ANSIBLE

Drawbacks before C.M:-

Well before configuration management arrival there were drawbacks in operations that is deploying and managing servers has become very tough for system admins as data centers grew

In the agile methodology development team was releasing software’s frequently but operations team were spending more time in configuring the systems and hence there was lag between development team and operations team

Configuration management:-

Configuration management is configuring software on top of hardware. Anything you configure the services

In an organization we have data centers with many servers which have different architecture, different applications

It is difficult to manage all those things and we need to configure each and every server like

1. Creating user account for employees and revoking their accounts while leaving in an organization
2. Patching o.s on all servers
3. Taking backups of files, data and o.s
4. Configuring services
5. Deploying applications
6. Documentation steps for each and every task , for referring next time

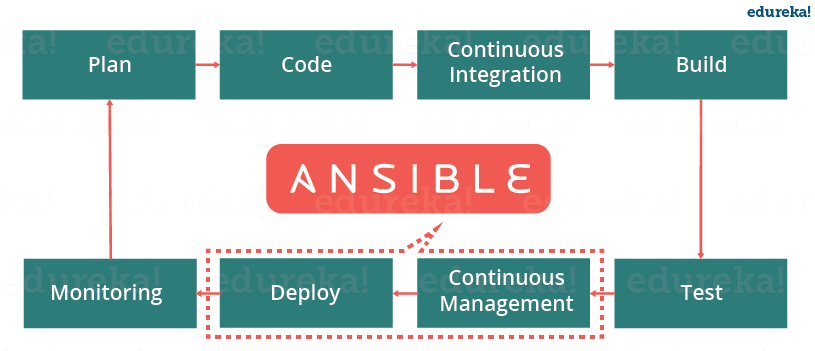
Why config management:-

1. Increase uptime(H.A) of machines
2. To improve performance
3. To prevent errors
4. To reduce cost of man hours

Ansible:-

Ansible is an automation engine, which automates configuration management, software provisioning, application deployment & orchestration tool.

* Provisioning means providing or supplying something (package, files & configuration settings), Ansible makes sure that the required packages are downloaded and installed in order to provision my application. (if I get a mail to install ftp server that is provisoining)
* Configuration management lets you configure on ‘n’ number of systems
* Automation is automating entire IT infrastructure where Ansible automates cloud provisioning, configuration management, application deployment & intra service orchestration
* Orchestration means what should be installed first and what should be installed next, and in which machine it should be installed



Here where Ansible comes into picture in devops.

idempotency:-

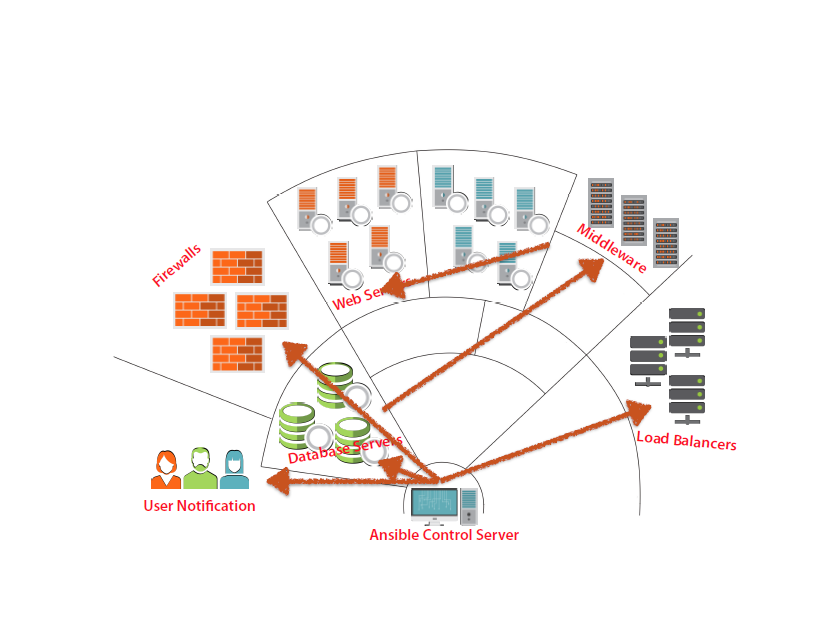
If you write a script to install httpd on remote machine. It does, and again if you try to install it will over write

But configuration management follows a principle called idempotence, although you run ansible playbook multiple times, it maintains your desired state, unless there is a change in the operation (If you write a playbook to install httpd, it will install. If you re run the playbook then ansible checks if httpd is installed or not, if installed ansible doesn’t do any operation on remote hosts)

orchestration:-

Orchestration is a process where we group all multiple automated tasks and execute them in order

Ex: - in multi-tier architecture we first setup db servers, then web servers, then load balancers and at last monitoring

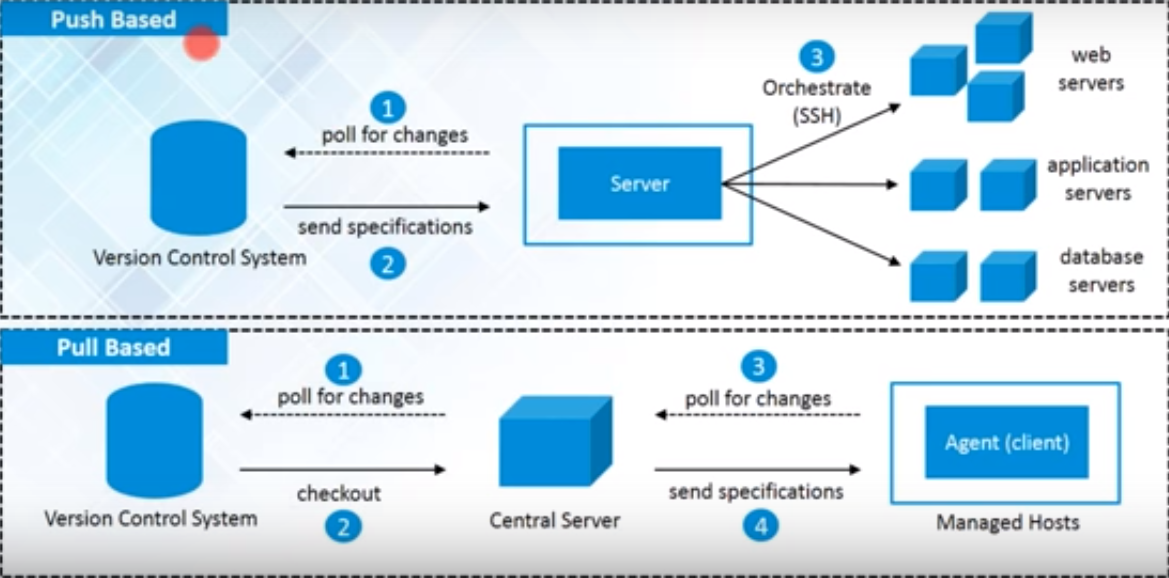


why ansible:-

1. Agentless = no need to install Ansible in client system
2. ssh = for secure connection over network between two unix machines, for windows it uses WinRM
3. Push based mechanism
4. Easy and fast to setup with minimal requirements
5. Powerful, flexible & efficient
6. Redhat support
7. Uses python (can create own plugins)

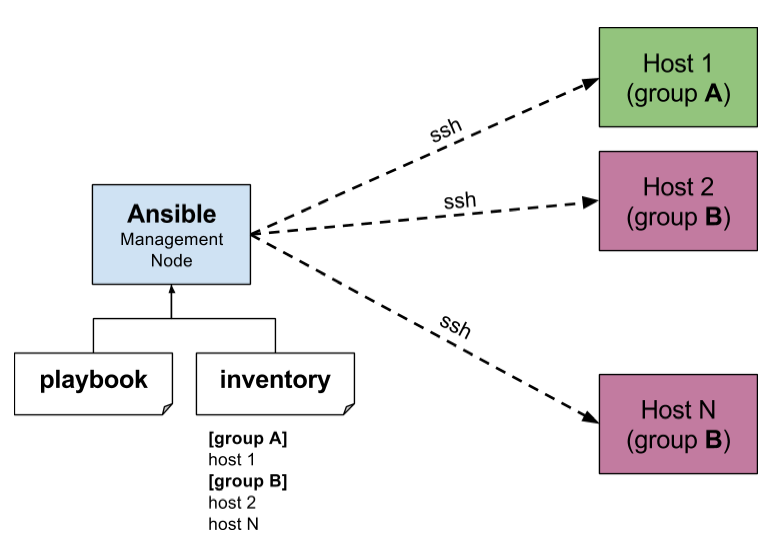
why not ansible:-

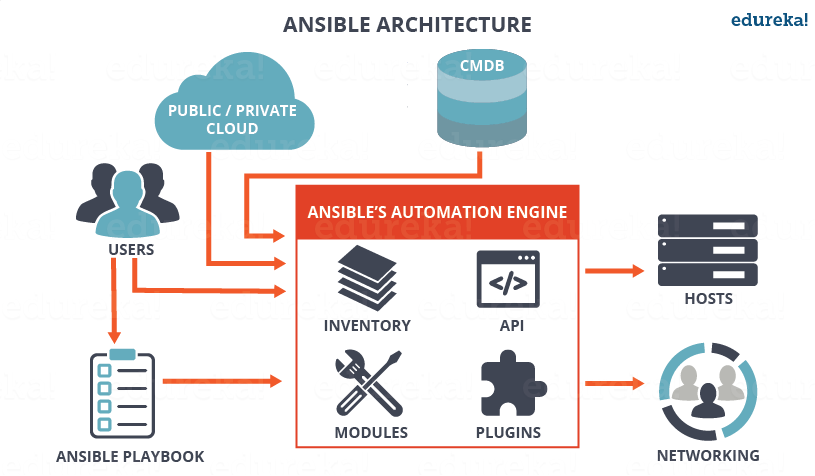
1. It takes more time to execute than chef

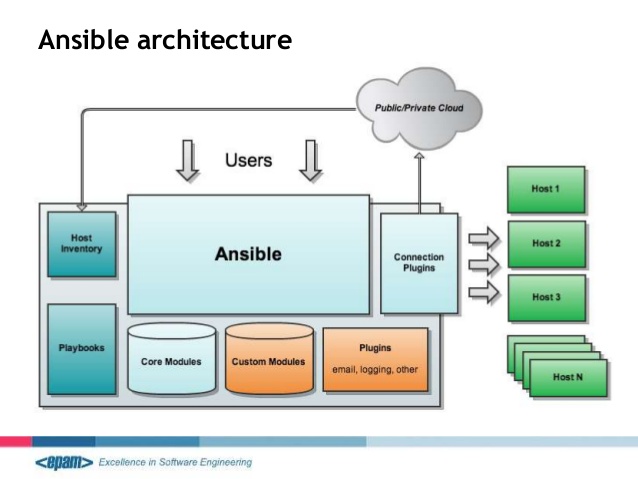
Push vs. pull based:- 

In the first figure we have vcs, central server and web, DB servers. Web, DB servers are connected to central server via ssh .The central server looks for any changes in vcs at regular intervals of time. If there are any changes those changes will be sent to central server and you can directly push onto all node machines (as configured on main server ex:-if you want the changes only for web servers then you have to configure for web servers) ex:-Ansible, salt stack

But in pull based the central server keeps on checking for any code change in vcs. If it detects then the code will be sent to central server. Now the agents which is installed on client system constantly checks central server for any configuration changes, whenever it finds changes then agent pulls the changes then the client gets effected ex:- chef, puppet

Ansible architecture:- 





Ansible workflow:-

We will define state of a machine (through code in yaml format in a playbook file) for some hosts. Now we execute that playbook on controller machine, and ansible does ssh to some hosts and gather some facts/details of the hosts. If there is a mismatch between the state of a machine and gathered facts then ansible will update it/ take action

Ex:- you defined a code where state of machine of a host should be installing apache. When you invoke ansible, it does ssh and gather some facts now ansible compares weather apache is installed on remote host, if not installed ansible will install else ansible will not take any action

Ansible terminology:-

1. Controller machine = the machine where Ansible is installed
2. Inventories = it is a file which consists of all host names/ipaddress
3. Hosts = the remote machines
4. Playbooks = a simple files where automation code is defined through tasks using YAML format.

Playbook --contains--🡪plays--contains--🡪tasks--contains--🡪modules

1. Tasks = a block that defines single procedure to be executed ex:- install a package
2. Modules = it deals with packages, creating or changing files. Ansible has lot of built-in-modules. When you run playbooks actually modules gets executed on hosts

Ex:- copies file from local m/c to host

Starts, restarts, stop service

Installs or removes packages

1. Play = a provisioning executed from start to finish is called play
2. Handlers = used to change status like restarting, stopping services
3. Plugins = special kind of modules which executes on controller machine

State of the machine:-

State of the machine defines how the host you wanted to be (ante particular server lo nuvu ela kaavali ankuntunnavo)

Ex: - you want the tomcat to be installed on host so that is the state of a machine

You wanted the tomcat to be restarted on a host machine so that is state of a machine

Infrastructure as a code(IAC):-

Configuration management works on the concept called IAC. For each users requirement we will write a code and define the machine. Through ansible you will execute

Benfits:-

1. Repeat and resuse
2. Testing
3. Better than scripts

config files:-

1. ANSIBLE\_CONFIG = environment variables
2. ansible.cfg = current directory
3. .ansible.cfg = users home directory
4. /etc/ansible/ansible.cfg = default ansible config file

You will define how ansible should work in configuration file. The default config file of ansible is /etc/ansible/ansible.cfg but ansible will follow precedence order

When you run ansible, it will check for ANSIBLE\_CONFIG file as which user is running and weather he has environment variable or not. If nothing is defined ansible will come to current directory from where you invoke ansible and checks for ansible.cfg file. If nothing is defined ansible will go to users home directory and checks for .ansible.cfg file. If ansible could not find any configuration at last it will go to /etc/ansible/ansible.cfg

inventory:-

Ansible recognizes hosts through inventory file. It is a file where we mention all host names/ipaddress,

/etc/ansible/hosts-------is ansible inventory file

localhost.com

**[webservers]**

host1.com

host2.com

**[dbservers]**

db1.com

db2.com

Ansible will run all hosts one after the other

* When we test machines for one time we don’t keep in ansible inventory file. So we define our own inventory file. We put all host names in a file and make it inventory file as follows

#ansible -i inventory-filename (-m module)/playbook

ex:- ansible -i myhostfile -m ping

by default ansible inventory is /etc/ansible/hosts.

First ansible will check for inventory in this location and give priority <current-directory where user is executing ansible>/hosts. If not found then it looks for /home/user/.ansible/hosts, if it finds ok else it looks for /etc/ansible/hosts

pattern:-

Patterns are used to decide which host to manage.

Generally we have lot of hostnames in inventory file. When you run ansible to execute some task. Ansible will do it for all the hosts in inventory file but we want ansible to pick only few machines or group of machines then in that case patterns are helpful.

#ansible hostpattern -m modulename -a argument

Ex:- ansible demo[0] --list -host

In demo group you have selected only 1st machine

patterns:-

1. all hosts--- \* ------- 192.168.1.\*
2. specific hostname/groupname----- , -------db,demo
3. union (join two groups/hosts)------ : ------demo[0:-1], db:web

0= 1st

1=2nd

-1=last

1. intersection---- :& ----
2. exclusion (excludes that host)---- :! -----demo[0]

ad-hoc:-

adhoc is nothing but one liner commands or arbitrary commands to run quickly.

#ansible host -m modulename

Ex:- #ansible all -s -m ping

1. Pinging all servers

#ansible hostname -m ping

1. Rebooting all servers

#ansible hostname –a “/sbin/reboot”

1. To copy a file

#ansible hostname –m copy -a “src=xxx dest=yyy”

1. To manage a service

#ansible hostname –m service –a “name=xxx state=yyy”

1. Deleting a user

#ansible all –m user –a “name=xxx state=yyy”

1. setup module

#ansible localhost -m setup

The setup module connects to the host and gather facts for all kind of details like ipaddress, mountpoints, cpu arch, disksace, memory…..

STATES IN ANSIBLE:-

1. present = will install a package
2. absent = will remove a package
3. latest = will update a package
4. started = will start a service
5. stopped = will stop a service
6. reloaded = will reload a service
7. restarted = will restart a service

module:-

modules are also called as task plugins, which do actual work in ansible(in ansible we call commands as module)

* note:- if module gets executed successfully o/p will be in yellow color else green color

1. copy module

ansible demo -m copy -a “src=mani dest=/home/ansible”

user module

ansible demo -s –m user -a “name=raju”

1. user account module

ansible demo –s –m user –a “name=mani ”----to create user

ansible demo –s –m user –a “name=mani state=absent ”----to delete user

facts:-

when you run ansible against a host, then ansible will gather all the properties of the host

to see facts of a host. We have a module called “setup”

ex:- ansible demo -m setup

if you don’t want all the details in o/p then you can filter out

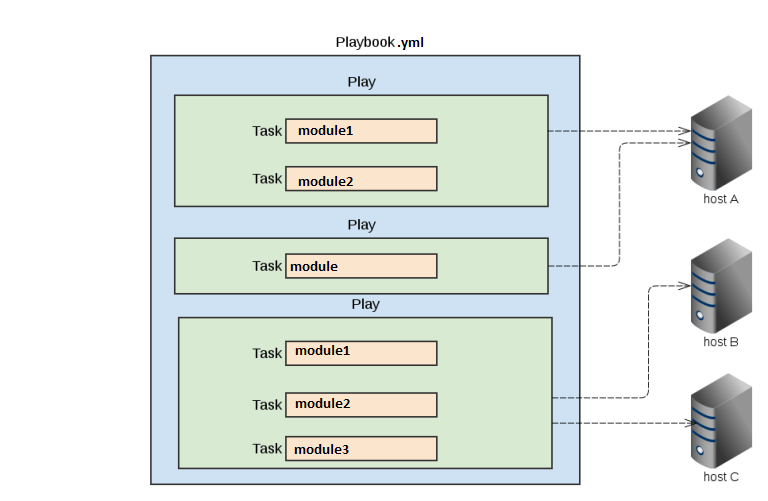
ex:- ansible demo -m setup -a ‘filter=ipv4’

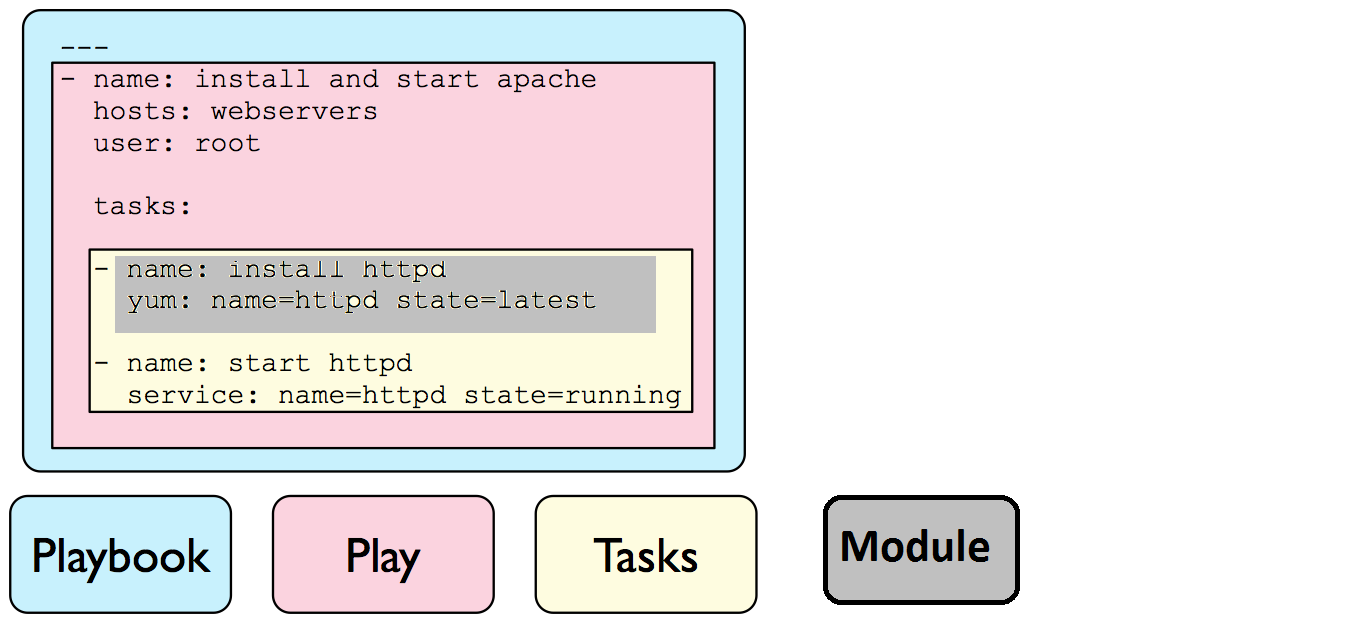
now ipv4 properties will not be shown in o/p

playbook:-

playbook is a file where we define the state of a machine through tasks to run repetitively on all hosts

each playbook compose one or more plays, each play contain one or more tasks, each tasks contains modules. When you run playbook, modules gets executed.





We use yaml language to write a playbook. Playbook should end with .yml extension. In playbook we define everything in order what to do first and what to do next

yaml basics:-

Yaml is a data serialization language. yaml file starts with list, each item in the list uses key-pair value. The starting syntax of a file is 3 hyphens (with no space between them)

Ex:- employer.yml

---

*# An employee record*

Employer1: -------list

name: Martin

job: admin --------key-value pair

skill: devops ---------members of list

...

\*we have to maintain space alignment called indentation

tasks:-

tasks are responsible for calling module. Tasks run sequentially

tasks:

- name: ensure apache is at the latest version

yum: name=httpd state=latest

- name: write the apache config file

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

play sections:-

playbook contains plays each play contains into 3 sections

1. target section:-

here we define hosts on which, play should run and how it should run (through ssh)

--- # my target section

- hosts: all

user: ansible

sudo: yes

connection: ssh

1. task section:-

we define all the modules in an order we want them to be run by ansible

tasks:

- name: install httpd on target machine

action: yum name=httpd state=installed

- name: ensure apache is at the latest version

yum: name=httpd state=latest

\*you can give either yum or action

1. variable section:-

we define variables here to store some value and reuse it

vars:

* pkg\_nam: httpd

tasks:

- name: install httpd on target machine

yum: name=’{{ pkg\_name}}’ state=installed

there are three types to use variables

1. hard-coding (vars):-

giving one value to variable and can be reused only for that playbook

vars:

* pkg\_nam: httpd

tasks:

- name: install httpd on target machine

yum: name=’{{ pkg\_name}}’ state=installed

1. file (vars\_files):-

if a variable should be used for many playbooks then we will define variable in a file

ex:- var.yml

pkgname: httpd

pb1.yml

vars\_files:

- var.yml

tasks:

- name: install httpd

yum: pkg=’{{ pkgname}}’ state=installed

1. user prompt variable (vars\_prompt):-

During execution of playbook, prompts for input then we will pass a value and that value will be stored in a variable ( In automation nothing should ask for users prompt, so we never use it )

Ex:- pb2.yml

vars\_prompt:

- name: url---#whtaever user passes will be stored in url

prompt: “enter an url” -------user passes url

tasks:

- name: install package

debug: msg=”the name is {{ url}}”

1. passing variable (command line):-

we can pass value of an variable from command line

#ansible-playbook playbook.yml --extra-vars “pkgname=value”

tasks:

- name: install apache

yum: pkg=’{{ pkgname}}’ state=installed

#ansible-playbook pb6.yml --extra-vars “pkgname=httpd”

handlers:-

handlers is a special kind of a task which you run at the end of all the tasks. If there is a change in the state, the change will be notified to the handlers and handlers gets executed

tasks:

- name: install httpd

yum: name=httpd state=installed

notify: restart httpd

handlers:

- name: restart httpd

service: name=httpd state=restarted

\*notify and handler name should be same

Here we have defined in such a way that if there is change in httpd config file, the change of the system is notified to the handler and handler gets executed

\*most of the time handlers are used for restarting the service or rebooting the system

outline of playbook:-

When writing a playbook we have to make sure everything in target section (hosts, user, sudo, date, applications, connection, logs, services..) variable section (define variables) task section (tasks handlers notifiers)

vi playbook1.yml

- - - #playbook outline

- hosts: all

user: ansible

sudo: yes

date/time

connection: ssh

gather\_facts: no

vars:

- pkgname: httpd

tasks:

- name: install httpd

yum: name=’{{ pkgname}}’ state=installed

notify: restart httpd

- name: log all the packages

raw: yum list installed > /home/ansible/abc

- name: date and time

raw: /usr/bin/date > /home/ansible/xyz

handlers:

- name: restart httpd

service: name=’{{ pkgname}}’ state=restarted

dry run:-

it is a way to check whether syntax of playbook is correct or not. Also to test how playbook is behaving before actually running (testing a model car before actually we manufacture)

dry run is used in production to avoid syntax errors while running

#ansible-playbook file.yml --check

Asynchronous action and polling:-

Asynchronous mode means tasks running parallel on different machines (if tasks are independent)

1. async:-

by default ansible will run all the tasks from top to bottom approach assume if one task is taking long time all other tasks have to wait. So async lets the time taking task to execute in background and allows other tasks to run. (async value will be the, max time ansible will wait till the task gets executed)

#async=max time

Ex:- assume there is a task called yum update, we have defined 3000sec if the task takes more than 3000sec, then async will run in background

1. poll:-

poll option frequently checks whether a task is completed or not

#poll=time

Ex:- assume there is a task called yum update, we have defined poll as10sec, now for every 10 sec ansible will check if task has completed or not

tasks:

- name: install httpd

yum: name=httpd state=installed

async: 300

poll: 10

1. fork/parllelism:-

ansible connect to remote server one at a time (ansible executes modules on all the hosts one after the other), but fork allows you to run/connect 5 servers parallel by default. If you want to change you need to modify in ansible.cfg file

fork value is defined 5 the we can run playbook on 5 hosts

vault:-

ansible vault is used to encrypt or decrypt the senstive information (paaswords, ssh keys) in playbook or files instead of leaving as plain text

to encrypt or decrypt we use a tool called ansible vault. We use this when duplicating some environment. They can use but can’t edit

1. creating a new encrypted playbook:-

You create a new encrypted playbook and you define all your tasks. For the first time you set password

So whenever you run/edit playbook ansible will ask for a password. If you edit playbook using vi you will see in encrypted format

#ansible-vault create filename.yml

Ex: - ansible-vault create pb1.yml

new passwd:

retype passwd:

1. edit encrypted playbook:-

To edit an encrypted playbook we can’t use vi

#ansible-vault edit filename.yml

Ex: - ansible-vault edit pb1.yml

passwd:

1. change password of playbook:-

If you want to change a password of a playbook

#ansible-vault rekey filename.yml

Ex: - ansible-vault rekey pb1.yml

passwd:

new passwd:

retype passwd:

1. decrypt a playbook:-

We use this when we want to remove encryption for a playbook

#ansible-vault decrypt filename.yml

Ex:- ansible-vault decrypt pb1.yml

passwd:

1. encrypt existing playbook:-

You already have a playbook and you want to encrypt that then we use this

#ansible-vault encrypt filename.yml

Ex: - ansible-vault decrypt pb1.yml

Newpasswd:

Retype passwd:

Include statement:-

Instead of writing common tasks in every playbook, it is better to write all common tasks in a file and mention/include in playbook where we wanted. So that we can use more than once in a playbook. Ansible checks for include file in current directory

tags:-

If you have large playbook and you wanted to run specific task then these tags are helpful

#ansible-playbook filename .yml --tags “taskname”

Ex:-vi pb6.yml

tasks:

- name: first\_task

yum: name=httpd state=installed

- name: second\_task

yum: name=telnet state=absent

#ansible-playbook pb6.yml --tags “first\_name”

Now first task will be executed

#ansible-playbook filename .yml --skip-tags “taskname”

Except that task all other tasks in laybook gets executed

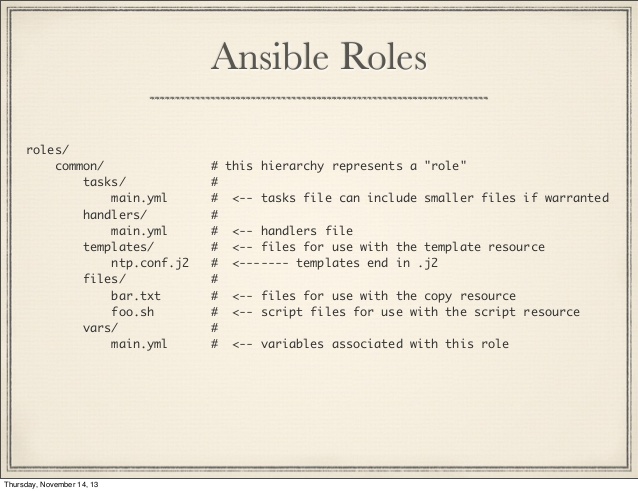
roles:-

If there is a large playbook and you add lot of functionality or more tasks, then it will become difficult to maintain a single playbook and difficult to identify a particular task if at all we want to modify

The best way to maintain a playbook is to split and store in smaller files and reuse them. Roles allows you create minimal playbooks and also organizes playbook

We split playbook, based on roles and each corresponding file is put in corresponding role. Roles has framework or directory structure

Ex: - we write all tasks in a file under task directory, vars related things is put in var directory, handlers, default, meta….



There are few roles in ansible like tasks, vars, meta, default, handler, templates, hosts.

For each role we create a directory and inside that directory we create corresponding yaml file ex:- tasks-----main.yml

In /home/ansible create a directory called “playbook”, under playbook directory create a master playbook and write all target section and mention role (as web server/db server……). Now create a directory called “roles” under playbook directory. Now create a directory of a role which mentioned in master playbook say “webservers” under roles directory. Inside webservers create “tasks” directory and inside it create “main.yml” file, inside main.yml file we define our tasks. Similarly we create directories for handlers, vars, meta, templates, default,

Roles-tasks:

If you directly write tasks in master playbook then ansible will not execute tasks. First ansible is going to check for main.yml file in tasks directory under roles and then execute

If at all ansible want to execute the tasks which you have mentioned in playbook then you have to setup something called as pre tasks and post tasks

Pre\_tasks:

- name: start role

raw: date > /home/ansible/pretask.org

roles:

- webservers

Post\_tasks:

- name: end role

raw: date > /home/ansible/pretask.org

Now ansible will execute pretask before executing roles and then execute roles and then post tasks

Roles-handler:

Now create a directory called “handlers” under role (webserver) inside handler directory create main.yml file

handlers:

- name: restart httpd

yum: name=httpd state=installed

ignore\_errors: yes

Roles-vars:

Now create a directory called “vars” under role (webserver) inside vars directory create main.yml file

vars:

- redhat\_apache: httpd

debian\_apache: apache2

\*vars have higher precedence than defaults, in roles. Ansible is gonna check in roles directory, if not mentioned it will look for defaults directory, if you have defined

Roles-configuring alternative roles paths:

Default path of roles is /home/ansilble/playbooks/roles:/etc/ansible/roles:pwd

By default ansible is going to check for roles in current directory where you are running. if not found then check for /etc/ansible/roles

You can also define your own path of roles in /etc/ansible/ansible.cfg

#roles\_path= ur own path

Roles-conditional execution:

We give conditional statement when a task matches our requirement

Ex: install httpd on redhat systems else ignore tasks:

- name: install apache on centos

yum: name=httpd state=latest

when: ansible\_os\_family == “redhat”

ignore\_errors: yes

With\_items:-

With\_items is a module in ansible, which takes the values one after the other

Ex:- to create many users, install many packages

tasks:

- name: to install packages

user: name={{ item}} state=present with\_item: - httpd

- nfsd

- vsftpd

when condition:-

Execute the task only when it meets the condition (or) condition is true

Ex:- I want to install httpd only when o.s is fedora based, if it is debian based os apache2 should be installed

tasks:

- name: install httpd on ubuntu

command: apt-get-y install apache2 when: ansible\_os\_family==”debian”

- name: install httpd on centos

yum: pkg=httpd state=latest when: ansible\_os\_family==”fedora”

do-untill:- (opposite of when)

Execute the task only when condition is false/ as long as the condition is false it iterates

tasks:

- name: install apache

yum: pkg=httpd state=latest - name: verify service status of apache shell: systemctl status httpd register: result until: result.stdout.find(“active(running)”)!=-1 retries: 5 delay: 5 debug: var=result

ignore errors:-

we know ansible runs tasks one after the other. Assume if one task failed and rest of the tasks wont execute, but you want to ignore the failed task and move on. In that case this module is helpful

tasks:

- name: install httpd

yum: pkg=httpd state=installed ignore\_errors: yes

- name: install apache

apt: pkg=apache state=restarted

in this case if task1 fails ansible ignores and moves to next tasks

debug:- s

ansible galaxy:-

ansible galaxy is an ansible hub for sharing roles. If you want role for mysql you can download from ansible galaxy

#ansible-galaxy install username.rolename

#ansible-galaxy install geerlingguy.apache geerlingguy.mysql

By default ansible roles will be downloaded into /etc/ansible/roles. You can override these settings

We can download multiple roles

#username.rolename[, version, version]

Ex:- username.role1, v1.0, v2.0

INVOKING/RUNNING A PLAYBOOK:-

We can run ansible either with adhoc commands or playbooks

# ansible-playbook filename.yml

MODULES:-

All modules are clearly explained in ansible modules folder

When you type password in ansible playbook, you can see but if that passwords needs to be hidden then you must use secrets, to use secrets you must install some package

LOOPS & CONDITIONS:-

1. when:-

when condition in ansible works like if condition, assume you have to update a system. So there will be windows, ubuntu, centos, redhat

so you have to give condition like when system is redhat update via yum, when system is ubuntu update via apt

1. with\_item:-

with\_items condition in ansible works like for loop, assume you have to install many packages then in that case we use with\_items