

DEVOPS

NEW MATERIAL

VISUVALPATH

MANOJ XEROX

**Gayarti nager,Behind-Huda-Ameerpet
SOFT WARE INSTITUTES MATERIAL AVAILABLE**

CELL :9542556141

INDEX

1.ANSIBLE	5-56
2.ANT	57-84
3.CHEF	85-140
4.DOCKERS&CONTAINERS	141-162
5.GIT&GITHUB	163-204
6.JENKINS	205-226
7.LINUX BASICS &SHELL SCRIPTING	227-294
8.MAVEN	295-320
9.NAGIOS	321-342
10.PUPPET	343-382
11.PYTHON	383-466
12.VAGRANT	467-488

INDEX

Ansible

Ansible

Installation and configuration

Communication framework

Password less Authentication

Ansible command line

Ansible playbooks

Ansible roles

Ansible tags

Ansible Vault

Ansible variables- precedence

Ansible jinja2 templates

Ansible galaxy

Chef,Puppet,Ansible, Saltstack these tools are for the same purpose – configuration management.

System admin/Operational tasks

provision server

deploy

configure

Monitor

Scale

Secure

Application reliability and scalability directly affect the business profits

Difficult to manage thousands of servers.

As the application grows in functionality the day to day activities takes more time

We need to control and flexibility, as well as ease of use.

Automation:

Automation of these tasks is becoming extremely important, hence DevOps

Smart Automation is the key

Recognize the most common tasks and automate

-Defined way of doing things is Recipes

In today's world everything is code

kernel

OS

shell

Application

Configuration management tools

Provisioning resources

Ansible

Ansible in short is an IT Automation, configuration management and provisioning tool.

Simple to start

Scales up on demand

No agents are needed

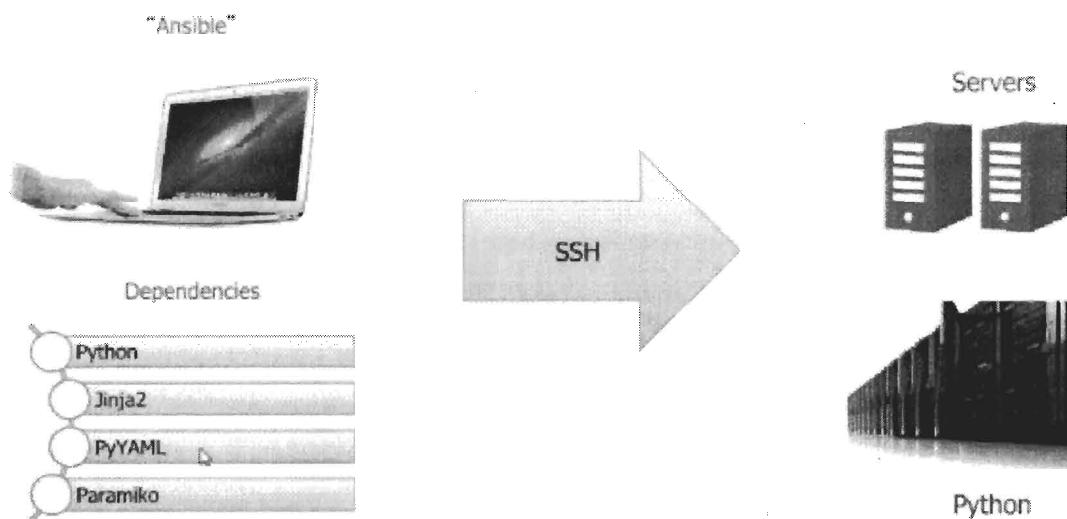
sshd on remote machine can be used

Parallel by default

Automaton language is like English

212 core modules

Ansible is a tool to manage systems and their configuration. Without the need for a client installed agent and with the ability to launch programs with command line, it seems to fit between classic configuration management like Puppet on one hand and ssh/dsh on the other.



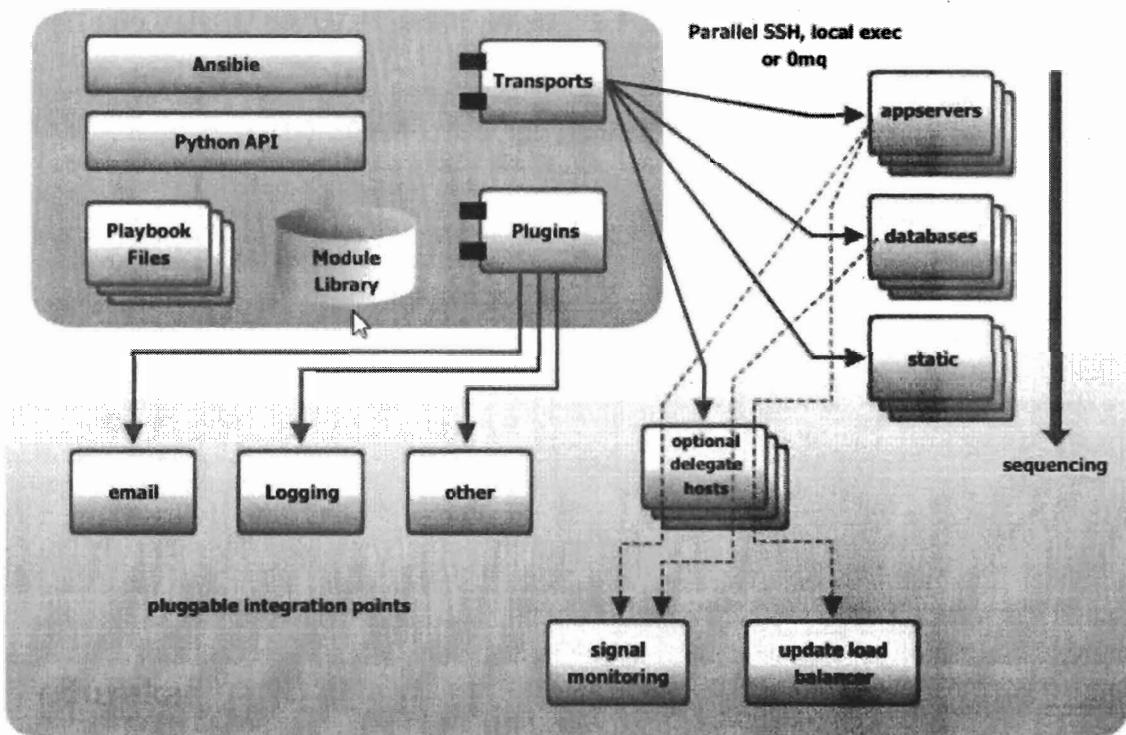
Ansible based on python

Jinja2 supports templates for Ansible – templating engine

PyYAML – yet another markup language – playbooks are written in this language

Paramiko – kind of ssh implementation.

Ansible Architecture



Playbook files – infrastructure as code is maintained in this files

Python API – execute commands using API

Module – is a resource – file/package/command/item trying to configure

transports – by default ssh – to talk to servers

plugins – email logging other

playbooks are executed on parallel fashion

Ansible terminology

Inventory: file contains list of hosts, groups or variables.

Modules: Actually do the work

Plugins: call back actions, and other hooks

Facts: Data gathered from target hosts

Playbooks: Collection of plays

Plays: Loops over a list of tasks mapped to list of hosts

Tasks: Invokes a module to do the work

Ansible Install:

ssh-key has to setup on both the nodes

Ansible server talks to managed nodes using ssh

Default location of inventory: /etc/ansible/hosts

Its in INI format

Ansible repository Ansible Galaxy

How to use Ansible

```
ansible inventory -m modulename
```

```
ansible-playbook
```

```
Ansible-playbook playbook.yml
```

Ansible command mode:

add hosts to /etc/ansible/hosts

and configure password less authentication

Ansible needs to know the hosts its going to serve. They can be managed on the central server in /etc/ansible/hosts or in a file configured in the shell variable ANSIBLE_HOSTS. The hosts can be listed as IP addresses or host names, and can contain additional information like user names, ssh port and so on:

Installation and configuration:

1. Install ansible on the node using

```
yum install ansible
```

2. Add nodes to ansible server using inventory file

```
/etc/ansible/hosts
```

3. Generate ssh keys and setup password less authentication between server and clients

4. Perform jobs either using ansible command line or playbooks.

Ansible command line:

```
[root@ctx1p05 RHEL5]# ansible all -m ping
```

```
ctx1p10.in.vp.com | SUCCESS => {
```

```
    "changed": false,
```

```
"ping": "pong"  
}  
  
ctx1p11.in.vp.com | SUCCESS => {  
    "changed": false,  
    "ping": "pong"
```

[root@ctx1p05 RHEL5]# ansible all -a "touch /tmp/hello"

ctx1p10.in.vp.com | SUCCESS | rc=0 >>

ctx1p11.in.vp.com | SUCCESS | rc=0 >>

to execute command on local machine (ansible server)

ansible all -i "localhost," -c local -m shell -a 'echo hello world'

to execute command on remote single machine (ansible client)

ansible all -i "ctx1p10" -m shell -a 'echo hello world'

Grouping the nodes:

Nodes can be grouped into groups like webserver dbserver application server etc in hosts file

[root@ctx1p05 RHEL5]# ansible webservers -m ping

ctx1p10.in.vp.com | SUCCESS => {

"changed": false,

```
"ping": "pong"  
}  
  
[root@ctx1p05 RHEL5]#
```

[webservers]

ctx1p10.in.vp.com

[dbservers]

ctx1p11.in.vp.com

to execute a command on local/ansible server

```
ansible all -i "localhost," -c local -m shell -a 'hostname'
```

```
root@ctx1p05 ~]# ansible all -i "localhost," -c local -a 'hostname'
```

```
localhost | SUCCESS | rc=0 >>
```

ctx1p05.in.vpath.com

to execute command on single host

```
[root@ctx1p05 ~]# ansible all -i "reviewb.in.vpath.com," -a 'hostname'
```

```
reviewb.in.vpath.com | SUCCESS | rc=0 >>
```

reviewb.in.vpath.com

Ansible Playbooks:

A sample playbook:

```
[root@ctx1p05 playbooks]# cat httpd.yaml
```

```
---
```

```
- hosts: webservers
```

```
tasks:
```

```
- name: install httpd
```

```
yum: pkg=httpd state=installed
```

```
[root@ctx1p05 playbooks]#
```

```
[root@ctx1p05 playbooks]# ansible-playbook httpd.yaml
```

```
PLAY [webservers] ****
```

```
TASK [setup] ****
```

```
ok: [ctx1p10.in.vp.com]
```

```
TASK [install httpd] ****
changed: [ctx1p10.in.vp.com]
```

PLAY RECAP

```
*****
ctx1p10.in.vp.com      : ok=2  changed=1  unreachable=0  failed=0
```

```
[root@ctx1p05 playbooks]#
```

Playbooks:

hosts defined on which target this configuration is pushed

user: to login to remote server

vars:

http_port:80

max_client: 200

Variables are declared and can be passed , wherever http_port is used, 80 will be replaced

Tasks: collection of actions we need to perform

each action will have a name

action will call the module

name: ensure apache is at the latest version

action: yum pkg=httpd state=latest

here yum is the module for rhel

we are passing some variables to yum

1st one is package name, 2nd one is action to install/remove/latest

for specific version we need to replace state with version

action: template src=httpd.j2 dest=/etc/httpd.conf

we want to create a file /etc/httpd.conf with template file httpd.j2

here we are using template

notify:

- restart apache

Whenever you perform this action (when template is notified), this action has to be performed

-name ensure httpd is running

action=service name =httpd state=started

handlers:

-name: restart apache

action: service name=httpd state=restarted

to understand handles, execute below playbook 2 times

```
[root@ctx1p05 playbooks]# cat template.yaml
```

- hosts: webservers

tasks:

- name: copy application code to destination on clients

```
template: src=index.html.j2 dest=/var/www/html/index.htm  
notify: restart apache  
  
handlers:  
  
- name: restart apache  
  
service: name=httpd state=restarted
```

1st run

```
[root@ctx1p05 playbooks]# ansible-playbook template.yaml
```

```
PLAY [webservers] ****
```

```
TASK [setup] ****
```

```
ok: [ctx1p11.in.ibm.com]
```

```
TASK [copy application code to destination on clients] ****
```

```
ok: [ctx1p11.in.ibm.com]
```

PLAY RECAP

```
*****
```

```
ctx1p11.in.ibm.com : ok=2  changed=0  unreachable=0  failed=0
```

2nd run:

Modify something in index.html on ansible master and deploy

```
[root@ctx1p05 playbooks]# ansible-playbook template.yaml
```

```
PLAY [webservers] *****
```

```
TASK [setup] *****
```

```
ok: [ctx1p11.in.ibm.com]
```

```
TASK [copy application code to destination on clients] *****
```

```
changed: [ctx1p11.in.ibm.com]
```

```
RUNNING HANDLER [restart apache] *****
```

```
changed: [ctx1p11.in.ibm.com]
```

PLAY RECAP

```
*****
```

```
ctx1p11.in.ibm.com      : ok=3  changed=2  unreachable=0  failed=0
```

Note: if there is any change only handles will be used to restart service

To list all tasks in a module

```
[root@ctx1p05 playbooks]# ansible-playbook apache.yaml --list-tasks
```

```
playbook: apache.yaml
```

```
play #1 (webservers): webservers    TAGS: []
```

tasks:

```
ensure apache is installed    TAGS: []
```

```
ensure apache is running  TAGS: []
```

```
copy files to document root    TAGS: []
```

```
copy application code to document root  TAGS: []
```

```
[root@ctx1p05 playbooks]#
```

```
[root@ctx1p05 playbooks]# ansible-playbook apache.yaml --list-tasks
```

```
playbook: apache.yaml
```

```
play #1 (webservers): webservers    TAGS: []
```

tasks:

```
ensure apache is installed    TAGS: []
```

```
ensure apache is running  TAGS: []
```

```
copy files to document root    TAGS: []
```

```
copy application code to document root  TAGS: []
```

```
[root@ctx1p05 playbooks]# ansible-playbook --check apache.yaml
```

```
PLAY [webservers] ****
```

TASK [setup] *****

ok: [ctx1p10.in.ibm.com]

TASK [ensure apache is installed] *****

ok: [ctx1p10.in.ibm.com]

TASK [ensure apache is running] *****

ok: [ctx1p10.in.ibm.com]

TASK [copy files to document root] *****

ok: [ctx1p10.in.ibm.com]

TASK [copy application code to document root] *****

ok: [ctx1p10.in.ibm.com]

PLAY RECAP

ctx1p10.in.ibm.com : ok=5 changed=0 unreachable=0 failed=0

ansible-playbook php.yaml --^Cmit @/etc/ansible/playbooks/php.retry

to retry a failed playbook execution

```
[root@ctx1p05 playbooks]# ansible-playbook php.yaml --limit  
@/etc/ansible/playbooks/intaractive.retry
```

```
PLAY [webservers] *****
```

```
TASK [setup] *****
```

```
ok: [ctx1p10.in.ibm.com]
```

```
TASK [xyz] *****
```

```
changed: [ctx1p10.in.ibm.com]
```

```
PLAY RECAP
```

```
*****
```

```
ctx1p10.in.ibm.com : ok=2  changed=1  unreachable=0  failed=0
```

Online Ansible Playbook repository : Ansible – galaxy

Ansible tower – licensed version of Ansible

Ansible is not supported on Windows, its supported only on Linux

in Ansible modules are tools

playbooks are instruction manuals

5 basic components in Ansible

1)Inventory:

A text file that contains information of servers and systems in the infrastructure

Group hosts into categories

Define any variables required by specific hosts (ex: ssh username and password)

2)Ansible configuration:

custom configuration can be defined here

3)Modules

It is the command center of the system

2 types

core modules – developed by Ansible

other modules – developed by community

4)Playbooks

Play: It is a single task or set of tasks which can be executed on hosts with the help of modules

it Is set of plays built in specific order sequence to produce an expected outcome

5) Ansible global configuration

It is a file that contains global variables

execution types:

remote

local : when python is not installed on the agent

ex: router

Handles: wait for event to occur

once event occurs with notify action, handler which is mapped that event will gets executed

hosts: webservers

user: root

example playbooks

```
[root@ctx1p05 playbooks]# cat apache.yaml
```

```
---
```

```
- hosts: webservers
```

```
    tasks:
```

```
        - name: ensure apache is installed
```

```
            yum: pkg=httpd state=latest
```

```
        - name: ensure apache is running
```

```
            service: name=httpd state=running enabled=yes
```

```
        - name: copy files to document root
```

```
            copy: src=cloud.png dest=/var/www/html/cloud.png
```

```
- name: copy application code to document root
  template: src=index.html.j2 dest=/var/www/html/index.html
  notify: restart apache
handlers:
- name: restart apache
  service: name=httpd state=restarted
[root@ctx1p05 playbooks]#
```

```
[root@ctx1p05 playbooks]# cat index.html.j2
```

```
<head>
  <title>Vp Ansible Demo</title>
</head>
<body>
  <h1>
    Welcome to Visual path Ansible demo.
  </h1>
  <br/><br/><br/>
  <p>
    
  </p>
</body>
```

```
</html>
```

```
[root@ctx1p05 playbooks]#
```

```
roles:
```

```
[root@ctx1p05 playbooks]# cat myrole.yaml
```

```
---
```

```
- hosts: webservers
```

```
  roles:
```

```
    - webservers
```

```
[root@ctx1p05 playbooks]#
```

```
include:
```

```
[root@ctx1p05 playbooks]# cat myrole.yaml
```

```
---
```

```
- hosts: webservers
```

```
  roles:
```

```
    - webservers
```

```
[root@ctx1p05 playbooks]# cat sample.yaml
```

```
---
```

```
- hosts: webservers
```

```
  tasks:
```

```
    - include: /etc/ansible/tasks/file.yaml
```

```
- hosts: dbservers
  tasks:
    - include: /etc/ansible/tasks/file.yaml
```

```
[root@ctx1p05 playbooks]#
```

include is for making the code look simpler, manageable

small playbooks can be created and can be included

To see what hosts would be affected by a playbook before running it, try

```
root@ctx1p05 playbooks]# ansible-playbook apache.yaml --list-hosts
```

```
playbook: apache.yaml
```

```
play #1 (webservers): webservers    TAGS: []
```

```
pattern: [u'webservers']
```

```
hosts (1):
```

```
ctx1p10.in.vp.com
```

Ansible all –list-hosts

shows the nodes

Ansible all –s –m shell –a “mount –a”

installed – will install

latest - will upgrade if available

absent – will remove

with –check option – dry run method, will not do changes to machine

deprication_warnings = make it to no in cfg

-v is for verbose

--vv explain completely

--vvv full verbose

tags

to execute based on the task

with_loop

to execute in a loop

Interactive play book

include:

```
[root@ctx1p05 playbooks]# cat sample.yaml
```

- hosts: webservers

tasks:

```
- include: /etc/ansible/tasks/file.yaml
```

- hosts: dbservers

tasks:

```
- include: /etc/ansible/tasks/file.yaml
```

Roles:

Yaml file will grow with time as and when the tasks are increased

Directory structures and templates

```
[root@ctx1p05 playbooks]# cat myrole.yaml
```

```
---
```

```
- hosts: webservers
```

```
  roles:
```

```
    - webservers
```

```
[root@ctx1p05 playbooks]#
```

Using ansible facts to get os family

```
---
```

```
- hosts: ext1
```

```
  tasks:
```

```
    - name: install httpd
```

```
      yum: name=httpd state=present
```

```
      when: ansible_os_family == "RedHat"
```

```
    - name: install httpd
```

```
      apt: name=httpd state=present
```

```
      when: ansible_os_family == "Debian"
```

```
[root@ctx1p05 playbooks]#
```

Getting info about the facts

```
ansible ext1 -m setup --tree /tmp/facts
```

using user module to add user

```
ansible webservers -s -m user -a "name=user1 uid=20000 shell=/bin/bash"
```

including multiple roles in playbook:

```
[root@reviewb playbooks]# cat role.yaml
```

```
- hosts: webservers
```

```
roles:
```

```
  - common
```

```
  - webservers
```

```
[root@reviewb playbooks]#
```

```
[root@reviewb playbooks]# cat /etc/ansible/roles/common/tasks/main.yaml
```

```
- name: create a dummy file for common role
```

```
  command: touch /tmp/dummy2role
```

```
[root@reviewb playbooks]# cat /etc/ansible/roles/webservers/tasks/main.yaml
```

```
- name: ensure apache is installed
```

```
  yum: pkg=httpd state=latest
```

```
- name: ensure apache is running
```

```
  service: name=httpd state=running enabled=yes
```

```
- name: copy files to document root
```

```
  copy: src=cloud.png dest=/var/www/html/cloud.png
```

```
- name: copy application code to document root
```

```
template: src=index.html.j2 dest=/var/www/html/index.html

[root@reviewb playbooks]# cat /etc/ansible/roles/webservers/templates/index.html.j2

<head>
  <title> Ansible </title>
</head>

<body>
  <h1>
    Welcome to Ansible Roles  </h1>
  <br/><br/><br/>
  <p>
    
  </p>
</body>
</html>
```

parallel vs serial execution:

change forks=15 in cfg file

```
[root@ctx1p05 playbooks]# cat parallel.yaml
```

Parallel execution:

```
---
```

```
- hosts: all
```

```
  tasks:
```

```
    - name: sleep for 5 seconds
```

```
shell: /bin/sleep 5
```

Serial execution:

```
[root@ctx1p05 playbooks]# cat serial.yaml
```

```
---
```

```
- hosts: all
```

```
serial: 1
```

```
tasks:
```

```
- name: sleep for 5 seconds
```

```
shell: /bin/sleep 5
```

```
[root@ctx1p05 playbooks]#
```

Serial:1 is for executing the playbook on single machine (one by one)

Serial:2 will execute on each of 2 machines (first 2 machines parallelly, then goes to third,forth)

pre and post tasks execution

```
[root@ctx1p05 playbooks]# cat preposttask.yaml
```

```
---
```

```
- hosts: webservers
```

```
serial: 1
```

```
pre_tasks:
```

```
- name: install package
```

```
yum: name=httpd state=installed
```

```
- name: copy new template
```

```
copy: src=/index1.html dest=/var/www/html/index.html
- name: restart http
  service: name=httpd state=restarted
- name: sleep for 5 sec
  shell: /bin/sleep 5
tasks:
- name: deploy website content
  copy: src=/index2.html dest=/var/www/html/index.html
- name: restart http
  service: name=httpd state=restarted
post_tasks:
- name:
  service: name=httpd state=restarted
[root@ctx1p05 playbooks]#
```

Roles:

Roles define what each type of server has to perform

Sample:

```
- name: install and configure webservers
```

```
hosts: webservers
```

```
remote_user: ec2-user
```

```
sudo: yes
```

roles:

```
- webservers
```

```
- common
```

Here common and webservers role is associated with hosts webservers

```
[root@ctx1p05 playbooks]# cat myrole.yaml
```

```
---
```

```
- hosts: webservers
```

```
  roles:
```

```
    - webservers
```

```
[root@ctx1p05 playbooks]# cat /etc/ansible/roles/webservers/tasks/main.yaml
```

```
- name: ensure apache is installed
```

```
  yum: pkg=httpd state=latest
```

```
- name: ensure apache is running
```

```
  service: name=httpd state=running enabled=yes
```

```
- name: copy files to document root
```

```
  copy: src=cloud.png dest=/var/www/html/cloud.png
```

```
- name: copy application code to document root
```

```
  template: src=index.html.j2 dest=/var/www/html/index.html
```

```
  notify: restart apache
```

Playbook to create users

```
[root@ctx1p05 playbooks]# cat create.yaml
```

```
---
```

```
- hosts: webservers
```

tasks:

```
- name: add several users  
  user: name={{ item.name }} state=present groups={{ item.groups }}  
  with_items:  
    - { name: 'testuser1', groups: 'root' }  
    - { name: 'testuser2', groups: 'root' }
```

[root@ctx1p05 playbooks]#

Playbook to explain notify:

imagine a setup where you copy a file, and if that file was copied (so not there before or changed in the meantime) , we need to restart sshd:

- hosts: webservers

remote_user: liquidat

tasks:

```
- name: copy file  
  copy: src=~/tmp/test.txt dest=~/test.txt
```

notify:

```
  - restart sshd
```

handlers:

```
- name: restart sshd  
  service: name=sshd state=restarted  
  sudo: yes
```

Using Inventory file to provide the variables/values:

```
[root@ctx1p05 playbooks]# cat inventory
web1 ansible_ssh_host=9.184.184.145 ansible_ssh_user=john ansible_ssh_pass=root123
[root@ctx1p05 playbooks]# ansible web1 -i inventory -a "touch /tmp/hello15thfeb2017"
web1 | SUCCESS | rc=0 >>
```

```
[root@ctx1p05 playbooks]# cat echo2.yaml
```

```
---
- hosts: all
  tasks:
    - name: run echo command
      command: /usr/bin/touch /tmp/hellofeb
```

```
ansible-playbook -i inventory echo2.yaml
```

Ansible Vault

Installing Ansible vault:

Comes by default with ansible

From Ansible 1.5, “Vault” is a feature that allows keeping sensitive data such as passwords or keys in encrypted files, rather than as plaintext in your playbooks or roles. These vault files can then be distributed or placed in source control.

How to encrypt data

Encrypt files with ansible vault

- AES 256 encryption

Ansible will decrypt at the runtime

To encrypt the passwords for privacy ansible vault can be used

ansible-vault encrypt intaractive.yaml

ansible-playbook intaractive.yaml --ask-vault-pass

```
[root@ctx1p05 playbooks]# ansible-playbook intaractive.yaml --ask-vault-pass
```

Vault password:

Install which package? [telnet]:

```
PLAY [webservers] ****
```

```
TASK [setup] ****
```

ok: [ctx1p11.in.ibm.com]

```
TASK [ensure apache is installed] ****
```

ok: [ctx1p11.in.ibm.com]

```
PLAY RECAP
```

```
*****
```

```
ctx1p11.in.ibm.com : ok=2  changed=0  unreachable=0  failed=0
```

ansible-vault encrypt <> - to encrypt the file

ansible-valut edit <> - to edit the file

ansible-vault rekey – to change password

Ansible until loop:

Sometimes we may want to retry a task until a certain condition is met. Here's an example:

```
[root@ctx1p05 playbooks]# cat until.yaml
```

```
- hosts: webservers
```

```
tasks:
```

```
  - action: shell ifconfig
```

```
    register: result
```

```
    until: result.stdout.find("ens192") != -1
```

```
    retries: 5
```

```
    delay: 10
```

```
[root@ctx1p05 playbooks]# ansible-playbook until.yaml
```

```
PLAY [webservers] ****
```

```
TASK [setup] ****
```

```
ok: [ctx1p11.in.ibm.com]
```

```
TASK [command]
```

```
****
```

```
changed: [ctx1p11.in.ibm.com]
```

```
PLAY RECAP
```

```
****
```

```
ctx1p11.in.ibm.com      : ok=2  changed=1  unreachable=0  failed=0
```

```
[root@ctx1p05 playbooks]#
```

The above example run the shell module recursively till the module's result has "all systems go" in its stdout or the task has been retried for 5 times with a delay of 10 seconds. The default value for "retries" is 3 and "delay" is 5.

The task returns the results returned by the last task run. The results of individual retries can be viewed by -vv option. The registered variable will also have a new key "attempts" which will have the number of the retries for the task.

Ansible if loop:

Jinja2 template language can be used to populate the template files on the destination with loops as below:

If loop

For loop

If loop can be used to populate the destination template file using input variables we specify during playbook execution.

```
[root@ctx1p05 role1]# pwd  
/etc/ansible/roles/role1  
[root@ctx1p05 role1]# ls -lrt  
total 0  
drwxr-xr-x. 2 root root 21 Feb 17 02:37 templates  
drwxr-xr-x. 2 root root 21 Feb 17 02:42 vars  
[root@ctx1p05 role1]#
```

```
[root@ctx1p05 role1]# cd templates/
[root@ctx1p05 templates]# ls -lrt
total 4
-rw-r--r--. 1 root root 295 Feb 17 02:37 myvar.j2
[root@ctx1p05 templates]# cat myvar.j2
{% if param1 is defined and param2 is defined and param3 is defined %}
myvariable: 'param1,param2,param3'

{% elif param1 is defined and param2 is defined %}
myvariable: 'param1,param2'

{% elif param1 is defined %}
myvariable: 'param1'

{% else %}
myvariable: 'default-param'

{% endif %}

[root@ctx1p05 templates]#
```

```
[root@ctx1p05 role1]# cd vars/
[root@ctx1p05 vars]# ls
main.yml

[root@ctx1p05 vars]# cat main.yml
myvariable: 'param1,param2'

[root@ctx1p05 vars]#
```

```
[root@ctx1p05 playbooks]# cat role1.yml
```

```
- hosts: xyz  
gather_facts: yes  
# connection: local  
become: yes
```

tasks:

```
- template: src=/etc/ansible/roles/role1/templates/myvar.j2  
dest=/etc/ansible/roles/role1/vars/main.yml
```

```
- debug: var=myvariable
```

roles:

```
- { role: role1 }
```

```
[root@ctx1p05 playbooks]#
```

```
[root@ctx1p05 playbooks]# ansible-playbook role1.yml -e "param1=value1 param2=value2  
param3=value3"
```

```
PLAY [xyz] ****
```

```
TASK [setup] ****
```

```
ok: [ctx1p18.in.ibm.com]
```

```
TASK [template] ****
```

```
changed: [ctx1p18.in.ibm.com]
```

```
TASK [debug]
```

```
*****
```

```
ok: [ctx1p18.in.ibm.com] => {
```

```
    "myvariable": "param1,param2"
```

```
}
```

```
PLAY RECAP
```

```
*****
```

```
ctx1p18.in.ibm.com : ok=3  changed=1  unreachable=0  failed=0
```

On the destination machine:

```
root@ctx1p18:/etc/ansible/roles/role1/vars# cat main.yml
```

```
myvariable: 'param1,param2,param3'
```

example 2:

```
[root@ctx1p05 playbooks]# ansible-playbook role1.yml
```

```
PLAY [xyz] ****
```

```
TASK [setup] ****
```

```
ok: [ctx1p18.in.ibm.com]
```

```
TASK [template] ****
```

```
changed: [ctx1p18.in.ibm.com]
```

```
TASK [debug]
```

```
*****
```

```
ok: [ctx1p18.in.ibm.com] => {
```

```
    "myvariable": "param1,param2"
```

```
}
```

```
PLAY RECAP
```

```
*****
```

```
ctx1p18.in.ibm.com      : ok=3  changed=1  unreachable=0  failed=0
```

```
[root@ctx1p05 playbooks]#
```

On destination machine:

```
root@ctx1p18:/etc/ansible/roles/role1/vars# cat main.yml
```

```
myvariable: 'default-param'
```

Template if loop:

```
[root@ctx1p05 playbooks]# pwd
```

```
/etc/ansible/playbooks
```

```
[root@ctx1p05 playbooks]# cat role2.yml
```

```
- hosts: xyz
```

```
gather_facts: yes
```

```
become: yes
```

```
tasks:
```

```
- template: src=/etc/ansible/roles/role2/templates/myvar.j2  
dest=/etc/ansible/roles/role2/vars/main.yml
```

```
- debug: var=myvariable
```

```
roles:
```

```
- { role: role2 }
```

```
[root@ctx1p05 playbooks]#
```

```
[root@ctx1p05 role2]# pwd
```

```
/etc/ansible/roles/role2
```

```
[root@ctx1p05 role2]# ls -rlt
```

```
total 0
```

```
drwxr-xr-x. 2 root root 21 Feb 17 05:47 vars
```

```
drwxr-xr-x. 2 root root 21 Feb 17 06:00 templates
```

```
[root@ctx1p05 role2]# cat templates/myvar.j2
```

```
Tower name is: {{ hostvars['ctx1p18.in.ibm.com']['ansible_hostname'] }}
```

```
The epochs of the clients are:
```

```
{% for host in groups['xyz'] %}
```

```
- {{ hostvars[host]['ansible_date_time']['epoch'] }}
```

```
{% endfor %}
```

```
[root@ctx1p05 role2]#
```

```
[root@ctx1p05 playbooks]# ansible-playbook role2.yml
```

```
PLAY [xyz] ****
```

```
TASK [setup] ****
```

```
ok: [ctx1p18.in.ibm.com]
```

```
TASK [template] ****
```

```
changed: [ctx1p18.in.ibm.com]
```

```
TASK [debug]
```

```
*****
```

```
ok: [ctx1p18.in.ibm.com] => {
```

```
    "myvariable": "param1,param2"
```

```
}
```

```
PLAY RECAP
```

```
*****
```

```
ctx1p18.in.ibm.com      : ok=3  changed=1  unreachable=0  failed=0
```

```
[root@ctx1p05 playbooks]#
```

```
root@ctx1p18:/etc/ansible/roles/role1/vars# cat /etc/ansible/roles/role2/vars/main.yml
```

Tower name is: ctx1p18

The epochs of the clients are:

- 1487329302

Writing variables to index.html using roles:

```
[root@ctx1p05 playbooks]# cat /etc/ansible/playbooks/role3.yml
```

- hosts: webservers

vars:

http_port: 80

company: example.com

roles:

- { role: role3 }

```
[root@ctx1p05 playbooks]# cat /etc/ansible/roles/role3/tasks/main.yml
```

- name: Copy index file.

template:

```
src: /etc/ansible/roles/role3/templates/myvar.j2  
dest: /var/www/html/index.html
```

```
[root@ctx1p05 playbooks]# cat /etc/ansible/roles/role3/templates/myvar.j2
```

```
<head>  
  <title> Ansible </title>  
</head>  
<body>  
  <h1>  
    Welcome to Ansible {{ http_port }} {{ company }}  
  </h1>  
  <br/><br/><br/>  
  <p>  
      
  </p>  
</body>  
</html>
```

How to use facts variables inside template:

Get the facts from the below command:

```
ansible -m setup ctx1p11.in.ibm.com
```

example:

```
ctx1p11.in.ibm.com | SUCCESS => {  
    "ansible_facts": {  
        "ansible_all_ipv4_addresses": [  
            "9.182.76.101"  
        ],  
        "ansible_all_ipv6_addresses": [  
            "fe80::d039:46f6:236:45f4"  
        ],  
        "ansible_architecture": "x86_64",  
        "ansible_bios_date": "10/13/2009",  
        "ansible_bios_version": "6.00",  
        "ansible_cmdline": {  
            "BOOT_IMAGE": "/vmlinuz-3.10.0-514.el7.x86_64",  
            "LANG": "en_US.UTF-8",  
            "crashkernel": "auto",  
            "quiet": true,  
            "rd.lvm.lv": "rhel/swap",  
            "rhgb": true,  
            "ro": true,  
            "root": "/dev/mapper/rhel-root"  
        }  
    }  
}
```

These variables can be directly used in templates

Example: To display on site using index.html

Galaxy:

Download a role nfs server setup and use it to export a file system

Download the role and copy , extract it to roles

Create a playbook

```
[root@ctx1p05 playbooks]# cat nfs.yaml
```

```
- hosts: webservers
```

```
roles:
```

```
  - ansible-role-nfs-master
```

Go thr readme

Provide directory name in defaults/main.yml

```
[root@ctx1p05 ansible-role-nfs-master]# cat defaults/main.yml
```

```
nfs_exports: [/abc]
```

```
nfs_exports: { "/home/public *(rw,sync,no_root_squash)" }
```

Ansible galaxy role mysql

Download the cookbook

Check the compatible versions/OS matrix

Download and create a playbook and use it to install mysql

geerlingguy.mysql

ansible-galaxy install geerlingguy.mysql

Ansible variable precedence:

Variables precedence or priority is in this order:

Variables declared in playbooks along with role

Variables declared in vars directory in the role

Variables defined in defaults

How to declare variables in playbook

Example 1:

```
[root@ctx1p05 playbooks]# cat role3.yml-bk
```

```
- hosts: webservers
```

```
vars:
```

```
    http_port: 80
```

```
    company: example.com
```

```
roles:
```

```
    - { role: role3 }
```

```
[root@ctx1p05 role3]# cat templates/myvar.j2
```

```
<head>
```

```
    <title> Ansible </title>
```

```
</head>

<body>
    <h1>
        Welcome to Ansible ROLES
    </h1>
    <h1>
        Welcome to {{ company }}
    </h1>
    <h1>
        Port is {{ port_num }}
    </h1>

    <br/><br/><br/>

    <p>
        
    </p>
</body>

[root@ctx1p05 role3]# cat tasks/main.yml

---
- name: Copy index file.
  template:
    src: /etc/ansible/roles/role3/templates/myvar.j2
    dest: /var/www/html/index.html
```

Example2:

```
[root@ctx1p05 playbooks]# cat role3.yml
```

```
- hosts: webservers
  roles:
    - role3

[root@ctx1p05 playbooks]# cd /etc/ansible/roles/role3/
[root@ctx1p05 role3]# ls
tasks templates vars

[root@ctx1p05 role3]# cat vars/main.yml
port_num: 8081
company: XYZ123

[root@ctx1p05 role3]# cat /etc/ansible/playbooks/role3.yml
- hosts: webservers
  roles:
    - { role: role3, port_num: 9090 } ➔ this has the highest precedence

[root@ctx1p05 role3]# cat vars/main.yml
port_num: 8081
company: XYZ123

[root@ctx1p05 role3]# cat templates/myvar.j2
<head>
  <title> Ansible </title>
</head>
<body>
  <h1>
    Welcome to Ansible ROLES
  </h1>
```

```
</h1>

<h1>
    Welcome to {{ company }}
</h1>

<h1>
    Port is {{ port_num }}
</h1>

<h1>
    My host name is {{ hostvars[groups['webservers'][0]].ansible_default_ipv4 }}
</h1>

<br/><br/><br/>

<p>
    
</p>

</body>
```

File module to create a file/directory

```
[root@ctx1p05 playbooks]# cat file.yml
```

```
---
```

```
- hosts: webservers
```

```
tasks:
```

```
  - name: create a directory
```

file:

path: /etc/xyz

state: directory

mode: 755

if state=directory → creates directory

if nothing is given -> creates file

Register module to store value from command line:

```
[root@ctx1p05 playbooks]# cat register.yml
```

- hosts: webservers

tasks:

- shell: cat /etc/motd

register: motd_contents

- name: motd contains the word hi

shell: echo "motd contains the word hi"

when: motd_contents.stdout.find('hi') != -1

```
[root@ctx1p05 playbooks]# ansible-playbook register.yml
```

```
PLAY [webservers] ****
```

```
TASK [setup] ****
```

```
ok: [ctx1p11.in.ibm.com]
```

```
TASK [command]
```

```
****
```

```
changed: [ctx1p11.in.ibm.com]
```

```
TASK [motd contains the word hi] ****
```

```
changed: [ctx1p11.in.ibm.com]
```

```
PLAY RECAP
```

```
****
```

```
ctx1p11.in.ibm.com : ok=3  changed=2  unreachable=0  failed=0
```

Ansible cheatsheat

ANSIBLE CHEAT SHEET

\$ansible

- call a shell command:
-a "\$command"
- call with sudo rights:
--sudo
- test if machine responds:
-m ping
- call an arbitrary module:
-m \$module -a "\$argument"
- gather specific facts:
-m setup -a "filter='*distri*'"

\$ansible-playbook

- call a book:
\$host some/playbook.yaml
- run in parallel:
-f 10
- list affected hosts:
--list-hosts"

basic playbook.yaml

YAML format:

- hosts:all
- launch task:
 - name: check avail
ping:
- variables:
 - vars:
 - rack: green
- include tasks:
 - tasks:
 - { include: tasks/my.yml, user=alice }
- include handlers:
 - handlers:
 - { include: handlers/my.yml }
- call dependency:
 - tasks:
 - name: copy file
copy: src=a dest=b
notify:
 - restart sshd
 - handlers:
 - name: restart sshd

inventory

- general inventory:
/etc/ansible/hosts
- shell variable:
\$ANSIBLE_HOSTS
- normal entry:
www.example.com
- multi-definition:
db[0-9].example.com
- custom ip:
ansible_ssh_host
- grouping:
[group]
- specific user:
ansible_ssh_user
- specific port:
ansible_ssh_port

complex playbook.yaml

- loop over items:
 - name: copy file
copy: src={{item.src}}
dest={{item.dst}}
- with_items:
 - {src: a, dst: b}
 - {src: k, dst: l}
- conditionals:
 - name: reboot Debian
command: shutdown -r now
when: ansible_os_family == "Debian"
- Roles:
 - hosts: webserver
roles:
 - common
 - dbservers
 - roles/
 - common/
files/
templates/
tasks/main.yml
handlers/main.yml
vars/main.yml
meta/main.yml
webservers/
...

ANT

ANT is Another neat tool

It is Build Automation tool

It is written in java, uses xml configuration file

Compiling java classes

Package classes into jar files

Need for a Build Tool

On an average, a developer spends a substantial amount of time doing mundane tasks like build and deployment that include:

Compiling the code

Packaging the binaries

Deploying the binaries to the test server

Testing the changes

Copying the code from one location to another

To automate and simplify the above tasks, Apache Ant is useful. It is an Operating System build and deployment tool that can be executed from the command line.

Ant was created by James Duncan Davidson (the original author of Tomcat).

It was originally used to build Tomcat, and was bundled as a part of Tomcat distribution.

Ant was born out of the problems and complexities associated with the Apache Make tool.

Ant was promoted as an independent project in Apache in the year 2000. The latest version of Apache Ant as on May 2014 is 1.9.4.

Features of ANT:

Ant is the most complete Java build and deployment tool available.

Ant is platform neutral and can handle platform specific properties such as file separators.

Ant can be used to perform platform specific tasks such as modifying the modified time of a file using 'touch' command.

Ant scripts are written using plain XML. If you are already familiar with XML, you can learn Ant pretty quickly.

Ant is good at automating complicated repetitive tasks.

Ant comes with a big list of predefined tasks.

Ant provides an interface to develop custom tasks.

Ant can be easily invoked from the command line and it can integrate with free and commercial IDEs.

Installing Apache Ant

Install JDK first

Ensure that the JAVA_HOME environment variable is set to the folder where your JDK is installed.

Download the binaries from <http://ant.apache.org>

Unzip the zip file to a convenient location c:\folder. using Winzip, winRAR, 7-zip or similar tools.

Create a new environment variable called ANT_HOME that points to the Ant installation folder, in this case c:\apache-ant-1.8.2-bin folder.

Append the path to the Apache Ant batch file to the PATH environment variable. In our case this would be the c:\apache-ant-1.8.2-bin\bin folder.

Verifying Apache Ant Installation

To verify the successful installation of Apache Ant on your computer, type ant on command prompt.

```
C:\>ant -version
```

```
Apache Ant(TM) version 1.8.2 compiled on December 20 2010
```

Ant's build file, called build.xml should reside in the base directory of the project. However there is no restriction on the file name or its location. You are free to use other file names or save the build file in some other location.

Skeleton of xml :

```
<?xml version="1.0"?>

<project name="Hello World Project" default="info">

<target name="info">

    <echo>Hello World - Welcome to Apache Ant!</echo>

</target>

</project>
```

In xml the root element is project

Targets defined, Within targets we can have tasks

Each of the task can depend on other task

Jar depends on compile

Sample 2:

By default, ant looks for build.xml in the current directory.

If any other xml file has to be used, -buildfile arg can be used as below

```
[root@ctx3p11 ant]# cat build.xml_11

<?xml version="1.0"?>

<project name="Hello World Project" default="info">

<target name="info">

    <echo>Hello World - Welcome to Apache Ant!</echo>
```

```
</target>  
</project>
```

```
[root@ctx3p11 ant]# ant -buildfile build.xml_11
```

```
Buildfile: build.xml_11
```

```
info:
```

```
[echo] Hello World - Welcome to Apache Ant!
```

```
BUILD SUCCESSFUL
```

```
Total time: 0 seconds
```

Sample 3:

```
<?xml version="1.0"?>  
  
<project name="Hello World Project" default="deploy">  
  
<target name="deploy" depends="package">  
  
    <echo>Deploy - Welcome to Apache Ant!</echo>  
  
</target>  
  
<target name="package" depends="clean,compile">  
  
    <echo>package - Welcome to Apache Ant!</echo>  
  
</target>  
  
<target name="clean" >  
  
    <echo>clean - Welcome to Apache Ant!</echo>  
  
</target>
```

```
<target name="compile" >  
    <echo>comiple - Welcome to Apache Ant!</echo>  
</target>  
</project>
```

[root@ctx3p11 ant]# ant clean

Buildfile: build.xml

clean:

[echo] clean - Welcome to Apache Ant!

BUILD SUCCESSFUL

Total time: 0 seconds

[root@ctx3p11 ant]#

[root@ctx3p11 ant]# ant deploy

Buildfile: build.xml

clean:

[echo] clean - Welcome to Apache Ant!

compile:

```
[echo] compile - Welcome to Apache Ant!
```

package:

```
[echo] package - Welcome to Apache Ant!
```

deploy:

```
[echo] Deploy - Welcome to Apache Ant!
```

BUILD SUCCESSFUL

Total time: 0 seconds

```
[root@ctx3p11 ant]#
```

```
[root@ctx3p11 ant]# javac HelloWorld.java
```

```
[root@ctx3p11 ant]# java HelloWorld
```

Hello, World

```
[root@ctx3p11 ant]# cat HelloWorld.java
```

```
public class HelloWorld {  
    public static void main(String[] args) {  
        // Prints "Hello, World" in the terminal window.  
        System.out.println("Hello, World");  
    }  
}
```

```
}
```



```
//
```



```
//
```



```
//
```



```
//
```



```
//
```



```
[root@ctx3p11 ant]#
```

Any shell/python script can be executed from ant as below

Executing shell script:

```
[root@ctx3p11 ant]# cat shell.xml
```



```
<?xml version="1.0"?>
```



```
<project name="Hello World Project" default="test">
```



```
<target name="test">
```



```
 <exec executable="/bin/bash">
```



```
   <arg value="/shell.sh"/>
```



```
 </exec>
```



```
</target>
```



```
</project>
```

```
[root@ctx3p11 ant]# cat /shell.sh
```

```
date > /fileant
```

```
[root@ctx3p11 ant]#
```

Executing shell script with command line args:

```
[root@ctx3p11 ant]# cat shell.xml
```

```
<?xml version="1.0"?>

<project name="Hello World Project" default="test">

<target name="test">

<exec executable="/bin/bash">

<arg value="scr.sh"/>

<arg value="1234"/>

</exec>

</target>

</project>
```

```
[root@ctx3p11 ant]# cat scr.sh
```

```
#!/bin/bash

set -x

date > /fileshell;

echo "$1">>> /fileshell
```

```
[root@ctx3p11 ant]# cat /fileshell
```

```
Fri Feb 24 12:36:54 IST 2017
```

```
1234
```

```
[root@ctx3p11 ant]#
```

Executing python script:

```
[root@ctx3p11 ant]# cat python.xml  
<?xml version="1.0"?>  
  
<project name="Hello World Project" default="test">  
  
<target name="test">  
  
<exec executable="/usr/bin/python">  
  
  <arg value="/1.py"/>  
  
  <arg value="/tmp"/>  
  
</exec>  
  
</target>  
  
</project>
```

```
[root@ctx3p11 ant]# cat /1.py
```

```
#!/usr/bin/python  
  
import os  
  
import sys  
  
os.system("date > /fileant1")
```

```
[root@ctx3p11 ant]#
```

Deploying the files using scp :

```
<?xml version="1.0"?>  
  
<project name="Hello World Project" default="test">  
  
<target name="test">
```

```
<exec executable="scp">  
    <arg value="ifile"/>  
    <arg value="root@ctx3p07.in.company.com:/tmp"/>  
</exec>  
</target>  
</project>
```

Need to configure password less authentication

Compiling a java program using ANT:

```
[root@ctx3p11 ant]# cat helloworld.xml  
<?xml version="1.0" encoding="ISO-8859-1"?>  
<project name="MyTask" basedir=". " default="jar">  
  
<target name="clean" description="Delete all generated files">  
    <delete dir="classes"/>  
    <delete file="HelloWorld.jar"/>  
</target>  
  
<target name="compile" description="Compiles the Task">
```

```
<javac srcdir="src" destdir="classes"/>

</target>

<target name="jar" description="JARs the Task">

    <jar destfile="HelloWorld.jar" basedir="classes"/>

</target>

</project>
```

[root@ctx3p11 ant]#

Using the properties file for defining the variables:

build.properties file can be used to define custom variables.

Few default variables can also be used as shown below

[root@ctx3p11 ant]# cat build.xml

```
<?xml version="1.0"?>

<project name="Hello World Project" default="info">

<target name="info">

    <echo>Ant version is ${ant.version} - you are in ${dir} </echo>

    <echo>Build version is ${buildversion} - base dir is ${basedir} </echo>

    <echo>ant project name is ${ant.project.name} </echo>

    <echo>ant file is ${ant.file} </echo>
```

```
</target>
```

```
</project>
```

```
[root@ctx3p11 ant]# cat build.properties
```

```
buildversion=1.0
```

```
dir="/root/ant"
```

```
[root@ctx3p11 ant]# ant -buildfile build.xml -propertyfile build.properties
```

```
Buildfile: build.xml
```

```
info:
```

```
[echo] Ant version is Apache Ant version 1.7.1 compiled on April 26 2010 - you are in  
"/root/ant"
```

```
[echo] Build version is 1.0 - base dir is /root/ant
```

```
[echo] ant project name is Hello World Project
```

```
[echo] ant file is /root/ant/build.xml
```

```
BUILD SUCCESSFUL
```

```
Total time: 0 seconds
```

```
[root@ctx3p11 ant]#
```

Data types:

Ant provides a number of predefined data types

They are the services that are built into product already

Fileset:

It is collection of files

Used as filter to exclude or include files that match a particular pattern

```
<fileset dir="${src}" casesensitive="yes">  
    <include name="**/*.java"/>  
    <exclude name="**/*abc*"/>  
</fileset>
```

The fileset selects all .java files in the source folder except those contain the word 'Stub'. The case-sensitive filter is applied to the fileset which means a file with the name Samplestub.java will not be excluded from the fileset.

Pattern set:

```
<patternset id="java.files.without.stubs">  
    <include name="src/**/*.java"/>  
    <exclude name="src/**/*Stub*"/>
```

```
</patternset>
```

pattern set is a pattern that allows to filter files or folders easily based on certain patterns. Patterns can be created using the following meta characters:

? - Matches one character only.

* - Matches zero or many characters.

** - Matches zero or many directories recursively.

Filter set

Using a filterset data type along with the copy task, you can replace certain text in all files that matches the pattern with a replacement value.

A common example is to append the version number to the release notes file, as shown in the following code.

```
<copy todir="${output.dir}">  
<fileset dir="${releasenotes.dir}" includes="**/*.txt"/>
```

```
<filterset>  
    <filter token="VERSION" value="${current.version}" />  
</filterset>  
</copy>
```

Path:

The path data type is commonly used to represent a class-path. Entries in the path are separated using semicolons or colons. However, these characters are replaced at the run-time by the executing system's path separator character.

The classpath is set to the list of jar files and classes in the project, as shown in the example below.

```
<path id="build.classpath.jar">  
    <pathelement path="${env.J2EE_HOME}/${j2ee.jar}" />  
    <fileset dir="lib">  
        <include name="**/*.jar" />  
    </fileset>  
</path>
```

To create a directory:

```
[root@ctx3p11 ant]# cat dir.xml

<?xml version="1.0"?>

<project name="Hello World Project" default="create-java-dir">

<!-- Target #1. Set property value depends on check result -->

<target name="check-dir">

<available property="javadir1" file="/root/ant" type="dir"/>

</target>

<!-- target #2. Create dir 'javadir1' if doesn't exist -->

<target name="create-java-dir" depends="check-dir" unless="/root/ant/javadir1">

<mkdir dir="/root/ant/javadir1"/>

</target>

</project>
```

```
[root@ctx3p11 ant]# ant -buildfile dir.xml
```

```
Buildfile: dir.xml
```

```
check-dir:
```

```
create-java-dir:
```

```
  [mkdir] Created dir: /root/ant/javadir1
```

```
BUILD SUCCESSFUL
```

```
Total time: 0 seconds
```

```
[root@ctx3p11 ant]#
```

Sample project:

Project.xml

```
<?xml version="1.0"?>

<project name="Helloworld" basedir=". default="deploy">

    <property name="src.dir" value="src"/>

    <property name="web.dir" value="web"/>

    <property name="build.dir" value="build"/>

    <property name="name" value="Helloworld"/>

    <path id="master-classpath">
        <fileset dir="${web.dir}">
            <include name="*.jar"/>
        </fileset>
        <pathelement path="${build.dir}"/>
    </path>

    <target name="build" description="Compile source tree java files" >
```

```
<mkdir dir="${build.dir}">

<javac destdir="${build.dir}" source="1.5" target="1.5">
  <src path="${src.dir}" />
  <classpath refid="master-classpath" />
</javac>

</target>

<target name="clean" description="Clean output directories">
  <delete>
    <fileset dir="${build.dir}">
      <include name="**/*.class" />
    </fileset>
  </delete>
</target>

<target name="build-jar" depends="build">
  <mkdir dir="build/jar"/>
  <mkdir dir="build/classes"/>
  <jar destfile="build/jar/HelloWorld.jar" basedir="build/classes">
    <manifest>
      <attribute name="Main-Class" value="HelloWorld" />
    </manifest>
  </jar>
</target>
```

```
<target name="build-war" depends="build-jar">

<mkdir dir="build/war"/>

<war destfile="build/war/Helloworld.war" webxml="${web.dir}/web.xml">
    <classes dir="${build.dir}/web"/>
</war>

</target>

<target name = "deploy" depends = "build-war" description = "Deploy application">

<mkdir dir="deploy"/>

<copy todir = "${deploy.path}" preservelastmodified = "true">
    <fileset dir = "${build.dir}">
        <include name = "**/*.*"/>
    </fileset>
</copy>
</target>

</project>
```

~

```
[root@ctx3p11 ant]# cat build.properties

buildversion=1.0

dir="/root/ant"

deploy.path = /root/ant/deploy

[root@ctx3p11 ant]#
```

```
[root@ctx3p11 ant]# ant -buildfile project.xml -propertyfile build.properties
```

```
Buildfile: project.xml
```

```
build:
```

```
build-jar:
```

```
 [jar] Building MANIFEST-only jar: /root/ant/build/jar/Helloworld.jar
```

```
build-war:
```

```
 [war] Building war: /root/ant/build/war/Helloworld.war
```

```
deploy:
```

```
 [copy] Copying 2 files to /root/ant/deploy
```

```
BUILD SUCCESSFUL
```

```
Total time: 0 seconds
```

```
[root@ctx3p11 ant]#
```

Example2:

```
[root@ctx3p11 ant]# cat project.xml
<?xml version="1.0"?>
<project name="Helloworld" basedir=". " default="deploy">
    <property name="src.dir" value="src"/>
    <property name="web.dir" value="web"/>
    <property name="build.dir" value="build/classes"/>
    <property name="name" value="Helloworld"/>

    <path id="master-classpath">
        <fileset dir="${web.dir}">
            <include name="*.jar"/>
        </fileset>
        <pathelement path="${build.dir}"/>
    </path>

    <target name="build" description="Compile source tree java files" >
        <mkdir dir="${build.dir}"/>
        <javac destdir="${build.dir}" >
            <src path="${src.dir}"/>
            <classpath refid="master-classpath"/>
        </javac>
    </target>
```

```
<target name="clean" description="Clean output directories">

    <delete>

        <fileset dir="${build.dir}">

            <include name="**/*.class"/>
            <include name="**/*.jar"/>
            <include name="**/*.war"/>

        </fileset>

    </delete>

</target>
```

```
<target name="build-jar" depends="build">

    <mkdir dir="build/jar"/>

    <jar destfile="build/jar/Helloworld.jar" basedir="build/classes">

        <manifest>

            <attribute name="Main-Class" value="HelloWorld"/>

        </manifest>

    </jar>

</target>
```

```
<target name="build-war" depends="build-jar">

    <mkdir dir="build/war"/>
    <mkdir dir="build/web"/>

    <war destfile="build/war/Helloworld.war" webxml="${web.dir}/web.xml">

        <classes dir="build/web"/>

    </war>
```

```
</war>

</target>

<target name = "deploy" depends = "build-war" description = "Deploy application">
    <mkdir dir="deploy"/>
    <copy todir = "${deploy.path}" preservelastmodified = "true">
        <fileset dir = "${build.dir}">
            <include name = "**/*.*"/>
        </fileset>
    </copy>
</target>
</project>
```

```
[root@ctx3p11 ant]# ant -buildfile project.xml -propertyfile build.properties
```

```
Buildfile: project.xml
```

```
build:
```

```
    [mkdir] Created dir: /root/ant/build/classes
    [javac] Compiling 1 source file to /root/ant/build/classes
```

```
build-jar:
```

```
    [mkdir] Created dir: /root/ant/build/jar
    [jar] Building jar: /root/ant/build/jar/HelloWorld.jar
```

build-war:

```
[mkdir] Created dir: /root/ant/build/war  
[mkdir] Created dir: /root/ant/build/web  
[war] Building war: /root/ant/build/war/Helloworld.war
```

deploy:

```
[mkdir] Created dir: /root/ant/deploy  
[copy] Copying 1 file to /root/ant/deploy
```

BUILD SUCCESSFUL

```
[root@ctx3p11 build]# ls -lrt  
total 16  
drwxr-xr-x 2 root root 4096 Feb 24 12:59 classes  
drwxr-xr-x 2 root root 4096 Feb 24 12:59 jar  
drwxr-xr-x 2 root root 4096 Feb 24 12:59 web  
drwxr-xr-x 2 root root 4096 Feb 24 12:59 war
```

```
[root@ctx3p11 build]# cd jar/
```

```
[root@ctx3p11 jar]# ls
```

```
HelloWorld.jar
```

```
[root@ctx3p11 jar]# ls -lrt  
total 4  
-rw-r--r-- 1 root root 700 Feb 24 12:59 HelloWorld.jar
```

```
[root@ctx3p11 jar]# jar xf HelloWorld.jar
```

```
[root@ctx3p11 jar]# ls -lrt  
-bash: l: command not found  
  
[root@ctx3p11 jar]# ls -lrt  
-bash: l: command not found  
  
[root@ctx3p11 jar]# ls -lrt  
-bash: l: command not found  
  
[root@ctx3p11 jar]# ls -lrt  
total 12  
-rw-r--r-- 1 root root 700 Feb 24 12:59 HelloWorld.jar  
drwxr-xr-x 2 root root 4096 Feb 24 12:59 META-INF  
-rw-r--r-- 1 root root 341 Feb 24 12:59 HelloWorld.class  
  
[root@ctx3p11 jar]# cd ..  
  
[root@ctx3p11 build]# ls -lrt  
total 16  
drwxr-xr-x 2 root root 4096 Feb 24 12:59 classes  
drwxr-xr-x 2 root root 4096 Feb 24 12:59 web  
drwxr-xr-x 2 root root 4096 Feb 24 12:59 war  
drwxr-xr-x 3 root root 4096 Feb 24 12:59 jar  
  
[root@ctx3p11 build]# cd war/
```

Jar file is created if classes exists in base directory

[\[root@ctx3p11 jar\]# ls -rlt](#)

total 4

-rw-r--r-- 1 root root 700 Feb 24 12:59 HelloWorld.jar

[root@ctx3p11 jar]# pwd

/root/ant/build/jar

[root@ctx3p11 jar]# jar xf HelloWorld.jar

[root@ctx3p11 jar]# ls -lrt

total 12

-rw-r--r-- 1 root root 700 Feb 24 12:59 HelloWorld.jar

drwxr-xr-x 2 root root 4096 Feb 24 12:59 META-INF

-rw-r--r-- 1 root root 341 Feb 24 12:59 HelloWorld.class

[root@ctx3p11 jar]# java HelloWorld

Hello, World

[root@ctx3p11 jar]#

CHEF

Content:

Chef Introduction

Chef Architecture

Chef Components

Chef setup and configuration

Chef Boot strap method

Chef cookbooks

Chef Run lists

Chef Templates

Chef data bags

Chef roles

Chef Environments

Chef search criterion

Chef Command line execution

Chef super market

Berks tool

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).

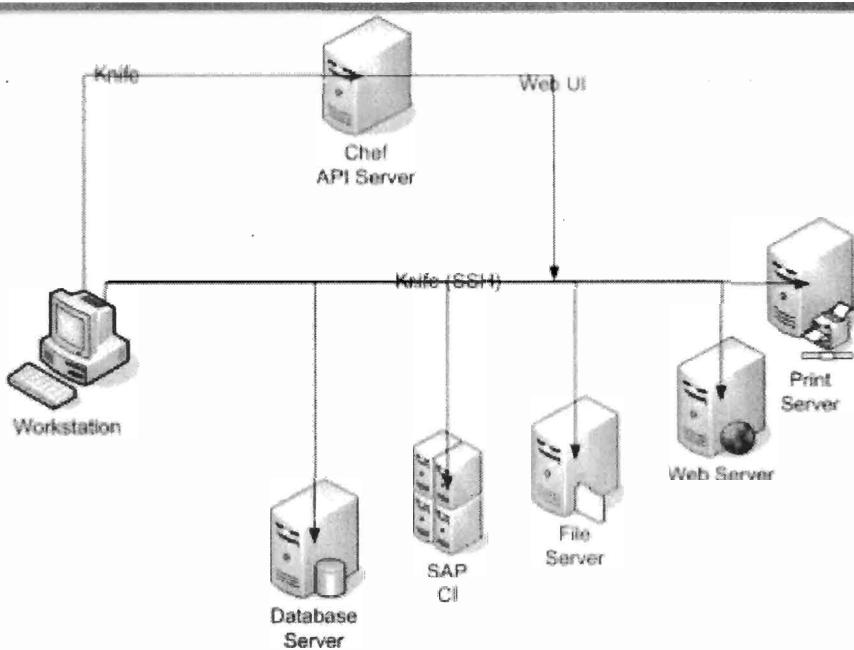
Nodes are the machines—physical, virtual, cloud, and so on—that are under management by Chef. The chef-client is installed on each node and is what performs the automation on that machine.

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 Architecture



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 '$1$JsvHslasdfjVEroftprNn4JHtDi'
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 as well

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 7.1.1 version

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 market, 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  
end
```

```
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']

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'].eq?(('inet') || params['family'].eq?('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.)

Dockers and Containers

Virtualization:

Virtualization is the ability to run multiple operating systems on a single physical system and share the underlying hardware resources.

It is the process by which one server hosts the appearance of many computers.

Virtualization is used to improve IT throughput and costs by using physical resources as a pool from which virtual resources can be allocated.

Different types of virtualizations:

Bare metal hypervisor ex: Vmware ESX

Hosted Hypervisor ex: KVM

Software partitioning/Dockers

LXC (Linux Containers) is an operating-system-level virtualization method for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel

Uses kernel features calls cgroups (Linux Kernel control groups)— that allows limitation and prioritization of resources like CPU/memory, /i/o, network

Namespace isolation – that allows complete isolation of an application view of Operating system environment including process trees ,networking ,user ids and mounted file system

LXC combines kernel's cgroups and support for isolated namespaces to provide an isolated environment for applications

Note: LXC is a linux technology. It does not work on Windows or Mac.

Dockers are useful in:

To avoid having too much software on the machine

To try deploying a web app

To try provisioning scripts on fake machine first them try on actual machine

Dockers leverage LXC – Light weight alternative to full virtualization such as provided by traditional hypervisors like KVM VMware Xen or ESXi

Installing docker:

Steps for RHEL :

1. Execute below command on the node

```
sudo yum-config-manager --add-repo
```

```
https://docs.docker.com/engine/installation/linux/repo_files/centos/docker.repo
```

```
sudo yum -y install docker-engine
```

```
sudo systemctl start docker
```

```
service docker status
```

Ensure the below file is created in /etc/yum.repos.d

```
cat /etc/yum.repos.d/docker.repo
```

```
root@reviewb ~]# cd /etc/yum.repos.d/
```

```
[root@reviewb yum.repos.d]# ls
```

```
iso.repo
```

```
[root@reviewb yum.repos.d]# cat /docker
```

```
cat: /docker: Is a directory
```

```
[root@reviewb yum.repos.d]# cat /docker.repo
```

```
[docker-main]
```

```
name=Docker Repository
```

```
baseurl=https://yum.dockerproject.org/repo/main/centos/7/
```

```
enabled=1
```

```
gpgcheck=1
```

```
gpgkey=https://yum.dockerproject.org/gpg
```

```
[docker-testing]

name=Docker Repository

baseurl=https://yum.dockerproject.org/repo/testing/centos/$releasever/

enabled=0

gpgcheck=1

gpgkey=https://yum.dockerproject.org/gpg
```

```
[docker-beta]

name=Docker Repository

baseurl=https://yum.dockerproject.org/repo/beta/centos/7/

enabled=0

gpgcheck=1

gpgkey=https://yum.dockerproject.org/gpg
```

```
[docker-nightly]

name=Docker Repository

baseurl=https://yum.dockerproject.org/repo/nightly/centos/7/

enabled=0

gpgcheck=1

gpgkey=https://yum.dockerproject.org/gpg
```

docker --help

--config=~/docker Location of client config files

```
-D, --debug           Enable debug mode
-H, --host=[]        Daemon socket(s) to connect to
-h, --help            Print usage
-l, --log-level=info Set the logging level
--tls                Use TLS; implied by --tlsverify
--tlscacert=~/.docker/ca.pem Trust certs signed only by this CA
--tlscert=~/.docker/cert.pem Path to TLS certificate file
--tlskey=~/.docker/key.pem Path to TLS key file
--tlsverify          Use TLS and verify the remote
-v, --version         Print version information and quit
```

Commands:

```
attach  Attach to a running container
build   Build an image from a Dockerfile
commit  Create a new image from a container's changes
cp      Copy files/folders between a container and the local filesystem
create  Create a new container
diff    Inspect changes on a container's filesystem
events  Get real time events from the server
exec   Run a command in a running container
export  Export a container's filesystem as a tar archive
history Show the history of an image
images  List images
import  Import the contents from a tarball to create a filesystem image
info   Display system-wide information
```

inspect Return low-level information on a container, image or task

kill Kill one or more running container

load Load an image from a tar archive or STDIN

login Log in to a Docker registry.

logout Log out from a Docker registry.

logs Fetch the logs of a container

network Manage Docker networks

node Manage Docker Swarm nodes

pause Pause all processes within one or more containers

port List port mappings or a specific mapping for the container

ps List containers

pull Pull an image or a repository from a registry

push Push an image or a repository to a registry

rename Rename a container

restart Restart a container

rm Remove one or more containers

rmi Remove one or more images

run Run a command in a new container

save Save one or more images to a tar archive (streamed to STDOUT by default)

search Search the Docker Hub for images

service Manage Docker services

start Start one or more stopped containers

stats Display a live stream of container(s) resource usage statistics

stop Stop one or more running containers

swarm Manage Docker Swarm

tag Tag an image into a repository
top Display the running processes of a container
unpause Unpause all processes within one or more containers
update Update configuration of one or more containers
version Show the Docker version information
volume Manage Docker volumes
wait Block until a container stops, then print its exit code

Run 'docker COMMAND --help' for more information on a command.

Steps:

1. Install docker using yum

yum install docker

2. download the image using docker pull

docker pull ubuntu

docker pull rhel6

3. List dockers available

docker ps → to list running containers

docker ps -a → lists all containers including killed ones

4. start a docker using

docker run -t -i ubuntu

5. To stop a docker

docker stop <container_id>

6. to start a docker

docker start <container_id>

7.to login

docker attach <>

Commands:

Docker pull is to pull images

Docker ps to display running containers

Docker ps -a to display all running containers including the dead ones

Docker inspect is to show all values

Docker stop/start – is to stop or start a container

Docker attach/detach is to attach or detach to a container

Docker run is to run the container, if image is not available docker pulls from docker hub repository

Examples:

```
[root@rscthydnet1 ~]# docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	1967d889e07f	2 weeks ago	167.9 MB

→ ***sudo docker run -t -i ubuntu:latest*** → downloads Ubuntu docker

checks local machine first, if not available downloads the image from net

```
[root@rscthydnet1 LINUX]# sudo docker run -t -i ubuntu
```

Unable to find image 'ubuntu:latest' locally

latest: Pulling from ubuntu

0847857e6401: Pull complete

f8c18c152457: Pull complete

```
8643975d001d: Pull complete  
d5802da4b3a0: Pull complete  
fe172ed92137: Pull complete  
Digest: sha256:5349f00594c719455f2c8e6f011b32758dc326d8e225c737a55c15cf3d6948c  
Status: Downloaded newer image for ubuntu:latest  
root@d47c9e2cbf15:/#
```

it will login to docker

```
root@d47c9e2cbf15:/# id  
uid=0(root) gid=0(root) groups=0(root)  
root@d47c9e2cbf15:/# cat /etc/*release  
DISTRIB_ID=Ubuntu  
DISTRIB_RELEASE=16.04  
DISTRIB_CODENAME=xenial  
DISTRIB_DESCRIPTION="Ubuntu 16.04.1 LTS"  
NAME="Ubuntu"  
VERSION="16.04.1 LTS (Xenial Xerus)"  
ID=ubuntu  
ID_LIKE=debian  
PRETTY_NAME="Ubuntu 16.04.1 LTS"  
VERSION_ID="16.04"  
HOME_URL="http://www.ubuntu.com/"
```

```
SUPPORT_URL="http://help.ubuntu.com/"

BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"

VERSION_CODENAME=xenial

UBUNTU_CODENAME=xenial

root@d47c9e2cbf15:/#
```

We can download the docker and load it

```
docker load -i centos_docker_image.tar
```

run apt-get update to work with apt-get

```
root@d47c9e2cbf15:/# apt-get update

Get:1 http://ports.ubuntu.com/ubuntu-ports xenial InRelease [247 kB]

Get:2 http://ports.ubuntu.com/ubuntu-ports xenial-updates InRelease [95.7 kB]

Get:3 http://ports.ubuntu.com/ubuntu-ports xenial-security InRelease [94.5 kB]

Get:4 http://ports.ubuntu.com/ubuntu-ports xenial/main Sources [1103 kB]

Get:5 http://ports.ubuntu.com/ubuntu-ports xenial/restricted Sources [5179 B]

Get:6 http://ports.ubuntu.com/ubuntu-ports xenial/universe Sources [9802 kB]

Get:7 http://ports.ubuntu.com/ubuntu-ports xenial/main x86_64el Packages [1470 kB]

Get:8 http://ports.ubuntu.com/ubuntu-ports xenial/universe x86_64el Packages [9485 kB]

Get:9 http://ports.ubuntu.com/ubuntu-ports xenial-updates/main Sources [256 kB]

Get:10 http://ports.ubuntu.com/ubuntu-ports xenial-updates/restricted Sources [1872 B]

Get:11 http://ports.ubuntu.com/ubuntu-ports xenial-updates/universe Sources [136 kB]

Get:12 http://ports.ubuntu.com/ubuntu-ports xenial-updates/main x86_64el Packages [487 kB]

Get:13 http://ports.ubuntu.com/ubuntu-ports xenial-updates/universe x86_64el Packages [395 kB]

Get:14 http://ports.ubuntu.com/ubuntu-ports xenial-security/main Sources [54.7 kB]
```

```
Get:15 http://ports.ubuntu.com/ubuntu-ports xenial-security/restricted Sources [1872 B]
Get:16 http://ports.ubuntu.com/ubuntu-ports xenial-security/universe Sources [13.8 kB]
Get:17 http://ports.ubuntu.com/ubuntu-ports xenial-security/main x86_64el Packages [171 kB]
Get:18 http://ports.ubuntu.com/ubuntu-ports xenial-security/universe x86_64el Packages [55.7 kB]
Fetched 23.9 MB in 1min 2s (382 kB/s)
```

Reading package lists... Done

Install packages

```
apt-get install vim
apt-get install openssh
apt-get install vim
apt-get install iutils-ping
apt-get install net-tools
apt-get install ksh
```

to ge the container id

```
root@rscthydnet1 ~]# docker ps -a
CONTAINER ID        IMAGE       COMMAND       CREATED      STATUS      PORTS
NAMES
docker stop container_ID
docker start container_ID
```

```
docker run -d --name ubuntu-docker1 ubuntu
```

```
[root@rscthydnet1 ~]# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
20354b3035a3	ubuntu	"/bin/bash"	6 seconds ago	Up 3 seconds	
romantic_booth					

docker attach 20354b3035a3

[root@rscthydnet1 ~]# docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
20354b3035a3	ubuntu	"/bin/bash"	8 minutes ago	Up 18 seconds	
romantic_booth					

[root@rscthydnet1 ~]# docker stop 20354b3035a3

20354b3035a3

[root@rscthydnet1 ~]# docker start 20354b3035a3

20354b3035a3

[root@rscthydnet1 ~]# docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					
20354b3035a3	ubuntu	"/bin/bash"	9 minutes ago	Up 57 seconds	
romantic_booth					

```
5f3e16925382    ubuntu    "/bin/bash"    9 minutes ago   Exited (127) 9 minutes ago  
admiring_montalcini  
  
24eedbddd026    ubuntu    "/bin/bash"    11 minutes ago  Exited (0) 11 minutes ago  
ubuntu-docker1  
  
d47c9e2cbf15    ubuntu    "/bin/bash"    3 hours ago    Exited (127) 11 minutes ago  
sleepy_elion  
  
[root@rscthydnet1 ~]#
```

```
[root@rscthydnet1 ~]# docker attach d47c9e2cbf15
```

```
You cannot attach to a stopped container, start it first
```

```
[root@rscthydnet1 ~]# docker start d47c9e2cbf15
```

```
d47c9e2cbf15
```

```
[root@rscthydnet1 ~]#
```

```
[root@rscthydnet1 ~]# docker attach d47c9e2cbf15
```

```
root@d47c9e2cbf15:#
```

```
root@d47c9e2cbf15:#
```

```
root@d47c9e2cbf15:#
```

```
root@d47c9e2cbf15:/# ls -lrt
```

```
total 828
```

```
drwxr-xr-x. 2 root root 6 Apr 12 2016 home
```

```
drwxr-xr-x. 2 root root 6 Apr 12 2016 boot
```

```
drwxr-xr-x. 10 root root 97 Nov 1 08:20 usr
```

```
drwxr-xr-x. 2 root root 6 Nov 1 08:20 srv
```

```
drwxr-xr-x. 2 root root 6 Nov 1 08:20 opt
```

```
drwxr-xr-x. 2 root root 6 Nov 1 08:20 mnt
```

```
drwxr-xr-x. 2 root root 6 Nov 1 08:20 media
drwxr-xr-x. 2 root root 22 Nov 1 08:21 lib64
drwxr-xr-x. 11 root root 4096 Nov 1 08:21 var
dr-xr-xr-x. 12 root root 0 Nov 14 04:07 sys
drwxr-xr-x. 2 root root 6 Nov 16 08:25 rsctfvt
drwxr-xr-x. 9 root root 4096 Nov 16 08:31 lib
-rw-r----- 1 root root 809234 Nov 16 08:34 sg3_utils-1.41-3.fc24.x86_64.rpm
drwxrwxrwx. 2 root root 8192 Nov 16 08:36 _linux_2
drwxr-xr-x. 5 root root 74 Nov 16 08:38 run
drwxr-xr-x. 2 root root 4096 Nov 16 08:40 bin
drwxr-xr-x. 2 root root 4096 Nov 16 08:40 sbin
drwxr-xr-x. 72 root root 4096 Nov 16 08:45 etc
drwxrwxrwt. 2 root root 6 Nov 16 08:45 tmp
drwx----- 4 root root 94 Nov 16 10:32 root
dr-xr-xr-x. 583 root root 0 Nov 16 10:44 proc
drwxr-xr-x. 5 root root 380 Nov 16 10:44 dev
root@d47c9e2cbf15:/#
root@d47c9e2cbf15:/#
root@d47c9e2cbf15:/#
```

to make container running always

```
docker run -d <docker name> sh /script.sh
```

this script,sh should be a never ending loop

something like:

```
while true  
do  
sleep 1  
done
```

Port forwarding:

Start a container from host

And we can map the host and container's ports

Container 's app can be accessed from host machine using port forwarding concept

Example:

Create a docker and do a mapping between host port 8080 to container 8080

docker run -p 8080:8080 jenkins:latest

docker run -d -p 8080:8080 jenkins:latest -> to run in the background

docker rm <name of the container> to remove the container, first stop the container and remove.

Detaching/attaching – we can do on a running container

Attaching/detaching a shell

Stop/start – stopping a service .

Docker rmi – to remove image

Docker pull xyz:latest

Docker run -it xyz /bin/bash

Install software

Apt-get update

Apt-get install <>

Customize the image

Exit the container

To push the image to Docker Hub:

Docker commit -m "customized image" -a "authorname" <containerid> <nameofaccount> <nameof the image>:v2

docker commit -m "ubuntupython" -a "santosh" f56be454c884 santoshdevops/ubuntu:python:v1

sha256:7f093647f35a39965d101859c269da63884fef22bc574829786408513efbe7b8

***docker commit -m "jenkinscustomzed3" -a "santosh" fd4e1be10bd2
santoshdevops/jenkinscustomzed3:v200***

docker login

Note:

We need to create a login in docker hub, and use the same for pushing images

Give account name and password

Docker push nameofdockerimage

To create a public repository:

docker push santoshdevops/ubuntu:python

then try to delete the image from local machine

and try to pull same image

Docker File:

To automate the image creation

Create docker file

touch Dockerfile

Add the instructions to file

Command Description

ADD Copies a file from the host system onto the container

CMD The command that runs when the container starts

ENTRYPOINT

ENV Sets an environment variable in the new container

EXPOSE Opens a port for linked containers

FROM The base image to use in the build. This is mandatory and must be the first command in the file.

MAINTAINER An optional value for the maintainer of the script

ONBUILD A command that is triggered when the image in the Dockerfile is used as a base for another image

RUN Executes a command and save the result as a new layer

USER Sets the default user within the container

VOLUME Creates a shared volume that can be shared among containers or by the host machine

WORKDIR Set the default working directory for the container

cat Dockerfile

```
FROM ubuntu:latest  
MAINTAINER Visualpath  
RUN apt-get update && apt-get install -y ruby
```

Docker build -t newdockerimage:v3 .

Creates container

Do all the stuff

Create a new container

Remove the old container

ADD

CMD similar to run,

Run gets executed while building

CMD gets executed while running the container

USER to set user id

VOLUME – attach a directory from a host machine

WORKDIR – launch into default dir

Docker bridge ip 172.17.0.1 – advanced switch

This is the gateway for containers

Example of building image from Dockerfile:

docker build -t myimage_t .

docker run --name my_first_instance -t myimage_t

To stop all the docker containers

```
docker stop $(docker ps -a -q)
```

to remove all the docker images

```
docker rm $(docker ps -a -q)
```

Docker images are stored in - /var/lib/docker/devicemapper/

(based on the device used, here device used is devicemapper)

Can we directly copy the image from one machine to other?

Yes:

save the docker image as a tar file:

```
docker save -o <save image to path> <image name>
```

Then copy your image to a new system with regular file transfer tools such as cp or scp. After that you will have to load the image into docker:docker load -i <path to image tar file>

```
docker load -i <path to image tar file>
```

Can we control the cpu /memory running for a container?

Yes:

If we have 2 containers, one for the database and one more for the web server

```
sudo docker run -c 614 -dit --name db postgres /postgres.sh
```

```
sudo docker run -c 410 -dit --name web nginx /nginx.sh
```

Will give 60% to the db container (614 is 60% of 1024) and 40% to the web container.

To share volumes between 2 containers:

```
[root@reviewb ~]# cd /vol2  
[root@reviewb vol2]# ls -lrt  
total 4  
-rw-r--r--. 1 root root 92 Mar 4 10:10 Dockerfile
```

```
[root@reviewb vol2]# cat Dockerfile  
  
FROM ubuntu:latest  
  
RUN mkdir /myvol  
  
RUN echo "hello world" > /myvol/greeting  
  
VOLUME /myvol
```

docker build -t vol2 . → build the image with directory /myvol

docker run -it vol2 /bin/bash → creates container1

docker run -it --volumes-from <1st container id> ubuntu:latest bash → creates container2 with shared directory /myvol

file /myvol will be shared across 2 dockers

Sharing file between Host and Container:

on host:

docker volume create --name DataVolume1 --> creates a docker volume

docker run -ti --rm -v DataVolume1:/datavolume1 ubuntu --> creates container, upon exit docker will be deleted

on docker:

```
echo "Example1" > /datavolume1/Example1.txt
```

on host:

```
docker volume inspect DataVolume1
```

output

```
[  
 {  
   "Name": "DataVolume1",  
   "Driver": "local",  
   "Mountpoint": "/var/lib/docker/volumes/datavolume1/_data",  
   "Labels": null,  
   "Scope": "local"  
 }  
 ]
```

we can start a new docker with same data volume

```
docker run --rm -ti -v DataVolume1:/datavolume1 ubuntu
```

to create volume that persists when container is removed.

```
docker run -ti --name=Container2 -v DataVolume2:/datavolume2 ubuntu
```

Docker logs:

On RHEL

cat /var/log/messages | grep docker

or

journalctl -u docker.service

Docker logs based on OS:

Ubuntu (old using upstart) - /var/log/upstart/docker.log

Ubuntu (new using systemd) - journalctl -u docker.service

Boot2Docker - /var/log/docker.log

Debian GNU/Linux - /var/log/daemon.log

CentOS - /var/log/daemon.log | grep docker

CoreOS - journalctl -u docker.service

Fedora - journalctl -u docker.service

Red Hat Enterprise Linux Server - /var/log/messages | grep docker

OpenSuSE - journalctl -u docker.service

OSX - ~/Library/Containers/com.docker.docker/Data/com.docker.driver.amd64-linux/log/docker.log

GIT and GITHUB

GIT:

- Version control software
- Open source
- git-scm.com

Goals of GIT

Simple

speed

Non linear

distributed

Large

GitHUB:

Collaboration software, hosts GIT repositories

Repository to store the files/projects/code

Entire linux code runs on GITHUB

GITHUB Workflow:

Branching:

Create a branch on master replica, as the changes should not be done on master

Master should always has to be deployable

Branch is exact copy of master

Commits

sort of making changes to branches.

Pull request

Open pull request on github

Comparing the change with master and show other people about your changes

Collaboration:

People may give suggestions or go ahead

Merge

Commits the branch

Cloning Repository:

Cloning is for making local copy from remote copy in GITHUB and working on a project

GITHUB is remote repository

Entire project can be copied to locally and work on it without internet access

Multiple people can work on GITHUB

When we push our changes to github, other users can pull the request and see them

GIT commands:

git clone <https://github.com/username/repository>

git branch feature-branch

git checkout featurebranch

git status

git add index.html

git commit

git push

git fetch

git merge

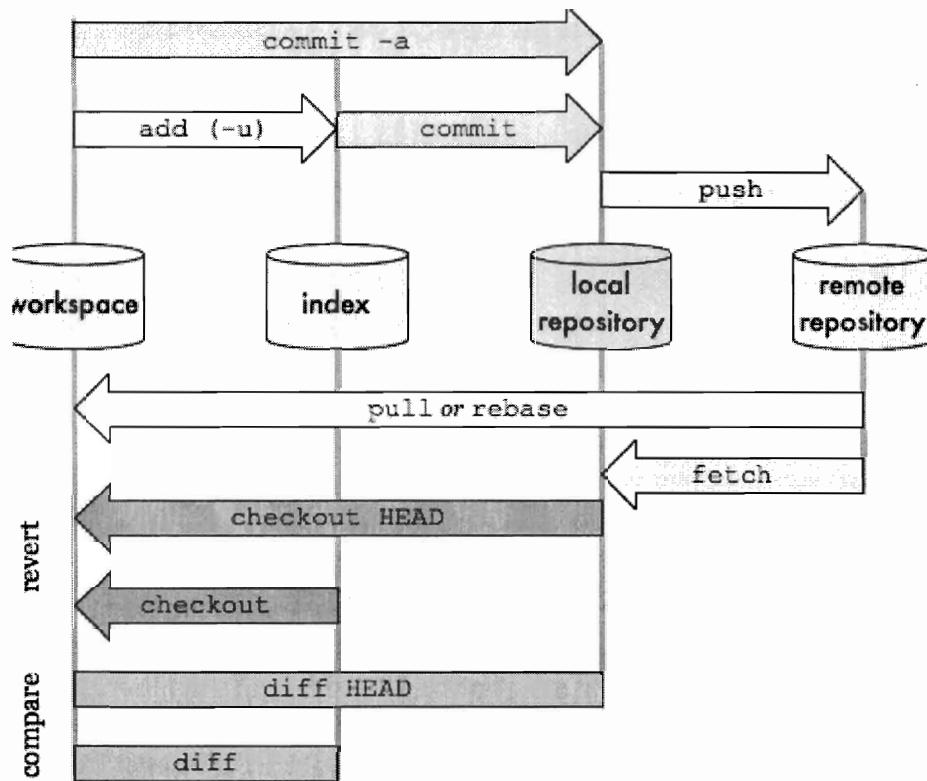
git pull

git diff

git stash

git rebase

GIT Work flow



With every commit we make there will be Commit id

When the commit was made date and time (parent commit)

Before the commit what was the previous commit etc

It is a 40 character SHA1 hash includes Bobs (reduced version of the code , files) and metadata

Git commit has 2 steps

2 step commit

Git add → which files you want to add

Staging files will be added to staging area

Git commit → all files will be included in github with 40 character sha1 hash

Git merge:

Remote- git hub

Locally – git merge

Merge conflicts:

When there is a change in same line in 2 branches

Try git status

And open the file in editor

Keep the change you want to keep

Rebase:

Git rebase to change the history

Below are commits

A B C D
E F

The changes look like A E B F C D

But if you want the history to be like

A B C D E F

Use rebase

How to undo last commit:

git revert

creating a new commit with opposite changes

other commands:

git reset

git commit-amend

git cherry-pick

Command line:

To clone the project

Create user id and password on the git hub site

Then create a new repository on github

Then clone that repository using

```
git clone https://github.vp.com/user1-git/EXCITE
git config --global user.email santoshdevops1@gmail.com
git config --global user.name santosh
```

from a linux machine:

1. add the ssh key of the linux machine to git hub

Go to <https://github.vp.com/settings/ssh>

Click on ssh keys

Add a new key

Copy key from machine and paste it on github

```
[root@reviewb .ssh]# git clone ssh://github.vp.com/santosh-chennuri/EXCITE
Cloning into 'EXCITE'...
remote: Counting objects: 15, done.
remote: Total 15 (delta 0), reused 0 (delta 0), pack-reused 15
Receiving objects: 100% (15/15), done.
[root@reviewb .ssh]# ls -lrt
total 16
-rw-----. 1 root root 405 Nov 29 12:08 authorized_keys
-rw-r--r--. 1 root root 405 Dec 29 09:53 id_rsa.pub
-rw-----. 1 root root 1675 Dec 29 09:53 id_rsa
-rw-r--r--. 1 root root 3428 Jan 1 08:28 known_hosts
drwxr-xr-x. 3 root root 72 Jan 1 08:30 EXCITE
[root@reviewb .ssh]#
```

```
sangit@VP089-PBXCEE7 MINGW64 ~ (master)
$ cd EXCITE/
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ pwd
/c/users/VP_ADMIN/EXCITE
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ ls -lrt
total 1
-rw-r--r-- 1 sangit 197121 27 Nov 15 05:56 README.md
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ cat README.md
# EXCITE
Sample project
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

Modify the read me file

And git status shows the uncommitted change in red

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

Add a new file to the directory and git status shows it

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   README.md
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    file-added
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git add README.md
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   README.md
```

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    file-added
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git add file-added
warning: LF will be replaced by CRLF in file-added.
The file will have its original line endings in your working directory.
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ echo $?
0
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   README.md
    new file:   file-added
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git commit -m "made the changes "
[master f74d55f] made the changes
 2 files changed, 2 insertions(+)
 create mode 100644 file-added
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git commit
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working tree clean
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
```

To add all the files at once

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git add --all
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
```

To commit all the files at once

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ pwd
/c/Users/VP_ADMIN/EXCITE

sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git commit --all
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working tree clean

sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

Example: git commit -all -m "adding all the files"

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git commit --all -m "committing all the changes"
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working tree clean

sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git log
commit f74d55fd058515567a34f8d53c7a64baeb63e81c
Author: sangit1 <sanchenn@in.vp.com>
Date:   Tue Nov 15 06:08:20 2016 +0530
```

made the changes

```
commit 4ae15896e770acb13543f739d6d9641aeb5854c2
Author: User1 K. Git <user1.git@in.vp.com>
Date: Tue Nov 15 05:55:39 2016 +0530
```

```
Initial commit
commit f74d55fd058515567a34f8d53c7a64baeb63e81c → 40 character sha1 code commit code
```

push the changes to master

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git push
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 366 bytes | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.vp.com/user1-git/EXCITE
  4ae1589..f74d55f master -> master
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

To create a new branch and switch to it:
git checkout -b <branch>

To switch to a branch
git checkout <branch>

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git checkout -b branch1
Switched to a new branch 'branch1'
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$ git branch
* branch1
  master
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$
```

Change a file in the branch

And add

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$ git add --all
warning: LF will be replaced by CRLF in file-added.
```

The file will have its original line endings in your working directory.

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$ git status
On branch branch1
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   file-added
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$ git branch
* branch1
  master

sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$ git push --set-upstream origin branch1
Total 0 (delta 0), reused 0 (delta 0)
To https://github.vp.com/user1-git/EXCITE
 * [new branch]      branch1 -> branch1
Branch branch1 set up to track remote branch branch1 from origin.
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$ git commit --all -m "committing changes to branch called branch1"
[branch1 8ab53f1] committing changes to branch called branch1
 1 file changed, 1 insertion(+)
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$ git status
On branch branch1
Your branch is ahead of 'origin/branch1' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working tree clean
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$ git push --set-upstream origin branch1
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 347 bytes | 0 bytes/s, done.
```

```
Total 3 (delta 0), reused 0 (delta 0)
To https://github.vp.com/user1-git/EXCITE
  f74d55f..8ab53f1 branch1 -> branch1
Branch branch1 set up to track remote branch branch1 from origin.
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (branch1)
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git pull
remote: Counting objects: 1, done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From https://github.vp.com/user1-git/EXCITE
  f74d55f..fa3eba8 master      -> origin/master
Updating f74d55f..fa3eba8
Fast-forward
 file-added | 1 +
 1 file changed, 1 insertion(+)
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git branch --delete branch1
Deleted branch branch1 (was 8ab53f1).
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git branch --delete branch1
Deleted branch branch1 (was fa3eba8).
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$ git branch
* master
```

```
sangit@VP089-PBXCEE7 MINGW64 ~/EXCITE (master)
$
```

How to merge branch with the master:

git checkout -b br2 → creates a new branch br2

create a file, touch filebr2

modify the file filebr2

```
git add filebr2  
git commit filebr2  
git status  
move back to master  
git checkout master  
git merge br2 → merges branch2 with the master brahnch  
git push → will push all changes to github/remote repository
```

To configure the linux client

Install git on linux machine with internet access

Make sure it pings

```
[root@rscthydnet1 .ssh]# git clone ssh://github.vp.com/user1-git/EXCITE
```

```
Cloning into 'EXCITE'...
```

```
remote: Counting objects: 15, done.
```

```
remote: Compressing objects: 100% (11/11), done.
```

```
remote: Total 15 (delta 0), reused 10 (delta 0), pack-reused 0
```

```
Receiving objects: 100% (15/15), done.
```

```
[root@rscthydnet1 .ssh]# ls -lrt
```

```
total 16
```

```
-rw-r--r--. 1 root root 409 Nov 15 12:44 id_rsa.pub
```

```
-rw-----. 1 root root 1679 Nov 15 12:44 id_rsa
```

```
-rw-r--r--. 1 root root 191 Nov 15 12:53 known_hosts
```

```
-rw-r--r--. 1 root root 138 Nov 15 13:07 config
```

```
drwxr-xr-x. 3 root root 68 Nov 15 13:09 EXCITE
```

```
[root@rscthydnet1 .ssh]# cd EXCITE/
```

```
[root@rscthydnet1 EXCITE]# ls -lrt
```

```
total 12
```

```
-rw-r--r--. 1 root root 47 Nov 15 13:09 README.md
```

```
-rw-r--r--. 1 root root 59 Nov 15 13:09 Gittest.txt
```

```
-rw-r--r--. 1 root root 73 Nov 15 13:09 file-added
```

```
[root@rscthydnet1 EXCITE]#
```

```
[root@rscthydnet1 EXCITE]#
```

Staging is a step before the commit process in **git**. That is, a commit in **git** is performed in two steps: **staging** and actual commit. As long as a changeset is in the **staging area**, **git** allows you to edit it as you like (replace **staged** files with other versions of **staged** files, remove changes from **staging**, etc.)

git pull -> gets all latest commits to local repos, to working copy

git push → copies local files to remote repos

git fetch → copies remote commits to local repos, but not to working copy

git remote update → updates local copy with remote changes

workspace

index

local repos

remore repos

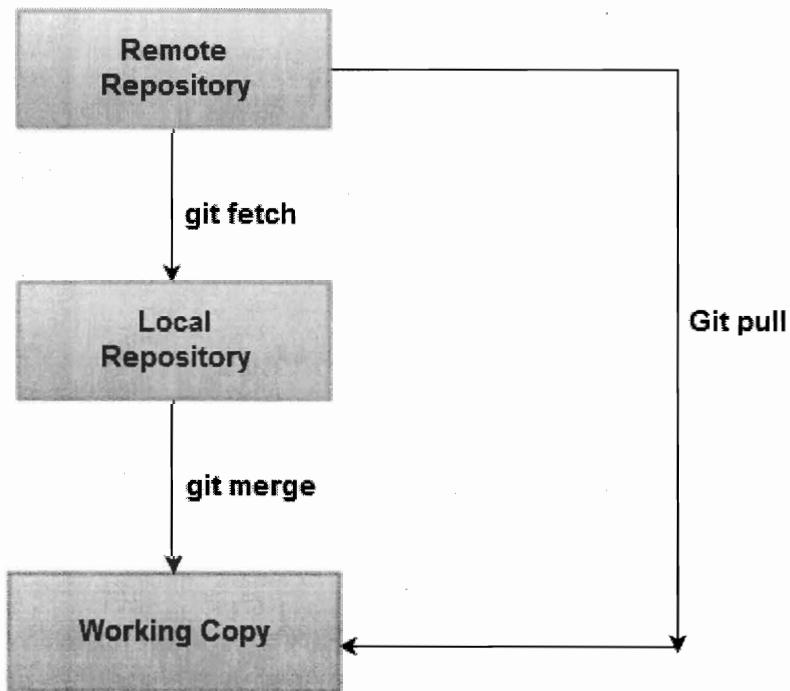
#git fetch

#git diff origin

```
diff --git a/file b/file
index 332e8fc..3d5eaf4 100644
--- a/file
+++ b/file
@@ -4,4 +4,3 @@ hi
abc
xyz
this is for pull
-this is for push
git merge
```

git pull = git fetch + git merge

its always recommended to do git fetch , review changes and then do merge



To delete the local branch use:

```
$ git branch -d branch_name
```

To delete the branch from the remote repository:

```
git push origin --delete <branch_name>
```

Master is local branch

Origin is remote branch

The below a and b does same thing

a)

```
mkdir repo
```

```
cd repo
```

```
git init
```

```
git remote add origin git://github.com/cmcculloh/repo.git
```

```
git fetch --all
```

```
git pull origin master
```

b)

```
git clone git://github.com/cmcculloh/repo.git
```

git stash - temporary storage box for the changes you are not ready to commit

```
git branch
```

```
git stash
```

git checkout -b newbr

change something save changes

git stash --> puts all changes in a box

there is something wrong with master , so we need to go back to master and do some changes

git checkout master --> correct it

then go back to branch newbr

git stash pop --index

git diff

create a file and modify

git add .

`git diff --cached` --> difference between staging and committed version

`git diff` - working and staged version

`git commit`

`git diff HEAD` - working dir and commit dir

Git rebase:

To check the history of commits:

`git log --graph --all -oneline`

To check files in a branch:

```
$ git ls-tree -r --name-only master
build.xml
cloud.png
file
file123
file12345
file123890
file89089
file67890898989
index.html
index.html.j2
```

```
$ git ls-tree -r --name-only branch123
build.xml
cloud.png
file
file123
file12345
file123890
file89089
file67890898989
index.html
index.html.j2
```

to find out difference between 2 files in 2 diff branches

```
ADMINIBM@IBM847-PC0BY40U MINGW64 /123/sample (branch123)
$ git diff master:file branch123:file
```

To remove a file

git rm -f file

git reset HEAD file → unstage changes

git rebase:

D → E

A → B → C

F

Git clone

Git add .

Git commit .

git remote add origin
<https://github.com/santoshdevops/sample2.git>

git push -u origin master

add 2 files and commit

```
$ git log --graph --all --oneline
* aa4d11c Second
* 5525681 first
```

1. flow oall the tools - CI\CD complete flow

Code in git -> Jenkins Jobs (maven for compiling and Ansible to push the code)

2. For java app what are the best tools which is used for CI and CD tools.-----

Jenkins + Ansible

3. Advantage of puppet ---

Configuration Management

4. High level arch of chef -----

chef workstation -> Chef server -> Chef Node

5. Default name given to the repository (origin is the default name given to any repository) -----

6. Cloud based deployment experience -----

AWS

7. My contribution in the recent project and day to day activity --

--

Writing puppet modules

8. Various plugin used in ur project -----

Backup, git , Delivery pipeline, Build pipeline , Cucumber , dashboard, Build graph etc.

9. How do u do roll back deployment, for eg- installed an app with version 6 and for some reason -----

By using git tags.

10.I have to roll back to version 5.9, how to roll back to version 5.9. ----

11.What is the achievement you have done in ur exp of devops -- --

12.Automation done using puppet,chef and ansible ----

13.What problem CI solves, advantage of CI. ----

What code is updated , It gets compiled automatically.

14.Different scripts used in ur project and what is the purpose of the scripts ----

15.How u have implemented devops in ur project ----

started writing puppet modules for infrastructure.

Ansible

16.Y do u use ansible for automation deployment, jenkins can also do the auto deployment, then y ansible is used y not jenkins. - Jenkins is a CI tool and ansible is very easy for pushing code to multiple systems.

17.different types of inventories in ansible - /etc/ansible/hosts, group_vars and host_vars.

18.Diff bet ansible and chef - ansible is agent less ans works on push method and chef works on pull method and has server-client.

19.in which language we write playbooks in ansible -

YAML

20.if we want system information of machine how we will get the data with ansible like we use facter in puppet - using module setup.

21.Activity that we use with ansibles or advantage Functionality or purpose of ansible using in your project - idempotency , simple because of modules.

22.Ansible Configuration files and process required to take control of other 3 machines - controller and entried in /etc/ansible/hosts.

23.Functionality use with ansible in ur project and what is the adv of having ansible in ur project---

For deployments and provisioning resources on AWS.

24.Activity we can do using ansible like main advantage of using ansible --- idempotency.

25.How to launch an LDAP server using ansible ---

Used LDAP module from galaxy as a reference and modified it according to our environment.

26.How ansible works.. and what are the playbook written other than the basic playbooks -- we are using even to provision AWS resources.

27.For example there are 4 aps server running and 4 websers running and we have a new code change and this change should automatically copy to these different servers using ansible, how I will come to know new code is generated. --- use Jenkins with POLL SCM option and after artifact is generated using ansible to push suing copy module. (make sure serial: 1).

28.Have u done any automation like app goes down then for self healing of the app kind of the thing. Have u written any playbook. (make sure serial: 1). We have written for Tomcat.

Maven and Ant

29.what is maven from where we get the pom.xml for the jobs.—

Maven is a build tool.

```
mvn archetype:generate
```

30.what is pom.xml ---

Any Maven project must have a pom.xml file. POM is the Maven project descriptor just like the web.xml file in your Java EE web application, or the build.xml file in your Ant project.

31.maven and ant difference and what are the different scenario where it is used----

Difference between Ant and Maven. Ant and Maven both are build tools provided by Apache. The main purpose of these technologies is to ease the build process of a project. Ant doesn't has formal conventions, so we need to provide information of the project structure in build.xml file.

32.what is ant and from where we get the build.xml for the jobs.---

33.how pom.xml will read the variables which we use in the jenkin like version.----

By invoking maven

34.what is dependency in pom.xml---

POM inheritance

35.Invoke maven task is there in post build option.

Jenkins

**36.are we in admin part of jenkins like installation of jenkins ----
yes**

**37different ways to install a plugin's in jenkins - using manage
plugins.----**

**38.how you created jobs in jenkins other than the GUI like new
item option.. other option is job dsl. ---- using Jenkins API**

**39.Why jenkin is required if we can do the same thing with script
like automatic. - Continous Integration and its plugin feature.**

**40.How many jobs are there in jenkins (for eg – ant , maven) -
ant is default Maven you have to install, , Gradle.----**

**41.steps for the installation of jenkins tomcat server (for tomcat
server install command is different) - see our copy.yml.**

42.yum install Jenkins will run on which server ? - Master server.

43.how do you do a depoyemnt. - using shell scripts

44.are you doing the deployment in an application server - yes.

**45.What is sonar qube and How to Integrate sonar qube with
jenkins - code metrics.**

46.Main reason of using jenkins - CI.

- 47.Commands to start jenkin manually through command prompts like how to up the server - service jenkins start.**
- 48.How to Safe restart the jenkins - Restart when no jobs are running.**
- 49.How to create a backup in jenkins, what is the plug in name complete process. From which directory it will take the back up (jenkin_home) directory. - Backup Plugin.**
- 50.Step to setup a jenkins and how to create a new job - New item-> Frees style.**
- 51.How to deploy a Custom build in jenkins - usinfg build tools such as ant and maven.**
- 52.Where we store the artifacts which gets generated from jenkins - local and nexus .**
- 53.How to bypass the github like directly using the code and put it in jenkins--- t**
- 54.Jenkins used a entriprised or a open source u used - ---open source.**
- 55.Different Pluging used in jenkins ---build pipeline, Backup, git**
- .

Git

- 56.pull request in git hub - ---git pull**
- 57.fork a repository in git hub----**

Forking Projects

If you want to contribute to an existing project to which you don't have push access, GitHub encourages forking the project. When you land on a project page that looks interesting and you want to hack on it a bit, you can click the "fork" button in the project header to have GitHub copy that project to your user so you can push to it. This way, projects don't have to worry about adding users as collaborators to give them push access. People can fork a project and push to it, and the main project maintainer can pull in those changes by adding them as remotes and merging in their work.

58.Diff between svn and git ---svn is centralized version control and git is Distributed Version Control.

Linux

59.How to delete different lines in a file in linux eg:- line 1,4,6,10. ---

Sed '1d;4d;6d;10d' filename

60.difference between soft link and hard link in linux---

Hard Link acts like a mirror copy of the original file. These links share the same inodes. Changes made to the original or hard linked file will reflect in the other. When you delete Hard Link nothing will happen to the other file. Hard links can't cross file systems.

Soft Link is actual link to the original file. These Links will have a different Inodes value. Soft link points to the original file so if original file is deleted then the soft link fails. If you delete the Soft Link, nothing will happen to file. The reason for this is, the actual file or directory's inode is different from the "soft link" created file's inodes. Soft links can cross file systems

61.can we create a soft link and hard link for file and dir---

Hard linkink of directorties is not possible

62.need to create a user and dont want the user to get the shell access--- put shell as /bin/false

63.Top command in linux--- top (to check memory, process usage)

63.How to check dynamic log files in linux--- tail -f filename

Shell

64.Define a shell script---

A shell script is a text file that contains a sequence of commands for a UNIX-based operating system. It's called a shell script because it combines into a "script" in a single file a sequence of commands that would otherwise have to be presented to the system from a keyboard one at a time.

65.What are the shell script written in ur projects---

66.How to execute a shell file in debug mode---

Set +x

67.File contains my name 10 times we have to write the script where I have to count the number of times my name appears in the file. ---

vi abc.txt

#!/bin/bash

grep \$1 | wc -l

sh abc.sh filename

Nagios

68.What types of monitoring and what are the monitoring u have done on daily basis - Responding to the alerts. Adding new nodes and services on those nodes. ---

I have written puppet nrpe module.

69.How nagios works or how u monitor the servers - ---NRPE.

70.written any plugin in nagios ---- I have plugins generated from Nagios plugin installation and plugins from exchange.nagios.com.

Puppet:

71.What is puppet?---

Puppet is a configuration management tool though it can be used for orchestration also.

72.How to install puppetserver?---

Puppet is available in both open source and Enterprise edition.

Open source is available in EPEL.

Enterprise has to be downloaded from puppetlabs site.

73.How to setup environment in puppet?---

By setting Modulepath and Manifest destinations in puppet.conf.

74.What is puppet.conf/ what is puppet configuration file name? ---

Puppet.conf is main configuration file in Puppet.

75.Sections in puppet.conf file? ---

main, master and agent

76.How to setup agent in puppet? ---

By installing puppet rpm.

77.Where do we mention puppet server information in puppet agent? ---

In puppet.conf under agent section.

78.What port puppetmaster listens? ---

see below notes

79.What port puppet agent listens? ---

see below notes

80.Where do we set puppet agent node definition in puppetmaster? ---

in site.pp file.

81.Default & environment both? ---

production

82.What are puppet manifests? ---

file should have and extension of .pp file and inside this file we will define resources using puppet DSL.

83.Location of puppet manifests, default and environment both? ---
Are mentioned in puppet.conf

84.Name 10 puppet resources? ---
check your cheat sheet.

85.What are selectors? ---

condition in puppet can be set using if/else, case and selectors.

86.What are classes in puppet and how to define it? ---

class is a collection of resources. is specified inside manifest file.

87.Installing multiple packages in by using variables? ---
variables can be defined using \$variablename.

ex: \$name1 = 'httpd'

```
class httpd {
  package {'httpd':
    package_name => $name1,
  }
}
```

89.What modules in puppet and how to create it? ---

Modules is collection of the files, templates, manifests required to implement a scenario.

Example: Setting up Web server

90.How to use puppetforge modules? ---

To download puppet modules using puppet module install.

91.Conditional statements in puppet? ---

if/else, case and selectors/

92.How to finetune or customize puppetforge modules? ---

Download the module from puppetlabs and modify it according to our environment.

Chef :

93.what is chef work flow----

Workstation -> Server -> Nodes

94.how we get the properites of node----

Ohai

Tool used to detect attributes on a node and then provide attributes to chef-client at the start of every chef-client run

95.what is resource ----

Resources:

A statement of configuration policy within a recipe Describes the desired state of an element in the infrastructure and steps needed to configure

96.what is diff b/w chef server and chef client----

Chef-Client:

Agent that runs locally on the node that is registered with the chef server

Chef Server:

Chef server is the hub for all configuration data, stores cookbooks, and the policies applied to the node

97.what is run list-----

Nodes receive their policy based off of roles and individual node configurations

A run list defines the order in which you want your recipes to run during convergence

98.what is diff b/w cookbook and recipe-----

Recipes are made up of a collection of resources

Cookbooks are made up of a collection of recipes

99.how bootstrap will work & what is process-----

Nodes should be bootstrapped and managed from the workstation

Nodes should be assigned roles and environments

Attributes specific to roles/environments should be configured accordingly

100.which language chef deploy?

ruby

What is Puppet?

Infrastructure automation and configuration management tool

Enforces the defined state of the infrastructure

Puppet can automate tasks on thousands of machines.

Puppet enables infrastructure as code

Puppet allows configuration consistency across nodes

Puppet enables quick provisioning of new machines in an environment

Puppet allows DevOps admins to write declarative instructions using the Puppet language

DevOps admins write code using the Puppet DSL to express the desired state of a node

Code is written inside of classes and classes are assigned to node

Node classification is the process of assigning a class to a node for processing

Puppet Ports

- 3000 - Web-based installer of puppet master
- 8140 puppet requests

- 61613 - orchestration requests
- 443 for console management

Example: Puppet language DSL

```
{  
  Class motd  
  {  
    file { "/etc/motd":  
      ensure => 'file',  
      source =>  
      "puppet:///modules/motd/motd",  
    }  
  }  
}
```

Puppet Master

Compile and serve configuration catalogs to agent nodes

Issue orchestration commands via MCollective

Availability of puppet enterprise web interface console

Collect data and compile reports from agent nodes

What is a catalog?

The catalog is a document downloaded from the puppet master to the agent that describes

the desired state for each resource that should be managed. It may also specify dependency information for the resources that should be managed in a certain order. This is essentially a compiled version of the DSL and is compiled on the Puppet master and stored in PuppetDB if PuppetDB is used.

Puppet enterprise includes MCollective which is an orchestration system that allows the admin to invoke different commands in parallel across a select group of nodes.

Puppet Process

**Install Puppet master and Puppet agents
Create configurations (modules/classes) that declare a resources desired state
Assign configurations to nodes**

Puppet nodes (agent)

Nodes are any virtual or physical system that is able to run Puppet Agent and is specifically supported by puppet agent.

Required Ports:

8140 for puppet requests

61613 for orchestra0on requests

Puppet.conf

Puppet Enterprise: /etc/puppetlabs/puppet/puppet.conf

Puppet OpenSource /etc/puppet/puppet.conf

Config Sections:

[main] – Global section used by all services/servers

[master] – Use by Puppet master and puppet cert command

[agent] – Used by the Puppet agent service (even if the agent runs on the master)

[user] – Used most commonly by the Puppet apply command

Resource Abstraction Layer

A system configuration is a collection of resources that make up the state of your system

Users

Cronjobs

Packages installed

Services

Etc.

Resources

When building modules we are using the puppet DSL to declare the desired state of resources on a node

Fundamentally all we are doing with Puppet is managing resources on a large and automated scale.

In Puppet we are using resource types to define instances of a resource on a node

Chef

Common Chef Terminology

Recipes:

Fundamental configuration element within an organization

Cookbook:

Defines a scenario and is the fundamental unit of configuration and policy distribution

Chef--Client:

Agent that runs locally on the node that is registered with the chef server

Convergence:

Occurs when chef--client configures the system/node based off the information collected from chef--server

Configuration Drift:

Occurs when the node state does not reflect the updated state of policies/configurations on the chef server

Resources:

A statement of configuration policy within a recipe Describes the desired state of an element in the infrastructure and steps needed to configure

Provider:

Defines the steps that are needed to bring the piece of the system from its current state to the

desired state

Attributes:

Specific details about the node, used by chef-client to understand current state of the node, the state of the node on the previous chef-client run, and the state of the node at the end of the client run

Data-bags:

A global variables stored as JSON data and is accessible from the Chef server

Workstation:

A computer configured with Knife and used to synchronize with chef-repo and interact with chef server

Chef Server:

Chef server is the hub for all configuration data, stores cookbooks, and the policies applied to the node

Knife:

Command line tool which provides an interface between the local chef-repo and chef-server

client.rb

Configuration file for chef-client located at /etc/chef/client.rb on each node

Ohai:

Tool used to detect attributes on a node and then provide attributes to chef-client at the start of every chef-client run

Node Object:

Consists of run-list and node attributes that describe states of the node

Chef-Repo:

Located on the workstation and installed with the starter kit, should be synchronized with a version

control system and stores Cookbooks, roles, data bags, environments, and configuration files

Organization:

Used in chef enterprise server to restrict access to objects, nodes environments, roles, data--bags etc

Environments:

Used to organize environments (Prod/Staging/Dev/QA) generally used with cookbook versions

Idempotence

Means a recipe can run multiple times on the same system and the results will always be identical

Jenkins

Agenda:

1. Installing Jenkins, setting up, configuration
2. Security LDAP - authorization
3. How to schedule a job
4. integration:

 Integration with GIT/GITHUB

 Integration with Ant

 Integration with Maven

 Integration with Ansible

 Integration with Nexus

- 5.Troubleshooting (location of log files of startup, failed plugin and etc)

6. how to schedule backups

7. restore from backups

8. Configure Slaves

9. Monitoring the workload

10. Managing plugins

What is Jenkins:

Leading open source continuous Integration server

Highly configurable system by itself

400+ community developed plugins

Multi platform

Easy to use

Free & Open source

<https://jenkins.io/index.html>

Features:

- Jenkins provides continuous Integration services for software deployment.
- Based on tomcat
- Originally started as Hudson then renamed as Jenkins
- leading open source continuous integration server.
- Flexibility
- Jenkins is a highly configurable system by itself.
- 400+ community developed plugins
- Less Effort to Integrate existing existing test buckets.
- Multi platform
- Easy to use
- Free & Open Source
- Documentation: <https://jenkins.io/index.html>

Failing is Ok, but we need to fail fast

Catching a bug in test costs less than catching in production

Code quality review

Proper documentation

This can be done using custom scripts, but as the project increases it is difficult to maintain.

There was split in Hudson when oracle took over SUN,

90% Hudson and Jenkins are same

Build can be started by

Commit a version in version control

Scheduling via cron like mechanism

Tests can be executed every week/hour etc

If A fails do B, if A passes do C – jobs can be dependent like a pipeline

What is a build:

Build has many components – one of them is version control

No matters how many times a code changes, build has to happen, code has to be compiled.

Build also has to be tested

Communicate the results to all stake holders

- Product owner
- System admin
- QA
- Developer

History – can be maintained

Continuous delivery vs Continuous deployment

Continuous Deployment: deploying the product to customers continuously without approval, this is automatic

Continuous delivery: approval is needed, ready to deploy. This is not automatic

Test cases have to be automated for Continuous Integration

Download Jenkins:

Download LTS – long term support , not weekly build which is not safe

Download native packages not war packages

Jenkins rpm comes with webserver

Download Stable Jenkins rpm from <http://pkg.jenkins-ci.org/redhat-stable/>

1. Install it using rpm -ivh

```
root@ctx3p12 MISC]# rpm -ivh jenkins-2.9-1.1.noarch.rpm
```

```
warning: jenkins-2.9-1.1.noarch.rpm: Header V4 DSA/SHA1 Signature, key ID d50582e6: NOKEY
```

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

```
Updating / installing...
```

```
1:jenkins-2.9-1.1      ##### [100%]
```

2. Install Java using yum

3. Start the Jenkins service

```
[root@ctx3p12 MISC]# sudo systemctl start jenkins.service
```

```
[root@ctx3p12 MISC]# cat /var/lib/jenkins/secrets/initialAdminPassword
```

a45c1a99da7c49518ce07b7db91950e4

[root@ctx3p12 MISC]#

firewall-cmd --permanent --zone=public --add-port=8080/tcp

to open port 8080 on Jenkins server

Jenkins by default listens on port 8080

So open <http://FQDN:8080>

To change port

/etc/sysconfig/jenkins

http port to 8089

jenkins home

/var/lib/jenkins/config.xml

How to install

sudo yum -y install java

java is a prerequisite for Jenkins.

yum whatprovides service

sudo wget -O /etc/yum.repos.d/jenkins.repo <http://pkg.jenkins-ci.org/redhat/jenkins.repo>

- Downloads the repo file from Jenkins site

```
sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key
```

- Imports the key from Jenkins site for authentication

```
sudo yum install jenkins
```

- Downloads and installs the Jenkins packages from Jenkins repository

```
sudo systemctl status jenkins.service
```

- Start the service and check the status – it should be Active

Download Jenkins

Install it

Configure plugins

Install all of them

Install GIT related them

Installing all may slow down Jenkins

Its not recommended to enable auto refresh

URL/restart → will restart Jenkins

Manage Jenkins → Configure global security → click on enable security

Authorization → any one can do any thing

Security realm Jenkins own user database

LDAP is for corporate security to sign up using intranet id

Manage Jenkins → project based authority → enable everything for admin enable all permissions

Setup security

Click on Enable-security

Difference between save and apply

Apply will remain in the same, save will go back to previous page

Manage Jenkins -> configure system

No. of executors – no. of core you have on CPU

GIT hub server - create user id in git hub server

Add git hub server

Go to client

Git clone <https://github.com/RSCTFVT/>

Commit back from SVN

Add a file here

Git add *

Git commit –m “some commit”

Jenkins was developed by Kohsuke Kawaguchi.

Backup of Jenkins:

1. Take backup of home directory using tar -cvf command

```
tar -cvf jenkins-backup-21dec.tar Jenkins
```

```
gzip jenkins-backup-21dec.tar
```

then copy it to other machine

Incase of crash, extract this file in newly created Jenkins server

2. Take backup of single job using xml

<http://ctx3p11.in.company.com:8080/job/RSCTBAT/config.xml>

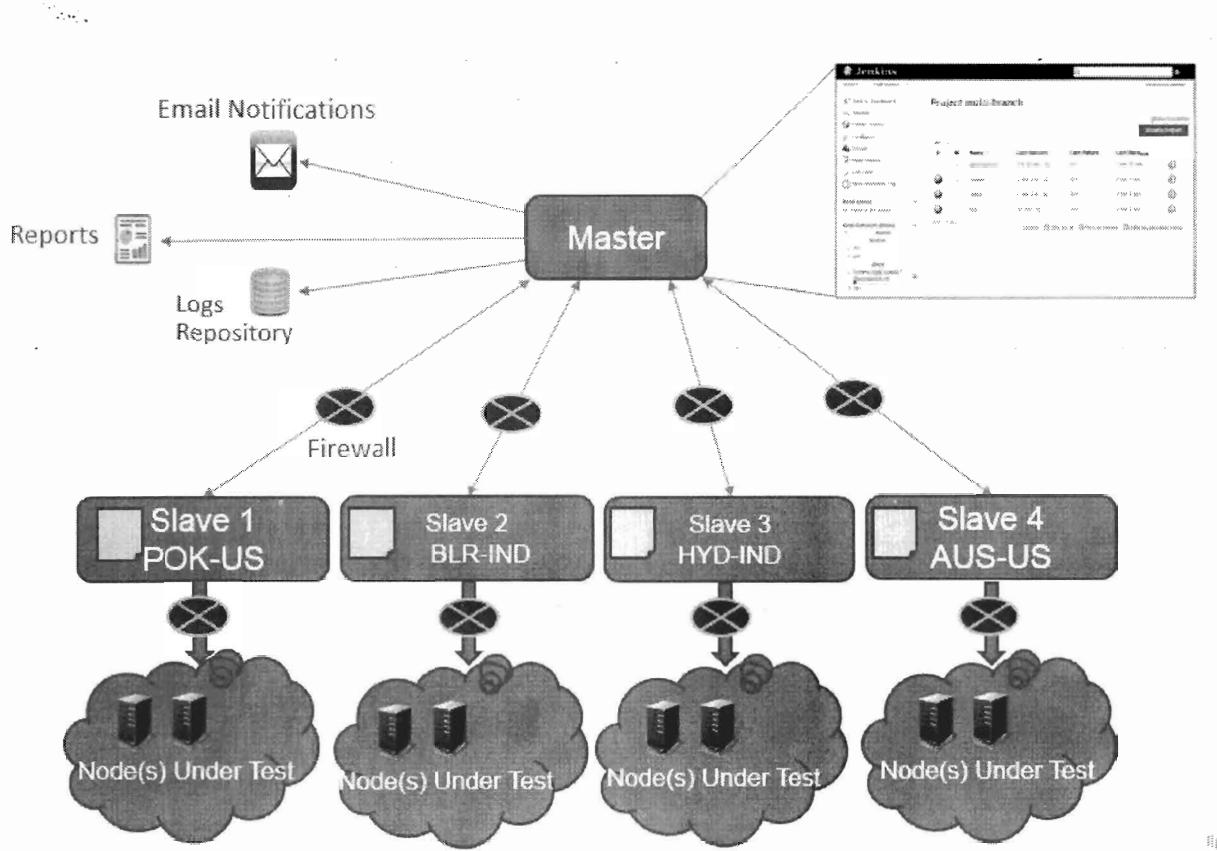
open a job with config.xml extension and take backup in a file.

After installation:

```
[root@jupiter-vm751 ~]# cat /var/lib/jenkins/secrets/initialAdminPassword
```

```
1b5561f487c14ea3b9c6c386eca33fcf
```

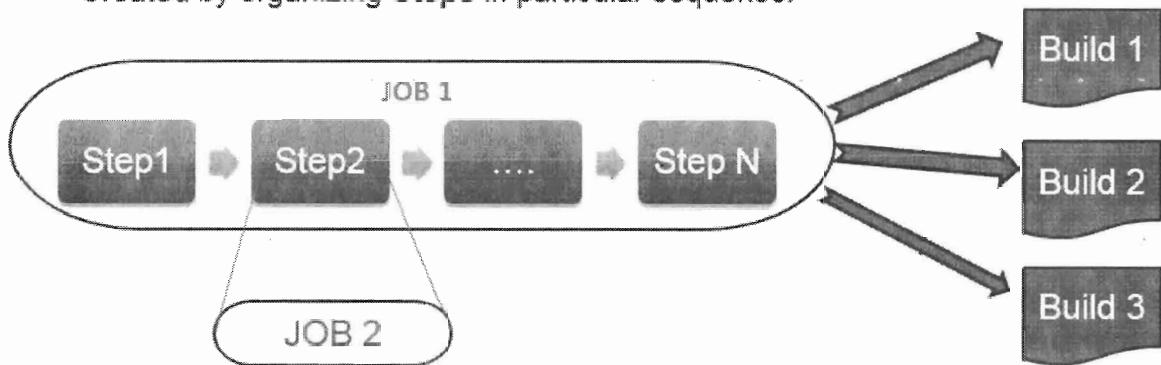
Configuring Slaves:



Jenkins Basic Terminology

- **Job**

- They all refer to runnable tasks that are controlled / monitored by Jenkins.
- Created by organizing **Steps** in particular sequence.



- **Build** : Result of one run of a Job.
- <https://wiki.jenkins-ci.org/display/JENKINS/Terminology>

GIT

Git is becoming popular in Devops world

Git is to keep track of changes

Ansible/puppet/chef/vagrant – files we store on GITHUB

Why GIT :

1. Build & Release - devops eng works with developers
2. For storing automation scripts
3. Open source - so many open source projects are hosted on GIT (Linux, chef, puppet etc)

System admins take backup by copying.

Cp file file_1

Cp file_1 file_2

Its difficult to track

Or we can name with date

That is also difficult to track

If there are many scripts – how to handle ?

There should be a centralized location to keep track of all changes – that is called version control system

Don't keep track

Save numbered zip file

Fomal version control

Diff , windiff – can see the difference

Even that is difficult

Developed by Linux torwalds – keep track of source code

Used for just for source code

Do not use for file storage

Its fast

Don't need access to server

Good at merging simultaneous changes

Everyone is using it

Bitbucket is alternative

GIT HUB – to store our scripts

Singup to github

Just give username and password

Login to it

Create new repository

Other CI tools:

Hudson

Bamboo

Microsoft tfs – team foundation server

Mercuyrail

Configuring slaves on the Jenkins master:

1. Create jenkins user on the slave

```
useradd -d "/home/jenkins" -c "jenkins user" -s "/bin/bash" Jenkins
```

and set the password to slave as root

```
passwd Jenkins
```

2. Configure password less authentication on the server and slave for the Jenkins user

Login as Jenkins to master and execute

```
ssh-copy-id -i ~/.ssh/id_rsa.pub jenkins@rscthycloud1
```

3. Go to Jenkins → manage nodes → New node,

4. Select as Permanent agent

5. Select options as below

The screenshot shows the Jenkins 'Nodes' management interface. On the left sidebar, 'New Node' is highlighted. The main form is for creating a new node named 'slave1'. It includes fields for description, number of executors (set to 10), remote root directory ('/home/jenkins'), labels ('slave1'), usage ('Use this node as much as possible'), launch method ('Launch slave agents via SSH'), host ('slave1'), and credentials ('jenkins*****'). Under 'Availability', the option 'Keep this agent online as much as possible' is checked. At the bottom, there are checkboxes for 'Node Properties' such as 'Date pattern for the BUILD_TIMESTAMP variable', 'Environment variables', and 'Tool Locations', none of which are checked. A 'Save' button is visible at the bottom left.

Once slave is added, make it online by

Clicking on

Manage Jenkins → Manage Node → Click on Slave → Launch agent

This will make the slave online

Integration with GIT:

Install git on slave

Register your self on github and create a sample project

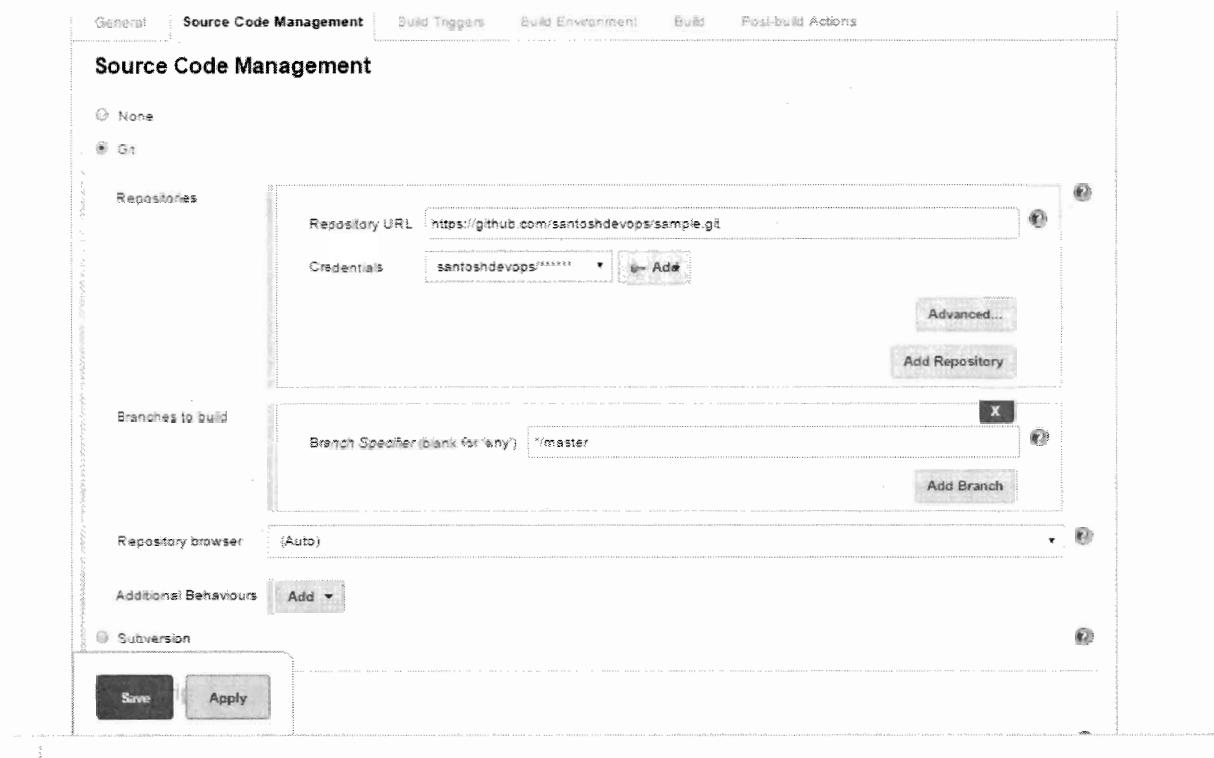
Add git plugin to Jenkins

Create a new job

Select Source Code Management as git

Provide the Repository URL as the git url (copy from clone or download option on git hub)

Provide the credentials (Same as github)



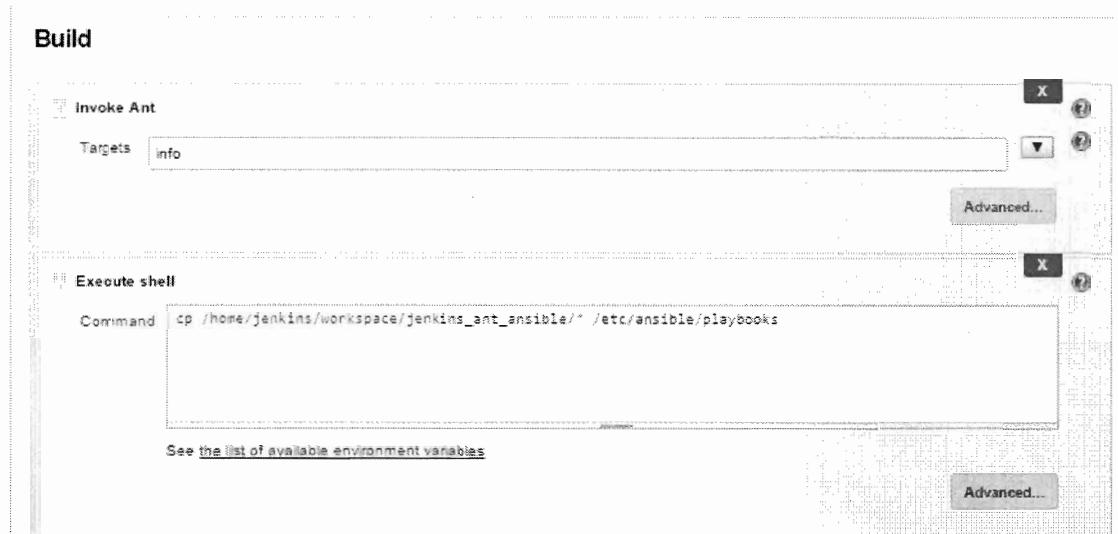
Select build triggers as Poll SCM for every minute - * * * * *

To run this project on a particular slave

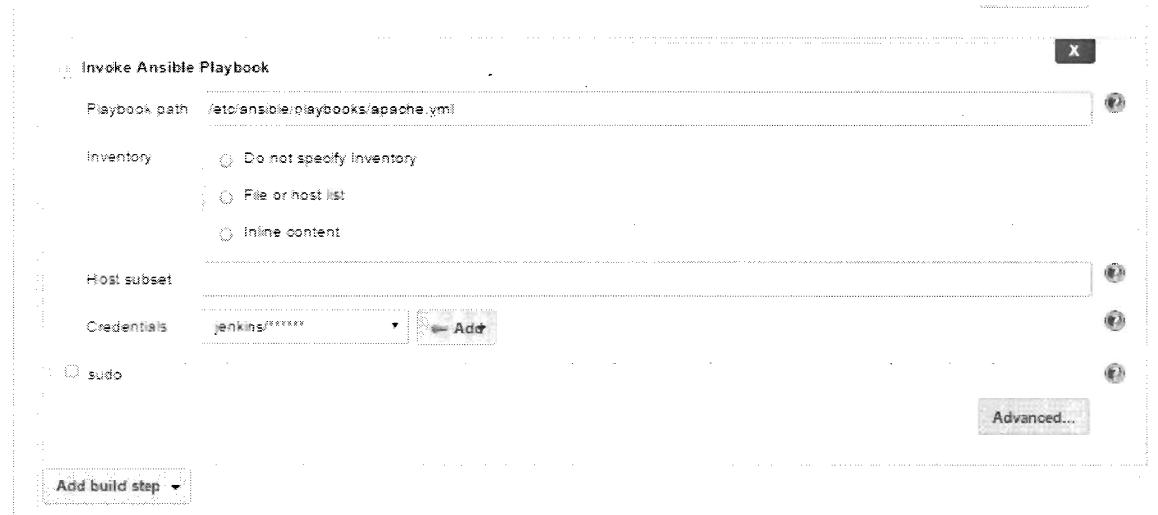
Click the checkbox

Restrict where this project can be run and select label of the slave.

Ant:



Ansible



Integration with Nexus build repository:

Nexus is a build repository stores the builds in a hierarchical directory structure.

Nexus can

- Gives a centralized repository for managing all popular component formats.
- Gain insight into component security, license, and quality issues.
- Improve productivity by efficiently distributing components to developers around the corner, or around the world.
- Modernize software development with intelligent staging and release functionality.
- Sleep comfortably with world-class support and training.

How to install Nexus:

```
cd /usr/local/  
wget http://sonatype.org/downloads/nexus-latest-bundle.tar.gz  
gunzip nexus-latest-bundle.tar.gz  
tar -xvf nexus-latest-bundle.tar  
mv nexus-2.14.3-02 sonatype-work /usr/local
```

```
ln -s nexus-2.14.3-02 nexus
```

```
cd nexus  
export RUN_AS_USER=root
```

./nexus start

Make sure nexus is running by executing “ ps -ef | grep -i nexus ”

admin/admin123 – default password for nexus login.

Open nexus GUI:

URL: <http://rscthydnet2:8081/nexus/#welcome>

Create a repository in Nexus to host builds

Go to repository → add hosted repository

The screenshot shows the 'New Hosted Repository' configuration page. The fields are as follows:

- Repository ID: gameoflife
- Repository Name: gameoflife
- Repository Type: hosted
- Provider: Maven2
- Format: maven2
- Repository Policy: Release
- Default Local Storage Location (empty)
- Override Local Storage Location (empty)

In the Jenkins add the plugin **Nexus Artifact Uploader**

Also install the timestamp plugin to use timestamp variable in the job

Configure the job as below

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Project name: March17nexus

Description:

[Plain text] Preview

Change date pattern for the BUILD_TIMESTAMP (build timestamp) variable

Date and Time Pattern: YYMMDDHHMMSS

Discard old builds

GitHub project

Nexus artifact uploader

Nexus Details

Nexus Version: NXUS2

Protocol: HTTP

Nexus URL: cmq201.ibm.com:8081/nexus

Credentials: admin*****

Group: DEV

Version: \$BUILD_ID

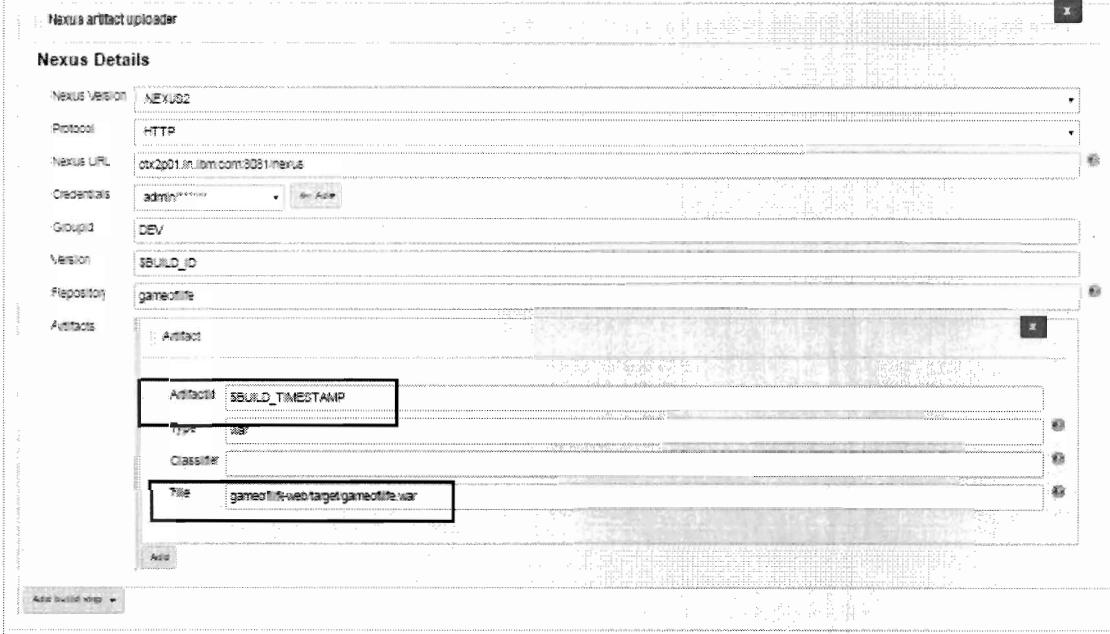
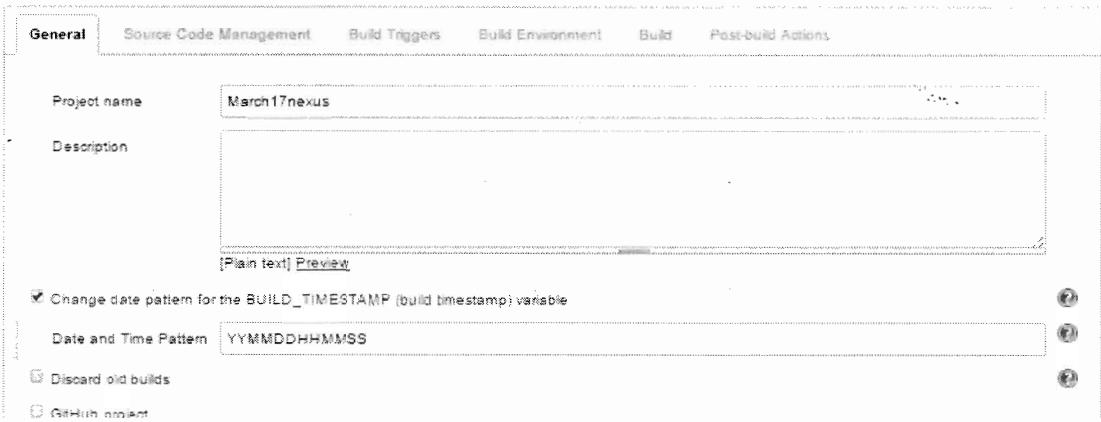
Repository: gametlife

Artifacts:

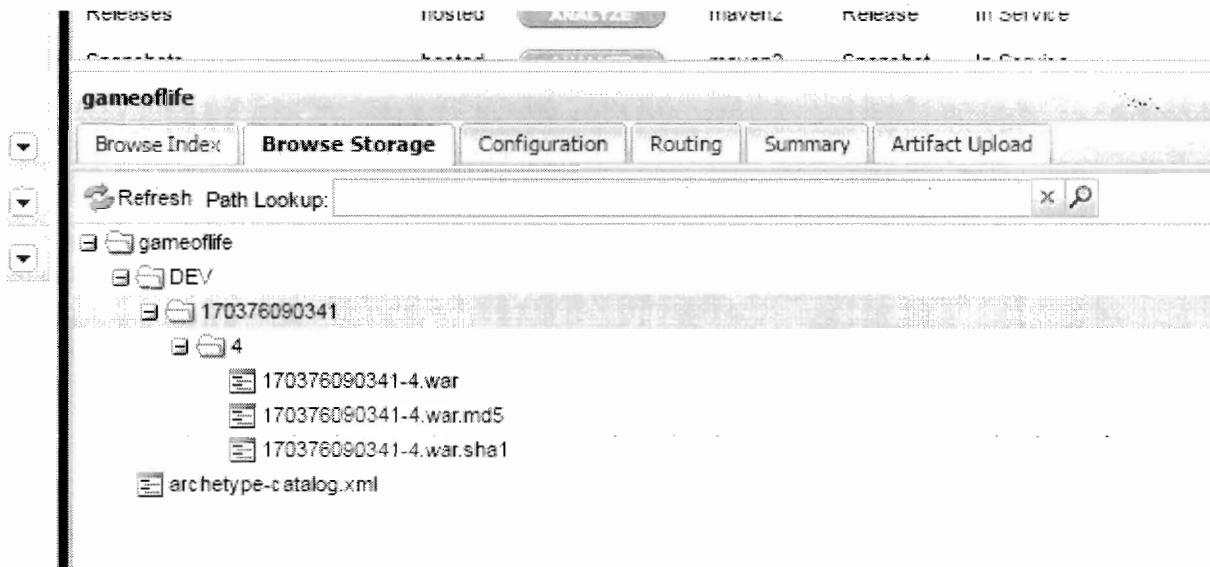
Artifact: \$BUILD_TIMESTAMP	Type: war
Classifier:	Title: gameLife-web-target-gameLife.war

Add

Add build step



After job is executed nexus creates repos as below



And it can be accessed using a link

Index of /repositories/gameoflife/DEV/170376090341/4

Name	Last Modified	Size	Description
Parent Directory			
170376090341-4.war	Fri Mar 17 09:21:16 IST 2017	3192487	
170376090341-4.war.md5	Fri Mar 17 09:21:17 IST 2017	32	
170376090341-4.war.sha1	Fri Mar 17 09:21:17 IST 2017	40	

Jenkins logs:

Jenkins logs will be stored under :

/var/log/jenkins/jenkins.log

Some commonly used plugins:

Extended choice parameter

Build with parameters

Rebuild now

Html report generator

Junit report generator

Thinkbackup

Monitoring

Linux Basics & Shell Scripting

Linux Introduction:

Linux is Open Source Operating system developed by a Finnish student LinuS Torvalds in 1991.

It is free to download and use.

Linux is more reliable

Linux is compatible on many h/w like Macs, Mainframes, super computers, cell phones etc.

Very resistant to malware such as viruses, spyware etc.

What is Operating System:

Operating system is the software which acts as the interface between the hardware and the software we want to run on the hardware

Ex: Microsoft office is the software

Microsoft windows is operating system

Laptop/desktop is hardware

Ex:

Mobile is hardware

Examples of mobile hardware:

Symbiosis

Ios

android is OS

Application is the software

Unix was developed in 1970 by Denis retchi at Bell labs

Linux was developed by Linux torwalds in the year 1990.

Unix vs Linux

Unix is called mother of OS which laid foundation to Linux

Unix is designed mainly for mainframes and is in enterprises

Linux is for computer users, developers, servers

Linux is free, Unix is not

Unix runs on specific hardware only (AIX on IBM boxes)

Linux runs on many h/w

Linux Architecture

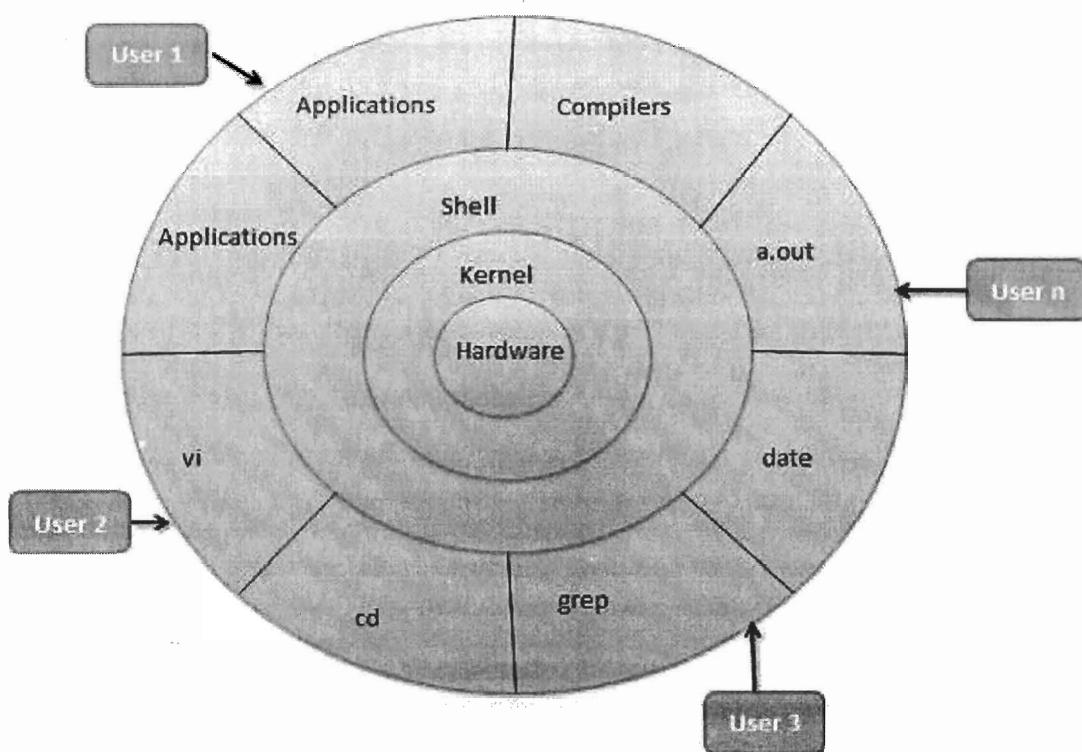
Linux has

H/w includes HDD RAM CPU

Kernel Heart of OS, talks to H/w, provides low level services to upper layer components

Shell interface between Kernel and user

Applications/Utilities – provide most of functions to users.



Shell:

Shell is the screen that user use to interact with operating system

Windows shell is the GUI – click on icons – it is called GUI shell

LUI – Line User interface – looks like DOS (Disk Operating system)prompt

Type a command and hit enter

Line user interface much more powerful

With LUI we get prompt to work on .

Bash shell is the default shell in Linux

ROOT:

Like Administrator for windows machine

Highest level of user/anything

As root anything can be performed

In Linux users will have home folders

Home directory – is the highest level of any user folder

root user – highest user

root directory – highest folder, home directory

Capitazalation:

Upper case and lower case letters are different

Home/home/hOME- all are same in windows

All are different in case of Linux

Because each letter is ASCII character in Linux

Same for username

Windows is case insensitive, 2 files cannot be created wth names File, file.

Linux is case sensitive, 2 files can be created with names File, file

Server vs Desktop:

Server version is stripped down version of Linux – no GUI, lot of other tools are not installed automatically, it has what a server needs.

Desktop version: gives GUI version of Linux – looks like Windows/Mac.

Every distro has desktop and server editions ex: readhat/Ubuntu etc

As a desktop OS, Linux is not best choice.

Linux is good for Server configurations.

For setting up for website, does need to reboot often.

Linux Distributions:

People started developing their own code out of kernel developed by Linus

So there are many versions of linux which is called distributions/flavours

Example:

Redhat

Ubuntu

Google android

Fedora

Centos

Every distribution has a different purpose

Trustix linux – most secure

Laptop desktop – Ubuntu

Enterprise (On servers)– Redhat linux - customer center to support

DSL - very small distro – 53 MB

Open source:

All open source software is not free (usage is free, support is not free)

Everyone is allowed to see the source code

Ex: mysql database

They will give free software, but support they will charge

They get money from support

Open source software can be used at home/personal use/ at test lab

But cannot be used on production server, we need license to use them.

Linux Boot process

What happens when a power button is pressed on the server/laptop

BIOS – Basic Input Output System – does POST Poweron Self Test – checks whether all i/o devices are working fine (mouse monitor keyboard RAM HDD etc), searches loads and executes boot loader program

MBR – Master boot record :

Located in the first sector of bootable disk. ex: /dev/sda or /dev/hda

first program to be executed in Linux, of 512 bytes,

it has 3 components

1) primary bootloader of 446 bytes

2) partition table info in next 64 bytes

3) MBR validation check in last 2 bytes

So it loads and executes GRUB into memory

The Master Boot Record (MBR) is the information in the first sector of any hard disk or diskette that identifies how and where an operating system is located so that it can be boot (loaded) into the computer's main storage or random access memory.

The Master Boot Record is also sometimes called the "partition sector" or the "master partition table" because it includes a table that locates each partition that the hard disk has been formatted into. In addition to this table, the MBR also includes a program that reads the boot

sector record of the partition containing the operating system to be booted into RAM. In turn, that record contains a program that loads the rest of the operating system into RAM.

GRUB – Grand Unified Boot loader - responsible for selecting OS, loads kernel into memory

if multiple OS images are present, one image can be chosen using GRUB

GRUB displays a splash screen, waits for sometime, if user does not choose anything, it loads default kernel

So it loads and executes Kernel

Kernel –

executes init program

PID 1

init

Looks at /etc/inittab file to decide run level

Runlevel programs

Basic runlevels are:

0 – halt

1 – Single user mode

2 – Multiuser, without NFS

3 – Full multiuser mode CLI no graphics with network

4 – unused

5 – X11 – graphics, multiuser mode with network.

6 – reboot

Run levels:

Run level 0 – /etc/rc.d/rc0.d/

Run level 1 – /etc/rc.d/rc1.d/

Run level 2 – /etc/rc.d/rc2.d/

Run level 3 – /etc/rc.d/rc3.d/

Run level 4 – /etc/rc.d/rc4.d/

Run level 5 – /etc/rc.d/rc5.d/

Run level 6 – /etc/rc.d/rc6.d/

Under the /etc/rc.d/rc*.d/ directories, you would see programs that start with S and K.

Programs starts with S are used during startup. S for startup.

Programs starts with K are used during shutdown. K for kill.

There are numbers right next to S and K in the program names. Those are the sequence number in which the programs should be started or killed.

Linux Installation

Linux can be installed using CD/USD/using iso image/using network installation like kickstart

Or single machine can be installed using iso (which can be downloaded from net)

Linux can be installed using iso

iso – international standard organization

example:

redhat-7.3-x86_64.ga.iso

Directory structure:

```
/  
  /root  
  /tmp  
  /dev/ /dev/sda, /dev/sdb  
  /bin/  
  /lib  
  /usr  
  /var  
  /etc  
  /home  
  /boot  
  /opt
```

iso;

RHEL-7.2-x86_64.iso → RHEL OS on x86_64 hardware

SUSE-12.SP1-ppc64le.iso → SLES on ppc64le hardware

Ubuntu-16.10-x86_64.iso → Ubuntu on x86_64 hardware

FQDN – Fully qualified Domain Name –

ctx1p25 – shortname

ctx1p25.in.xyz.com – FQDN

Basic Commands

ls; ls -lrt

ls list files in a directory

ls -lrt → lists files in long list format with time stamp

root@ctx2p02:~# ls -lrt

total 44

-rwxrwxrwx 1 root root 179 Nov 24 06:31 3.sh

-rwxrwxrwx 1 root root 277 Nov 24 07:07 fib.sh

-rwxrwxrwx 1 root root 103 Nov 24 07:19 1.sh

-rwxrwxrwx 1 root root 424 Nov 24 09:21 2.sh

-rwxr--r-- 1 root root 22 Nov 24 10:34 file

-rwxrwxrwx 1 root root 171 Nov 24 20:45 7.sh

-rwxrwxrwx 1 root root 206 Nov 24 21:58 fact.sh

-rw-r--r-- 1 root root 82 Nov 24 21:59 8.sh

-rwxrwxrwx 1 root root 936 Nov 25 05:49 10.sh

-rwxrwxrwx 1 root root 73 Nov 25 08:42 until.sh

-rw-r--r-- 1 root root 9 Nov 25 09:14 file90

root@ctx2p02:~#

cd; cd - ; cd ../../ abs vs rel path, cd ~

cd changes directory

cd .. → goes one directory up

cd - → takes to previous directory

cd ~ → take to user home directory

mkdir , mkdir -p

`mkdir` → create directory

`mkdir -p` → creates path

`rm , rm -rf`

`rm` → remove a directory

`rm -rf` → removes directory recursively forcibly

`rmdir` → remove a directory

`pwd` → shows present working directory

`umask` → decides the default permissions with which a file/directory will be created by a user.

`chmod` → change permissions of a user

`chown` → changes user

`mv` → move a file or a rename

`cp, cp -r`

`cp` → copies a file

`cp -r` → copies all files in a directory.

`scp` → secure copy protocol

syntax:

to transfer a file file from machine with ip 192.168.100.100 to 192.168.100.101

to remote directory /tmp from /var

`scp /var/file username@192.168.100.101:/tmp`

to transfer from 192.168.100.101 to 100

`scp username@192.168.100.101:/tmp/file .`

`ftp` – file transfer protocol

syntax: `ftp ip` or `ftp FQDN`

mget tp get all files with pattern matching

ssh – secure shell – to connect to a linux machine, listens on port 22

telnet - to connect to linux box, listens on port 23

ssh is preferred as the password is sent encrypted over network

mount

to attach a file system from remote server to local machine.

for this NFS server has to be configured

syntax:

mount -o nfsvers=3 192.168.100.101:/mount /mnt

mounts file system from /mount from 192.168.100.101 to local machine in the mount point /mnt

umask to decide the default permissions with which a file/directory will be created by a user.

Default

Umask 022

For a directory:

Read permission – to list out the files

Write - create rename delete files within directory.

Execute – change the directory

redirection operator > >>

> this operator creates a new file , overwrites if any file exists

>> this operator creates a new file, appends to file, if the file exists

pipe → to pass output of one command to other command as input

cat → to display content of file on terminal

Shell properties

control U → remove everything on the line

control A → to go beginning of line

control E → go to end of line

control R reverse → search the history based on a word

control w → remove word

Editors:

vi vim editor

insert mode – press i to go to insert mode

command mode - press ESC to go to command mode

yy to copy a line

p to paste

dd to cut

2yy – copies 2 lines

:1 → take to 1st line in file

:\$ → takes to last line

:set nu → sets numbers

:se ic → case insensitive

:se nonum → removes numbers

:se noic → removes case insensitive

:w → save file

:wq save and quit

:q quit

:q! quit without saving

working on multiple file

vim → Improved version of vi editor

can open multiple files

split files

vim -o file1 file2 → to open 2 files

vim -O file1 file2

copy few lines from file1 to file2

to switch between files control+w

nano

control +x

control + x 11

grep

awk

sed

umask normal user 002

root 022

head → displays first n lines

syntax: head -10 file

tail → displays last n lines

syntax: tail -10 file

nohup

- ➔ to execute a command in background in no hangup mode
- ➔ nohup command &
- ➔ logs output to nohup.out

to display contents of a file

less

more

sed - stream editor

grep - global regular expression print

awk

package management commands

redhat:

rpm – redhat package manager

rpm –ivh → to install

rpm –qa → to search

rpm –e → to remove

Ubuntu

dpkg

dpkg –l → to list installed packages

dpkg –i → to install

dpkg –r → to remove

dpkg –purge → to remove completely

To install packages to handle dependencies automatically

for redhat/centos/fedora

yum

yum install <>

to configure:

/etc/yum.repos.d/iso.repo → to install from iso

/etc/yum.repos.d/redhat.repo → from net

yast → on SUSE linux

AIX commands:

lslpp -l → to list installed packages

installp → to install

grep command:

grep = global regular expression print

to search a pattern in a file

grep -i → case insensitive

grep -w → to search for a word

grep -n → to display number of pattern match

grep -q

grep -A3

grep -v → -ve search

egrep '|'

fgrep old

egrep

extended grep

grep uses basic regular expressions where the plus is treated literally, any line with a plus in it is returned.

grep "+" myfile.txt

egrep on the other hand treats the "+" as a meta character and returns every line because plus is interpreted as "one or more times".

egrep "+" myfile.txt

fgrep

fixed grep

will not take meta characters

```
grep "." myfile.txt
```

The above command returns every line of myfile.txt. Do this instead:

```
fgrep "." myfile.txt
```

Then only the lines that have a literal '.' in them are returned. fgrep helps us not bother escaping our meta characters.

? one

* 0 or more

[]

telnet

ssh

ping

traceroute

nslookup

tar

untar

zip

yum

yast

apt-get

rpm

dpkg

top

pstree

vmstat

sar

iostat

stat

cpio

dd

fdisk -l

lsblk

partitions – primary

extended

DNS

LDAP

reverse DNS

ftp protocol flow

configuring yum – how it works internally

file system types: ext2 3 4 xfs zfs

find

find . -name xx

find . - mtime 5

find . -name xx -exec rm

LVM – creating

creating vg, lv, fs

RAIDs

software raid

hardware raid

how to configure raid on Linux

Linux Package installation

For installing a single package:

RHEL /CentOS/Fedora/SLES - rpm -ivh <xyz.rpm>

Ubuntu – dpkg -l <xyz.deb>

For installing all dependent packages:

Configure the repository

Use the tools

RHEL / CentOS/Fedora – yum

Ubuntu – apt-get

SLES – yast

Yum can be configured in many ways:

- 1) using iso
- 2) Using internet
- 3) Using ftp sites

Using iso:

- 1) mount iso to local directory

```
mount -o loop RHEL-7.3-20161019.0-Server-x86_64-dvd1.iso /mnt
```

- 2) update /etc/yum.repos.d/iso.repo

```
[iso]
name=iso
baseurl=file:/mnt/
enabled=1
gpgcheck=0
```

- 3) install/remove packages using yum

```
yum install <>
```

```
yum remove <>
```

Shell scripting

The first line of shell script is called She-Bang

she bang line:#!

that instructs the shell to execute this script using the interpreter

Comment:# → that command won't be executed

command line arguments and return status

\$? → gives Return status of a command

\$# - no. of args

\$@ - all args

\$* - all args

\$0 – file name

\$1 – first argument

\$n – nth arg

\$\$ - PID

echo – to print

sh –vx to debug

1.cleanup script

```
# Cleanup
```

```
# Run as root, of course.
```

```
cd /var/log
```

```
cat /dev/null > messages
```

```
cat /dev/null > wtmp  
echo "Log files cleaned up."
```

2. a proper cleanup script:

```
#!/bin/bash  
  
# Proper header for a Bash script.  
  
# Cleanup, version 2  
  
# Run as root, of course.  
  
# Insert code here to print error message and exit if not root.
```

```
LOG_DIR=/var/log  
  
# Variables are better than hard-coded values.  
cd $LOG_DIR
```

```
cat /dev/null > messages  
cat /dev/null > wtmp
```

```
echo "Logs cleaned up."
```

```
exit # The right and proper method of "exiting" from a script.  
  
# A bare "exit" (no parameter) returns the exit status  
#+ of the preceding command.
```

The sha-bang (#!)

[1] at the head of a script tells your system that this file is a set of commands to be fed to the command interpreter indicated. The #! is actually a two-byte

[2] magic number, a special marker that designates a file type, or in this case an executable shell script (type man magic for more details on this fascinating topic).

Immediately following the sha-bang is a path name. This is the path to the program that interprets the commands in the script, whether it be a shell, a programming language, or a utility. This command interpreter then executes the commands in the script, starting at the top (the line following the sha-bang line), and ignoring comments. [3]

```
#!/bin/sh  
#!/bin/bash  
#!/usr/bin/perl  
#!/usr/bin/tcl  
#!/bin/sed -f  
#!/bin/awk -f
```

Each of the above script header lines calls a different command interpreter

More commonly seen in the literature as she-bang or sh-bang. This derives from the concatenation of the tokens sharp (#) and bang (!)

Case statement:

```
case "$variable" in  
abc) echo "\$variable = abc" ;;  
xyz) echo "\$variable = xyz" ;;  
esac
```

Hello world program

```
#!/bin/bash  
STR="Hello World!"  
echo $STR
```

Difference between \$@ and \$*:

both are same when used without quotes with echo

when used with quotes, with IFS – field separator

\$@ prints does not print any thing

\$* prints first character of IFS in between args

```
for i in "$@"
do
echo $i # loop $# times
done
```

```
for i in "$*"
do
echo $i # loop 1 times
done
```

It's safer to use "\$@" instead of \$. When you use multiword strings as arguments to a shell script, it's only "\$@" that interprets each quoted argument as a separate argument.

As the output above suggests, if you use \$*, the shell makes a wrong count of the arguments.

```
#!/bin/bash  
#A scrip to explain command line args  
echo "filename $0";  
echo $1  
echo $2  
echo $3  
echo $4  
echo $5  
echo $6  
echo $7  
echo $8  
echo $9
```

```
[root@reviewb ~]# vi xyz.sh  
#!/bin/bash  
# take 2 numbers as input from user and add them  
#echo "Enter a";
```

```
#read a;  
#echo "Enter b";  
#read b;  
#sum=0;
```

```
a=$1;  
b=$2;  
  
sum=`expr $a + $b`;  
echo $sum;
```

```
[root@r8r3m2kvm tmp]# cat 2.sh  
#!/bin/bash  
  
#A scrip to explain command line args  
echo "filename $0";  
echo $1  
echo $2  
echo $3  
echo $4  
echo $5  
echo $6  
echo $7  
echo $8  
echo $9
```

```
echo $10
echo $11
echo $12
echo $13
echo $14
```

```
[root@r8r3m2kvm tmp]# ./2.sh 1 2 3 4 5 6 7 8 9 10 11 12 13
```

```
filename ./2.sh
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
```

```
[root@r8r3m2kvm tmp]# cat 2.sh
```

```
#!/bin/bash
#A scrip to explain command line args
echo "filename $0";
echo "Bfore shift";
```

```
echo $1  
echo $2  
echo $3  
shift;  
echo "After shift";  
echo $1  
echo $2  
echo $3
```

[root@r8r3m2kvm tmp]# ./2.sh 1 2 3

filename ./2.sh

Bfore shift

```
1  
2  
3
```

After shift

```
2  
3
```

[root@r8r3m2kvm tmp]#

[root@r8r3m2kvm tmp]# cat 3.sh

```
#!/bin/bash  
echo "enter a variable";  
read n;
```

```
echo "You have entred $n";
echo "enter a variable";
read m;
echo "You have entred $m";
l=`expr $n + $m`;
echo "The sum is $l";
```

[root@r8r3m2kvm tmp]# ./3.sh

enter a variable

1

You have entred 1

enter a variable

2

You have entred 2

The sum is 3

[root@r8r3m2kvm tmp]#

Define a variable using

var=10

vehicle=bus

Rules:

don't put spaces a = 10 wrong

a=10 right

variables are case sensitive

variable can be printed using

echo \$ var

Pipes

ls | grep hello

filters

bc-linux calculator

if- take 2numbers and compare

#!/bin/bash

#program to show if

echo "enter a"

read a

echo "enter b"

read b

if [a==b]

then

print "a and b are equal"

else

print "a and b are not equal"

fi

test: to check a condition is true or not

```
if [ test $1 -gt 0 ]
```

```
then
```

```
echo "$1 is +ve"
```

```
fi
```

if conditions:

-s file- empty

-f file exists

-d directory – r read

-w write

-x execute

```
if [ ]
```

```
then
```

```
do this
```

```
fi
```

```
if []
```

then

dothis

else

dothis

fi

if []

then

dothis

elif []

then

do this

else

dothis

fi

if []

then

dothis

else

if []

then

fi

fi

fi

1)

if [-f file]

then

echo "file exists";

fi

if[-f file]

then

echo "fileexists"

else

echo"filedoes notexists";

fi

\$n=100

if [\$n -eq 100]

then

echo "n is 100 ";

elif [\$n -gt 100]

then

echo "nisgtthatn100";

else

```
echo "count is less than 100";  
fi
```

test-

```
#!/usr/bin/bash  
  
echo "Enter file name";  
read filename  
  
if [ -f $filename ]  
then  
    echo "$filename exists";  
else  
    echo "$filename does not exist, creating file";  
    touch $filename;  
    if [ $? -eq 0 ]  
    then  
        echo "file is created successfully";  
    else  
        echo "file is not created";  
    fi  
fi
```

~

When to use double equal to vs when to use -eq

When working on numbers its recommended to use clike syntax

```
#!/usr/bin/bash
```

```
echo "enter a";
```

```
read a;
```

```
echo "enter b";
```

```
read b
```

```
if (( $a >= $b ))
```

```
then
```

```
echo "a is greater"
```

```
else
```

```
echo "b is greater"
```

```
fi
```

```
[root@rscthydnet1 ~]# cat while.sh
```

```
#!/usr/bin/bash
```

```
# while loop
```

```
i=100;
```

```
while (( $i >= 0 ))
```

```
do
```

```
echo $i;
```

```
i=`expr $i - 1`;
```

```
done
```

```
[root@rscthydnet1 ~]#
```

When working on strings use -eq,

Compare 2 decimal numbers

```
#!/usr/bin/bash
```

```
echo "enter a";
```

```
read a;
```

```
echo "enter b";
```

```
read b
```

```
if [ "$a" == "$b" ]
```

```
then
```

```
echo "equal"
```

```
else
```

```
echo "not equal"
```

```
fi
```

for decimal comparisons c-like syntax or -eq does not work

While loop :

the below script prints numbers from 100 to 1

operators

-eq equal to

-gt greater than

-lt lesser than

-le less than or equal to

-ge greater or equal to

```
[root@r8r3m2kvm ~]# cat while.sh
#!/usr/bin/bash
i=100;
while [ $i -ge 0 ]
do
echo $i;
i=`expr $i - 1`;
done
```

the below script will print numbers from 1 to 10

```
#!/usr/bin/bash  
i=0;  
while [ $i -le 10 ]  
do  
echo $i;  
i=`expr $i + 1`;  
done
```

expr is the command to perform arithmetic operations

For loop:

Below script prints 1 2 3 4 5 on the screen

```
#!/usr/bin/bash  
for i in 1 2 3 4 5  
do  
echo $i  
done
```

```
#!/usr/bin/bash  
for ((i=0; i <= 5; i++ ))  
do
```

```
for ((j=0; j <=5; j++ ))  
do  
echo -n $i;  
done  
echo " ";  
done
```

the above scripts o/p is:

```
[root@r8r3m2kvm ~]# ./for2.sh
```

```
000000
```

```
111111
```

```
222222
```

```
333333
```

```
444444
```

```
555555
```

Case statement:

```
[root@r8r3m2kvm ~]# cat case.sh
```

```
#!/usr/bin/bash

echo "Enter number"
read n;
case $n in
"1") echo "you have entered 1";;
"2") echo "you have entered 2";;
"3") echo "you have entered 3";;
"4") echo "you have entered 4";;
*) echo "you have entered something else";;
esac

[root@r8r3m2kvm ~]#
```

Case statement is to select one of the options

```
#!/usr/bin/bash

echo "Enter your vehicle type";
echo "Enter 1 for bus, 2 for car, 3 for bike, 4 for van";
read n;
case $n in
"1") echo "the fee for bus is 100";;
"2") echo "the fee for car is 50";;
"3") echo "the fee for bike is 20";;
"4") echo "the fee for van is 80";;
*) echo "you have entered something else, please choose the right option";;
esac
```

Fibonacci series:

```
#!/bin/bash

echo "How many number of terms to be generated ?"
read n

x=0
y=1
i=2

echo "Fibonacci Series up to $n terms :"
echo "$x"
echo "$y"

while [ $i -lt $n ]
do
    i=`expr $i + 1`
    z=`expr $x + $y`
    echo "$z"
    x=$y
    y=$z
done
```

~

```
#!/bin/bash

echo "How many number of terms to be generated ?"
read n

x=0
y=1
```

```
i=2  
echo "Fibonacci Series up to $n terms :"  
echo "$x"  
echo "$y"  
while [ $i -lt $n ]  
do  
    i=`expr $i + 1`  
    z=`expr $x + $y`  
    echo "$z"  
    x=$y  
    y=$z  
done
```

Functions:

- Saves lot of time.
- Avoids rewriting of same code again and again
- Program is easier to write.
- Program maintains is very easy.

```
[root@r8r3m2kvm ~]# cat fun.sh
#!/usr/bin/bash
sayhello()

{
echo "Hello, I am inside the function";
}

sayhello;
```

```
[root@r8r3m2kvm ~]#
```

```
[root@r8r3m2kvm ~]# cat fun.sh
#!/usr/bin/bash
sayhello()

{
echo "Hello, I am inside the function";
}

sayhello;
```

The above function prints hello when executed.

function has 2 parts
declaration /definition
calling function

The below script prints words,

uses function output in a for loop

```
#!/usr/bin/bash
generate_list ()
{
    echo "one two three"
}
for word in $(generate_list)
do
    echo "$word"
done
```

the below is example for
function with arguments

function within a function

```
root@ctx2p02:~# ./1.sh
```

```
Hello abc
```

```
Hi
```

```
root@ctx2p02:~# cat 1.sh
```

```
#!/bin/bash
```

```
sayhello()
```

```
{
```

```
echo "Hello $1";
```

```
sayhi;
```

```
return 100;
```

```
}
```

```
sayhi()
```

```
{
```

```
echo "Hi";
```

```
}
```

```
sayhello abc;
```

function in command line

```
root@ctx2p02:~# . 1.sh
```

```
Hello abc
```

```
Hi
```

```
root@ctx2p02:~# sayhello
```

Hello

Hi

```
root@ctx2p02:~# sayhi
```

Hi

factorial of a number:

```
root@ctx2p02:~# cat 3.sh
```

```
#!/bin/bash
```

```
factorial()
```

```
{
```

```
if [ "$1" -gt "1" ]
```

```
then
```

```
i=`expr $1 - 1`;
```

```
j=`factorial $i`;
```

```
k=`expr $1 \* $j`;
echo $k;
else
echo 1
fi
}

echo "Enter a number";
read x;
factorial $x;
```

getopts:

to take inputs from command line

example:

```
#!/bin/bash
# Usage: ani -n -a -s -w -d
#
```

```
#  
  
# help_ani() To print help  
  
#  
  
help_ani()  
{  
    echo "Usage: $0 -n -a -s -w -d"  
    echo "Options: These are optional argument"  
    echo " -n name of animal"  
    echo " -a age of animal"  
    echo " -s sex of animal "  
    echo " -w weight of animal"  
    echo " -d demo values (if any of the above options are used "  
    echo " their values are not taken)"  
    exit 1  
}  
  
#  
  
isdef=0  
  
na=Moti  
  
age="2 Months"  
  
sex=Male  
  
weight=3Kg  
  
#  
  
#if no argument  
  
#
```

```
if [ $# -lt 1 ]; then
    help_ani
fi
while getopts n:a:s:w:d opt
do
    case "$opt" in
        n) na="$OPTARG";;
        a) age="$OPTARG";;
        s) sex="$OPTARG";;
        w) weight="$OPTARG";;
        d) isdef=1;;
        \?) help_ani;;
    esac
done
if [ $isdef -eq 0 ]
then
    echo "Animal Name: $na, Age: $age, Sex: $sex, Weight: $weight (user
define mode)"
else
    na="Pluto Dog"
    age=3
    sex=Male
    weight=20kg
    echo "Animal Name: $na, Age: $age, Sex: $sex, Weight: $weight (demo
mode)"

```

fi

Regular expressions:

^ –Caret/Power symbol to match a starting at the beginning of line.

\$ –To match end of the line

* –0 or more occurrence of previous character.

. –To match any single character

? – matches one or no characters (The preceding item is optional and will be matched, at most, once.)

[] –Range of character

[^char] –negate of occurrence of a character set

<word> –Actual word finding

\ –Escape character

```
root@ctx2p02:/var/tmp# ls -l | grep ^d
```

```
drwx----- 3 root root 4096 Oct  7 19:42 systemd-private-  
56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ
```

```
drwx----- 3 root root 4096 Nov 20 11:15 systemd-private-  
df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex
```

```
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# grep '^#' 1.sh
```

```
#!/bin/bash
```

```
root@ctx2p02:/var/tmp# ls -lrt | grep sh$
```

```
-rwxrwxrwx 1 root root 179 Nov 24 06:31 3.sh
```

```
-rwxrwxrwx 1 root root 277 Nov 24 07:07 fib.sh  
-rwxrwxrwx 1 root root 103 Nov 24 07:19 1.sh  
-rwxrwxrwx 1 root root 424 Nov 24 09:21 2.sh  
-rwxrwxrwx 1 root root 171 Nov 24 20:45 7.sh  
-rwxrwxrwx 1 root root 206 Nov 24 21:58 fact.sh  
-rw-r--r-- 1 root root 82 Nov 24 21:59 8.sh  
-rwxrwxrwx 1 root root 936 Nov 25 05:49 10.sh  
-rwxrwxrwx 1 root root 73 Nov 25 08:42 until.sh  
root@ctx2p02:/var/tmp#
```

finding empty lines in a file

```
root@ctx2p02:/var/tmp# grep '^$' *
```

1.sh:

1.sh:

2.sh:

2.sh:

2.sh:

2.sh:

2.sh:

3.sh:

3.sh:

3.sh:

8.sh:

8.sh:

8.sh:
8.sh:
8.sh:
8.sh:
8.sh:
8.sh:
8.sh:
8.sh:
8.sh:
8.sh:
8.sh:
8.sh:
8.sh:
fact.sh:
fact.sh:
fact.sh:
file:
grep: systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ: Is a directory
grep: systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex: Is a directory

grep -E 'word1|word2' filename

root@ctx2p02:/var/tmp# ls -lrt | grep 'syste*d*'
drwx----- 3 root root 4096 Oct 7 19:42 systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ
drwx----- 3 root root 4096 Nov 20 11:15 systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex
root@ctx2p02:/var/tmp# ls -lrt | grep 'un*I*'

```
-rwxrwxrwx 1 root root 73 Nov 25 08:42 until.sh  
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# ls -lrt | grep '[1-9].sh'  
-rwxrwxrwx 1 root root 179 Nov 24 06:31 3.sh  
-rwxrwxrwx 1 root root 103 Nov 24 07:19 1.sh  
-rwxrwxrwx 1 root root 424 Nov 24 09:21 2.sh  
-rwxrwxrwx 1 root root 171 Nov 24 20:45 7.sh  
-rw-r--r-- 1 root root 82 Nov 24 21:59 8.sh  
-rw-r--r-- 1 root root 0 Nov 28 05:21 9.sh  
root@ctx2p02:/var/tmp#
```

[a-z] –Match's any single char between a to z.

[A-Z] –Match's any single char between A to Z.

[0-9] –Match's any single char between 0 to 9.

[a-zA-Z0-9] – Match's any single character either a to z or A to Z or 0 to 9

[!@#\$%^] — Match's any ! or @ or # or \$ or % or ^ character.

```
root@ctx2p02:/var/tmp# ls | grep '[abc]'
```

a
aa
aaa
fact.sh
fib.sh

```
systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ
```

```
systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex
```

```
root@ctx2p02:/var/tmp#
```

```
[root@reviewb abc]# ls -lrt /etc/*release
```

```
-rw-r--r--. 1 root root 52 Sep 27 15:12 /etc/redhat-release
```

```
-rw-r--r--. 1 root root 495 Sep 27 15:12 /etc/os-release
```

```
lrwxrwxrwx. 1 root root 14 Nov 17 15:15 /etc/system-release -> redhat-release
```

```
[root@reviewb abc]# ls -lrt /etc/*release*
```

```
-rw-r--r--. 1 root root 45 Sep 27 15:12 /etc/system-release-cpe
```

```
-rw-r--r--. 1 root root 52 Sep 27 15:12 /etc/redhat-release
```

```
-rw-r--r--. 1 root root 495 Sep 27 15:12 /etc/os-release
```

```
lrwxrwxrwx. 1 root root 14 Nov 17 15:15 /etc/system-release -> redhat-release
```

```
/etc/lsb-release.d:
```

```
total 0
```

```
-rw-r--r--. 1 root root 0 Sep 3 2014 desktop-4.1-noarch
```

```
-rw-r--r--. 1 root root 0 Sep 3 2014 desktop-4.1-amd64
```

```
-rw-r--r--. 1 root root 0 Sep 3 2014 core-4.1-noarch
```

```
-rw-r--r--. 1 root root 0 Sep 3 2014 core-4.1-amd64
```

```
[root@reviewb abc]#
```

```
root@ctx2p02:/var/tmp# ls | grep '[^abc]'
```

10.sh
1.sh
2.sh
3.sh
7.sh
8.sh
9.sh
fact.sh
fib.sh
file
file90
systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ
systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex
until.sh
root@ctx2p02:/var/tmp#

root@ctx2p02:/var/tmp# grep '\[' *

10.sh:if [\$# -lt 1]; then
10.sh:if [\$isdef -eq 0]
2.sh:if [\$? -eq 0]
3.sh:if ["\$1" -gt "1"]
7.sh:while [\$k -lt \$n]
fact.sh: if ["\$1" -gt "1"]; then
fib.sh: while [\$i -lt \$n]

```
grep: systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ: Is a directory
```

```
grep: systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex: Is a directory
```

```
until.sh:until [ $a -lt 10 ]
```

```
root@ctx2p02:/var/tmp#
```

{n} – n occurrence of previous character

{n,m} – n to m times occurrence of previous character

{m, } – m or more occurrence of previous character.

```
root@ctx2p02:/var/tmp# ls -l | grep -E 'a{2}'
```

```
-rw-r--r-- 1 root root 0 Nov 28 05:28 aa
```

```
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# ls -l | grep -E 'a{1,3}'
```

```
total 56
```

```
-rw-r--r-- 1 root root 8 Nov 28 05:24 a
```

```
-rw-r--r-- 1 root root 0 Nov 28 05:28 aa
```

```
-rw-r--r-- 1 root root 0 Nov 28 05:29 aaa
```

```
-rwxrwxrwx 1 root root 206 Nov 24 21:58 fact.sh  
drwx----- 3 root root 4096 Oct  7 19:42 systemd-private-  
56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ  
drwx----- 3 root root 4096 Nov 20 11:15 systemd-private-  
df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex  
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# ls -l | grep -E 'u+'  
-rwxrwxrwx 1 root root 73 Nov 25 08:42 until.sh  
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# ls -l | grep -E 'a|b'  
total 56  
-rw-r--r-- 1 root root 8 Nov 28 05:24 a  
-rw-r--r-- 1 root root 0 Nov 28 05:28 aa  
-rw-r--r-- 1 root root 0 Nov 28 05:29 aaa  
-rwxrwxrwx 1 root root 206 Nov 24 21:58 fact.sh  
-rwxrwxrwx 1 root root 277 Nov 24 07:07 fib.sh  
drwx----- 3 root root 4096 Oct  7 19:42 systemd-private-  
56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ  
drwx----- 3 root root 4096 Nov 20 11:15 systemd-private-  
df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex  
root@ctx2p02:/var/tmp#
```

AWK

```
root@ctx2p02:/var/tmp# ls -lrt | awk '{print $9}'
```

3.sh

fib.sh

1.sh

2.sh

7.sh

fact.sh

8.sh

10.sh

until.sh

file90

9.sh

a

file

aa

aaa

```
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# ls -lrt | awk '{print $8 "\t" $9}'
```

```
19:42 systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ
```

```
11:15 systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex
```

```
06:31 3.sh
07:07 fib.sh
07:19 1.sh
09:21 2.sh
20:45 7.sh
21:58 fact.sh
21:59 8.sh
05:49 10.sh
08:42 until.sh
09:14 file90
05:21 9.sh
05:24 a
05:27 file
05:28 aa
05:29 aaa
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# ls -lrt | awk '{print $8 "," $9}'
```

```
,
```

```
19:42,systemd-private-56136dd90e4f4901bdc3fa6db5ac108b-systemd-
timesyncd.service-dxHQKZ
```

```
11:15,systemd-private-df9fa18523f14518a8244c0bacba7692-systemd-
timesyncd.service-ENs3ex
```

```
06:31,3.sh
```

```
07:07,fib.sh
```

```
07:19,1.sh
```

09:21,2.sh
20:45,7.sh
21:58,fact.sh
21:59,8.sh
05:49,10.sh
08:42,until.sh
09:14,file90
05:21,9.sh
05:24,a
05:27,file
05:28,aa
05:29,aaa
root@ctx2p02:/var/tmp#

root@ctx2p02:/var/tmp# ls -lrt| awk '/a/ {print \$0}'
total 56
drwx----- 3 root root 4096 Oct 7 19:42 systemd-private-
56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ
drwx----- 3 root root 4096 Nov 20 11:15 systemd-private-
df9fa18523f14518a8244c0bacba7692-systemd-timesyncd.service-ENs3ex
-rwxrwxrwx 1 root root 206 Nov 24 21:58 fact.sh
-rw-r--r-- 1 root root 8 Nov 28 05:24 a
-rw-r--r-- 1 root root 0 Nov 28 05:28 aa
-rw-r--r-- 1 root root 0 Nov 28 05:29 aaa
-rw-r--r-- 1 root root 0 Nov 28 05:47 ba
root@ctx2p02:/var/tmp#

```
root@ctx2p02:/var/tmp# ls -lrt | awk '/a/{++cnt} END {print "Count =", cnt}'
```

```
Count = 8
```

```
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# ls -lrt | awk 'FNR == 2 {print}'
```

```
drwx----- 3 root root 4096 Oct 7 19:42 systemd-private-  
56136dd90e4f4901bdc3fa6db5ac108b-systemd-timesyncd.service-dxHQKZ
```

```
root@ctx2p02:/var/tmp# ls -lrt | grep ^- | awk 'FNR == 5 {print $9}'
```

```
7.sh
```

```
root@ctx2p02:/var/tmp#root@ctx2p02:/var/tmp#
```

to print the j'th field of the i'th line

```
awk -v i=5 -v j=3 'FNR == i {print $j}'
```

```
root@ctx2p02:/var/tmp# ls -lrt | grep ^- | awk -v i=5 -v j=9 'FNR == i{print $j}'
```

```
7.sh
```

```
root@ctx2p02:/var/tmp#
```

Syntax:

```
BEGIN { .... initialization awk commands ...}
```

```
{ .... awk commands for each line of the file...} END
```

```
{ .... finalization awk commands ...}
```

```
root@ctx2p02:/var/tmp# ls -l | awk 'BEGIN {sum=0} {sum=sum+$5} END {print sum}'
```

```
10693
```

```
root@ctx2p02:/var/tmp#
```

SED

By default, the sed command replaces the first occurrence of the pattern in each line and it won't replace the second, third...occurrence in the line.

`sed 's/unix/linux/' file.txt` → replaces word unix with linux

`sed 's/unix/linux/2'` file.txt → replaces 2nd occurrence of word

to replace all the occurrences of file

`sed 's/unix/linux/g'` file.txt

Replacing from nth occurrence to all occurrences in a line.

`sed 's/unix/linux/3g'` file.txt - The following sed command replaces the third, fourth, fifth... "unix" word with "linux" word in a line.

There might be cases where you want to search for the pattern and replace that pattern by adding some extra characters to it. In such cases & comes in handy. The & represents the matched string.

`sed 's/unix/{&}'` file.txt

root@ctx2p02:/var/tmp# cat file.txt | sed 's/unix/{&}/g '

{unix} {unix}

linux

{unix}

root@ctx2p02:/var/tmp#

The /p print flag prints the replaced line twice on the terminal. If a line does not have the search pattern and is not replaced, then the /p prints that line only once.

```
root@ctx2p02:/var/tmp# sed 's/unix/linux/p' file.txt
```

linux unix

linux unix

linux

linux

linux

```
root@ctx2p02:/var/tmp# cat file.txt
```

unix unix

linux

unix

Replacing string on a specific line number.

You can restrict the sed command to replace the string on a specific line number. An example is

```
root@ctx2p02:/var/tmp# sed '3 s/unix/linux/' file.txt
```

unix unix

linux

linux

```
root@ctx2p02:/var/tmp# sed '1,3 s/unix/linux/' file.txt
```

linux unix

linux

linux

```
root@ctx2p02:/var/tmp#
```

```
root@ctx2p02:/var/tmp# sed '2,$ s/unix/linux/' file.txt
```

unix unix

linux

linux

Here \$ indicates the last line in the file. So the sed command replaces the text from second line to last line in the file.

```
sed '2 d' file.txt
```

deletes the 2nd line from file

```
sed '5,$ d' file.txt
```

Sed as grep command

sed command can be used as grep

```
>grep 'unix' file.txt  
>sed -n '/unix/ p' file.txt
```

Here the sed command looks for the pattern "unix" in each line of a file and prints those lines that has the pattern.

You can also make the sed command to work as grep -v, just by using the reversing the sed with NOT (!).

```
>grep -v 'unix' file.txt  
>sed -n '/unix/ !p' file.txt
```

after:

```
root@ctx2p02:/var/tmp# sed '/unix/ a "Add a new line"' file.txt
```

unix unix

"Add a new line"

linux

unix

"Add a new line"

before :

```
root@ctx2p02:/var/tmp# sed '/unix/ i "Add a new line"' file.txt  
"Add a new line"
```

unix unix

linux

"Add a new line"

unix

lower case to upper case

sed 'y/ul/UL/' file.txt

it can be used to replace an entire line with a new line. The "c" command to sed tells it to change the line.

```
>sed '/unix/ c "Change line"' file.txt
```

```
root@ctx2p02:/var/tmp# sed '/unix/ c "Change line"' file.txt
```

```
"Change line"
```

```
linux
```

```
"Change line"
```

to make the change permanent

```
root@ctx2p02:/var/tmp# sed -i '/unix/ c "Change line"' file.txt
```

```
root@ctx2p02:/var/tmp# cat file.txt
```

```
"Change line"
```

```
linux
```

```
"Change line"
```

```
root@ctx2p02:/var/tmp#
```

Maven

Maven is a project management tool/ build automation tool

Maven follows convention than configuration

Maven asks to follow convention

Pom.xml

Define

mvn clean install – maven compile run tests and create jar file

we can add dependencies plugins apache-common etc

maven gets them from internet.

Maven is very tiny after installation

It uses plugin framework

It downloads plugins on the fly and uses them

To run junit test, for first time it downloads sure shark plugin

Maven will compile and dump into repos

Dependencies can be fetched from repos

Convetion

Plugins

Pom.xml – project object model

Repositories

Directory/folder structure in java:

My project

src/main/java – main source code

src/main/resource – property file or config files

src/test/java – junit test

targets – copies all classes/jar files to targets

Maven automates build test deployment with very minimal convention.

If we follow the directory structure, maven takes care of test of things

Maven uses pom.xml - project object model

Using pom Compiles and builds a jar file out of project

Command to use – mvn

Maven downloads plugins on the fly and uses them

Repository: Maven will compile and create jar files into a repository

mvn clean

Or mvn install

Maven executes the life cycle phases

maven install - processing resources

compile

execute test

package project into jar/war files

install it

goals in maven are

Phases	Goal
Process-resources	resources:resource
Compile	compile:compile

Process test resources

Execute tests

Package jar:jar

Install install:install

Install is p

Maven jar:jar

First jar is Package life cycle phase and second jar is actual goal

Phase:goal

Maven makes life of java developer easy

Maven is project management tool java application tool

Developer might be using

Sprint mvc

Struts

Jpa – to connect to data base

War files

Deploy to tomcat websphere

Ant – we need to write config file manually

Maven archifacts – used to develop web applications

Power of Maven

POM

Build life cycle

- 1) App.java → App.class
- 2) App.Test.java ➔ App.Test.java.class
- 3) Run the unit test
- 4) Combine all jar files

Src/main

Src/test

Example POM:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.companyname.project-group</groupId>
  <artifactId>project</artifactId>
  <version>1.0</version>
</project>
```

All POM files require the project element and three mandatory fields: groupId, artifactId, version.

Projects notation in repository is groupId:artifactId:version.

Root element of POM.xml is project and it has three major sub-nodes

groupId This is an Id of project's group. This is generally unique amongst an organization or a project. For example, a banking group com.company.bank has all bank related projects.

artifactId This is an Id of the project. This is generally name of the project. For example, consumer-banking. Along with the groupId, the artifactId defines the artifact's location within the repository.

POM contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal. Some of the configuration that can be specified in the POM are following:

project dependencies

plugins

goals

build profiles

project version

developers

mailing list

Before creating a POM, we should first decide the project group (groupId), its name(artifactId) and its version as these attributes help in uniquely identifying the project in repository.

version This is the version of the project. Along with the groupId, It is used within an artifact's repository to separate versions from each other. For example:

com.company.bank:consumer-banking:1.0

com.company.bank:consumer-banking:1.1

Goals in maven

clean

compile

install

package

jar

deploy

Build life cycle:

Prepare resources

Compile

Package

Install

The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked. For example, consider the command below. The clean and package arguments are build phases while the dependency:copy-dependencies is a goal.

```
mvn clean dependency:copy-dependencies package
```

Here the clean phase will be executed first, and then the dependency:copy-dependencies goal will be executed, and finally package phase will be executed

Clean phase example:

```
[root@reviewb consumerBanking]# ls -lrt target/classes/com/companyname/bank/App.class
-rw-r--r--. 1 root root 555 Jan 4 05:23 target/classes/com/companyname/bank/App.class

[root@reviewb consumerBanking]# mvn clean

[INFO] Scanning for projects...

[INFO]

[INFO] -----
[INFO] Building consumerBanking 1.0-SNAPSHOT

[INFO] -----
[INFO]

[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ consumerBanking ---

[INFO] Deleting /root/maven/consumerBanking/target

[INFO] -----
[INFO] BUILD SUCCESS

[INFO] -----
[INFO] Total time: 0.647 s

[INFO] Finished at: 2017-01-04T05:37:07+05:30

[INFO] Final Memory: 6M/56M

[INFO] -----
[root@reviewb consumerBanking]# ls -lrt target/classes/com/companyname/bank/App.class
```

ls: cannot access target/classes/com/companyname/bank/App.class: No such file or directory

Sample pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-
v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.companyname.bank</groupId>
  <artifactId>consumerBanking</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>consumerBanking</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
    
```

```
<scope>test</scope>
</dependency>
</dependencies>
</project>
```

Command to execute:

```
mvn clean package
```

```
cd /root/maven/consumerBanking/target/classes
```

```
[root@reviewb classes]# java -classpath . com.companyname.bank.App
```

```
Hello World!
```

```
[root@reviewb classes]#
```

```
[root@reviewb classes]# cd /root/maven/consumerBanking/
```

```
[root@reviewb consumerBanking]# maven clean package
```

```
-bash: maven: command not found
```

```
[root@reviewb consumerBanking]# mvn clean package
```

```
[INFO] Scanning for projects...
```

```
[INFO]
```

```
[INFO] -----
```

```
[INFO] Building consumerBanking 1.0-SNAPSHOT
```

```
[INFO] -----  
[INFO]  
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ consumerBanking ---  
[INFO] Deleting /root/maven/consumerBanking/target  
[INFO]  
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ consumerBanking ---  
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is  
platform dependent!  
[INFO] skip non existing resourceDirectory /root/maven/consumerBanking/src/main/resources  
[INFO]  
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ consumerBanking ---  
[INFO] Changes detected - recompiling the module!  
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform  
dependent!  
[INFO] Compiling 1 source file to /root/maven/consumerBanking/target/classes  
[INFO]  
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @  
consumerBanking ---  
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is  
platform dependent!  
[INFO] skip non existing resourceDirectory /root/maven/consumerBanking/src/test/resources  
[INFO]  
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ consumerBanking ---  
[INFO] Changes detected - recompiling the module!  
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform  
dependent!  
[INFO] Compiling 1 source file to /root/maven/consumerBanking/target/test-classes
```

[INFO]

[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ consumerBanking ---

[INFO] Surefire report directory: /root/maven/consumerBanking/target/surefire-reports

TESTS

Running com.companyname.bank.AppTest

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.028 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

[INFO]

[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ consumerBanking ---

[INFO] Building jar: /root/maven/consumerBanking/target/consumerBanking-1.0-SNAPSHOT.jar

[INFO] -----

[INFO] BUILD SUCCESS

[INFO] -----

[INFO] Total time: 5.574 s

[INFO] Finished at: 2017-01-04T05:23:46+05:30

[INFO] Final Memory: 15M/56M

[INFO] -----

[root@reviewb consumerBanking]#

Difference between ant and Maven

Ant	Maven
Ant doesn't have formal conventions , so we need to provide information of the project structure in build.xml file.	Maven has a convention to place source code, compiled code etc. So we don't need to provide information about the project structure in pom.xml file.
Ant is procedural , you need to provide information about what to do and when to do through code. You need to provide order.	Maven is declarative , everything you define in the pom.xml file.
There is no life cycle in Ant.	There is life cycle in Maven.
It is a tool box .	It is a framework .
It is mainly a build tool .	It is mainly a project management tool .
The ant scripts are not reusable .	The maven plugins are reusable .
It is less preferred than Maven.	It is more preferred than Ant.

Maven life cycle:

validate - validate the project is correct and all necessary information is available

compile - compile the source code of the project

test - test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed

package - take the compiled code and package it in its distributable format, such as a JAR.

verify - run any checks on results of integration tests to ensure quality criteria are met

install - install the package into the local repository, for use as a dependency in other projects locally

deploy - done in the build environment, copies the final package to the remote repository for sharing with other developers and projects.

Clean phase pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.companyname.projectgroup</groupId>
  <artifactId>project</artifactId>
  <version>1.0</version>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-antrun-plugin</artifactId>
        <version>1.1</version>
    
```

```
<executions>

    <execution>
        <id>id.pre-clean</id>
        <phase>pre-clean</phase>
        <goals>
            <goal>run</goal>
        </goals>
        <configuration>
            <tasks>
                <echo>pre-clean phase</echo>
            </tasks>
        </configuration>
    </execution>

    <execution>
        <id>id.clean</id>
        <phase>clean</phase>
        <goals>
            <goal>run</goal>
        </goals>
        <configuration>
            <tasks>
                <echo>clean phase</echo>
            </tasks>
        </configuration>
    </execution>

</executions>
```

```
<execution>
  <id>id.post-clean</id>
  <phase>post-clean</phase>
  <goals>
    <goal>run</goal>
  </goals>
  <configuration>
    <tasks>
      <echo>post-clean phase</echo>
    </tasks>
  </configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
</project>
```

```
[root@reviewb consumerBanking]# cat superpom.xml | grep -i phase
<phase>clean</phase>
<phase>process-test-resources</phase>
<phase>process-resources</phase>
<phase>package</phase>
<phase>compile</phase>
<phase>test-compile</phase>
```

```
<phase>test</phase>  
<phase>install</phase>  
<phase>deploy</phase>  
<phase>site</phase>  
<phase>site-deploy</phase>
```

```
[root@reviewb consumerBanking]# cat superpom.xml | grep -i goal
```

```
<goals>  
  <goal>clean</goal>  
</goals>  
  
<goals>  
  <goal>testResources</goal>  
</goals>  
  
<goals>  
  <goal>resources</goal>  
</goals>  
  
<goals>  
  <goal>jar</goal>  
</goals>  
  
<goals>  
  <goal>compile</goal>  
</goals>  
  
<goals>  
  <goal>testCompile</goal>  
</goals>
```

```
<goals>
  <goal>test</goal>
</goals>
<goals>
  <goal>install</goal>
</goals>
<goals>
  <goal>deploy</goal>
</goals>
<goals>
  <goal>site</goal>
</goals>
<goals>
  <goal>deploy</goal>
</goals>
```

```
[root@reviewb consumerBanking]# mvn clean compile
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building consumerBanking 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ consumerBanking ---
```

[INFO] Deleting /root/maven/consumerBanking/target

[INFO]

[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ consumerBanking ---

[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!

[INFO] skip non existing resourceDirectory /root/maven/consumerBanking/src/main/resources

[INFO]

[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ consumerBanking ---

[INFO] Changes detected - recompiling the module!

[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!

[INFO] Compiling 1 source file to /root/maven/consumerBanking/target/classes

[INFO] -----

[INFO] BUILD SUCCESS

[INFO] -----

[INFO] Total time: 3.239 s

[INFO] Finished at: 2017-01-04T05:42:37+05:30

[INFO] Final Memory: 12M/56M

[INFO] -----

[root@reviewb consumerBanking]#

How to generate pom.xml :

```
archetype:generate -DgroupId=com.companyname.bank -DartifactId=consumerBanking -  
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

it generates a pom.xml

```
[root@reviewb 2]# mvn archetype:generate -DgroupId=com.companyname.bank -  
DartifactId=consumerBanking -DarchetypeArtifactId=maven-archetype-quickstart -  
DinteractiveMode=false
```

```
[INFO] Scanning for projects...
```

```
[INFO]
```

```
[INFO] -----
```

```
[INFO] Building Maven Stub Project (No POM) 1
```

```
[INFO] -----
```

```
[INFO]
```

```
[INFO] >>> maven-archetype-plugin:2.4:generate (default-cli) > generate-sources @ standalone-pom  
>>>
```

```
[INFO]
```

```
[INFO] <<< maven-archetype-plugin:2.4:generate (default-cli) < generate-sources @ standalone-pom  
<<<
```

```
[INFO]
```

```
[INFO] --- maven-archetype-plugin:2.4:generate (default-cli) @ standalone-pom ---
```

```
[INFO] Generating project in Batch mode
```

```
[INFO] -----
```

```
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-  
quickstart:1.0
```

```
[INFO] -----
```

```
[INFO] Parameter: basedir, Value: /2
```

```
[INFO] Parameter: package, Value: com.companyname.bank
```

```
[INFO] Parameter: groupId, Value: com.companyname.bank  
[INFO] Parameter: artifactId, Value: consumerBanking  
[INFO] Parameter: packageName, Value: com.companyname.bank  
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
```

```
[INFO] project created from Old (1.x) Archetype in dir: /2/consumerBanking
```

```
[INFO] -----
```

```
[INFO] BUILD SUCCESS
```

```
[INFO] -----
```

```
[INFO] Total time: 17.184 s
```

```
[INFO] Finished at: 2017-01-04T05:56:02+05:30
```

```
[INFO] Final Memory: 14M/56M
```

```
[INFO] -----
```

```
[root@reviewb 2]# ls -lrt
```

```
total 0
```

```
drwxr-xr-x. 3 root root 32 Jan 4 05:56 consumerBanking
```

```
[root@reviewb 2]# cd consumerBanking/
```

```
[root@reviewb consumerBanking]# ls
```

```
pom.xml src
```

```
[root@reviewb consumerBanking]# vi pom.xml
```

```
[root@reviewb consumerBanking]#
```

```
[root@reviewb consumerBanking]# cat pom.xml
```

```
<project xmlns="http://maven.apache.org/POM/4.0.0"  
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-  
        v4_0_0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>

<groupId>com.companyname.bank</groupId>

<artifactId>consumerBanking</artifactId>

<packaging>jar</packaging>

<version>1.0-SNAPSHOT</version>

<name>consumerBanking</name>

<url>http://maven.apache.org</url>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>
```

Phase vs goal:

phase	Goal
process-resources	resources:resources
compile	compile:compile
process test resources	
execute test resources	
package	jar:jar
install	install:install

mvn install --> executes life cycle phase

goal is a task

single goal is bound to life cycle phase

if we select package as the phase using mvn package, all the earlier phases till package (process-resources,compile,process test resources, execute test resources,package) will be executed

or we can execute maven using goal also

mvn compile:compile syntax: mvn plugin:task

all the goals till compile will be executed.

jar is plugin jar is the task

```
[root@reviewb consumerBanking]# mvn jar:jar
```

```
[INFO] Scanning for projects...
```

```
[INFO]
[INFO] -----
[INFO] Building consumerBanking 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-cli) @ consumerBanking ---
[WARNING] JAR will be empty - no content was marked for inclusion!
[INFO] Building jar: /2/consumerBanking/target/consumerBanking-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.503 s
[INFO] Finished at: 2017-01-04T06:03:16+05:30
[INFO] Final Memory: 7M/56M
[INFO] -----
[root@reviewb consumerBanking]#
```

Repository manager:

How to improve the maven build performance:

Mvn install or build → maven downloades third party binaries from repository and dump on the machine somewhere

.M2 directory

Maven takes time to download

Apache access tool

Spring framework

Hibernate

All jars if they are declared as dependencies they will be pulled from public repository.

To overcome this issue: we have maven local repos

Acts as proxy for public repos

Acts as internal project repos

Settings.xml → maven config file, instead of pulling from public repos, pulls from local repos

Nexus is a easy maven repos from sonatype

Nexus is a repository

We can configure like: use only licensed apache jar files

Deploy using maven:

Define repository as below

[...]

```
<distributionManagement>
  <repository>
    <id>internal.repo</id>
    <name>MyCo Internal Repository</name>
```

```
<url>Host to Company Repository</url>  
</repository>  
</distributionManagement>  
[...]
```

Define server details in settings.xml

[...]

```
<server>  
<id>internal.repo</id>  
<username>maven</username>  
<password>foobar</password>  
</server>
```

[...]

Nagios

Nagios is a powerful monitoring system that provides instant awareness of organization's mission critical IT infrastructure

Nagios allows to detect and repair problems and mitigate future issues before they affect end users and customers

Nagios is very old tool, its very stable

If data base is down, its like business loss for organization

Other tools: Zabbix

Pre-req:

Always have a separate host as monitoring server/node and do not install it with other critical services

Nagios runs on multiple platforms like Linux, Solaris and Windows and has very minimum system requirements by today's standards.

Nagios as application is very small footprint, does not require too many resources.

Plan for infrastructure upgrades before outdated systems cause failures

Respond to issues at the first sign of a problem

Automatically fix problems when they are disabled.

Ensure IT infrastructure outages have a minimal effect on org

Monitor your business process.

High availability – 99.999%

Nagios Architecture:

Nagios plugins can be written in shell script/perl

Nagios monitors nodes/agents using plugins/perl plugins

Install Nagios:

Download the package and install it

Dependencies:

PHP – front end Nagios is developed in PHP

HTTP

start httpd and enable it

htpasswd /etc/nagios/passwd nagiosadmin

⇒ It will store the username and password in /etc/nagios/passwd in encrypted manner

systemctl start Nagios

systemctl enable Nagios

systemctl restart Nagios

visit the link:

<http://<ip>/nagios>

By default, Nagios installs some plugins like disk space

Check_swap

Check_users

Check_ping

Check_ssh

Check_load

Check_http

Check_disk

Check_procs

Technical overview

Hosts

Services

Monitoring features

Hosts unreachable

Monitors local host

Ping is OK

Plugins in Nagios:

```
[root@ctx2p10 plugins]# pwd
```

Plugins path:

```
/usr/lib64/nagios/plugins
```

```
[root@ctx2p10 plugins]# ./check_ping
```

```
check_ping: Could not parse arguments
```

Usage:

```
check_ping -H <host_address> -w <wrta>,<wpl>% -c <crta>,<cpl>%
```

```
[-p packets] [-t timeout] [-4|-6]
```

```
[root@ctx2p10 plugins]#
```

```
[root@ctx2p10 plugins]# ls -lrt
```

```
total 664
```

```
-rwxr-xr-x 1 root root 34595 Mar 20 2013 check_nrpe
```

```
drwxr-xr-x 2 root root 4096 Aug 31 2013 eventhandlers
```

```
-rwxr-xr-x 1 root root 2791 Sep 11 2015 utils.sh
```

```
-rwxr-xr-x 1 root root 46352 Sep 11 2015 urlize
```

```
-rwxr-xr-x 1 root root 49376 Sep 11 2015 negate
```

```
-rwxr-xr-x 1 root root 48168 Sep 11 2015 check_users
```

```
-rwxr-xr-x 1 root root 66352 Sep 11 2015 check_procs
```

```
-rwxr-xr-x 1 root root 68184 Sep 11 2015 check_ping
```

```
-rwxr-xr-x 1 root root 54424 Sep 11 2015 check_load
```

```
-rwxr-xr-x 1 root root 140720 Sep 11 2015 check_http
```

```
-rwxr-xr-x 1 root root 70104 Sep 11 2015 check_disk
```

```
-rwxr-xr-x 1 root root 66632 Sep 11 2015 check_by_ssh
```

```
[root@ctx2p10 plugins]#
```

Custom plugins for Nagios:

We can write our own plugins in shell,perl or python

Plugins may be either in C or interpreted scripts

Plugins are executable files run by Nagios to determine the status of host

All plugins must be written following some guide lines

NRPE (Nagios Remote Plugin Executor)

How to add a node to Nagios server:

1. Create a file under the directory

```
[root@ctx2p10 objects]# pwd  
/etc/nagios/objects  
  
[root@ctx2p10 objects]# ls -lrt  
total 60  
  
-rwxrwxrwx 1 root root 4019 Aug 31 2013 windows.cfg  
-rwxrwxrwx 1 root root 3208 Aug 31 2013 timeperiods.cfg  
-rwxrwxrwx 1 root root 11158 Aug 31 2013 templates.cfg  
-rwxrwxrwx 1 root root 3293 Aug 31 2013 switch.cfg  
-rwxrwxrwx 1 root root 3124 Aug 31 2013 printer.cfg
```

```
-rwxrwxrwx 1 root root 7704 Aug 31 2013 commands.cfg  
-rwxrwxrwx 1 root root 2166 Oct 17 18:38 contacts.cfg_old  
-rwxrwxrwx 1 root root 2169 Oct 17 18:40 contacts.cfg  
-rwxrwxrwx 1 root root 5504 Oct 18 07:52 localhost.cfg  
-rwxrwxrwx 1 root root 1375 Oct 18 18:10 ctx3p10.cfg  
-rwxr-xr-x 1 root root 1375 Dec 20 06:10 ctx3p11.cfg  
[root@ctx2p10 objects]#
```

Ctx3p11.cfg

```
# Adding contents to linux-hosts.cfg file from (localhost.cfg & templates.cfg) file #  
  
# nano /usr/local/nagios/etc/linux-hosts.cfg  
  
#####  
#  
#  
  
# HOST DEFINITION  
  
#  
#####  
#  
  
# Define a host for the remote machine  
  
  
define host{  
    use      linux-server ; Name of host template to use  
                ; This host definition will inherit all variables that are defined  
                ; in (or inherited by) the linux-server host template definition.
```

```

host_name      ctx3p11.in.company.com

alias          RHEL 7.0

address        9.182.76.65

}

define service{

    use           local-service ; Name of service template to use

    host_name    ctx3p11.in.company.com

    service_description PING

    check_command   check_ping!100.0,20%!500.0,60%

}

#define service{

#  use           generic-service

#  host_name    ctx3p11.in.company.com

#  service_description # Memory Used

#  check_command  check_nrpe!check_mem

# }


```

~

2. Open the file /etc/nagios/nagios.cfg and add a new line

```

# Definitions for monitoring the local (Linux) host

cfg_file=/etc/nagios/objects/localhost.cfg

cfg_file=/etc/nagios/objects/ctx3p10.cfg

cfg_file=/etc/nagios/objects/ctx3p11.cfg

```

3. Restart the Nagios server

```
[root@ctx2p10 objects]# service nagios restart
```

Running configuration check...done.

Stopping nagios: done.

Starting nagios: done.

```
[root@ctx2p10 objects]#
```

New host will appear in the hosts section

4. Validate the configuration using below command

```
[root@ctx2p10 plugins]# nagios -v /etc/nagios/nagios.cfg
```

Nagios Core 3.5.1

Copyright (c) 2009-2011 Nagios Core Development Team and Community Contributors

Copyright (c) 1999-2009 Ethan Galstad

Last Modified: 08-30-2013

License: GPL

Website: <http://www.nagios.org>

Reading configuration data...

Read main config file okay...

Processing object config file '/etc/nagios/objects/commands.cfg'...

Processing object config file '/etc/nagios/objects/contacts.cfg'...

Processing object config file '/etc/nagios/objects/timeperiods.cfg'...

Processing object config file '/etc/nagios/objects/templates.cfg'...

Processing object config file '/etc/nagios/objects/localhost.cfg'...

Processing object config file '/etc/nagios/objects/ctx3p10.cfg'...

Processing object config file '/etc/nagios/objects/ctx3p11.cfg'...

Processing object config directory '/etc/nagios/conf.d'...

Read object config files okay...

Running pre-flight check on configuration data...

Checking services...

Checked 11 services.

Checking hosts...

Checked 3 hosts.

Checking host groups...

Checked 1 host groups.

Checking service groups...

Checked 0 service groups.

Checking contacts...

Checked 1 contacts.

Checking contact groups...

Checked 1 contact groups.

Checking service escalations...

Checked 0 service escalations.

Checking service dependencies...

Checked 0 service dependencies.

Checking host escalations...

Checked 0 host escalations.

Checking host dependencies...

Checked 0 host dependencies.

Checking commands...

Checked 24 commands.

Checking time periods...

Checked 5 time periods.

Checking for circular paths between hosts...

Checking for circular host and service dependencies...

Checking global event handlers...

Checking obsessive compulsive processor commands...

Checking misc settings...

Total Warnings: 0

Total Errors: 0

Things look okay - No serious problems were detected during the pre-flight check

[root@ctx2p10 plugins]#

Nagios:

host groups: hosts can be combined to form a host group

Service group:

Services can be combined to form a service group

Customized plugins:

We can create our own plugins based on the requirement

Return code must be handled

0 – OK

1 – warning

2- critical

Based on the return code Nagios will change the color of the state.

Ping http ssh – these plugins will be executed on the current machine

To execute commands on the remote machine we need to use nrpe plugins

NRPE:

Install nrpe and nrpe-plugins on all the remote nodes

And modify

/etc/Nagios/nrpe.cfg

To point to Nagios server

And allow the ip

Customized plugins:

```
define command{
    command_name check_mem
    command_line /usr/local/nagios/libexec/check_mem.sh 80 90
}
```

```
define command{
    command_name check_rmc
    command_line /usr/local/nagios/libexec/check_rmc
}
```

Flow:

/etc/Nagios/Nagios.cfg

```
# Definitions for monitoring the local (Linux) host
cfg_file=/etc/nagios/objects/localhost.cfg
cfg_file=/etc/nagios/objects/ctx3p10.cfg
cfg_file=/etc/nagios/objects/ctx3p11.cfg
cfg_file=/etc/nagios/objects/ctx3p12.cfg
cfg_file=/etc/nagios/objects/ctx2p05.cfg
```

```
[root@ctx2p10 objects]# pwd  
/etc/nagios/objects  
  
[root@ctx2p10 objects]# ls -lrt  
total 76  
  
-rwxrwxrwx 1 root root 4019 Aug 31 2013 windows.cfg  
  
-rwxrwxrwx 1 root root 3208 Aug 31 2013 timeperiods.cfg  
  
-rwxrwxrwx 1 root root 11158 Aug 31 2013 templates.cfg  
  
-rwxrwxrwx 1 root root 3293 Aug 31 2013 switch.cfg  
  
-rwxrwxrwx 1 root root 3124 Aug 31 2013 printer.cfg  
  
-rwxrwxrwx 1 root root 2166 Oct 17 18:38 contacts.cfg_old  
  
-rwxrwxrwx 1 root root 2169 Oct 17 18:40 contacts.cfg  
  
-rwxrwxrwx 1 root root 1375 Oct 18 18:10 ctx3p10.cfg  
  
-rwxr-xr-x 1 root root 847 Dec 20 09:16 ctx2p05.cfg  
  
-rwxr-xr-x 1 root root 1526 Dec 20 15:32 ctx3p12.cfg  
  
-rwxrwxrwx 1 root root 5780 Dec 20 17:20 localhost.cfg  
  
-rwxr-xr-x 1 root root 1856 Dec 21 05:46 ctx3p11.cfg  
  
-rwxrwxrwx 1 root root 731 Dec 21 05:46 nrpe.cfg  
  
-rwxrwxrwx 1 root root 8261 Dec 21 09:13 commands.cfg  
  
[root@ctx2p10 objects]#
```

/etc/nagios/objects/Commands.cfg

```
define command{
    command_name check_mem
    command_line /usr/local/nagios/libexec/check_mem.sh 80 90
}
```

```
define command{
    command_name check_rmc
    command_line /usr/local/nagios/libexec/check_rmc
}
```

Check_mem.sh:

```
[root@ctx2p10 objects]# cat /usr/local/nagios/libexec/check_mem.sh 80 90
#!/bin/ksh

# Determine memory usage percentage on Linux servers.

# Original write for RHEL3 for PC1 Project - jlightner 05-Jul-2005

#
# Modified for RHEL5 on mailservers.

# -Some of the escapes previously required for RHEL3's ksh not needed on RHEL5.

# -Changed comparisons to allow for decimal rather than integer values.

# jlightner 23-Jan-2009

#
```

```
# Usage: check_mem.sh WARNING CRITICAL  
  
# Where WARNING and CRITICAL are the integer only portions of the  
# percentage for the level desired.  
  
# (i.e. 85% Warning & 95% Critical should be input only as "85 95".)  
  
# Define Levels based on input  
#  
WARNLEVEL=$1  
CRITLEVEL=$2  
  
# Setup standard Nagios/NRPE return codes  
#  
UNKNOWN_STATE=3  
CRITICAL_STATE=2  
WARNING_STATE=1  
OK_STATE=0  
  
# Give full paths to commands - Nagios can't determine location otherwise  
#  
BC=/usr/bin/bc  
GREP=/bin/grep  
AWK=/bin/awk  
FREE=/usr/bin/free  
TAIL=/usr/bin/tail
```

```
HEAD=/usr/bin/head
```

```
# total used free shared buffers cached
```

```
#Mem: 610612 551608 59004 0 96100 100996
```

```
# Get memory information from the "free" command - output of top two lines
```

```
# looks like:
```

```
# total used free shared buffers cached
```

```
# Mem: 8248768 6944444 1304324 0 246164 5647524
```

```
# The set command will get everything from the second line and put it into
```

```
# positional variables $1 through $7.
```

```
#
```

```
set `$FREE |$HEAD -2 |$TAIL -1`
```

```
# Now give variable names to the positional variables we set above
```

```
#
```

```
MEMTOTAL=$2
```

```
MEMUSED=$3
```

```
MEMFREE=$4
```

```
MEMBUFFERS=$6
```

```
MEMCACHED=$7
```

```
# Do calculations based on what we got from free using the variables defined
```

```
#
```

```
REALMEMUSED=`echo $MEMUSED - $MEMBUFFERS - $MEMCACHED | $BC`
```

```
USEPCT=`echo "scale=3; $REALMEMUSED / $MEMTOTAL * 100" |$BC -l`  
#USEPCT=`echo scale=3 "\n" $REALMEMUSED \ $MEMTOTAL \* 100 |$BC -l |$AWK -F\"{print  
$1}`  
  
# Compare the Used percentage to the Warning and Critical levels input at  
# command line. Issue message and set return code as appropriate for each  
# level. Nagios web page will use these to determine alarm level and message.  
  
#  
  
#if [ `echo "5.0 > 5" |bc` -eq 1 ]  
#then echo it is greater  
  
#else echo it is not greater  
  
#fi  
  
if [ `echo "$USEPCT > $CRITLEVEL" |bc` -eq 1 ]  
then echo "CRITICAL - Memory usage is ${USEPCT}% (${REALMEMUSED})"  
exit ${CRITICAL_STATE}  
  
elif [ `echo "$USEPCT > $WARNLEVEL" |bc` -eq 1 ]  
then echo "WARNING - Memory usage is ${USEPCT}% (${REALMEMUSED})"  
exit ${WARNING_STATE}  
  
elif [ `echo "$USEPCT < $WARNLEVEL" |bc` -eq 1 ]  
then echo "OK - Memory usage is ${USEPCT}% (${REALMEMUSED})"  
exit ${OK_STATE}  
  
else echo "Unable to determine memory usage."  
exit ${UNKNOWN_STATE}  
  
fi  
  
echo
```

```
check_rmc.sh  
#!/bin/bash  
  
function check_rmc()  
{  
    lssrc -s ctrmc | grep -w active;  
    if [ $? -eq 0 ]  
    then  
        return 0;  
    else  
        return 2;  
    fi  
}  
  
check_rmc
```

```
[root@ctx2p10 objects]# /usr/local/nagios/libexec/check_mem.sh 40 50  
OK - Memory usage is 13.200% (254980)  
  
[root@ctx2p10 objects]# /usr/local/nagios/libexec/check_mem.sh 10 20  
WARNING - Memory usage is 13.200% (255120)  
  
[root@ctx2p10 objects]# /usr/local/nagios/libexec/check_mem.sh 0 10  
CRITICAL - Memory usage is 13.300% (255908)  
  
[root@ctx2p10 objects]#
```

Check_mem is a script to monitor the memory usage on a machine

References:

Setting up Nagios on Ubuntu:

<https://help.ubuntu.com/lts/serverguide/nagios.html>

Debugging:

1. Make sure the script/plugin has appropriate permissions to execute on local and remote host
2. find . -name check_nrpe

./lib/nagios/plugins/check_nrpe

./check_nrpe -H host2 check_mem 80 90

NRPE v2.15

./check_nrpe -H host2 -c check_mem 80 90

OK - Memory usage is 32.200% (161996)

nag1#

on remote host:

```
define service{
    use          generic-service      ; Name of service template to use
    host_name    host2
    service_description  check_mem
    check_command   check_nrpe!check_mem!placeholder
}
```

On local host:

```
define service{
    use          generic-service      ; Name of service template to use
    host_name    localhost
    service_description  check_mem
    check_command   check_mem
}
```

Arguments are specified in the object (i.e. host or service) definition, by separating them from the command name with exclamation points (!) like so:

```
check_command      check_ping!200.0,80%!400.0,40%
```

To execute commands with arguments on remote server enable this parameter:

```
# COMMAND ARGUMENT PROCESSING

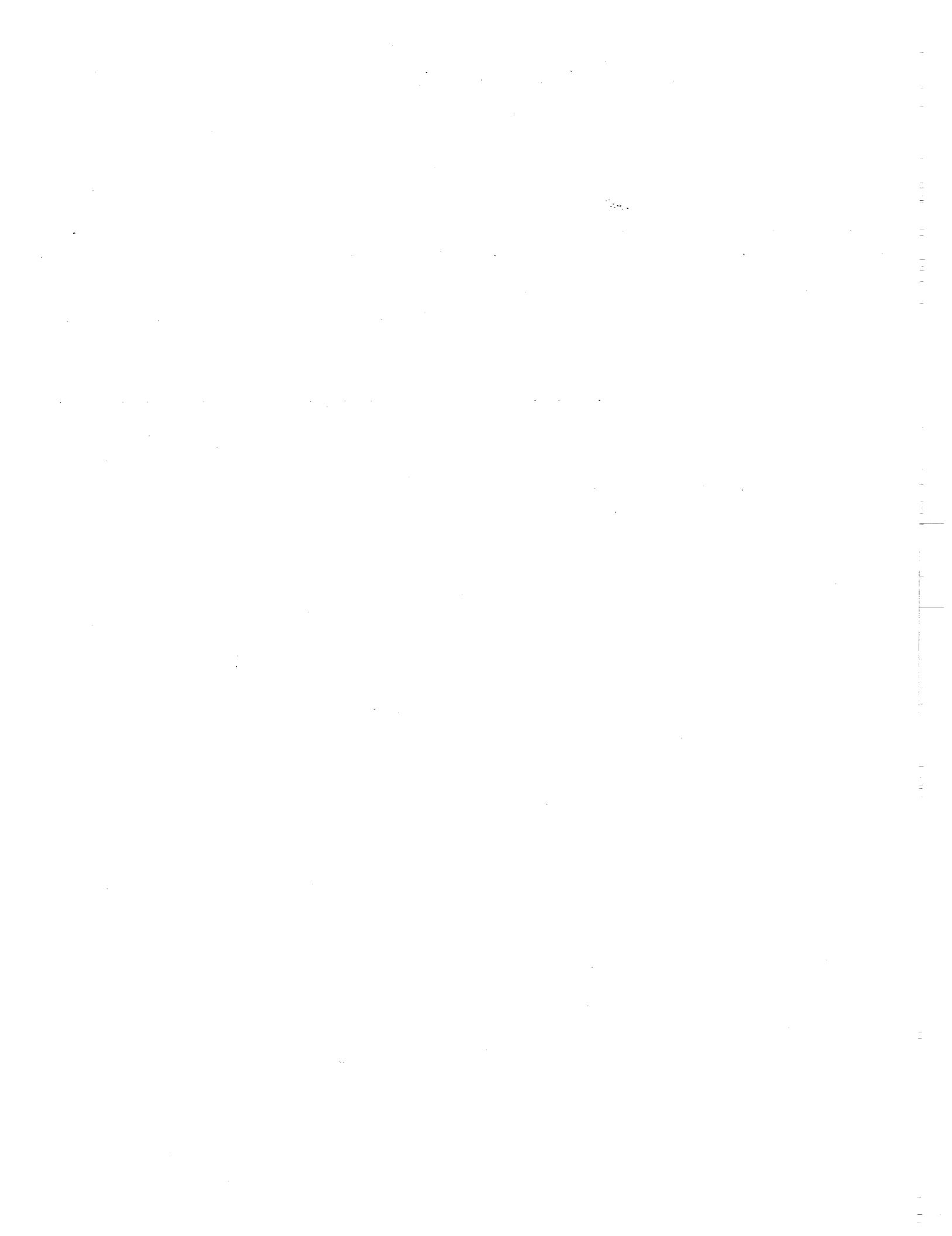
# This option determines whether or not the NRPE daemon will allow clients
# to specify arguments to commands that are executed. This option only works
# if the daemon was configured with the --enable-command-args configure script
# option.

#
# *** ENABLING THIS OPTION IS A SECURITY RISK! **

# Read the SECURITY file for information on some of the security implications
# of enabling this variable.

#
# Values: 0=do not allow arguments, 1=allow command arguments

dont_blame_nrpe=1
```



Puppet

Introduction:

Puppet is tool for configuration management and infrastructure management that is used for deploying, configuring and managing server machines

Can also be used for deployments

- Controls all the steps, right from bootstrapping to end of server life
- Can define configuration at the node level
- Can group them according to the roles
- Example: webserver database application
- Maintain consistency across nodes
- Example: if a change is done locally, it is rolled back to the original configuration.

Puppet is a tool for applying and managing system configurations across all nodes in org

Developed by puppet labs before a decade

A framework for system automation

A declarative Domain Specific Language – DSL

An Open source software written in ruby

Works on Linux, AIX, windows, mac OS

Note:

Note: Puppet server on windows is not supported.

But It can manage a windows client/node

Link: <https://docs.puppet.com/>

Puppet is free for managing 10 nodes.

It has GUI called puppet console (available in the puppet enterprise edition only)

Puppet word refers to entire puppet eco system (puppet master, slave etc)

It's a collection of tools

Runs in client server relationship

Server acts a master, and applies configuration to multiple client systems using puppet agent

Chef and salt works in the same way

Mechanics are different but core concepts are identical.

Master is sole repository of information

Puppet is leading market, 70% of people are using puppet.

Developer develops some code, and sysops faces one issue

Developer says, its not supported on some version x.y.z

In olden days it was fine for 2 weeks delay.

But nowadays software delivery itself is for 2 weeks , cuts down productivity

At the end of the day it gets resolved but how long it is taking to resolve is the question.

Operational tasks

- ➔ Provision
- ➔ Deploy
- ➔ Configure
- ➔ Monitor
- ➔ Scale
- ➔ Secure

Application reliability and scalability directly affect business profitability.

We need control flexibility as well as ease of use

Automation script/puppet:

How long it takes to deploy 1000 servers

Puppet allows to deploy configs very quickly.

Infrastructure as code:

Track

Test

Deploy

Reproduce

Scale

Dev are expected to test before deploying a change.

System admins assume new change will work and they revert if that does not work.

So system admins has to keep track of version control of configuration files

At any given point of time , we should be able to roll back to a specific version.

Infrastructure changes are also required be tested before deployment

Ability to keep track of all config changes – is a code.

Who did what changes

When was application was rolled off

Puppet modules are on module forge and git hub

Software related to puppet:

Factor: system profiling library

Collects info about systems

Ex: Node ip address, interface, OS, disk free, disk used, memory , version, 32 bit 64bit.users etc

Hiera: manages key value look up .

Memory free → 1GB

Data key → data value

Mcollective: True power of Puppet

Allows to do Complex orchestration of actions, sequencing etc...

Run specific action against specific nodes

Example: Configure first 10 nodes → then these 2

CLI driven framework.

PuppetDB: Data generated by puppet stored here

We can change db to my sql also

Persistent data storage.

Puppet Dashboard: GUI/puppet front end /external node classifier.

The full pledged puppet console is available for only enterprise

The Foremen: third party provisioning tool

Geppato: A puppet IDE based on Eclipse, can write puppet specific code.

Infrastructure as code:

Markets are constantly evolving and changing

Agile – change software very rapidly

We have 5 releases ahead

Good practices:

Versioning of code –

Previously we used to wait till dev team has finished

Nowadays,

Dev changed config file

DevOps knows that

Dev is ready for deployment

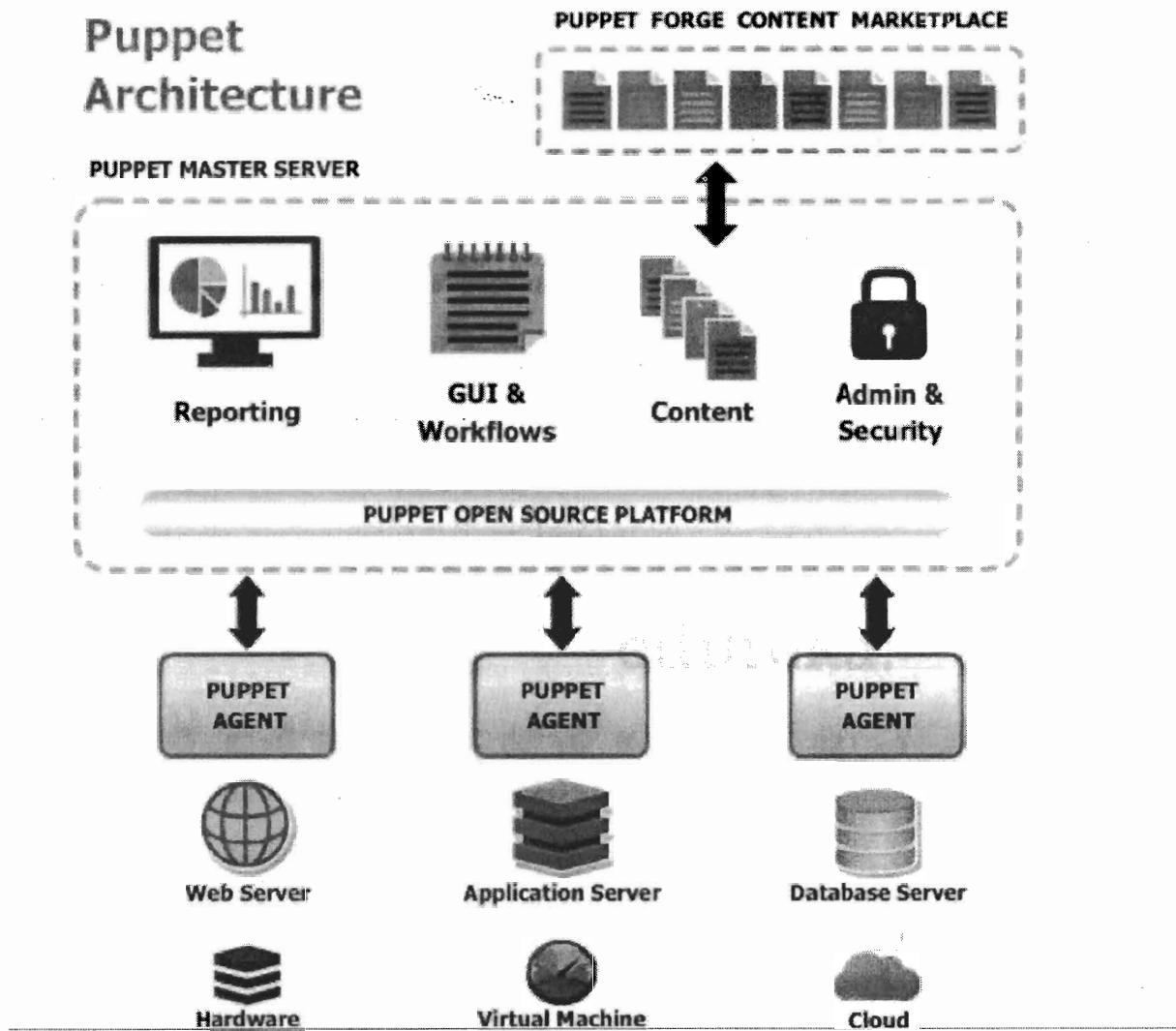
Take it deploy it

Inf as code: Both dev and ops team work together to model the required infrastructure as CODE.

All my configurations are code → this should be the idea.

Puppet Architecture:

Puppet Architecture



Enterprise puppet requires

100GB disk

8GB-16GB RAM available memory

How puppet works:

Puppet Master: The machine contains all the config for different hosts. Puppet master will run as daemon on this master server

Puppet Agent: This daemon which runs on each node and talks to the master

Connection: Connection between these 2 machines is made in a secure encrypted channel with the help of SSL.

Autosign: when node is added to master, node automatically sends key to master.

Installation & Configuration:

On CentOS/RHEL 6.5:

→ ***rpm -ivh https://yum.puppetlabs.com/el/6.5/products/x86_64/puppetlabs-release-6-10.noarch.rpm***

On CentOS/RHEL 7:

→***rpm -ivh https://yum.puppetlabs.com/el/7/products/x86_64/puppetlabs-release-7-10.noarch.rpm***

Install server

→ ***yum install puppet-server***

→ ***chkconfig puppetmaster on***

→ ***service puppetmaster start***

On CentOS/RHEL 6, where iptables is used as firewall, add following line into section ":OUTPUT ACCEPT" of /etc/sysconfig/iptables.

I -A INPUT -m state --state NEW -m tcp -p tcp --dport 8140 -j ACCEPT

service iptables restart

On CentOS/RHEL 7, where firewalld is used, the same thing can be achieved by:

firewall-cmd --permanent --zone=public --add-port=8140/tcp

firewall-cmd --reload

Client Installation

```
→# yum install puppet
```

```
→# chkconfig puppet on
```

On client change the puppet server in below file

```
→bash-4.1# cat /etc/sysconfig/puppet
```

```
# The puppetmaster server
```

```
PUPPET_SERVER=ctx1p14.in.ibm.com
```

```
# If you wish to specify the port to connect to do so here
```

```
#PUPPET_PORT=8140
```

```
# Where to log to. Specify syslog to send log messages to the system log.
```

```
#PUPPET_LOG=/var/log/puppet/puppet.log
```

```
# You may specify other parameters to the puppet client here
```

```
#PUPPET_EXTRA_OPTS="--waitforcert=500
```

```
→bash-4.1# cat /etc/puppet/puppet.conf
```

```
[main]
```

```
# The Puppet log directory.
```

```
# The default value is '$vardir/log'.
```

```
logdir = /var/log/puppet
```

```
# Where Puppet PID files are kept.
```

```
# The default value is '$vardir/run'.
```

```
rundir = /var/run/puppet
```

```
# Where SSL certificates are kept.
```

```
# The default value is '$confdir/ssl'.
```

```
ssldir = $vardir/ssl
```

```
[agent]
```

```
# The file in which puppetd stores a list of the classes
```

```
# associated with the retrieved configuration. Can be loaded in
```

```
# the separate ``puppet`` executable using the ``--loadclasses``
```

```
# option.
```

```
# The default value is '$confdir/classes.txt'.
```

```
classfile = $vardir/classes.txt
```

```
server=ctx1p14.in.ibm.com
```

```
use_cached_catalog=true
```

```
# Where puppetd caches the local configuration. An
```

```
# extension indicating the cache format is added automatically.
```

```
# The default value is '$confdir/localconfig'.
```

```
localconfig = $vardir/localconfig
```

→ service puppet start

→ puppet agent --test

With agent –test command nodes generate a certificate and sent it to server, server has to validate to establish communication.

on the pupper server

→ puppet cert list

Note : To establish the communication between master and the node, ntp time sync should be there between master and the agent.

here there will be a entry for client request

→ puppet cert sign client-node

As each puppet agent runs for the first time it submits certificate signing request (CSR) to the ceritificate authority (CA) puppet master

To check outstanding requests:

→puppet cert list

to sign a ceritificate

→ Puppet cert sign <name of client node>

Puppet DSL:

Resources

Variable and Facts

Resource relationships

Classes and modules

Classification

Other Terminology

DSL: Domain Specific Language

Puppet code is written in manifests (files with .pp extension)

Resources are grouped in classes

Classes and Configuration files are organized in modules.

When a client connects , the puppet master generates a catalog with the list of resources that the clients have to apply locally.

The puppet master has to classify the nodes and define it for each of them

The classes to include

Parameters to pass

Puppet env to use

By default all nodes are classified as production nodes

The catalog is generated by master according to the logic of our puppet code and data

Resource types are single units of configuration composed by

A type (package,service,file,user,mount,exec ...)

A title (how is called and referred)

Zero or more arguments

Type { 'title':

argument => value,

Orhter_arg=>value

}

Example for a file resource:

file {'motd':

```
path =>'/etc/motd',
content =>'Tomorrow is another day',
}
```

Install openssh :

Package { 'openssh' :

```
Ensure ➔ present,
}
```

Service { 'httpd'

```
Ensure ➔ running,
```

```
Enable -> true,
```

```
}
```

Variables:

Variable names are started with \$

Assigned with = sign

Nested arrays can be used.

Strict mode=true, if a variable is not having value, compilation will fail.

Array:

[\$a, \$b, \$c] = [1,2,3]

[\$a, [\$b,\$c]] = [1,[2,\$3]]

[\$a, \$b] = [1,[2]]

\$a=2

\$b= [2]

Hashes:

`[$a, $b] = {a => 10, b => 20}`

By default undefined variable will have undef

But puppet will stop execution

`File { "${homedir}/.vim":`

Ensure → directory.

}

`$apache::params::vhostdir`

To access a variable vhostdir variable from apache::params class

Scope:

Scope can be overridden in the local

Facts:

The command tool factor runs on clients and collects data that the server can use as variable

Facts are the system level values that cannot be changed

Ex:

Architecture → x86_64

Classes:

Classes are containers of different resources.

Class mysql (

```
{  
Root_password → 'default_value',  
Port → 3306,  
}  
  
Package { 'mysql-server':  
Ensure → present,  
}  
  
Service { 'mysql':  
Ensure → running  
}  
}
```

Packaging all the resources into one block

All packages services are put together in a class

Class definition

Class declaration -> 2 ways → include or class {'my class'}

include my class

Resource relationships:

If A and B resources are defined

A

B

So default order is executed

We can define relationship of resources using

1. Relationship metaparameters

2. Chaining arrows

3. Require() function

Before

A is before b

Automatically A will be applied first before b gets done

A requires B

Automatically B is applied first then A is applied

Notify –

A notify B

If A is changed, B gets notified, (may be B will be restarted/flushed)

Change configuration A

Restart apache B

If config is change, apache is restarted

B subscribe A

B says If A changes let me know

B will restart

Chaining operators

$A \rightarrow B$ first run A then run B

$A \sim B$ if A is changed, refresh/restart/notify B

Require

Class A{

Require B

Require C}

Declaring a class

Class { 'mysql':

Root_password → 'my_value'

Port→ '789'

}

Manifests can contain 4 or 5 or 10 classes.

If we want to separate functions and code data together

Modules can be used

Every single puppet manifest belong to a module

Only exception is site.pp

Module tools:

Install

Puppet module install puppetlabs-apache –version 0.0.2

List

Puppet module list

Puppet module uninstall <>

Puppet module search <> → searches forge site

Create a directory production

Under /etc/puppet/environments/production/modules

Add a config file environment.conf

Node.pp → pp stands for puppet programs

Cert dir:

/var/lib/puppet/ssl

puppet-master#cat environment.conf

modulepath = /etc/puppet/environments/production/modules

environment_timeout = 5s

puppet-master#pwd

/etc/puppet/environments/production

puppet-master#

Node 'web01' {

 Include apache

}

If this is include in site.pp

The node web01 starts executing class apache

Node 'web01', 'web02'

{

Include apache

}

Node /**^web\d+S/**

{

Include apache

}

Web01,Web02 ,web03 .. web100

Use if ip address is discouraged

Need to use FQDN

Manifests:

Puppet programs are called manifests

Manifests are composed of puppet code and their file names use .pp extension

The default manifest installed via apt is:

/etc/puppet/manifests/site.pp

→ mkdir -p /etc/puppet/manifests

Site.pp vs node.pp

By defaults clients get the changes from the master on pull request mode, for every 30 min

If the changes have to be reflected automatically use puppet agent -t on the clients

```
[root@ctx1p21 manifests]# cat site.pp
```

```
import 'nodes.pp'
```

While pulling requests, master checks for the manifests in site.pp

Here site.pp is importing nodes.pp

Nodes.pp can have individual nodes manifests as below

```
[root@ctx1p21 manifests]# cat nodes.pp
```

```
node 'ctx1p13.XX.XXX.com'
```

```
{
```

```
    file { '/etc/motd':
```

```
        content => "this is a file puppet MMMMaster\n"
```

```
}
```

```
    package { 'tigervnc':
```

```
        ensure => present,
```

```
}
```

```
    user { 'john1':
```

```
        ensure => present,
```

```
        comment => 'John user' ,
```

```
        home => '/home/john1' ,
```

```
        managehome => true,
```

```
    password =>
'$6$TEwb3UaixYC5.L2a$KunE.GvPMz2QEHIKBrP/v0G6vGEdJntuiZLzt8p3EdMaZ6V7SLL2WpPaY6.RsZ4IG3
zI9TLQVfVNZISyuz23g0'

}
}
```

The above example modifies /etc/inotd

Installs tigervnc packages

And creates a user called john

We can also have multiple nodes in the node.pp

```
node 'ctx1p13', 'ctx1p14', 'ctx1p15'
```

```
{
  include apache
}
```

```
node /^ctx\d+$/ {
  include apache
}
```

and add below code:

```
node 'client-node' {
    include custom_utils
}

class custom_utils {
    package { ['nmap','telnet','vim-enhanced','traceroute']:
        ensure => latest,
        allow_virtual => false,
    }
}
```

→service puppetmaster restart

puppet agent -t

puppet agent -t --debug

puppet agent -t --noop

Puppet terminology:

Example of a manifest

Resource declaration for user john:

To list default resource types that are available to puppet, enter following

Puppet resources –types

```
user { 'mitchell':  
    ensure  => present,  
    uid     => '1000',  
    gid     => '1000',  
    shell   => '/bin/bash',  
    home    => '/home/mitchell'  
}
```

The below example stops a process

```
service { 'multipathd':  
    ensure => 'stopped',  
    enable => 'false',  
}
```

The below example removes the package

```
Package { 'ntp' :  
    Ensure => absent,  
}
```

```
import "templates.pp"
import "nodes.pp"
import "classes/*"
import "groups/*"
import "users/*"
import "os/*"
```

/manifests/classes/ – Directory containing all classes

/manifests/site.pp – the primary manifest file

/manifests/templates.pp – Contains template nodes

/manifests/nodes.pp – Contains node definitions

/manifests/definitions/ – Contains all definitions

/manifests/groups/ – Contains manifests configuring groups

/manifests/os/ – Contains classes designed to configure nodes with particular operating systems

/manifests/users/ – Contains manifests configuring users

/manifest/files/ – Contains file server modules for Puppet distributable files

```
file { '/tmp/x/x.txt' :
```

```
    Ensure => present,  
}  

```

This manifest will fail if the directory /tmp/x does not exists

```
file {'/tmp/x/x.txt':
```

```
    Ensure => present,  
}  
  
file { '/tmp/x' :  
    Ensure => directory,  
}
```

Ideally this manifest should fail as /tmp/x does not exists for the first time

But puppet is intelligent enough to create directory first then create file.

Between 2 resources puppet finds the relationship between them and creates directory

```
file { 'x':  
    Ensure => directory,  
    Path => '/tmp/x'  
}  
  
file { 'r' :  
    ensure => present,  
    path => '/tmp/i/x.txt',  
}
```

```
file { 'i' :  
    ensure => directory,  
    path => '/tmp/i',  
    before => File ['r'],  
}
```

So here with use of before keyword,

File r is created before creating file called i

So the sequence is

Create file x

Create directory r

Create file i

Class:

Classes are containers of different resources.

We write code in classes

We don't have to use classes, but we recommend to use classes

Because its easy to read manifests with classes

Code can be reused

Class definition:

Does not execute any code:

Class example

```
{  
Code  
}
```

Class declaration : executes the code

Normal class – occurs when include keyword is called

Resource like class – declaration occurs when a class is declared like a resource like class {'example'}

```
class apache {  
  
  package { 'httpd':  
    ensure => 'present',  
  }  
  
  service {'httpd':  
    ensure => 'running',  
    require => Package["httpd"],  
  }  
}  
  
include apache
```

In production the site.pp looks like this

Modules:

Path: /etc/puppet/modules

Modules are for modularity, reusability.

Modules are self-contained bundles of code and data

Modules can be downloaded from puppet forge.

Every single manifest belong to a module.

Build small single-purpose modules

The site.pp manifest, contains site wide node specific code, does not belong to any module

Some more examples of manifests

How to execute a command using puppet

Class command_exec

```
{  
  exec { 'some command'  
    command => 'some command',  
    path => '/usr/sbin/'  
  }  
}
```

Factor:

Facter -p is the command to show all values

Hiera

A key value lookup tool

Can be organized and ordered nicely without touching the actual code

Just give hiera the data your module need

Hiera makes data separate from modules, so that module code can be untouched

Hiera data separation can be best explained with below example:

Ssh module without hiera:

```
class sshdconfig {  
    case $::osfamily {  
        Debian: {  
            $serviceName = 'ssh'  
        }  
        RedHat: {  
            $serviceName = 'sshd'  
        }  
    }  
}
```

```
file { "/etc/ssh/sshd_config":  
    owner => 'root',  
    group => 'root',
```

```
mode  => '0644',
#content => template("$module_name/sshd_config.erb"),
content => template("sshdconfig/sshd_config.erb"),
notify => Service[$serviceName],
}

}
```

```
service { $serviceName:
```

```
  ensure => 'running',
```

```
  enable => 'true',
```

```
}
```

```
}
```

```
include sshdconfig
```

Here the data is written in the manifest

If there are several os flavours we need to change the code every time, instead we can use hiera to get the data

As below:

```
class sshdconfig ( $serviceName = hiera("sshservicename") ){
```

```
  file { "/etc/ssh/sshd_config":
```

```
    owner  => 'root',
```

```
    group  => 'root',
```

```
    mode   => '0644',
```

```
    source => "puppet:///modules/sshdconfig/sshd_config",
```

```

    notify => Service[$serviceName],

}

service { $serviceName:
  ensure => 'running',
  enable => 'true',
}

}

include sshdconfig

```

here hiera is a function which can be used in manifests

:backends – Hiera supports yaml, json and puppet class backends.

```

[root@ctx1p13 sshdconfig]# pwd
/etc/puppet/modules/sshdconfig

[root@ctx1p13 sshdconfig]# puppet apply --test manifests/init.pp
Notice: Compiled catalog for ctx1p13.in.ibm.com in environment production in 0.20 seconds
Info: Applying configuration version '1478739772'
Notice: Finished catalog run in 0.09 seconds

[root@ctx1p13 sshdconfig]#

```

If there is no change in config file the module is never executed

Make change sshd_config and it executes module.

```
[root@ctx1p13 sshdconfig]# puppet apply --test manifests/init.pp

Notice: Compiled catalog for ctx1p13.in.ibm.com in environment production in 0.20 seconds

Info: Applying configuration version '1478739821'

Notice: /Stage[main]/Sshdconfig/File[/etc/ssh/sshd_config]/content:

--- /etc/ssh/sshd_config      2016-11-10 06:32:43.452946133 +0530
+++ /tmp/puppet-file20161110-30843-1ccq8q8-0  2016-11-10 06:33:41.595946106 +0530
@@@ -1,4 +1,4 @@

- #10th Nov 2nd run This file is managed by Puppet - Local changes will be DESTROYED.
+ #10th Nov 3rd rrun This file is managed by Puppet - Local changes will be DESTROYED.

#
# $OpenBSD: sshd_config,v 1.80 2008/07/02 02:24:18 djm Exp $
```

```
Info: Computing checksum on file /etc/ssh/sshd_config

Info: /Stage[main]/Sshdconfig/File[/etc/ssh/sshd_config]: Filebucketed /etc/ssh/sshd_config to
puppet with sum 22dea011d255c836b6cc32ce7cf273a

Notice: /Stage[main]/Sshdconfig/File[/etc/ssh/sshd_config]/content: content changed
'{md5}22dea011d255c836b6cc32ce7cf273a' to '{md5}fa81e2e095f2b34e7a7b4a153d98abe8'

Info: /Stage[main]/Sshdconfig/File[/etc/ssh/sshd_config]: Scheduling refresh of Service[sshd]

Notice: /Stage[main]/Sshdconfig/Service[sshd]: Triggered 'refresh' from 1 events

Notice: Finished catalog run in 0.33 seconds

[root@ctx1p13 sshdconfig]# ls -lrt /etc/ssh/sshd_config
-rw-r--r-- 1 root root 3997 Nov 10 06:33 /etc/ssh/sshd_config
```

```
[root@ctx1p13 sshdconfig]# date
```

Thu Nov 10 06:33:54 IST 2016

```
[root@ctx1p13 sshdconfig]# pwd
```

/etc/puppet/modules/sshdconfig

```
[root@ctx1p13 sshdconfig]#
```

Catalog:

Complete list of resources and their relationships that the puppet master generates for the client

Applied on the client after it has been received from master

Client uses RAL – Resource abstraction layer to execute actual system commands that convert abstract resources like

Package {'openssh'}

To their actual fulfillment on the system (apt-get, yum)

Catalog is saved by default:

/var/lib/puppet/client_data/catalog/\$certname.json

Creating a custom module:

puppet module generate santosh-devops

go inside Santosh-devops

```
cd manifests
```

```
open init.pp
```

```
[root@reviewb manifests]# cat site.pp
```

```
import 'node11.pp'
```

```
[root@reviewb manifests]# cat node11.pp
```

```
node 'ctx2p06.in.ibm.com'
```

```
{
```

```
    class {'apache':}
```

```
    class {'devops':}
```

```
}
```

```
class apache {
```

```
    package { 'httpd':
```

```
        ensure => 'present',
```

```
    }
```

```
    service {'httpd':
```

```
        ensure => 'running',
```

```
        require => Package["httpd"],
```

```
    }
```

```
}
```

```
[root@reviewb modules]# cd devops/
```

```
[root@reviewb devops]# ls
```

Gemfile manifests metadata.json Rakefile README.md spec tests

[root@reviewb devops]# tree

```
|── Gemfile
|── manifests
|   └── init.pp
|── metadata.json
|── Rakefile
|── README.md
|── spec
|   ├── classes
|   |   └── init_spec.rb
|   └── spec_helper.rb
└── tests
    └── init.pp
```

4 directories, 8 files

[root@reviewb devops]# cd manifests/

[root@reviewb manifests]# ls -lrt

total 4

-rw-r--r--. 1 root root 1194 Mar 19 06:32 init.pp

[root@reviewb manifests]# cat init.pp

```
# == Class: devops
```

```
#  
  
# Full description of class devops here.  
  
#  
  
# === Parameters  
  
#  
  
# Document parameters here.  
  
#  
  
# [*sample_parameter*]  
  
# Explanation of what this parameter affects and what it defaults to.  
  
# e.g. "Specify one or more upstream ntp servers as an array."  
  
#  
  
# === Variables  
  
#  
  
# Here you should define a list of variables that this module would require.  
  
#  
  
# [*sample_variable*]  
  
# Explanation of how this variable affects the function of this class and if  
# it has a default. e.g. "The parameter enc_ntp_servers must be set by the  
# External Node Classifier as a comma separated list of hostnames." (Note,  
# global variables should be avoided in favor of class parameters as  
# of Puppet 2.6.)  
  
#  
  
# === Examples  
  
#
```

```
# class { 'devops':  
#   servers => [ 'pool.ntp.org', 'ntp.local.company.com' ],  
# }  
  
#  
  
# === Authors  
  
#  
  
# Author Name <author@domain.com>  
  
#  
  
# === Copyright  
  
#  
  
# Copyright 2017 Your name here, unless otherwise noted.  
  
#  
  
class devops {  
  
$phpmysql = $osfamily ? {  
  'RedHat' => 'php-mysql',  
  'debian' => 'php5-mysql',  
  default => 'php-mysql',  
}  
  
}  
  
package {$phpmysql:  
  ensure => 'present',  
}  
  
}  
  
if $osfamily == 'RedHat'{  
  package {'php-xml':
```

```
ensure => 'present',  
}  
}  
}  
}  
  
class {'::apache':  
  docroot => '/var/www/html',  
  mpm_module => 'prefork',  
  subscribe => Package[$phpmysql],  
}  
  
class {'::apache::mod::php':}
```

Commonly used modules:

vcsrepo – change source code for version control like git

mysql

php

How to download a module:

Go to puppet forge site

Search in modules, download using the module command given



Python

Topics:

Introduction

Indentation

Numbers

strings

lists

tuples

dictionaries

loops

if

while

for

functions

modules

files

Python is a high level, interpreted and object oriented scripting language

Python was developed in 1990 by Guido Van Rossum, who works at google.

Python is quick, frictionless and nice language for automation.

Python is Interpreted

Python is processed at runtime by the interpreter. Similar to PHP and PERL, we don't have to compile python program before executing it

Python is Interactive

Using python prompt we can directly write programs.

Python is Object-Oriented

Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is Beginner's Language

Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History:

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python's feature highlights include:

Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

Easy-to-read: Python code is more clearly defined and visible to the eyes.

Easy-to-maintain: Python's source code is fairly easy-to-maintain.

A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Databases: Python provides interfaces to all major commercial databases.

GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable: Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

It supports functional and structured programming methods as well as OOP.

It can be used as a scripting language or can be compiled to byte-code for building large applications.

It provides very high-level dynamic data types and supports dynamic type checking.

It supports automatic garbage collection.

It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python is case sensitive a != A

The most up-to-date and current source code, binaries, documentation, news, etc. is available at the official website of Python:

Python Official Website : <http://www.python.org/>

Python documentation can be downloaded from the following site. The documentation is available in HTML, PDF, and PostScript formats.

Python Documentation Website : www.python.org/doc/

Interactive Mode Programming:

Invoking the interpreter without passing a script file as a parameter brings up the following prompt:

```
root# python
```

```
Python 2.5 (r25:51908, Nov 6 2007, 16:54:01)
```

```
[GCC 4.1.2 20070925 (Red Hat 4.1.2-27)] on linux2
```

```
Type "help", "copyright", "credits" or "license" for more info.
```

```
>>>
```

Type the following text to the right of the Python prompt and press the Enter key:

```
>>> print "Hello, Python!";
```

This will produce following result:

Hello, Python!

A Python identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).

Python does not allow punctuation characters such as @, \$, and % within identifiers. Python is a case sensitive programming language. Thus Manpower and manpower are two different identifiers in Python.

Here are following identifier naming convention for Python:

Class names start with an uppercase letter and all other identifiers with a lowercase letter.

Starting an identifier with a single leading underscore indicates by convention that the identifier is meant to be private.

Starting an identifier with two leading underscores indicates a strongly private identifier.

If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Reserved Words:

The following list shows the reserved words in Python. These reserved words may not be used as constant or variable or any other identifier names.

and

exec

not

assert

finally

or
break
for
pass
class
from
print
continue
global
raise
def
if
return
del
import
try
elif
in
while
else
is
with
except
lambda
yield

Lines and Indentation:

One of the first caveats programmers encounter when learning Python is the fact that there are no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. Both blocks in this example are fine:

```
if True:  
    print "True"  
  
else:  
    print "False"
```

However, the second block in this example will generate an error:

```
if True:  
    print "Answer"  
    print "True"  
  
else:  
    print "Answer"  
  
print "False"
```

Multi-Line Statements:

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example:

```
total = item_one + \  
       item_two + \  
       item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example:

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Quotation in Python:

Python accepts single ('), double ("") and triple (''' or '''''') quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes can be used to span the string across multiple lines. For example, all the following are legal:

```
word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python:

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the physical line end are part of the comment, and the Python interpreter ignores them.

```
#!/usr/bin/python3

# First comment

print "Hello, Python!"; # second comment
```

This will produce following result:

Hello, Python!

A comment may be on the same line after a statement or expression:

```
name = "Madisetti" # This is again comment
```

You can comment multiple lines as follows:

```
# This is a comment.  
# This is a comment, too.  
# This is a comment, too.  
# I said that already.
```

Variables

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals, or characters in these variables.

First program:

```
[root@reviewb var]# cat helloworld.py  
#!/usr/bin/python  
  
print "Hello World";  
  
[root@reviewb var]# ./helloworld.py  
Hello World  
[root@reviewb var]#
```

Getting variables from keyboard using input:

```
[root@reviewb var]# python  
Python 2.7.5 (default, Aug 2 2016, 04:20:16)  
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
```

Type "help", "copyright", "credits" or "license" for more information.

>>> print "Enter a number"

Enter a number

>>> x=input()

10

>>> x

10

>>> x=input()

"10"

>>> x

'10'

>>> type(x)

<type 'str'>

>>> x=input()

10

>>> x

10

>>> type(x)

<type 'int'>

>>>

>>> x=10.90

>>> type(x)

<type 'float'>

>>>

Note:

User does not have to specify the data type of the variable; python internally stores as per the value assigned

```
[root@reviewb var]# cat variables.py
```

```
#!/usr/bin/python  
x=input("Enter a value");  
y=input("Enter another value");  
print x;  
print y;
```

```
[root@reviewb var]# ./variables.py
```

```
Enter a value10
```

```
Enter another value11
```

```
10
```

```
11
```

```
[root@reviewb var]#
```

Getting variables from keyboard using raw_input:

Here the type is always string irrespective of quotes we give as input

```
[root@reviewb var]# python
```

```
Python 2.7.5 (default, Aug 2 2016, 04:20:16)
```

```
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
```

Type "help", "copyright", "credits" or "license" for more information.

```
>>> x=raw_input("Enter data")
```

Enter data10

```
>>> x
```

'10'

```
>>> type(x)
```

<type 'str'>

```
>>> x=raw_input("Enter data")
```

Enter data"str"

```
>>> type(x)
```

<type 'str'>

```
>>> x=raw_input("Enter data")
```

Enter data'str'

```
>>> type(x)
```

<type 'str'>

```
>>>
```

Standard Data Types

Numbers

Strings

List

Tuple

Dictionary

Numbers:

Number data types store numeric values. Number objects are created when you assign a value to them.
For example -

```
var1 = 1
```

```
var2 = 10
```

You can also delete the reference to a number object by using the del statement. The syntax of the del statement is -

```
del var1[,var2[,var3[....,varN]]]]
```

You can delete a single object or multiple objects by using the del statement. For example -

```
del var
```

```
del var_a, var_b
```

Python supports four different numerical types -

int (signed integers)

float (floating point real values)

complex (complex numbers)

Strings:

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes. Subsets of strings can be taken using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

Strings are immutable [read only]

The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example -

```
#!/usr/bin/python3
```

```
str = 'Hello World!'
```

```
print (str)      # Prints complete string
```

```
print (str[0])   # Prints first character of the string
```

```
print (str[2:5]) # Prints characters starting from 3rd to 5th  
print (str[2:]) # Prints string starting from 3rd character  
print (str * 2) # Prints string two times  
print (str + "TEST") # Prints concatenated string
```

This will produce the following result -

Hello World!

H

ll

lo World!

Hello World!Hello World!

Hello World!TEST

Lists:

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. For example -

Lists are mutable

```
#!/usr/bin/python3  
  
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
```

```
tinylis = [123, 'john']

print (list)      # Prints complete list
print (list[0])   # Prints first element of the list
print (list[1:3]) # Prints elements starting from 2nd till 3rd
print (list[2:])  # Prints elements starting from 3rd element
print (tinylis * 2) # Prints list two times
print (list + tinylis) # Prints concatenated lists
```

This produce the following result -

```
['abcd', 786, 2.23, 'john', 70.20000000000003]
abcd
[786, 2.23]
[2.23, 'john', 70.20000000000003]
[123, 'john', 123, 'john']
['abcd', 786, 2.23, 'john', 70.20000000000003, 123, 'john']
```

Tuples:

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ([]) and their elements and size can be changed, while tuples are enclosed in parentheses (()) and cannot be updated. Tuples can be thought of as read-only lists. For example -

Tuples are immutable, ready onlye

```
#!/usr/bin/python3
```

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
```

```
tinytuple = (123, 'john')

print (tuple)      # Prints complete tuple
print (tuple[0])   # Prints first element of the tuple
print (tuple[1:3]) # Prints elements starting from 2nd till 3rd
print (tuple[2:])  # Prints elements starting from 3rd element
print (tinytuple * 2) # Prints tuple two times
print (tuple + tinytuple) # Prints concatenated tuple
```

This produce the following result -

```
('abcd', 786, 2.23, 'john', 70.20000000000003)
abcd
(786, 2.23)
(2.23, 'john', 70.20000000000003)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.20000000000003, 123, 'john')
```

The following code is invalid with tuple, because we attempted to update a tuple, which is not allowed.
Similar case is possible with lists -

```
#!/usr/bin/python3

tuple = ('abcd', 786, 2.23, 'john', 70.2 )
list = [ 'abcd', 786, 2.23, 'john', 70.2 ]
tuple[2] = 1000 # Invalid syntax with tuple
list[2] = 1000 # Valid syntax with list
```

Dictionaries:

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({}) and values can be assigned and accessed using square braces ([]). For example -

```
#!/usr/bin/python3

dict = {}

dict['one'] = "This is one"

dict[2] = "This is two"

tinydict = {'name': 'john', 'code': 6734, 'dept': 'sales'}


print (dict['one'])      # Prints value for 'one' key
print (dict[2])         # Prints value for 2 key
print (tinydict)        # Prints complete dictionary
print (tinydict.keys()) # Prints all the keys
print (tinydict.values()) # Prints all the values
```

This produce the following result -

This is one

This is two

{'dept': 'sales', 'code': 6734, 'name': 'john'}

['dept', 'code', 'name']

['sales', 6734, 'john']

Dictionaries have no concept of order among elements. It is incorrect to say that the elements are "out of order"; they are simply unordered.

Note:

More than one entry per key is not valid, no duplicate key is allowed, key has to be unique across the dictionary

Keys are immutable. Strings, tuples, numbers can be used as dictionary keys but not lists []

```
[root@reviewb python]# cat dict.py
```

```
#!/usr/bin/python
```

```
#dictionary
```

```
dict={"name": "john", "age" : 25, "state": "Austin"}
```

```
print ("dict['name']: --> ", dict['name'])
```

```
print ("dict['age']: --> ", dict['age'])
```

```
print len(dict)
```

```
print str(dict)
```

```
[root@reviewb python]# ./dict.py
```

```
("dict['name']: --> , 'john')
```

```
("dict['age']: --> , 25)
```

```
3
```

```
{'age': 25, 'name': 'john', 'state': 'Austin'}
```

```
[root@reviewb python]#
```

```
>>> dict={"name": "john", "age" : 25, "state": "Austin"}
```

```
>>> dict.has_key("name")
```

```
True
```

```
>>> dict.has_key("name1")
```

```
False
```

```
>>> dict.items
```

```
<built-in method items of dict object at 0x2092560>
```

```
>>> dict.items()
```

```
[('age', 25), ('name', 'john'), ('state', 'Austin')]
```

```
>>>
```

Operators

Python language supports the following types of operators.

Arithmetic Operators

Comparison (Relational) Operators

Assignment Operators

Logical Operators

Bitwise Operators

Membership Operators

Identity Operators

Arithematic:

+ Addition	Adds values on either side of the operator.	$a + b = 31$
- Subtraction	Subtracts right hand operand from left hand operand.	$a - b = -11$

*	Multiplies values on either side of the operator	$a * b = 210$
Multiplication		
/ Division	Divides left hand operand by right hand operand	$b / a = 2.1$
% Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a = 1$
** Exponent	Performs exponential (power) calculation on operators	$a^{**}b = 10 \text{ to the power } 20$
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	$9//2 = 4$ and $9.0//2.0 = 4.0$, $-11//3 = -4$, - $11.0//3 = -4.0$

Sample program 1 – prints hello on screen

```

#!/usr/bin/python

#Defining a function

def main():
    print "Hello"

#Standard boiler plate that calls main function

if __name__ == '__main__':

```

```
main()
```

Sample program 2 – to print command line args

```
#!/usr/bin/python

import sys

#Defining a function

def main():

    print sys.argv

#Standard boiler plate that calls main function

if __name__ == '__main__':

    main()
```

```
[root@reviewb var]# ./1.py a ab c
```

```
['./1.py', 'a', 'ab', 'c']
```

```
[root@reviewb var]#
```

```
[root@reviewb var]# python
```

```
Python 2.7.5 (default, Aug 2 2016, 04:20:16)
```

```
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> import sys
```

```
>>> dir(sys)
```

```
'__displayhook__', '__doc__', '__excepthook__', '__name__', '__package__', '__stderr__', '__stdin__',
'__stdout__', '__clear_type_cache', '__current_frames', '__debugmallocstats', '__getframe', '__mercurial',
'api_version', 'argv', 'builtin_module_names', 'byteorder', 'call_tracing', 'callstats', 'copyright',
'displayhook', 'dont_write_bytecode', 'exc_clear', 'exc_info', 'exc_type', 'excepthook', 'exec_prefix',
```

```
'executable', 'exit', 'flags', 'float_info', 'float_repr_style', 'getcheckinterval', 'getdefaultencoding',
'getdlopenflags', 'getfilesystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof',
'gettrace', 'hexversion', 'long_info', 'maxint', 'maxsize', 'maxunicode', 'meta_path', 'modules', 'path',
'path_hooks', 'path_importer_cache', 'platform', 'prefix', 'ps1', 'ps2', 'py3kwarning', 'pydebug',
'setcheckinterval', 'setdlopenflags', 'setprofile', 'setrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout',
'subversion', 'version', 'version_info', 'warnoptions']
```

```
>>>
```

```
>>> help(sys)
```

Help on built-in module sys:

NAME

sys

FILE

(built-in)

MODULE DOCS

<http://docs.python.org/library/sys>

DESCRIPTION

This module provides access to some objects used or maintained by the interpreter and to functions that interact strongly with the interpreter.

Dynamic objects:

argv -- command line arguments; argv[0] is the script pathname if known

path -- module search path; path[0] is the script directory, else "

```
modules -- dictionary of loaded modules
```

```
>>> help(sys.exit)
```

Help on built-in function exit in module sys:

```
exit(...)
```

```
exit([status])
```

Exit the interpreter by raising SystemExit(status).

If the status is omitted or None, it defaults to zero (i.e., success).

If the status is numeric, it will be used as the system exit status.

If it is another kind of object, it will be printed and the system

exit status will be one (i.e., failure).

```
>>>
```

```
>>> len("Hello")
```

```
5
```

```
>>> len
```

```
<built-in function len>
```

```
>>>
```

Type in google

Python sys exit

Sample program 3:

```
[root@reviewb var]# cat 3.py
#!/usr/bin/python

import sys

def Hello(name):
    if name == 'john':
        name = name + '!!!!'
    else:
        Doesnotexists(name)
    print 'Hello', name

def main():
    Hello(sys.argv[1])

if __name__ == '__main__':
    main()
```

```
[root@reviewb var]#
```

```
[root@reviewb var]# ./3.py john
Hello john!!!!
```

There is a function Doesnotexists in the program which is not defined but it does not throw error if we provide the username john

Because it did not hit john

Python checks the line only when it runs

Python string type is enclosed in '' or " "

A = "this is a \"exercice\""

Strings in python are immutable.

a = "Hello"

a.lower() → hello

>>> a = "Hello"

>>> a.lower()

'hello'

>>>

>>> a

'Hello'

>>>

Here original a is unchanged

But it created a new string hello

>>> a

```
'Hello'
```

```
>>> a.find('e')
```

```
1
```

```
>>>
```

```
>>> a[0]
```

```
'H'
```

```
>>> a[1]
```

```
'e'
```

```
>>> a[10]
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
IndexError: string index out of range
```

```
>>>
```

```
>>> 'Hi %s I have %d rupees' % ('john', 42)
```

```
'Hi john I have 42 rupees'
```

```
>>>
```

```
>>> a
```

```
'Hello'
```

```
>>> a[1:3]
```

'el'

>>>

Starting at 1 and can go upto3 but not including 3

Python slicing

>>> a[1:]

'ello'

>>> a[:5]

'Hello'

>>>

-1 refers to right most character

>>> a[-1]

'o'

>>>

>>> a[:-3]

'He'

To omit last 3 characters

Time module:

```
>>> import time  
  
>>> time.time()  
1481415067.432948  
  
>>> time.localtime()  
  
time.struct_time(tm_year=2016, tm_mon=12, tm_mday=11, tm_hour=5, tm_min=41, tm_sec=36,  
tm_wday=6, tm_yday=346, tm_isdst=0)  
  
>>>  
  
>>> localtime = time.asctime( time.localtime(time.time()) )  
  
>>> print ("Local current time :", localtime)  
('Local current time :', 'Sun Dec 11 05:43:02 2016')  
  
>>>  
  
>>> time.sleep(10)
```

Sleeps the script for 10 sec

Loops:

If

While

For

Nested

If:

```
[root@reviewb python]# cat if.py
```

```
#!/usr/bin/python
```

```
print "Enter a number"  
x=input()  
if x>0:  
    print "x is positive"  
elif x<0:  
    print "x is negative"  
else:  
    print "x is zero"
```

[root@reviewb python]# ./if.py

Enter a number

-2

x is negative

[root@reviewb python]#

While:

[root@reviewb python]# cat while.py

```
#!/usr/bin/python  
  
#while  
  
i=0  
  
while i<100:  
    print i  
    i=i+1
```

prints 0 to 100 numbers

For:

[root@reviewb python]# cat for2.sh

```
#!/usr/bin/python

for i in range(1,6):
    for j in range(1,6):
        print i,
    print "\n"
```

[root@reviewb python]# ./for2.sh

1 1 1 1 1

2 2 2 2 2

3 3 3 3 3

4 4 4 4 4

5 5 5 5 5

Loop control statements:

break statement

Terminates the loop statement and transfers execution to the statement immediately following the loop.

continue statement

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

pass statement

The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

```
[root@reviewb python]# cat break.py
```

```
#!/usr/bin/python
```

```
i=0
```

```
while i<100:
```

```
    i=i+1
```

```
    if i==50:
```

```
        break
```

```
    print i
```

```
[root@reviewb python]# cat continue.py
```

```
#!/usr/bin/python
```

```
i=0
```

```
while i<100:
```

```
    i=i+1
```

```
    if i==50:
```

```
        continue
```

```
    print i
```

```
[root@reviewb python]# cat pass.py
```

```
#!/usr/bin/python
```

```
i=0
```

```
while i<10:
```

```
    i=i+1
```

```
    if i==5:
```

```
        pass
```

```
print "inside pass"
```

```
print i
```

```
[root@reviewb python]#
```

Functions:

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

Function blocks begin with the keyword def followed by the function name and parentheses (()).

Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

The first statement of a function can be an optional statement - the documentation string of the function or docstring.

The code block within every function starts with a colon (:) and is indented.

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function –

```
[root@reviewb python]# cat function.py
```

```
#!/usr/bin/python

def function(str):
    print str

function("Hello")

[root@reviewb python]# ./function.py
```

```
Hello
```

```
[root@reviewb python]#
```

Call by reference vs value:

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects in the calling function.

```
[root@reviewb python]# cat function2.py
#!/usr/bin/python
#function2

def function( list ):
    print ("values inside the function before change :", list)
    list[2]=100
    print ("values inside the function after change :", list)
    return
list = [10,20,30]
function( list )
print ("values outside the function after change :", list)
```

```
[root@reviewb python]# ./function2.py
('values inside the function before change :', [10, 20, 30])
('values inside the function after change :', [10, 20, 100])
('values outside the function after change :', [10, 20, 100])
[root@reviewb python]#
```

Function Arguments:

You can call a function by using the following types of formal arguments:

Required arguments

Keyword arguments

Default arguments

Variable-length arguments

Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

```
[root@reviewb python]# cat function.py
```

```
#!/usr/bin/python
```

```
def function(str):
```

```
    print str
```

```
function("Hello")
```

```
[root@reviewb python]# ./function.py
```

```
Hello
```

```
[root@reviewb python]# cat function.py
```

```
#!/usr/bin/python
```

```
def function(str):
```

```
    print str
```

```
function()
```

```
[root@reviewb python]# ./function.py  
Traceback (most recent call last):  
  File "./function.py", line 4, in <module>  
    function()  
TypeError: function() takes exactly 1 argument (0 given)  
[root@reviewb python]#
```

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

This allows you to skip arguments or place them out of order because the Python interpreter is able to use the keywords provided to match the values with parameters.

```
[root@reviewb python]# cat function.py  
#!/usr/bin/python  
  
def function(str):  
    print str  
  
function(str="string")  
[root@reviewb python]# ./function.py  
string  
[root@reviewb python]#
```

```
[root@reviewb python]# cat function3.py  
#!/usr/bin/python  
  
#function  
  
def function( name, age ):  
    print ("Name: ", name)  
    print ("Age: ", age)
```

```
return

function( age=40, name="john" )

[root@reviewb python]# ./function3.py

('Name:', 'john')

('Age:', 40)

[root@reviewb python]#
```

Default arguments:

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument. The following example gives an idea on default arguments, it prints default age if it is not passed –

```
[root@reviewb python]# cat function3.py

#!/usr/bin/python

#function

def function( name="default", age=40 ):

    print ("Name: ", name)

    print ("Age: ", age)

    return

function( age=40, name="hello" )

function( name="hi" )

function( age=10 )

[root@reviewb python]# ./function3.py
```

```
('Name:', 'hello')

('Age:', 40)

('Name:', 'hi')

('Age:', 40)

('Name:', 'default')

('Age:', 10)

[root@reviewb python]#
```

Variable length arguments:

You may need to process a function for more arguments than you specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.

Syntax for a function with non-keyword variable arguments is this –

```
def functionname([formal_args,] *var_args_tuple ):

    "function_docstring"

    function_suite

    return [expression]
```

An asterisk (*) is placed before the variable name that holds the values of all nonkeyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call. Following is a simple example –

```
[root@reviewb python]# cat function4.py
#!/usr/bin/python
#function
def function( arg1, *vartuple ):
    print "Output is ..."
    print (arg1)
    for var in vartuple:
        print var
    return
function( 10 )
function( 70,60,50 )
```

```
[root@reviewb python]# ./function4.py
```

```
Output is ...
```

```
10
```

```
Output is ...
```

```
70
```

```
60
```

```
50
```

```
[root@reviewb python]#
```

Lambda functions/The Anonymous Functions

These functions are called anonymous because they are not declared in the standard manner by using the def keyword. You can use the lambda keyword to create small anonymous functions.

Lambda forms can take any number of arguments but return just one value in the form of an expression. They cannot contain commands or multiple expressions.

An anonymous function cannot be a direct call to print because lambda requires an expression

Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.

Although it appears that lambda's are a one-line version of a function, they are not equivalent to inline statements in C or C++, whose purpose is by passing function stack allocation during invocation for performance reasons.

Syntax

The syntax of lambda functions contains only a single statement, which is as follows –

```
lambda [arg1 [,arg2,....argn]]:expression
```

Following is the example to show how lambda form of function works –

```
#!/usr/bin/python
```

```
# Function definition is here
```

```
sum = lambda arg1, arg2: arg1 + arg2
```

```
# Now you can call sum as a function
```

```
print ("Value of total : ", sum( 10, 20 ))
```

```
print ("Value of total : ", sum( 20, 20 ))
```

When the above code is executed, it produces the following result –

```
Value of total : 30
```

```
Value of total : 40
```

```
[root@reviewb python]# cat lambda.py
```

```
#!/usr/bin/python

sum = lambda arg1,arg2: arg1+arg2

print ("value of total: ", sum(10,20))

print ("value of total: ", sum(100,20))

[root@reviewb python]# ./lambda.py

('value of total: ', 30)

('value of total: ', 120)

[root@reviewb python]#
```

The return Statement

The statement `return [expression]` exits a function, optionally passing back an expression to the caller. A `return` statement with no arguments is the same as `return None`.

```
#!/usr/bin/python

def sum( a, b ):

    total = a+b;

    print "inside the function", total

    return total

total = sum(10,20)

print "outside the function", total

[root@reviewb python]# ./return.py

inside the function 30

outside the function 30
```

```
[root@reviewb python]#
```

Scope of variables in function:

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python –

Global variables

Local variables

Local vs Global scope:

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope. Following is a simple example –

```
[root@reviewb python]# cat scope.py
#!/usr/bin/python
#scope
total = 0 # global variable
def sum( a, b):
    total = a+b # here total is local variable
    print "Inside the function local", total
    return total
sum(10, 40)
print "Outside the function Global", total
```

```
[root@reviewb python]# ./scope.py
```

Inside the function local 50

Outside the function Global 0

```
[root@reviewb python]#
```

Modules:

A module allows to logically organize Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Any python source file can be used as module by using import statement

```
Import module1
```

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module

Python's from statement lets you import specific attributes from a module into the current namespace. The from...import has the following syntax –

From module1 import hi → imports hi function only

From module1 import hello → imports hello function only

From module1 import * → imports all functions

Locating Modules

When you import a module, the Python interpreter searches for the module in the following sequences

–

The current directory.

If the module isn't found, Python then searches each directory in the shell variable PYTHONPATH.

If all else fails, Python checks the default path. On UNIX, this default path is normally /usr/local/lib/python3/.

The module search path is stored in the system module sys as the sys.path variable. The sys.path variable contains the current directory, PYTHONPATH, and the installation-dependent default.

The PYTHONPATH Variable:

The PYTHONPATH is an environment variable, consisting of a list of directories. The syntax of PYTHONPATH is the same as that of the shell variable PATH.

Here is a typical PYTHONPATH from a Windows system:

```
set PYTHONPATH=c:\python34\lib;
```

And here is a typical PYTHONPATH from a UNIX system:

```
set PYTHONPATH=/usr/local/lib/python
```

```
reload() – to re execute the module
```

Files

The open Function

Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object, which would be utilized to call other support methods associated with it.

```
file object = open(file_name [, access_mode][, buffering])
```

Here are parameter details:

file_name: The file_name argument is a string value that contains the name of the file that you want to access.

access_mode: The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).

buffering: If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

different modes of opening a file -

Modes Description

r Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

rb Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.

r+ Opens a file for both reading and writing. The file pointer placed at the beginning of the file.

rb+ Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

w Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

wb Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

w+ Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

wb+ Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

a Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

ab Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

a+ Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

ab+ Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

R → opens for reading cursor is at beginning

R+ → opens for reading and writing, cursor is placed at the beginning of file

W → opens for writing only, overwrites if file is there

W+ → opens for reading and writing , overwrites the file

A → opens for appending, cursor at the end

A+ → both appending and reading , cursor at the end

```
#!/usr/bin/python

# Open a file

fo = open("foo.txt", "wb")

print ("Name of the file: ", fo.name)

print ("Closed or not : ", fo.closed)

print ("Opening mode : ", fo.mode)

fo.close()
```

```
[root@reviewb ~]# python
Python 2.7.5 (default, Aug 2 2016, 04:20:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> file=open('test','w')
>>> file.write("This is my first line\n")
>>> file.write("This is my second line\n"
...
>>> file.close()
>>>
[root@reviewb ~]# cat test
This is my first line
```

This is my second line

[root@reviewb ~]#

```
>>> file=open('test')
```

```
>>> file.read()
```

'This is my first line\nThis is my second line\n'

>>>

```
>>> file=open('test')
```

```
>>> print (file.read())
```

This is my first line

This is my second line

>>>

```
>>> file=open('test')
```

```
>>> file.readline()
```

'This is my first line\n'

```
>>> file.readline()
```

'This is my second line\n'

```
>>> file.readline()
```

"

>>>

Creating connection between file handle and file

Opening file

Closing file

Reading file

Writing file

```
>>> file.close()  
>>> file=open('test3','w')  
>>> file.write("This is my first line\n")  
>>> file.close()  
>>> file.open('test3')  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
AttributeError: 'file' object has no attribute 'open'  
>>> file=open('test3')  
>>> file.read(5)  
'This '  
>>> file.tell()  
5  
>>> file.read(5)  
'is my'  
>>> file.seek(0, 0)  
>>> file.read(5)  
'This '
```

```
>>>
```

File.read – to read a file , prints raw output

File.readline – to read line by line

File.write

File.tell

File.seek

```
>>> file=open('test3','r')
```

```
>>> for line in file:
```

```
...     print line
```

```
...     print "\n"
```

```
...
```

This is my second lineThis is my second lineThis is my second line

To read line by line

File=open("file",'r')

For line in file:

 Print line

Splitting the file using the delimiter:

```
>>> file=open("testfile",'r')
```

```
>>> for line in file:
```

```
...     words=line.split(" ")
```

```
...     print words
```

```
...
```

['This', 'is', 'my', 'test', 'file', 'for', 'exception', 'handling!!HelloHiiii\n']

```
['123abc']
```

```
>>>
```

Appending to a file

```
>>> file=open("test123",'a')  
>>> linesoftext=['a','b','c']  
>>> file.writelines(linesoftext)  
>>> file.close()
```

```
>>>
```

Exceptions:

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

Handling an exception

If you have some suspicious code that may raise an exception, you can defend your program by placing the suspicious code in a try: block. After the try: block, include an except: statement, followed by a block of code which handles the problem as elegantly as possible.

Syntax:

```
try:  
    You do your operations here  
.....
```

```
except Exception1:
```

If there is Exception1, then execute this block.

```
except Exception11:
```

If there is Exception11, then execute this block.

```
else:
```

If there is no exception then execute this block.

Note:

A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.

You can also provide a generic except clause, which handles any exception.

After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.

The else-block is a good place for code that does not need the try: block's protection.

This example opens a file, writes content in the file and comes out gracefully because there is no problem at all –

```
[root@reviewb python]# cat exception.py  
#!/usr/bin/python  
  
try:
```

```
fh = open("test", "w")
fh.write("This is my test file for exception handling!!!")

except IOError:
    print ("Error: can't find file or read data")

else:
    print ("Written content in the file successfully")

fh.close()

[root@reviewb python]# ./exception.py
Written content in the file successfully

[root@reviewb python]#
```

```
[root@reviewb python]# cat exception1.py
#!/usr/bin/python

try:
    fh = open("test", "r")
    fh.write("This is my test file for exception handling!!!")

except IOError:
    print ("Error: can't find file or read data")

else:
    print ("Written content in the file successfully")

fh.close()

[root@reviewb python]# ./exception1.py
Error: can't find file or read data

[root@reviewb python]#
```

In the above example, we are trying to write to a file, where we have opened with read permission hence it throws exception.

Try-Finally Clause:

We can use finally: block along with a try: block. The finally block is a place to put any code that must execute, whether the try-block raised an exception or not. The syntax of the try-finally statement is this -

try:

You do your operations here;

.....

Due to any exception, this may be skipped.

finally:

This would always be executed.

.....

```
[root@reviewb python]# cat exception3.py
```

```
#!/usr/bin/python
```

try:

```
fh = open("testfile123", "w")
```

```
fh.write("This is my test file for exception handling!!")
```

finally:

```
print ("Error: can't find file or read data")
```

```
fh.close()
```

```
[root@reviewb python]# ./exception3.py
```

```
Error: can't find file or read data
```

```
[root@reviewb python]# cat testfile123
```

```
This is my test file for exception handling!![root@reviewb python]#
```

Here the finally blocks executed even if there is no exception thrown by exception block.

```
[root@reviewb python]# cat exception4.py
```

```
#!/usr/bin/python
```

```
try:
```

```
    fh = open("testfile", "w")
```

```
try:
```

```
    fh.write("This is my test file for exception handling!!")
```

```
finally:
```

```
    print ("Going to close the file")
```

```
    fh.close()
```

```
except IOError:
```

```
    print ("Error: can't find file or read data")
```

```
[root@reviewb python]# ./exception4.py
```

```
Going to close the file
```

```
[root@reviewb python]#
```

When an exception is thrown in the try block, the execution immediately passes to the finally block. After all the statements in the finally block are executed, the exception is raised again and is handled in the except statements if present in the next higher layer of the try-except statement.

Next prog:

Int(var) will check if var is int or not:

```
[root@reviewb python]# python
Python 2.7.5 (default, Aug 2 2016, 04:20:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> var="xyz"
>>> int(var)

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'xyz'

>>> var="str"
>>> str(var)
'str'
```

A program to check if provided input is integer or not:

```
[root@reviewb python]# cat exception5.py
#!/usr/bin/python

# Define a function here.

def temp_convert(var):

    try:
        return int(var)

    except ValueError:

        print ("The argument does not contain numbers ", var)
```

```
# Call above function here.

temp_convert("xyz")
#temp_convert(123)

[root@reviewb python]# ./exception5.py

('The argument does not contain numbers ', 'xyz')

[root@reviewb python]#
```

Raise exception:

We can raise exceptions in several ways by using the raise statement. The general syntax for the raise statement is as follows.

Syntax

```
raise [Exception [, args [, traceback]]]
```

Here, Exception is the type of exception (for example, NameError) and argument is a value for the exception argument. The argument is optional; if not supplied, the exception argument is None.

The final argument, traceback, is also optional (and rarely used in practice), and if present, is the traceback object used for the exception.

Example

An exception can be a string, a class or an object. Most of the exceptions that the Python core raises are classes, with an argument that is an instance of the class. Defining new exceptions is quite easy and can be done as follows -

```
def functionName( level ):
    if level <1:
```

```
raise Exception(level)

# The code below to this would not be executed

# if we raise the exception

return level
```

Note: In order to catch an exception, an "except" clause must refer to the same exception thrown either class object or simple string. For example, to capture above exception, we must write the except clause as follows -

```
try:
    Business Logic here...
except Exception as e:
    Exception handling here using e.args...
else:
    Rest of the code here...
```

```
[root@reviewb python]# cat exception6.py
#!/usr/bin/python

def functionName( level ):
    if level <1:
        raise Exception(level)
    # The code below to this would not be executed
    # if we raise the exception

    return level
```

```
try:  
    l=functionName(-10)  
    #l=functionName(1)  
    print ("level=",l)  
  
except Exception as e:  
    print ("error in level argument",e.args[0])  
  
[root@reviewb python]# ./exception6.py  
('error in level argument', -10)  
[root@reviewb python]#
```

A program that explains exception try finally clauses clearly

```
[root@reviewb python]# cat exception7.py  
#!/usr/bin/python  
  
def divide(x,y):  
  
    try:  
        result = x/y  
    except ZeroDivisionError:  
        print "division by zero"  
    else:  
        print "result is", result  
    finally:  
        print "executing finally clause"
```

```
[root@reviewb python]# python
Python 2.7.5 (default, Aug 2 2016, 04:20:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.

>>> import os
>>> import sys
>>> from exception7 import *
>>> divide(2,2)
result is 1
executing finally clause
>>> divide(2,1)
result is 2
executing finally clause
>>> divide(4,2)
result is 2
executing finally clause
>>> divide(4,0)
division by zero
executing finally clause
>>>
```

User defined exceptions:

Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.

Here is an example related to RuntimeError. Here, a class is created that is subclassed from RuntimeError. This is useful when you need to display more specific information when an exception is caught.

In the try block, the user-defined exception is raised and caught in the except block. The variable e is used to create an instance of the class Networkerror.

```
class Networkerror(RuntimeError):
    def __init__(self, arg):
        self.args = arg
```

So once you defined above class, you can raise the exception as follows -

```
try:
    raise Networkerror("Bad hostname")
except Networkerror,e:
    print e.args
```

Object Oriented Python:

creating and instantiating a class

What is the difference between class and instance

Class is a blue print of for creating instances

why to use classes

not only python most of other languages are using classes

Classes allow us to logically group our data and function in way what is easy to reuse.

Classes contain attributes and methods.

method is a function

there is a company and we need to represent the employees in terms of class employee attributes:

name
email
sal

let us create a simple employee class

```
class Employee:  
    pass
```

class is blue print for creating instances
employee is a instances of employee class

Program 1:

```
[root@reviewb python]# cat class1.py  
#!/usr/bin/python  
  
class Employee:  
    pass  
emp_1 = Employee()  
emp_2 = Employee()  
  
print emp_1  
print emp_2  
  
[root@reviewb python]# ./class1.py  
<__main__.Employee instance at 0x7f431bc3d248>  
<__main__.Employee instance at 0x7f431bc3d200>  
[root@reviewb python]#
```

both of these are emp objects, created at diff locations

instance variables contain data that is unique to instance.

Program 2:

```
[root@reviewb python]# cat class1.py  
#!/usr/bin/python  
  
class Employee:  
    pass  
emp_1 = Employee()  
emp_2 = Employee()
```

```

print emp_1
print emp_2

emp_1.first = 'user1'
emp_1.last = 'test'
emp_1.email = 'user1.test@company.com'
emp_1.sal=100000

emp_2.first = 'user2'
emp_2.last = 'test'
emp_2.email = 'user2.test@company.com'
emp_2.sal=100000

print emp_1.email
print emp_2.email
[root@reviewb python]# ./class1.py
<__main__.Employee instance at 0x7ff221794200>
<__main__.Employee instance at 0x7ff22179a488>
user1.test@company.com
user2.test@company.com
[root@reviewb python]#

```

there is lot of manual work , and it is prone to mistakes

so we don't get much benefit this way

how to do automatically

we have to use special init method - initialize or constructor

by convention we call this instance self

Program 3:

```

[root@reviewb python]# cat class2.py
#!/usr/bin/python

class Employee:
    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.pay = pay
        self.email = first + '.' + last + '@company.com'

emp_1 = Employee('user1', 'test1', 50000)
emp_2 = Employee('user2', 'test2', 60000)

```

```
print emp_1.email
print emp_2.email
[root@reviewb python]# ./class2.py
user1.test1@company.com
user2.test2@company.com
[root@reviewb python]#
```

```
[root@reviewb python]# cat class2.py
#!/usr/bin/python

class Employee:
    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.pay = pay
        self.email = first + '.' + last + '@company.com'

emp_1 = Employee('user1', 'test1', 50000)
emp_2 = Employee('user2', 'test2', 60000)
```

```
print emp_1.email
print emp_2.email

print emp_1.first, emp_1.last
[root@reviewb python]# ./class2.py
user1.test1@company.com
user2.test2@company.com
user1 test1
[root@reviewb python]#
```

we can create a method full name so that it prints a full name automatically instance takes self by default as the first argument.

```
[root@reviewb python]# cat class2.py
#!/usr/bin/python
```

```
class Employee:
    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.pay = pay
```

```

        self.email = first + '.' + last + '@company.com'

def fullname(self):
    return '{} {}'.format(self.first, self.last)

emp_1 = Employee('user1', 'test1', 50000)
emp_2 = Employee('user2', 'test2', 60000)

print emp_1.email
print emp_2.email

#print emp_1.first, emp_1.last
print emp_1.fullname()
print Employee.fullname(emp_1)
#emp_1.fullname()

```

```

[root@reviewb python]# ./class2.py
user1.test1@company.com
user2.test2@company.com
user1 test1
user1 test1

```

Class Variables:

Instance variables are the ones which are set using the self-variable.

Example

Name

Email

Pay

Class variables are the variables that are shared among all the instances of class .

Employee class – annual raise – same for all employee

Regular methods, class methods and static methods:

Regular methods automatically takes self as the first arg.

Class method:

We can change a regular method to class method by using the decorators

`@classmethod`

```
[root@reviewb python]# cat class4.py
#!/usr/bin/python

#Defining the class variable

class Employee:

    raise_amount = 1.10

    def __init__(self, first, last, pay):
        self.first = first
        self.last = last
        self.pay = pay
        self.email = first + '.' + last + '@company.com'

    def fullname(self):
        return '{} {}'.format(self.first, self.last)
```

```
def apply_raise(self):
#           self.pay = int(self.pay * 1.04) # instead of giving manually here we can have variable for raise
amount
    self.pay = int(self.pay * Employee.raise_amount)
```

```
emp_1 = Employee('user1', 'test1', 100000)
```

```
emp_2 = Employee('user2', 'test2', 110000)
```

```
print emp_1.pay
```

```
emp_1.apply_raise()
```

```
print emp_1.pay
```

```
#print emp_1.fullname()
```

```
#print Employee.fullname(emp_1)
```

```
[root@reviewb python]#
```

```
[root@reviewb python]# cat class5.py
```

```
#!/usr/bin/python
```

```
# instance variable
```

```
class Employee:
```

```
    raise_amount = 1.04 # class variable
```

```
def __init__(self, first, last, pay):
    self.first = first # instance variables
    self.last = last
    self.pay = pay
    self.email = first + '.' + last + '@company.com'

def fullname(self):
    return '{} {}'.format(self.first, self.last)

def apply_raise(self):
    # self.pay = int(self.pay * 1.04) # instead of giving manually here we can have variable for raise amount
    self.pay = int(self.pay * self.raise_amount)

emp_1 = Employee('user1', 'test1', 50000)
emp_2 = Employee('user2', 'test2', 60000)

emp_1.raise_amount=1.05
emp_2.raise_amount=1.10

print Employee.raise_amount
print emp_1.raise_amount
print emp_2.raise_amount
```

```
#print emp_1.__dict__ #This is to print the name space of employee 1 , here raise_amount is instance variblae because we delclared it
```

```
#print emp_2.__dict__
```

```
[root@reviewb python]#
```

```
[root@reviewb python]# cat class6.py
```

```
#!/usr/bin/python
```

```
# Add one class variblae num_of_emps
```

```
class Employee:
```

```
    num_of_emps = 0 # class variable
```

```
    raise_amount = 1.04 # class variable
```

```
    def __init__(self, first, last, pay):
```

```
        self.first = first # instance variables
```

```
        self.last = last # instance variables
```

```
        self.pay = pay # instance variables
```

```
        self.email = first + '.' + last + '@company.com'      # instance variables
```

```
        Employee.num_of_emps += 1
```

```
    def fullname(self):
```

```
        return '{} {}'.format(self.first, self.last)
```

```
def apply_raise(self):
#      self.pay = int(self.pay * 1.04) # instead of giving manually here we can have variable for raise
amount

    self.pay = int(self.pay * self.raise_amount)

print "Before creating users"
print Employee.num_of_emps
print "Creating users ..."
emp_1 = Employee('user1', 'test1', 50000)
emp_2 = Employee('user2', 'test2', 60000)
emp_3 = Employee('user3', 'test3', 150000)
emp_4 = Employee('user4', 'test4', 160000)
print "After creating users"

emp_1.raise_amount=1.05

#print Employee.raise_amount
#print emp_1.raise_amount
#print emp_2.raise_amount

print Employee.num_of_emps
```

```
#print emp_1.__dict__ #This is to print the name space of employee 1 , here raise_amount is instance  
variblae because we delclared it  
  
#print emp_2.__dict__
```

```
[root@reviewb python]#
```

```
[root@reviewb python]# cat class7.py
```

```
#!/usr/bin/python
```

```
# class methods
```

```
class Employee:
```

```
    num_of_emps = 0  
  
    raise_amt = 1.04 # class variable  
  
    def __init__(self, first, last, pay):  
        self.first = first # instance variables  
        self.last = last  
        self.pay = pay  
        self.email = first + '.' + last + '@company.com'
```

```
Employee.num_of_emps += 1
```

```
def fullname(self):
    return '{} {}'.format(self.first, self.last)

def apply_raise(self):
#     self.pay = int(self.pay * 1.04) # instead of giving manually here we can have variable for raise
amount
    self.pay = int(self.pay * self.raise_amt)

@classmethod # Decorator
def set_raise_amt(cls,amount):
    cls.raise_amt = amount

emp_1 = Employee('user1', 'test1', 50000)
emp_2 = Employee('user2', 'test2', 60000)
Employee.set_raise_amt(1.05)

print Employee.raise_amt
print emp_1.raise_amt
print emp_2.raise_amt

[root@reviewb python]#
```

```
[root@reviewb python]# cat class8.py
```

```
#!/usr/bin/python
```

```
# static methods
```

```
# to find out given day is working day or not
```

```
#Regular methods pass instance as first arg (self), class methods pass class as first arg, static methods  
dont pass anything automatically they are like normal functions
```

```
class Employee:
```

```
    num_of_emps = 0
```

```
    raise_amt = 1.04 # class variable
```

```
    def __init__(self, first, last, pay):
```

```
        self.first = first # instance variables
```

```
        self.last = last
```

```
        self.pay = pay
```

```
        self.email = first + '.' + last + '@company.com'
```

```
Employee.num_of_emps += 1
```

```
    def fullname(self):
```

```
        return '{} {}'.format(self.first, self.last)
```

```
    def apply_raise(self):
```

```
        self.pay = int(self.pay * self.raise_amt)
```

```

@classmethod

def set_raise_amt(cls, amount):
    cls.raise_amt = amount


@staticmethod

def is_workday(day):

    if day.weekday() == 5 or day.weekday() == 6: #weekday is a method monday=0, sunday=6
        return False
    return True

emp_1 = Employee('user1', 'test1', 50000)
emp_2 = Employee('user2', 'test2', 60000)

import datetime

my_date = datetime.date(2016, 12, 11)

#print my_date #prints in 2016-12-11 format

print Employee.is_workday(my_date)

```

The below program explains Inheritance

```
[root@reviewb python]# cat class9.py
```

```
#!/usr/bin/python
```

```
#Inheritance
```

```
class Employee:
```

```
num_of_emps = 0

raise_amt = 1.04 # class variable

def __init__(self, first, last, pay):

    self.first = first # instance variables

    self.last = last

    self.pay = pay

    self.email = first + '.' + last + '@company.com'
```

```
Employee.num_of_emps += 1
```

```
def fullname(self):

    return '{} {}'.format(self.first, self.last)
```

```
def apply_raise(self):

    self.pay = int(self.pay * self.raise_amt)
```

```
class Developer(Employee): # inherited from parent class Employee

    pass

    raise_amt = 1.10 # if this is not specified the hike amount is taken from parent class which is 1.04
```

```
def __init__(self, first, last, pay, prog_lang):

    #super(Developer, self).__init__(first, last, pay)

    super(Employee, self).__init__(first, last, pay)

    self.prog_lang = prog_lang
```

```
dev_1 = Developer('user1', 'test1', 100000,'python')
```

```
dev_2 = Developer('user2', 'test2', 60000,'java')
```

```
print dev_1.email
```

```
print dev_2.email
```

```
print dev_1.pay
```

```
dev_1.apply_raise()
```

```
print dev_1.pay
```

```
print dev_1.prog_lan
```

```
[root@reviewb python]#
```

The below program explains Method overloading

```
[root@reviewb python]# cat class10.py
```

```
#!/usr/bin/python
```

```
#Method overloading
```

```
class Parent:
```

```
    def mymethod(self):
```

```
        print "Calling parent method"
```

```
class Child(Parent):
```

```
    def mymethod(self):
```

```
        print "Calling child method"
```

```
#     pass # if no method is declared in child class, it prints from parent class
```

```
c = Child() # creating instance of a class
```

```
c.mymethod() # child calls overridden method
```

```
[root@reviewb python]#
```

```
[root@reviewb python]# cat class11.py
```

```
#!/usr/bin/python  
#operator overloading  
#teach python how do add two objects
```

```
class Vector:
```

```
    def __init__(self, a, b):  
        self.a = a  
        self.b = b
```

```
    def __str__(self): # with str function it prints the values properly  
        return 'Vector (%d, %d)' % (self.a, self.b)
```

```
    def __add__(self,other):  
        return Vector(self.a + other.a, self.b + other.b)
```

```
v1 = Vector(2,10)
```

```
v2 = Vector(5,-2)
```

```
print (v1 + v2)

[root@reviewb python]#
```

The below program explains Inheritance

```
[root@reviewb python]# cat classinherit.py

#!/usr/bin/python

#Inheritance

class Employee:

    num_of_emps = 0

    raise_amt = 1.04 # class variable

    def __init__(self, first, last, pay):

        self.first = first # instance variables

        self.last = last

        self.pay = pay

        self.email = first + '.' + last + '@company.com'

    Employee.num_of_emps += 1

    def fullname(self):

        return '{} {}'.format(self.first, self.last)
```

```
def apply_raise(self):
    self.pay = int(self.pay * self.raise_amt)

class Developer(Employee): # inherited from parent class Employee
    raise_amt = 1.10 # if we comment this line rise will be 52000
    def __init__(self, prog_lang):
        Employee.__init__(self, first, last, pay)
        self.prog_lang = prog_lang

class Manager(Employee):
    def __init__(self, employees=None):
        Employee.__init__(self, first, last, pay)
        if employees is None:
            self.employees = []
        else:
            self.employees = employees
    def add_emp(self, emp):
        if emp not in self.employees:
            self.employees.append(emp)
    def remove_emp(self, emp):
        if emp not in self.employees:
            self.employees.remove(emp)
```

```
def print_emp(self):
    for emp in self.employees:
        print '-->', emp.fullname()

#dev_1 = Developer('user1', 'test1', 50000)
#dev_2 = Developer('user2', 'test2', 60000)

dev_1 = Developer('user1', 'test1', 50000, 'java')
dev_2 = Developer('user2', 'test2', 60000, 'python')

#dev_1 first looks for init method in Developer class, its not there so it gets from Employee

#print dev_1.email
#print dev_2.email

#print dev_1.pay
#dev_1.apply_raise()
#print dev_1.pay

mgr_1 = Manager('manager1', 'test1', 90000, [dev_1])
mgr_2 = Manager('manager2', 'test2', 190000, [dev_2])

print mgr_1.email
print mgr_2.email
```

The below program explains Inheritance .

```
[root@reviewb python]# cat classinherit.py-backup
#!/usr/bin/python

#Inheritance

class Employee:

    num_of_emps = 0

    raise_amt = 1.04 # class variable

    def __init__(self, first, last, pay):

        self.first = first # instance variables

        self.last = last

        self.pay = pay

        self.email = first + '.' + last + '@company.com'

    Employee.num_of_emps += 1
```

```
def fullname(self):
    return '{} {}'.format(self.first, self.last)

def apply_raise(self):
    self.pay = int(self.pay * self.raise_amt)

class Developer(Employee): # inherited from parent class Employee
    raise_amt = 1.10 # if we comment this line rise will be 52000

    def __init__(self, first, last, pay, prog_lang):
        super(Developer, self).__init__(first, last, pay)
        self.prog_lang = prog_lang

class Manager(Employee):
    def __init__(self, first, last, pay, employees=None):
        super(Manager, self).__init__(first, last, pay)
        if employees is None:
            self.employees = []
        else:
            self.employees = employees

    def add_emp(self, emp):
        if emp not in self.employees:
            self.employees.append(emp)

    def remove_emp(self, emp):
        if emp in self.employees:
            self.employees.remove(emp)
```

```
if emp not in self.employees:  
    self.employees.remove(emp)  
  
def print_emp(self):  
    for emp in self.employees:  
        print '-->', emp.fullname()  
  
#dev_1 = Developer('user1', 'test1', 50000)  
#dev_2 = Developer('user2', 'test2', 60000)  
  
dev_1 = Developer('user1', 'test1', 50000, 'java')  
dev_2 = Developer('user2', 'test2', 60000, 'python')  
  
#dev_1 first looks for init method in Developer class, its not there so it gets from Employee  
  
#print dev_1.email  
#print dev_2.email  
  
#print dev_1.pay  
#dev_1.apply_raise()  
#print dev_1.pay  
  
mgr_1 = Manager('manager1', 'test1', 90000, [dev_1])  
mgr_2 = Manager('manager2', 'test2', 190000, [dev_2])  
print mgr_1.email
```

```
print mgr_2.email
```

Modules:

Subprocess

```
proc = subprocess.Popen(["rsh" , nodename CMD], stdout=subprocess.PIPE, stderr=subprocess.PIPE)
output,error = proc.communicate()
print output
```

```
sys.path.append('modules')
```

```
sys.path.append("../..")
```

import os → to import the module

os.system() to use system function from module os

Vagrant

What is Vagrant

- Vagrant is open source tool to create Development environment and test the fixes/code/features before putting it on real environment
- It Creates and configures lightweight, reproducible, and portable development environments
- Most importantly it is an Open Source Software – Free!!
- It automates virtual machine creation using Oracle's Virtual Box
- The package/product is written in Ruby
- Avoiding all confusions, it is not a Virtual Machine solution, like Virtualbox, VMware.
- It is just a “helper”, a automated solution to interact easily with VirtualBox from Oracle.
- It sets-up a new VM based on the pre-configured Box that is specified on the Vagrantfile. The user does not have to open VirtualBox and configure the system, vagrant will deal with it.
- Static IPs are assigned to the VM, and is accessible from outside
- Necessary Ports can be enabled for access
- The best part is, once the job is done, the environment can be destroyed or removed
- The entire setup can be brought to life, in a few minutes, by clicking a single command

Advantages:

- Reduces the Setup time
- Very simple and straight forward
- Self servicing, based on Pull from a Base Box(each VM is based on a base box)
- Consistency can be maintaining by spawning multiple instances which are just identical.
- Repeatability (A new VM can be created and crushed in few minutes)
- Helps create a production-like VM in minutes
- With an agile driven development approach where testing becomes handy, virtualization tools like Vagrant, offer a great helping hand.

Setup and Configuration:

- 1. Download oracle virtualbox**
- 2. Install vagrant.exe**
- 3. Install gitbash**

mkdir vagrant

cd vagrant/

vagrant init → creates a file called Vagrantfile

Open this file:

Config.vm.box = "base" → box is a image

Vagrant will extract this images and create VM for us.

Box images are notisos – these are installed virtual machines

There are many boxes available we can directly install and use

Some boxes are official – provided by Ubuntu itself.

Vagrant is for Testing, training, not for production machines

Vagrant box add ubuntu/trusty64 → downloads the box

And change the Vagrantfile as below:

config.vm.box = "ubuntu/trusty64"

vagrant can do following things on VMs

restart

stop

start

edit

vagrant halt → stops/powers it off

vagrant destroy → will delete the VM

vagrant reload → reboots VM

vagrant reload --provision → setting up the VM from scratch ex: setting web server on a freshly installed VM.

Vagrant suspend – hibernates VM

Virtualbox – is a hypervisor which virtualizes the hardware.

Vmware workstation/vmware server/hyperv/

Vagrant is command line tool to automate

vagrant will create a virtual machine

vagrant up will read current directory and set up a virtual machine

default config:

memory = 512 MB

Vagrant is developed by Hashi corp

Sites to download boxes:

Example: <https://atlas.hashicorp.com/centos/boxes/7>

Vagrant downloads boxes from <https://vagrantcloud.com/>

vagrant init centos-7.0-x86_64

vagrant box add centos-7.0-x86_64 https://github.com/tommy-muehle/puppet-vagrant-boxes/releases/download/1.1.0/centos-7.0-x86_64.box

vagrant up

vagrant reload

Vagrant files:

Basic:

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

```
# vi: set ft=ruby :
```

```
# All Vagrant configuration is done below. The "2" in Vagrant.configure
```

```
# configures the configuration version (we support older styles for
```

```
# backwards compatibility). Please don't change it unless you know what
```

```
# you're doing.
```

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

```
# The most common configuration options are documented and commented below.  
# For a complete reference, please see the online documentation at  
# https://docs.vagrantup.com.
```

```
# Every Vagrant development environment requires a box. You can search for  
# boxes at https://atlas.hashicorp.com/search.  
config.vm.box = "ubuntu/trusty64"
```

```
# Disable automatic box update checking. If you disable this, then  
# boxes will only be checked for updates when the user runs  
# `vagrant box outdated`. This is not recommended.  
# config.vm.box_check_update = false
```

```
# Create a forwarded port mapping which allows access to a specific port  
# within the machine from a port on the host machine. In the example below,  
# accessing "localhost:8080" will access port 80 on the guest machine.  
# config.vm.network "forwarded_port", guest: 80, host: 8080
```

```
# Create a private network, which allows host-only access to the machine  
# using a specific IP.  
# config.vm.network "private_network", ip: "192.168.33.10"
```

```
# Create a public network, which generally matched to bridged network.  
# Bridged networks make the machine appear as another physical device on  
# your network.  
# config.vm.network "public_network"
```

```
# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
# argument is a set of non-required options.

# config.vm.synced_folder "../data", "/vagrant_data"

# Provider-specific configuration so you can fine-tune various
# backing providers for Vagrant. These expose provider-specific options.

# Example for VirtualBox:

#
# config.vm.provider "virtualbox" do |vb|
#   # Display the VirtualBox GUI when booting the machine
#   vb.gui = true
#
#   # Customize the amount of memory on the VM:
#   vb.memory = "1024"
#
# end

#
# View the documentation for the provider you are using for more
# information on available options.

# Define a Vagrant Push strategy for pushing to Atlas. Other push strategies
# such as FTP and Heroku are also available. See the documentation at
# https://docs.vagrantup.com/v2/push/atlas.html for more information.

# config.push.define "atlas" do |push|
#   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
#
# end
```

```
# Enable provisioning with a shell script. Additional provisioners such as
# Puppet, Chef, Ansible, Salt, and Docker are also available. Please see the
# documentation for more information about their specific syntax and use.

# config.vm.provision "shell", inline: <<-SHELL
#   apt-get update
#   apt-get install -y apache2
# SHELL

End
```

2. Bridged network:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.

Vagrant.configure("2") do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.

  # Every Vagrant development environment requires a box. You can search for
```

```
# boxes at https://atlas.hashicorp.com/search.  
config.vm.box = "ubuntu/trusty64"  
  
# Disable automatic box update checking. If you disable this, then  
# boxes will only be checked for updates when the user runs  
# `vagrant box outdated`. This is not recommended.  
# config.vm.box_check_update = false  
  
# Create a forwarded port mapping which allows access to a specific port  
# within the machine from a port on the host machine. In the example below,  
# accessing "localhost:8080" will access port 80 on the guest machine.  
# config.vm.network "forwarded_port", guest: 80, host: 8080  
  
# Create a private network, which allows host-only access to the machine  
# using a specific IP.  
# config.vm.network "private_network", ip: "192.168.33.10"  
  
# Create a public network, which generally matched to bridged network.  
# Bridged networks make the machine appear as another physical device on  
# your network.  
config.vm.network "public_network", bridge: "wlo1"  
  
# Share an additional folder to the guest VM. The first argument is  
# the path on the host to the actual folder. The second argument is  
# the path on the guest to mount the folder. And the optional third  
# argument is a set of non-required options.  
# config.vm.synced_folder "../data", "/vagrant_data"
```

```
# Provider-specific configuration so you can fine-tune various
# backing providers for Vagrant. These expose provider-specific options.

# Example for VirtualBox:

#
# config.vm.provider "virtualbox" do |vb|
#   # Display the VirtualBox GUI when booting the machine
#   vb.gui = true
#
#   # Customize the amount of memory on the VM:
#   vb.memory = "1024"
# end
#
# View the documentation for the provider you are using for more
# information on available options.

# Define a Vagrant Push strategy for pushing to Atlas. Other push strategies
# such as FTP and Heroku are also available. See the documentation at
# https://docs.vagrantup.com/v2/push/atlas.html for more information.

# config.push.define "atlas" do |push|
#   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
# end

# Enable provisioning with a shell script. Additional provisioners such as
# Puppet, Chef, Ansible, Salt, and Docker are also available. Please see the
# documentation for more information about their specific syntax and use.

# config.vm.provision "shell", inline: <<-SHELL
```

```
# apt-get update  
# apt-get install -y apache2  
# SHELL  
End
```

Note:

If host OS is linux (ubuntu/rhel) the wireless adapter will be wlo1

We need to bridge VM to the host to use the network on VM

By default VM gets NAT – (Network address translation) adapter

It gets private adapter.

VMs can talk to each other using NAT

If VM has to ping internet, we need to add Bridged adapter

```
$ cat Vagrantfile  
Vagrant.configure("2") do |config|  
  config.vm.box = "ubuntu/trusty64"  
  config.vm.network "public_network", bridge: "Intel(R) Dual Band  
Wireless-AC 7265 #2"  
  config.vm.define "myvm" do |myvm|  
    end  
  end
```

To login as root

Sudo su –

Set password for root using

passwd root

to enable ssh for root login

go to /etc/ssh/sshd_config

permitrootlogging yes

and restart ssh using

```
service ssh restart
```

3, Define memory

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.

Vagrant.configure("2") do |config|
  # The most common configuration options are documented and commented below.
  # For a complete reference, please see the online documentation at
  # https://docs.vagrantup.com.
```

```
# Every Vagrant development environment requires a box. You can search for
# boxes at https://atlas.hashicorp.com/search.

config.vm.box = "ubuntu/trusty64"

# Disable automatic box update checking. If you disable this, then
# boxes will only be checked for updates when the user runs
# `vagrant box outdated`. This is not recommended.

# config.vm.box_check_update = false

# Create a forwarded port mapping which allows access to a specific port
# within the machine from a port on the host machine. In the example below,
# accessing "localhost:8080" will access port 80 on the guest machine.

# config.vm.network "forwarded_port", guest: 80, host: 8080

# Create a private network, which allows host-only access to the machine
# using a specific IP.

# config.vm.network "private_network", ip: "192.168.33.10"

# Create a public network, which generally matched to bridged network.
# Bridged networks make the machine appear as another physical device on
# your network.

config.vm.network "public_network", bridge: "wlo1"

# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
```

```
# argument is a set of non-required options.

# config.vm.synced_folder "../data", "/vagrant_data"

# Provider-specific configuration so you can fine-tune various

# backing providers for Vagrant. These expose provider-specific options.

# Example for VirtualBox:

# config.vm.provider "virtualbox" do |vb|
#   # Display the VirtualBox GUI when booting the machine
#   vb.gui = true
#
#   # Customize the amount of memory on the VM:
#   vb.memory = "1024"
# end

# View the documentation for the provider you are using for more
# information on available options.

# Define a Vagrant Push strategy for pushing to Atlas. Other push strategies
# such as FTP and Heroku are also available. See the documentation at
# https://docs.vagrantup.com/v2/push/atlas.html for more information.

# config.push.define "atlas" do |push|
#   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
# end

# Enable provisioning with a shell script. Additional provisioners such as
# Puppet, Chef, Ansible, Salt, and Docker are also available. Please see the
```

```
# documentation for more information about their specific syntax and use.

# config.vm.provision "shell", inline: <<-SHELL
# apt-get update
# apt-get install -y apache2
# SHELL

End
```

4. Multi box

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

```
  config.vm.define "web" do |web|
    web.vm.box = "ubuntu/trusty64"
    web.vm.hostname = 'web'
  end
```

```
  config.vm.define "db" do |db|
    db.vm.box = "nrel/CentOS-6.5-x86_64"
    db.vm.hostname = 'db'
```

```
end
```

```
config.vm.define "control" do |control|
  control.vm.box = "ubuntu/trusty64"
  control.vm.hostname = "control"
```

```
end
```

```
end
```

config.vm.box = "ubuntu/trusty64" → global setting

we can ignore specifying the boxes – if we specify the global setting.

5. Windows:

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

```
# vi: set ft=ruby :
```

```
# All Vagrant configuration is done below. The "2" in Vagrant.configure
```

```
# configures the configuration version (we support older styles for
```

```
# backwards compatibility). Please don't change it unless you know what
```

```
# you're doing.
```

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

```
  # The most common configuration options are documented and commented below.
```

```
  # For a complete reference, please see the online documentation at
```

```
  # https://docs.vagrantup.com.
```

```
  # Every Vagrant development environment requires a box. You can search for
```

```
  # boxes at https://atlas.hashicorp.com/search.
```

```
  config.vm.box = "designerror/windows-7"
```

```
# Disable automatic box update checking. If you disable this, then
# boxes will only be checked for updates when the user runs
# `vagrant box outdated`. This is not recommended.
# config.vm.box_check_update = false

# Create a forwarded port mapping which allows access to a specific port
# within the machine from a port on the host machine. In the example below,
# accessing "localhost:8080" will access port 80 on the guest machine.
# config.vm.network "forwarded_port", guest: 80, host: 8080

# Create a private network, which allows host-only access to the machine
# using a specific IP.
# config.vm.network "private_network", ip: "192.168.33.10"

# Create a public network, which generally matched to bridged network.
# Bridged networks make the machine appear as another physical device on
# your network.
# config.vm.network "public_network"

# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
# argument is a set of non-required options.
# config.vm.synced_folder "../data", "/vagrant_data"

# Provider-specific configuration so you can fine-tune various
```

```
# backing providers for Vagrant. These expose provider-specific options.
```

```
# Example for VirtualBox:
```

```
#
```

```
# config.vm.provider "virtualbox" do |vb|
```

```
# # Display the VirtualBox GUI when booting the machine
```

```
# vb.gui = true
```

```
#
```

```
# # Customize the amount of memory on the VM:
```

```
# vb.memory = "1024"
```

```
# end
```

```
#
```

```
# View the documentation for the provider you are using for more
```

```
# information on available options.
```

```
# Define a Vagrant Push strategy for pushing to Atlas. Other push strategies
```

```
# such as FTP and Heroku are also available. See the documentation at
```

```
# https://docs.vagrantup.com/v2/push/atlas.html for more information.
```

```
# config.push.define "atlas" do |push|
```

```
# push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
```

```
# end
```

```
# Enable provisioning with a shell script. Additional provisioners such as
```

```
# Puppet, Chef, Ansible, Salt, and Docker are also available. Please see the
```

```
# documentation for more information about their specific syntax and use.
```

```
# config.vm.provision "shell", inline: <<-SHELL
```

```
# apt-get update
```

```
# apt-get install -y apache2
```

```
# SHELL
```

```
End
```

```
config.vm.network :forwarded_port, guest: 4000, host: 4000
```

network sample vagrantfile:

```
$ cat Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/trusty64"
  config.vm.network "public_network", bridge: "Intel(R) Dual Band
Wireless-AC 7265 #2"
  config.vm.define "myvm" do |myvm|
    end
  end
```

6. Provisioning using vagrant

Synced folders:

By default /vagrant directory will be created on the VM

And is shared with host machine where the vagrant file is created.

Whatever date we put in host will also be available in /vagrant

We can have bootstrap.sh for provisioning

Example:

Cat bootstrap.sh

```
#!/bin/bash
```

```
Apt-get update
```

```
Apt-get install apache2 -y
```

```
Apt-get install vim
```

```
Service apache2 restart
```

```
Give 777 permission
```

Line to be added in Vagrant file:

```
Config.vm.provision :shell, path: "apache.sh"
```

Sample file:

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

```
# vi: set ft=ruby :
```

```
# All Vagrant configuration is done below. The "2" in Vagrant.configure
```

```
# configures the configuration version (we support older styles for
```

```
# backwards compatibility). Please don't change it unless you know what
```

```
# you're doing.
```

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

```
  # The most common configuration options are documented and commented below.
```

```
  # For a complete reference, please see the online documentation at
```

```
  # https://docs.vagrantup.com.
```

```
  # Every Vagrant development environment requires a box. You can search for
```

```
  # boxes at https://atlas.hashicorp.com/search.
```

```
  config.vm.box = "ubuntu/trusty64"
```

```
  # Disable automatic box update checking. If you disable this, then
```

```
  # boxes will only be checked for updates when the user runs
```

```
  # `vagrant box outdated`. This is not recommended.
```

```
  # config.vm.box_check_update = false
```

```
  # Create a forwarded port mapping which allows access to a specific port
```

```
  # within the machine from a port on the host machine. In the example below,
```

```
# accessing "localhost:8080" will access port 80 on the guest machine.

# config.vm.network "forwarded_port", guest: 80, host: 8080


# Create a private network, which allows host-only access to the machine
# using a specific IP.

# config.vm.network "private_network", ip: "192.168.33.10"


# Create a public network, which generally matched to bridged network.

# Bridged networks make the machine appear as another physical device on
# your network.

config.vm.network "public_network", bridge: "wlo1"


# Share an additional folder to the guest VM. The first argument is
# the path on the host to the actual folder. The second argument is
# the path on the guest to mount the folder. And the optional third
# argument is a set of non-required options.

# config.vm.synced_folder "../data", "/vagrant_data"


# Provider-specific configuration so you can fine-tune various
# backing providers for Vagrant. These expose provider-specific options.

# Example for VirtualBox:

# config.vm.provider "virtualbox" do |vb|
#   # Display the VirtualBox GUI when booting the machine
#   vb.gui = true
#   #
#   # Customize the amount of memory on the VM:
```

```
vb.memory = "1024"
end
#
# View the documentation for the provider you are using for more
# information on available options.

# Define a Vagrant Push strategy for pushing to Atlas. Other push strategies
# such as FTP and Heroku are also available. See the documentation at
# https://docs.vagrantup.com/v2/push/atlas.html for more information.
# config.push.define "atlas" do |push|
#   push.app = "YOUR_ATLAS_USERNAME/YOUR_APPLICATION_NAME"
# end

# Enable provisioning with a shell script. Additional provisioners such as
# Puppet, Chef, Ansible, Salt, and Docker are also available. Please see the
# documentation for more information about their specific syntax and use.
# config.vm.provision "shell", inline: <<-SHELL
#   apt-get update
#   apt-get install -y apache2
# SHELL
config.vm.provision :shell, path: "bootstrap.sh"
# config.vm.network :forwarded_port, guest: 80, host: 4567
```

End

