['ntp']['conffile'] do
  source 'ntp.conf.erb'
  owner node['ntp']['conf_owner']
  group node['ntp']['conf_group']
  mode '0644'
  notifies :restart, "service[#{node['ntp']['service']}]" unless
node['ntp']['conf_restart_immediate']
```

```
  notifies :restart, "service[#{node['ntp']['service']}]", :immediately if node['ntp']['conf_restart_immediate']
  variables(
    lazy { { ntpd_supports_native_leapfiles: ntpd_supports_native_leapfiles } }
  )
end

if node['ntp']['sync_clock'] && !platform_family?('windows')
  execute "Stop #{node['ntp']['service']} in preparation for ntpdate" do
    command node['platform_family'] == 'freebsd' ? '/usr/bin/true' : '/bin/true'
    action :run
    notifies :stop, "service[#{node['ntp']['service']}]", :immediately
  end

  execute 'Force sync system clock with ntp server' do
    command node['platform_family'] == 'freebsd' ? 'ntpd -q' : "ntpd -q -u #{node['ntp']['var_owner']}"
    action :run
    notifies :start, "service[#{node['ntp']['service']}]"
  end
end

execute 'Force sync hardware clock with system clock' do
  command 'hwclock --systohc'
  action :run
  only_if { node['ntp']['sync_hw_clock'] && !(platform_family?('windows') || platform_family?('freebsd')) }
end

service node['ntp']['service'] do
  supports status: true, restart: true
  action [:enable, :start]
  timeout 120 if platform_family?('windows')
  retries 3
  retry_delay 5
end
```

How to pull changes from server and execute chef-client automatically:

Use crontab to schedule the chef-client jobs periodically.

```
 bash-3.2# crontab -l
#------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  *  command to be executed
```

30 * * * * - will execute the command every 30min

To execute chef-client

```
30 * * * * /usr/bin/chef-client
```

Public and Private keys:

Server

public and private keys are generated when user and organization is created

```
[root@reviewb .chef]# pwd
/root/.chef
[root@reviewb .chef]# ls -lrt
total 8
-rw-r--r--. 1 root root 1674 Dec 12 21:31 santosh.pem
-rw-r--r--. 1 root root 1674 Dec 12 21:32 org-validator.pem
[root@reviewb .chef]#
```

Client

```
[root@ctx1p11 ~]# ls -lrt /etc/chef/
total 20
-rw-r--r--. 1 root root 1674 Jan 19  2038 org-validator.pem
-rw-r--r--. 1 root root 1674 Jan 19  2038 santosh.pem
-rw-------. 1 root root 1676 Jan 19  2038 client.pem
drwxr-xr-x. 2 root root   36 Jan 19  2038 trusted_certs
-rw-r--r--. 1 root root  203 Jan 19  2038 client.rb
-rw-r--r--. 1 root root   16 Jan 19  2038 first-boot.json
[root@ctx1p11 ~]#
```

validator.pem is created on the client after boot strap is completed

client.pem will be created during first chef-client run

Work station

```
[root@ctx1p10 chef]# ls -lrt

total 8

-rw-r--r--. 1 root root 1674 Dec 18 23:19 org-validator.pem

-rw-r--r--. 1 root root 1674 Dec 18 23:19 santosh.pem

[root@ctx1p10 chef]#
```

same files will be copied to ~/.chef

```
[root@ctx1p10 .chef]# ls -lrt

total 8

-rw-r--r--. 1 root root 1674 Dec 18 23:22 santosh.pem

-rw-r--r--. 1 root root 1674 Dec 18 23:22 org-validator.pem

[root@ctx1p10 .chef]#
```

validator key is generated on the server while creating organization

validator key is trusted by the server, used by client to authenticate for the first time

then it uses client.pem

private key is generated using the validator key on the client

once the custom client is generated validator key can be removed from client

Every request made by the chef-client to the Chef server must be an authenticated request using the Chef server API and a private key.

When the chef-client makes a request to the Chef server, the chef-client authenticates each request using a private key located in /etc/chef/client.pem.

However, during the first chef-client run, this private key does not exist. Instead, the chef-client will attempt to use the private key assigned to the chef-validator,

located in /etc/chef/validation.pem. (If, for any reason, the chef-validator is unable to make an authenticated request to the Chef server, the initial chef-client run will fail.)