**VINAYAKA**

**DevOps:**

DevOps is a set of practices that combine software development and IT Operations. It aims to shorten the systems development life cycle and provides continuous delivery with high quality software.

**GIT Architecture:**



**Description:**

In git we have two repositories, Local repository and Remote repository. Remote repository is also called as central server where all the project code is kept and we create a local repo in our local workstation. Every individual user has their local workstation.

So local repository resides on the local workstation. From remote we will get a copy by using git clone and git pull commands. Initially we will get a copy from Remote repo which is nothing but a local repository.

The working directory is the folder where you are currently working on your Git project. You start with a working directory where all your files are stored. Then add your files to the staging area where ideally ready to connect your files back to the main branch on your Git project.

Once that’s done, you would want to push all the changes to a local repository where it will commit those files and make them ready for synchronization with the service. Then push your services out to the remote repository. An example of a remote repository is GitHub.

Later when you want to update your code, you can pull the latest changes from the remote repository so that your copy of the code is always the newest version of the software that the rest of the team is working on. One of the things that you can do is, as you are working on new features within the project, you can create branches. You can merge those branches with the main line code which will ensure that your code is of the highest quality and allows you to add new features to it without breaking the core code.

**Git Configuration:**

$ yum install git –y

$ git - - version

$ git config - - global user.name “smr.maheshkumar”

$ git config - - global user.email “[smr.maheshkumar@gmail.com](mailto:smr.maheshkumar@gmail.com)”

**To check above configuration:**

$ git config - - global - - list

$ cat ~/.gitconfig

**Jenkins GitHub Integration**

Install GitHub Integration Plugin and go to GitHub>Settings>Webhooks and enter as follows

http://52.15.102.192:8080/github-webook

**Git Initial Commands from Adding To Push:**

$ git init

$ git add .

$ git commit –m “first commit”

$ git remote add origin https://

$ git push –u origin master

**Git Logs:**

$ git log 🡺 shows the list of git logs

$ git log - - oneline 🡺 Shows list of logs with short commit id with file

**EX:** abcd third commit

Efgh second commit

Jklm first commit

$ git log file.txt 🡺 shows logs of file.txt

**Git Diff Command:**

ls

file1 file2 file3

Before add $ git diff file3 🡺 shows differences before the add

After add$ git diff - -stage file3.txt 🡺 shows differences after add

$ git commit –a –m “final” 🡺 editing file already tracked

$ git rev-list - -all - -count 🡺 To see all the commit counts

**Removing Files From WD & Local Repo:**

$ rm file.txt 🡺 Removes file from only WD

$ git rm file.txt 🡺 Remove both WD & Local Repo

$ git ls-files 🡺 Show the list of files related to Git

**Removing Files From Only Git Local Repo:**

$ git rm - - cached file.txt 🡺 Removes file from local repo but not WD

**GIT Ignore:**

To ignore files and to inform to the GIT to do not track the particular file, we will use git ignore.

$ vi .gitignore and mention the file name in the .gitignore

$ git add .

$ git commit –m “added gitignore”

$ git push –u origin master

**Git Checkout on Files:**

If we add some content in the file. To remove that changes, we will use git checkout command.

$ git checkout - - file.txt (or) $ git restore file.txt

**Git Reset:**

Remove changes from staging area

$ git reset Head file.txt

**Git Reset Mixed:**

Removes the commits with content

abcde third commit

fghijrf second commit

klmno first commit

$ git reset - - mixed HEAD~1

HEAD~2

**Git Reset Soft:**

abcde third commit

fghijsd second commit 🡺 becomes HEAD

klmno first commit

$ git reset - - soft HEAD~1

If we use this command 2nd commit will be the top most commit and changes will be in staged. We can get back the changes from staged

$ git commit –m “adding again” 🡺 we will get back from staged

If we want to remove from staging area use the below command

$ git reset HEAD file.txt

**Git Reset Hard:**

abcde third commit

fghijas second commit

kasmno first commit

$ git reset - hard HEAD~2 🡺 Permanently removes

**Overall Resets:**

Mixed Soft Hard

WD Staging X

$ git add git commit Permanently removed

$ git commit

**Git Branching:**

$ git branch –a 🡺 List all the branches

$ git branch production 🡺 Creates new branch production

$ git checkout production 🡺 Switches branch from master to production

$ git merge production master 🡺 Merges data from master to production

$ git push origin production 🡺 Pushes data to production

**Deleting Branches:**

$ git branch –d production 🡺 Delete in Local

$ git branch –D production 🡺 Force Delete

$ git push origin production - - delete 🡺 Delete in Remote Repo

**Deleting Files Along with the Commits:**

$ git reset abcdefg - - hard

$ git push –f origin master 🡺 Force Push

**Conflicts In Git:**

To remove the conflicts in the Git. First remove the unwanted lines in the files.

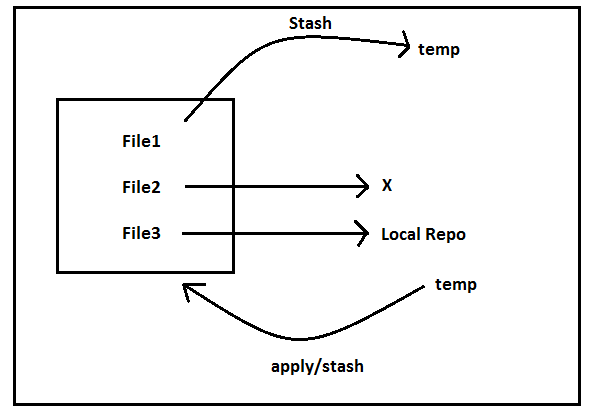
$ git add .

$ git commit –m “merge”

$ git push –u origin master

**Git Stash:**

Temporarily saves the file without pushing and we can come back and we can use it.



File1 file2 file3

If we want to stash file1

$ git status

File1 untracked

$ git stash 🡺 File1 will be moved to the temp

$ git status

Working tree clean

**Git Stash Pop:**

$ git stash list

stash@{0}: WIP on test

$ git stash pop stash@{0} 🡺 It will clear the stash. After this it is not available here.

**Git Stash Apply:**

$ git stash apply stash@{0}

$ git stash list [**Here stash is available**]

**Partial Stash:**

To stash the particular file, we will use the below command

$ git stash –p

It will ask file1 or file2 by y (or) n

$ git show stash@{0} 🡺 Show details of stash

**Delete The Stash:**

$ git stash clear 🡺 All stashes will clear

$ git stash drop stash@{1} 🡺 Particular stash will delete

**Git Reset:**

Removes changes from staging area

$ git reset HEAD file.txt

**Git Clone:**

$ git clone https://

**Git Pull:**

$ git pull https:// 🡺 Updates from central Repo

**Git Fetch:**

$ git fetch origin 🡺 Just show changes but **git pull** download the changes

**MAVEN**

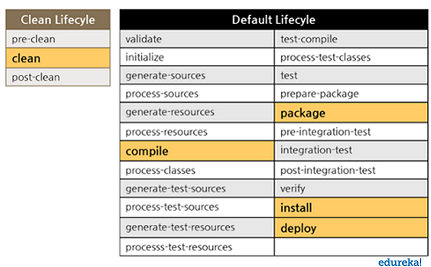
**Maven Architecture:**

## **Maven Architecture**

#### **architecture**

## **Maven life cycle, phases and goals**

#### **1. Maven life cycle**



There is a specific life cycle that Maven follows to deploy and distribute the target project.

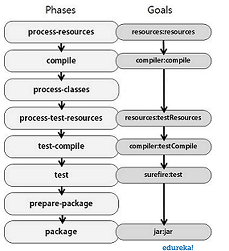
There are three built-in life cycles:

* **default** – This is the main life cycle of Maven as it is responsible for project deployment.
* **clean** – This life cycle is used to clean the project and remove all files generated by the previous build.
* **site** – The aim of this life cycle is to create the project’s site documentation.

Each life cycle is made up of a sequence of phases. The default build life cycle consists of 23 phases as it is the main build life cycle of Maven

On the other hand, clean life cycle consists of 3 phases, while the site life cycle is made up of 4 phases.

#### **2. Maven Phases**



A Maven phase is nothing but a stage in the Maven build life cycle. Each phase executes a specific task.

Here are a few important phases in the default build life cycle –

* **validate** –  This phase checks if all information necessary for the build is available
* **compile** – This phase compiles the source code
* **test-compile** – This phase compiles the test source code
* **test** – This phase runs unit tests
* **package** – This phase packages compiled source code into the distributable format (jar, war)
* **integration-test** – This phase processes and deploys the package if needed to run integration tests
* **install** – This phase installs the package to a local repository
* **deploy** – This phase copies the package to the remote repository

Maven executes phases in a specific order. This means that if we run a specific phase using the command such as mvn  <phase>, this won’t only execute the specified phase but all the preceding phases as well.

For example, if you run the command mvn deploy, that is, the deploy phase which is the last phase in the default build life cycle, then this will execute all phases prior to the deploy phase as well.

#### **3. Maven Goals**

A sequence of goals constitutes a phase and each goal executes a specific task. When you run a phase, then Maven executes all the goals in an order that are associated with that phase.  The syntax used is plugin:goal. Some of the phases and the default goals bound to them are as follows :

* compiler:compile – compile phase
* compiler:test – test-compile phase
* surefire:test – test phase
* install:install – install phase
* jar and war:war – package phase

A Maven plugin is a group of goals. However, these goals aren’t necessarily all bound to the same phase. For example, the Maven Failsafe plugin which is responsible for running integration tests. For unit testing, you need Maven surefire plugin.

**JAVA Installation:**

$ java - - version

$ yum remove java-1.7.0-openjdk –y

$ yum install java-1.8.0-openjdk –y

$ cd /usr/lib/jvm/java-1.8.0-openjdk-1.8.0222.blo-1e17\_7x86\_64

$ vi .bash\_profile

🡺 JAVA\_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0222.blo-1e17\_7x86\_64

🡺 Path=$JAVA\_HOME

echo $JAVA\_HOME

**Maven Installation:**

**Method 1:** yum install maven –y

**Method 2:**

cd /opt

wget https://

tar –xvzf apache

mv apache maven

vi .bash\_profile

M2\_HOME = /opt/maven

M2 = /opt/maven/bin

Path = $M2\_HOME:$M2

echo $M2

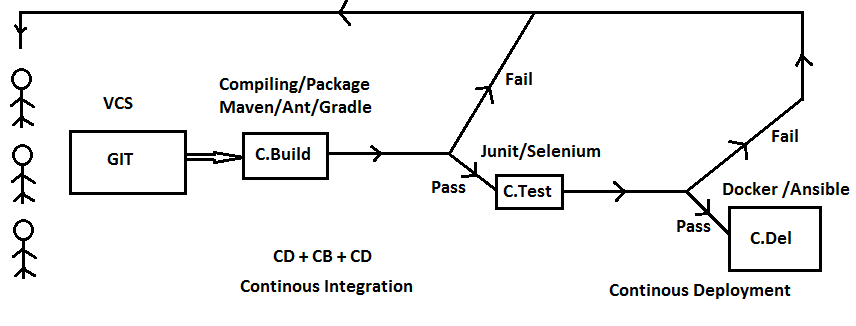
echo $M2\_HOME

mvn - - version

**JENKINS**

**Continous Integration:**

CI is a development practice that requires developers to integrate code into the shared repository at regular intervals several times a day.



**Problems before CI:**

* Chances of bugs are more
* Integration is done manually
* It will creates new work
* Error Prone
* Time Taking

**Benefits of CI:**

* Fixing the bugs is faster
* Less rework
* Less people
* Automation reduces risks of human error
* More flexible for changes

**Jenkins:**

Jenkins is a self contained, open source automation server which can be used to automate all sorts of tasks related to building, testing and delivering or deploying the software.

**Features of Jenkins:**

* Easy to setup
* Easy to configure
* Plugins
* Extensible (Lot of functionalities)
* Distributed

**Jenkins Installation:**

Get link from **jenkins.io/download** from centos

* yum install jenkins –y
* rpm –qa jenkins
* service jenkins start
* chkconfig jenkins on
* cat /var/lib/jenkins/secrets/initialAdminpassword

We will get Jenkins UI

**User Management**

Manage Jenkins 🡺 Config Global Security 🡺 Enable Security 🡺 LDAP Server details (or) Jenkins own database

**Matrix Based Security**

It enables what were the privilages that should give to the User. Under this we can give privilages to the Users.

**Project-Based Matrix Authorization Stategy**

Gives specific Jobs to the specific user.

🡺 Go to Job and Config in Jenkins 🡺 Under General 🡺 Enable Project-based Security

**Job:**

After running a sample Job

$ cd /var/lib/jenkins/workspace/jobs ls

Job1 Job2

$ cd Job1 ls

**builds config.xml nextbuildnumber**

**Builds:**

$ cd builds ls

1 2 3 4 5 legacyIds permalinks

$ cd 1

**Build.xml changelog.xml log**

**Logs 🡺** If we want to see the detail logs. We have to install Audit Trail(It will show who performed log at what time)

**Config.xml:**

Contains entore configuration of a particular job

**NextBuildnumber:**

If we want to change next build number.

🡺 vi nextbuildnumber and enter the build number that you’d like to go. Restart the Jenkins. So that you can get the nextbuildnumber as you want.

**Installation & Configuring Audit Trail:**

After installing the plugin Audit Trail. Goto Configuration 🡺 Auditrail

Log Location /tmp/jenkins\_logs

Log file size 10

Log File Count 5

$ cd /tmp/jenkins\_logs

$ cat jenkins\_logs

If we build any job now. We will get the detailed logs here.

$ cat jenkins\_logs | grep job1 (Displays only logs related to job1)

**CI/CD Pipeline**

🡺 Global Tool Configuration

🡺 Install JDK & Maven

**Free Style Project**

**Genaral** Description

**Source Code Management** Git

**Build Triggers** Poll SCM + Build after other Projects

**Build Environment**

**Build** Invoke Top Level Maven targets

**Post Build Actions** Publish Over SSH, Email Notifications & Projects to build

If we want to see in Pipeline view, we have to install **Build Pipeline** plugin

**Test Report:**

$ cd workspace/target/surefire-reports

🡺 Job configure 🡺 Post build actions 🡺 Publish Junit test report

**Publish Junit Test Report**

Test report XML’s /target/surefire-reports/\*.xml

**Producing Report For Code Analysis:**

Install **warnings next generation plugin**

Go to code analysis job 🡺 Post Build Actions 🡺 Record Compiler warnings

Tool PMD

Report file pattern **target/pmd.xml**

After building the job, we will get PMD warnings with congratulations job suceed.

**Agent / Slave setup:**

Install Jenkins in another EC2 instance

🡺 Manage Jenkins 🡺 Nodes 🡺 New Node 🡺 Give Node Name

**Name** slave1

**Remote root directory** /home/ec2-user/jenkins

**Launch method** Launch agents via SSH

**Host** 10.0.0.0/24

**Credentials** none Add

**Under Credentials Add**

**Username** ec2-user

**Private key** XXXXXXXXXXXXX

* Launch Agent
* Agent will add successfully
* Goto Particular job 🡺 Confg 🡺 Restrict where this project can be run 🡺 Slave

**Multi Configuration Project:**

**Docker**

**VM’s vs Containers:**

**Docker Containers:**

**Docker Architecture:**

* **Docker Client** Provides UI to Docker daemon
* **Docker daemon** To run command. Creating Images, containers etc
* **Docker Registry** Storage place for images

**Docker Installation:**

$ yum install docker –y

$ service docker start

$ chkconfig docker on

$ docker version 🡺 It will show client (UI) version & server version

$ docker - - version 🡺 Shows Docker version

$ docker images

$ docker ps 🡺 Shows only active containers

$ docker ps –a 🡺 Shows stopped containers including active

**Running Containers In Attached Mode:**

$ docker run –it myimage 🡺 we will go inside the container

root@bchked:/#

**|**

**Container**

To exit from the container there are two ways

* 1. exit 🡺 container will stopped after exit
  2. ctrl+p ctrl+q 🡺 container will active after exit

**Stop the Running Containers:**

$ docker stop container name or id

$ docker rm container 🡺 Removes the stopped container

$ docker rm –f container id or name 🡺 Force Remove

$ docker start container name or id 🡺 Start the container

**Running Container in Detached mode:**

$ docker run –itd myimage

**Itd 🡺** Ineractive Terminal Detached Mode

In this we don’t go into the shell of a container. It won’t be stop untill you stop the container.

If you want to go into the shell of the conatiner use the below command

**$ docker attach container id or name**

**(or)**

**$ docker exec –it container name or id /bin/bash**

**Removing docker images:**

$ docker rmi myimage

**Docker Volumes**

Back up of the containers

**Creating Docker volumes:**

$ docker volume ls 🡺 list all the volumes

$ docker volume create vol1 🡺 creates volume with vol1

$ docker volume inspect vol1 🡺 Inpects the vol1

{

“Mount point:”/var/lib/docker/volumes/vol1/\_data”,

}

cd /var/lib/docker/volumes/vol1/\_data 🡺 Location of docker volume

[root@vol1\_data]# 🡺 Entered into the docker volume

**Attaching Volumes to the containers:**

$ docker run –it –name container1 –v vol1:/data tomcat

**-v** flag indicates source & dest

We will enter into the shell

ls

cdkc ckhd data dev etc

**$ cd data ls**

If we create any file here, we will get the files in the destination

$ exit

$ docker volume ls

$ docker volume inspect vol1

If we delete the container, data will be backup automatically

**Attaching volume to the another containers**

$ docker run –it - - name container2 –v vol1:/data1 ubuntu

**Deleting Volume**

$ docker volume rm vol1

Delete unused volumes. We cannot remove volumes if containers is using the volumes.

**Creating Volume with V flag:**

$ docker run –it - - name container –v vol2:/data2 ubuntu

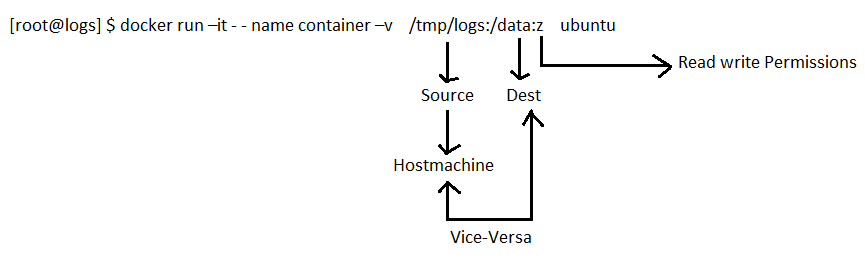
Creates volume vol2 along with the container

**Bind Mount:**

Access location from Hostfile to the container

root@pwd

tmp/logs 🡺 This is the location we have to share



**Migrating Docker Volumes: Check Once**

$ docker volume ls

$ docker volume inspect vol2

$ cd /var/lib/docker/volumes/vol2/\_data

[root@vol1\_data]# docker run –it - - name container –v vol2:/data ubuntu

It copies data from one container to another

$ docker run –it - - name container1 - - volumes – from container2 ubuntu

**Building Docker Images:**

Vi **Dockerfile**

FROM tomcat:latest

MAINTAINER Mahesh

COPY ./webapp.war /usr/local/tomcat/webapps

RUN apt-get-update

RUN apt-get install vim –y

CMD /bin/echo “Hello World”

$ docker build –t myimage .

Dockerfile \_\_\_\_\_\_\_\_\_build\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Docker images

**Docker Networks:**

$ docker network ls

Network ID Name Driver Scope

bridge 🡺 Default

host

none

$ docker network inspect bridge

Containers by default get into the **Bridge Network** and it will origins subnets in default bridge.

$ docker inspect container-id 🡺 Inspects container

**Ping:**

$ docker attach container

shell@]# ping 172.x.x.x.x 🡺 ip address

Containers with same bridge network will communicate with each other. Docker containers can communicate with each other through only ip address but not names.

**User Defined bridge Network In Docker:**

$ docker network create - - driver bridge bridge1

$ docker network ls

Network ID Name Driver Scope

Bridge1 bridge

bridge 🡺 Default bridge

host host

none null

If we want to run a container with our bridge network

$ docker run –itd - - name container - - network bridge1 ubuntu

In this containers can communicate with each other by name and also with the ip addressess

**Creating Own Networks:**

Docker network create newnet –subnet= 192.168.0.0/16

**Port Publishing:**

$ docker run –its - - name container –p 8080:80 nginx

**Removing Docker Network:**

$ docker network rm chdabcka

**Deploying Docker Containers via Jenkins:**

Dockerfile

**Job:**

/opt/docker

cd /home/dockeradmin;

docker build –t myimage .;

docker run –d - - name container –p 8080:8080 tomcat;

**Docker Swarm: Container Orchestration & Clustering Tool**

**Why Docker Swarm:**

* High availabilty of services
* Load Balancing
* Scaling the containers
* Rolling the updates

**Architecture:**

Manager Node 🡺 Manages

Worker Node 🡺 Load

Services

**Starting Docker Swarm:**

$ docker swarm init - - advertise – addr 10.x.x.x

|

Here ip address is which one we want as a Manager Node

We will get one link to add workers. Paste that command in worker nodes. It will join as workers.

$ docker node ls

$ docker swarm join-token worker 🡺 To retrieve the command

$ docker swarm join-token manager 🡺 Retrieve manager link

**Starting Services:**

$ docker service create –p 8000:80 - - name websvc nginx

$ docker service ls

$ docker service ps websvc

$ docker node update - - availabilty drain manager

$ docker node update - - availabilty active manager

**Scaling & Load balancing on Docker Swarm:**

$ docker service scale websvc=10

$ docker service ls

$ docker service ps websvc

$ docker ps

$ docker network ls

**ANSIBLE**

* useradd ansibleuser
* passwd ansibleuser
* visudo
* ansibleuser ALL=(ALL) NOPASSWD: ALL
* vi /etc/ssh/sshd\_config
* password Authentication yes
* service sshd reload
* ssh-keygen
* ssh-copy-id ssh [ansibleuser@10.x.x.x](mailto:ansibleuser@10.x.x.x)

Go to ansible location

$ cd /etc/ansible ls

ansible.cfg hosts roles

$ vi hosts

[allmyservers]

[ansibleuser1@10.x.x.x](mailto:ansibleuser1@10.x.x.x)

[ansibleuser2@10.x.x.x](mailto:ansibleuser2@10.x.x.x)

**Ping Command:**

$ ansible –m ping allmyservers

**Installing Application on Target Servers:**

$ ansible –m yum –a “name=docker state=present” allmyservers –b

| | |

Module Arguments Root

$ ansible –m service –a “name=docker state=started” allmyservers –b

**Ansible Playbooks:**

$ ansible-playbook myplaybook.yml - -syntax –check

$ ansible-playbook myplaybook.yml 🡺 To run the Playbook

**Ansible Roles:**

$ cd /etc/ansible/roles ls

roles]# ansible-galaxy init myrole

Role successfully created with the name as myrole

roles]# ls

myrole

$ cd myrole

myrole]# ls

**defaults files handlers meta README.md tasks templates tests vars**

**$ cd tasks**

$ vi main.yml

* name: Installing httpd service
* service: “name=httpd state=restarted”

**$ cd vars**

$ vi main.yml

* mypackages

-httpd

-git

-mysql

-docker

**$ vi myroleplaybook.yml**

* hosts: all

become: true

become\_true: root

roles:

**-myrole**

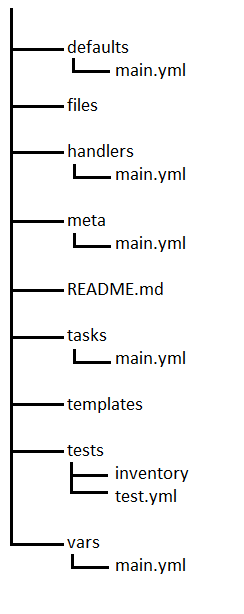
**To execute the above role**

ansible]# ansible-playbook myroleplaybook.tml

It will run the ansible playbooks with roles

$ yum install tree

myrole]# tree



8 Directories and 8 Files

**Sample Playbook:**

- hosts: allmyservers

become: true

become\_user: root

vars:

- mypackages:

- httpd

- git

- mysql

- docker

tasks:

- name: Installing httpd service

yum: "name={{mypackages}} state=present"

- name: starting httpd service

service: "name=httpd state=started"

- name: Deploying application

copy: "src=index.html dest=/var/www/html"

notify: "Restarting httpd service"

handlers:

- name: Restarting httpd service

service: "name=httpd state=restarted"

**ANSIBLE AR Shankar**

**What is Ansible:**

Radically simple open-source IT automation engine

**Ansible Automates:**

Configuration management

Provisioning

Application Deployment

Orchestration

**Why Ansible:**

Ansible uses YAML format and it is very simple and human readable. No special code skills requires.

Tasks executed in order

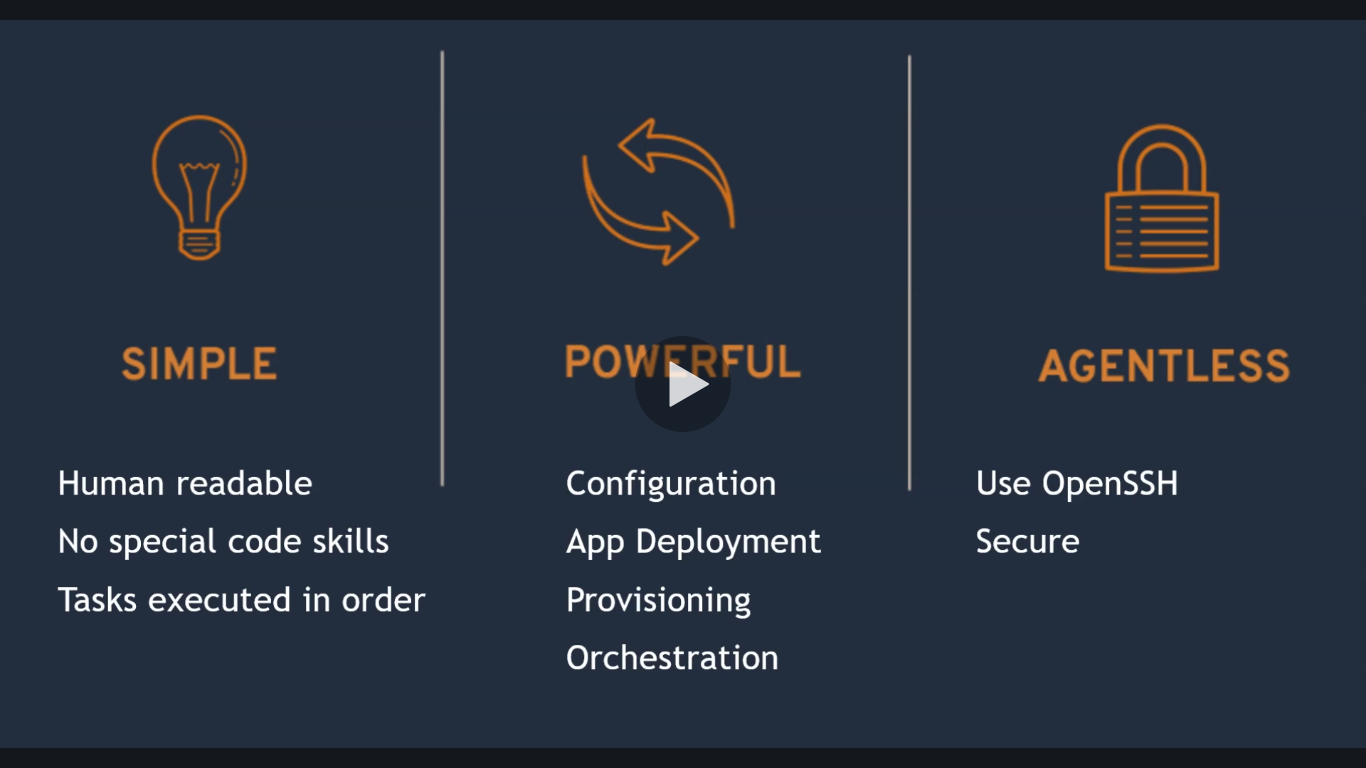
**Powerful:**

Configuration

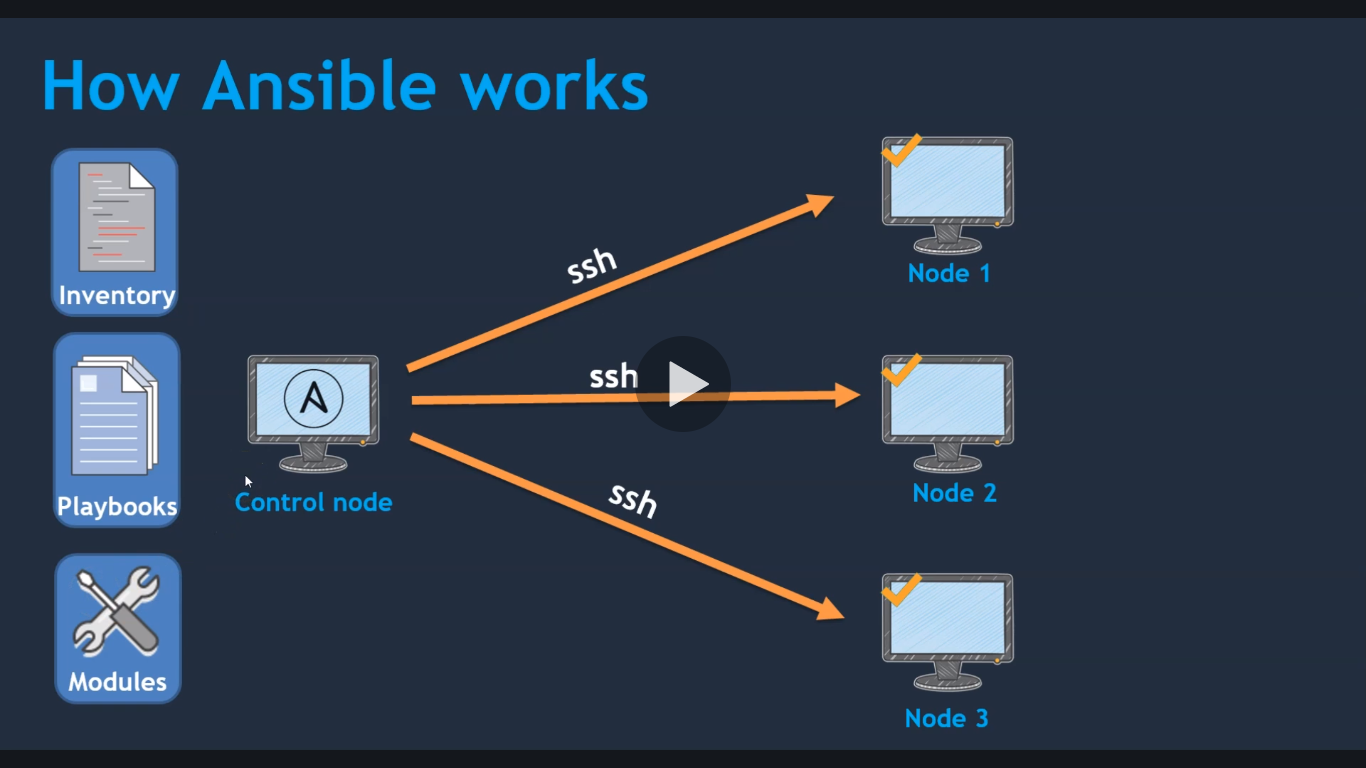
App Deployment

Provisioning

Orchestration



**Working of Ansible:**

**:**

**Ansible Terminology:**

**Control node:**

Any machine with Ansible installed

**Managed nodes:**

The network devices (servers) you manage with Ansible

**Inventory:**

A list of managed nodes. An inventory file is also sometimes called a “hostfile”.

**Modules:**

The unit of code Ansible executes. Each module has a partiicular functionality

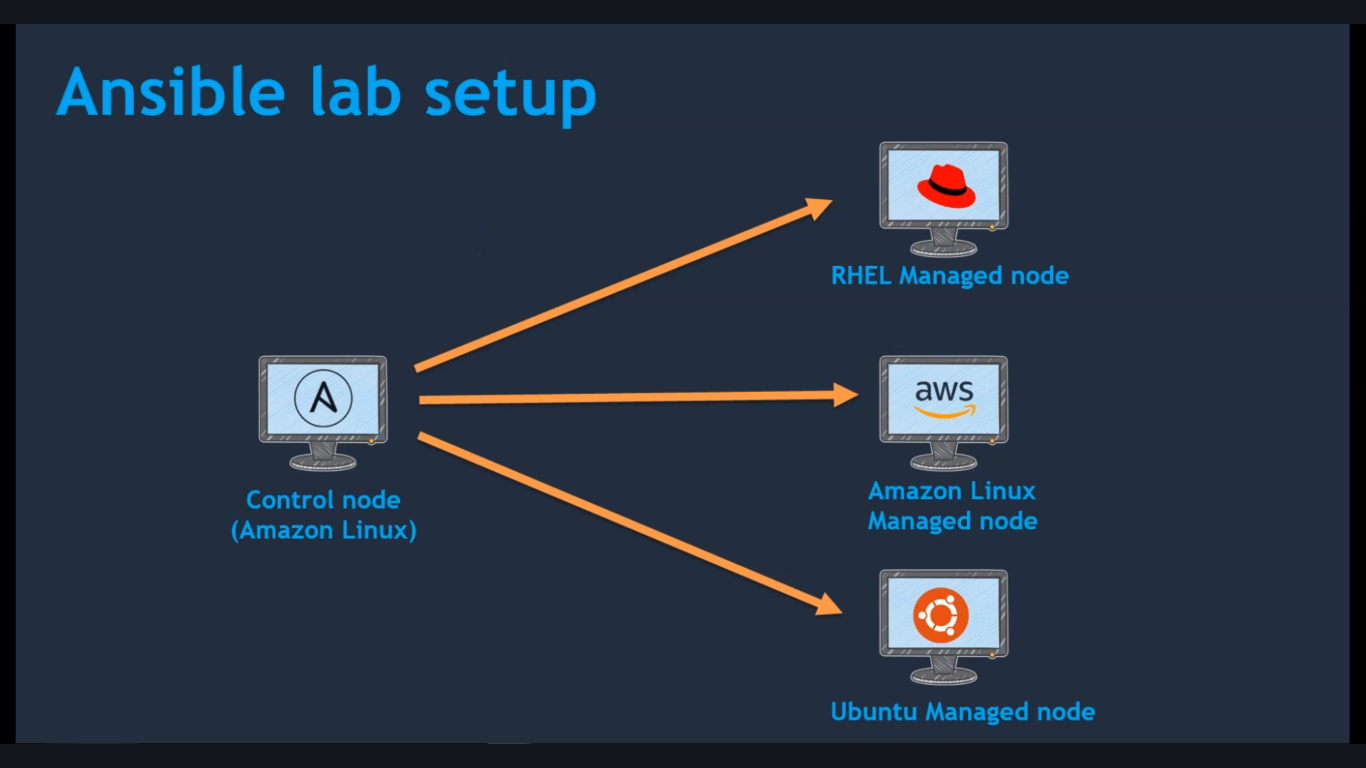
**Tasks:**

The units of action in Ansible

**Playbooks:**

Ordered list of tasks

**Ansible Lab Setup:**



* create an EC2 Instance
* hostname ansible-control-node
* vi /etc/hostname 🡺 ansible-control-node
* useradd ansadmin
* passwd ansadmin
* visudo
* ansadmin ALL=(ALL) NOPASSWD: ALL
* vi /etc/ssh/sshd\_config
* password authentication yes
* service sshd reload
* sudo su – ansadmin (Create keys under ansadmin user)
* ssh-keygen

**Ansible Installation and setup:**

### Pre-requisites

1. An AWS EC2 instance (on Control node)

### **Ansible Installation:**

#### On Amazon EC2 instance

1. Install python and python-pip

yum install python

yum install python-pip

1. Install ansible using pip check for version

pip install ansible

ansible --version

1. Create a user called ansadmin (on Control node and Managed host)

useradd ansadmin

passwd ansadmin

1. Below command grant sudo access to ansadmin user. But we strongly recommended using "visudo" command if you are aware vi or nano editor. (on Control node and Managed host)

echo "ansadmin ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

1. Log in as a ansadmin user on master and generate ssh key (on Control node)

sudo su - ansadmin

ssh-keygen

1. Copy keys onto all ansible managed hosts (on Control node)

ssh-copy-id ansadmin@<target-server>

1. Ansible server used to create images and store on docker registry. Hence install docker, start docker services and add ansadmin to the docker group.

yum install docker

# start docker services

service docker start

# add user to docker group

usermod -aG docker ansadmin

1. Create a directory /etc/ansible and create an inventory file called "hosts" add control node and managed hosts IP addresses to it.

### Validation Test:

1. Run ansible command as ansadmin user it should be successful (Master)

ansible all -m ping

**Ansible ad-hoc commands:**

An Ansible ad-hoc command uses the /usr/bin/ansible command-line tool to automate a single task on one or more managed nodes. Ad-hoc commands are quick and easy, but they are not reusable.

**Ping Module:**

$ ansible all –m ping

**Command Module:**

$ ansible all –m command –a “uptime”

$ ansible all –m command –a “date”

$ ansible all –m command –a “who”

**Stat Module:**

$ ansible all –m stat –a “path=/etc/hosts” 🡺 Checks file exists or not

**Yum Module:**

$ ansible all –m yum –a “name=git” –b

**User Modules: (Creating users by using user modules)**

$ ansible all –m user –a “name=Mahesh” –b

**Setup Module:**

$ ansible all –m setup 🡺 It gives entire system information about the managed nodes

**Ansible Inventory:**

Ansible works against multiple managed nodes or “hosts” in your infrastructure at the same time, using a list or group of lists known as Inventory.

Inventory file is a collection of hosts(nodes) which are managed by ansible control node.

Hosts information can be defined in following ways:

* Default Location: **/etc/ansible/hosts**
* Use –I option: ansible – I **my\_hosts**
* Defined in **ansible.cfg** file

**Ping our own hosts file:**

Create hosts file in your desired location.

After that you can ping our own host using the below command.

$ ansible all –m ping – i hosts

**\*\* If we want the our own hosts as a default.**

$ vi ansible.cfg 🡺

Under [defaults]

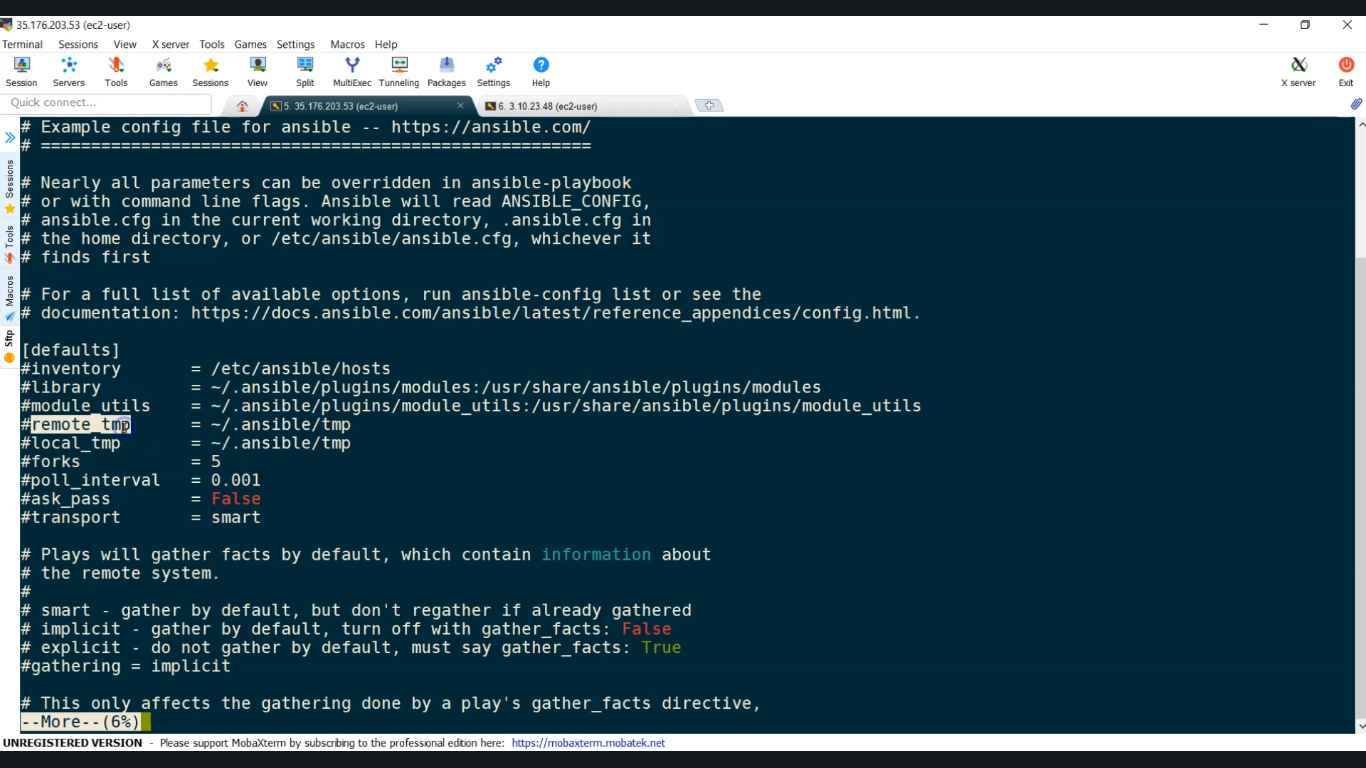
Inventory = /home/ansadmin/hosts

So that ansible will selects the default hosts location which you have specified in the **ansible.cfg**

**To ping only particular group:**

ansible **rhel** –m ping – i /etc/ansible/hosts

**If you don’t specify any privilages, Ansible will selects the default configuration which has been stored in ansible.cfg**



**\*\* In the command line we will use –b as a root user. To avoid this we can do as follow**

$ sudo vi /etc/ansible/ansible.cfg

Under [privilage escalation]

become = True

Now we can use any commands without **–b**

**\*\* Also, we can create our own ansible.cfg in ansadmin, we can create ansible.cfg own file as follow**

$ vi ansible.cfg

[Privilage\_escalation]

Become = True

Through this we can run the command without **-b**

# Ansible Configuration Settings

Ansible supports several sources for configuring its behavior, including an ini file named ansible.cfg, environment variables, command-line options, playbook keywords, and variables. See [Controlling how Ansible behaves: precedence rules](https://docs.ansible.com/ansible/latest/reference_appendices/general_precedence.html#general-precedence-rules) for details on the relative precedence of each source.

The ansible-config utility allows users to see all the configuration settings available, their defaults, how to set them and where their current value comes from. See [ansible-config](https://docs.ansible.com/ansible/latest/cli/ansible-config.html#ansible-config) for more information.

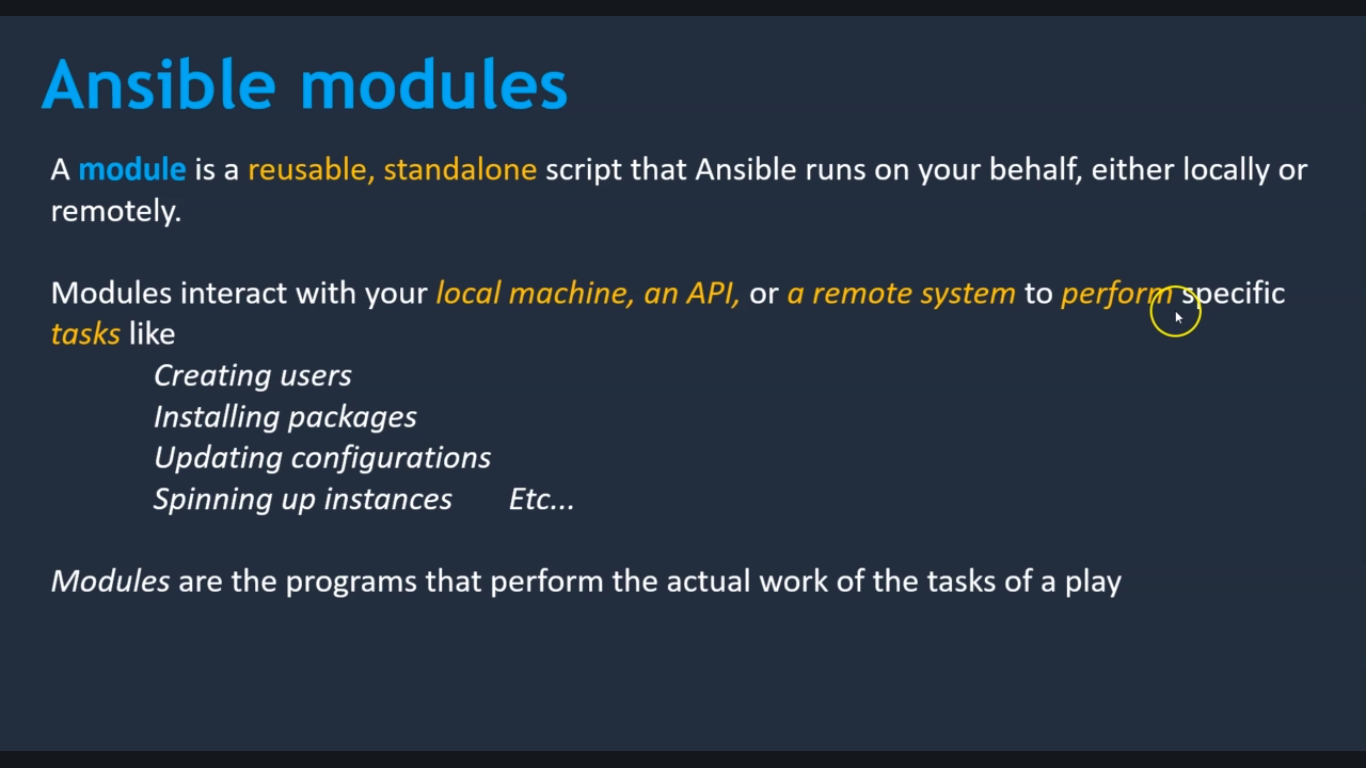
## The configuration file

Changes can be made and used in a configuration file which will be searched for in the following order:

* ANSIBLE\_CONFIG (environment variable if set)
* ansible.cfg (in the current directory)
* ~/.ansible.cfg (in the home directory)
* /etc/ansible/ansible.cfg

Ansible will process the above list and use the first file found, all others are ignored.

**Ansible Modules:**



# Working with Modules

* [Introduction to modules](https://docs.ansible.com/ansible/latest/user_guide/modules_intro.html)
* [Return Values](https://docs.ansible.com/ansible/latest/reference_appendices/common_return_values.html)
* [Module Maintenance & Support](https://docs.ansible.com/ansible/latest/user_guide/modules_support.html)
* [Module Index](https://docs.ansible.com/ansible/latest/modules/modules_by_category.html)

Ansible ships with a number of modules (called the ‘module library’) that can be executed directly on remote hosts or through [Playbooks](https://docs.ansible.com/ansible/latest/user_guide/playbooks.html#working-with-playbooks).

Users can also write their own modules. These modules can control system resources, like services, packages, or files (anything really), or handle executing system commands.

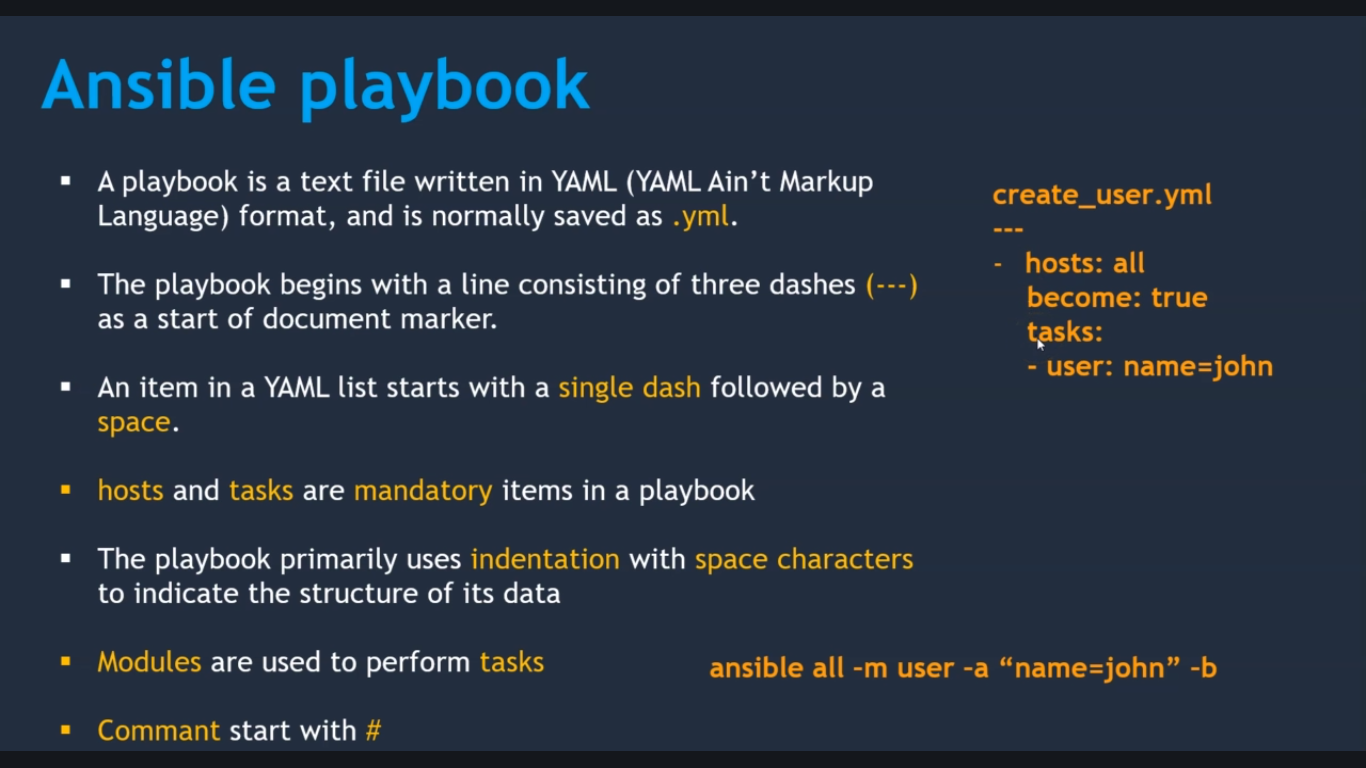
**$ ansible-doc –l 🡺 shows the list of the modules**

**Creating Own Ansible Playbook:**

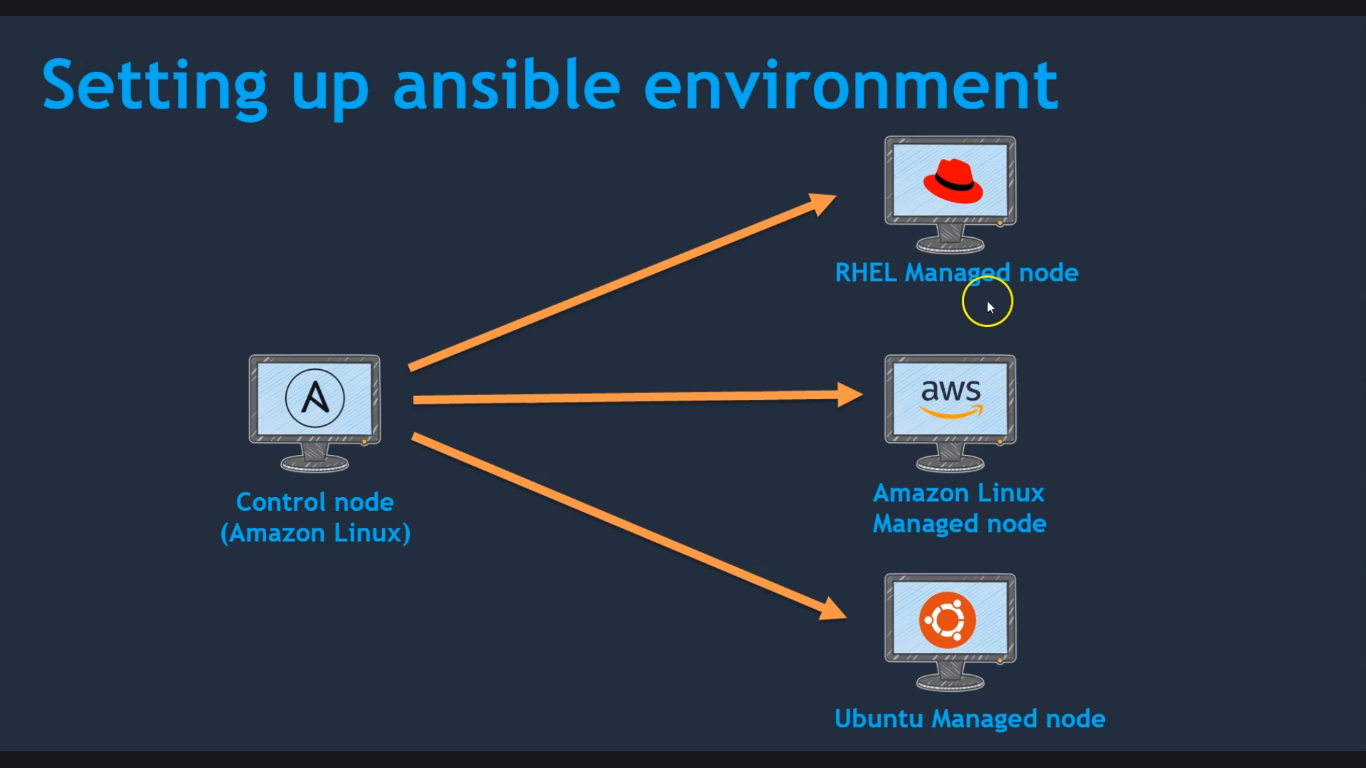
What is a Playbook?

Playbooks are essentially sets of instructions (plays) that you send to run on a single target or groups of targets (hosts). Think about the instructions you get for assembling an appliance or furniture. The manufacturer includes instructions so you can put the parts together in the correct order. When followed in order, the furniture looks like what was purchased.

**Ansible Playbooks Format:**



**Setting Up Ansible Environment:**



**Running A Playbook:**

**ls**

ansible.cfg hosts ansible\_user.yml

**Note: In unubtu system, it wont create home directory. That’s why while creating use the below command**

**$** useradd ansadmin -m –d /home/ansadmin ansadmin

Why Home directory: Bcz the keys will copy into the home directory.

**First Ansible Playbook:**

vi **first\_ansibleplaybook.yml**

---

- name: this playbook is to create a user

hosts: all

become: true

tasks:

- name: creating user john

user: name=john

(OR)

- name: creating user john

user:

- name: john

**Yum Module: (Installing Package):**

**vi yum\_module\_package.yml**

---

- name: this playbook install packages

hosts: **webservers**

become: true

tasks:

- name: install apckage

yum:

name: git

state: installed

**File Module – create/remove a file/directory:**

vi **create\_file.yml**

---

- name: this playbook creates a file or dir

hosts: all

become: true

tasks:

- name: creating a file

file:

path: /home/ansadmin/demofile

state: touch

Above file create the file in all the servers.

**Note**: Ansible command to list hosts

$ ansible all – i hosts - - list-hosts

**To Create a Directory:**

vi **create\_directory.yml**

- - -

- name: this playbook creates a file or dir

hosts: all

become: true

tasks:

- name: creating a file

file:

path: /home/ansadmin/dir1

state: directory

**To Remove The Above Directory:**

file:

path: /home/ansadmin/dir1

state: absent

**Copy Module:**

vi **index.html**

<h1!> This file should copy to the target servers

vi **copy\_file.yml**

---

- name: ansible playbook to copy a file

hosts: all

become: true

tasks:

- name: copy a file

copy:

src: /opt/ansible/index.html

dest: /home/ansadmin

mode: 0600

owner: john

**Install Apache on RHEL:**

**vi install\_httpd.yml**

---

- name: this playbook install httpd

hosts: webservers

become: true

tasks:

- name: install apckage

yum:

name: httpd

state: installed

- name: start httpd service

service:

name: httpd

state: started

**Removing the packages:**

**vi uninstall\_httpd.yml**

---

- name: this playbook uninstall httpd

hosts: webservers

become: true

tasks:

- name: stop httpd service

service:

name: httpd

state: stopped

- name: remove httpd service

yum:

name: httpd

state: removed

**Install Aache on Ubuntu:**

In ubuntu yum command will not work. Instead of **yum** command we will use **apt install**

**$ apt install git**

**$ apt install apache2**

**$ service apache2 start**

vi **install\_apche2.yml**

---

- name: install apache2 on ubuntu servers

hosts: dbservers

become: true

tasks:

- name: install apache2

apt:

name: apache2

state: present

- name: start apache2

service:

name: apache2

state: started

**Notify & Handlers In A Playbook:**

**vi notify\_handlers.yml**

---

- name: this playbook install httpd

hosts: webservers

become: true

tasks:

- name: install apckage

yum:

name: httpd

state: installed

notify: start httpd service

handlers:

- name: start httpd service

service:

name: httpd

state: started

**Note:** Notify and handlers names should be same as mentioned below

notify: start httpd service

handlers:

- name: start httpd service

**How Gathering Facts Works:**

Gathering facts will retrieve the sytem information from the remote system.

If we want to disable gathering facts, we will use **gather\_facts: no**

- - -

- name: this playbook creates a file or dir

hosts: all

become: true

**gather\_facts: no**

tasks:

- name: creating a file

file:

path: /home/ansadmin/dir1

state: directory

Why we disable **gathering facts**:

In real time it may 100 of servers, it will take longer time to gather all the facts.

**When Condition:**

vi **install\_httpd\_apache2.yml**

---

- name: this playbook install httpd and apache2 by using when condition

hosts: all

become: true

tasks:

- name: install package

yum:

name: httpd

state: installed

when: ansible\_os\_family == “Redhat”

- name: start apache

service:

name: httpd

state: started

when: ansible\_os\_family == “Redhat”

- name: install apache2

apt:

name: apache2

state: present

when: ansible\_os\_family == “Debian”

- name: start apache2

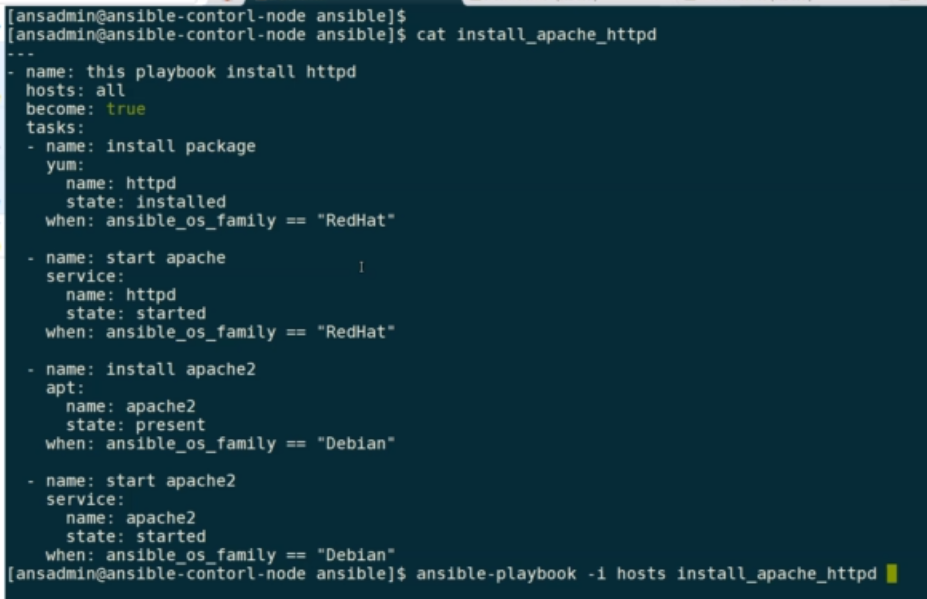
service:

name: apache2

state: started

when: ansible\_os\_family == “Debian”

**Note**: Below fig shows the above programming



**Uninstall Apache using When Condition:**

vi **unistall\_httpd\_using\_when\_condition.yml**

---

- name: this playbook uninstall httpd

hosts: webservers

become: true

tasks:

- name: stop httpd service

service:

name: httpd

state: stopped

when: ansible\_os\_family == “Redhat”

- name: remove httpd service

yum:

name: httpd

state: removed

when: ansible\_os\_family == “Redhat”

- name: stop apache2 services

service:

name: apache2

state: stopped

when: ansible\_os\_family == “Debian”

- name: uninstall apache2

apt:

name: apache2

state: removed

when: ansible\_os\_family == “Debian”

Above Programming Fig:



**Adding Copy Task To Apache Playbook:**

vi **copy\_indexfile\_task\_to\_apache\_playbook.yml**

<h1> Welcome to Apache tomcat </h1>

---

- name: this playbook install httpd and apache2 by using when condition

hosts: all

become: true

tasks:

- name: install package

yum:

name: httpd

state: installed

when: ansible\_os\_family == “Redhat”

- name: start apache

service:

name: httpd

state: started

when: ansible\_os\_family == “Redhat”

- name: install apache2

apt:

name: apache2

state: present

when: ansible\_os\_family == “Debian”

- name: start apache2

service:

name: apache2

state: started

when: ansible\_os\_family == “Debian”

- name: copy index.html

copy:

src: /opt/ansible/index.html

dest: /var/www/html

mode: 0666

**Lists & With\_items:**

Vi **lists\_with\_items.yml**

---

- name: this playbook installs so many packages

hosts: all

become: true

tasks:

- name: install packages

yum:

name: ['git', 'docker', 'make', 'gcc', 'wget', 'telnet', 'gzip'] While using Lists

name: "{{ item }}" While using With\_items. Now in latest version is not used

state: installed

with\_items:

- git

- make

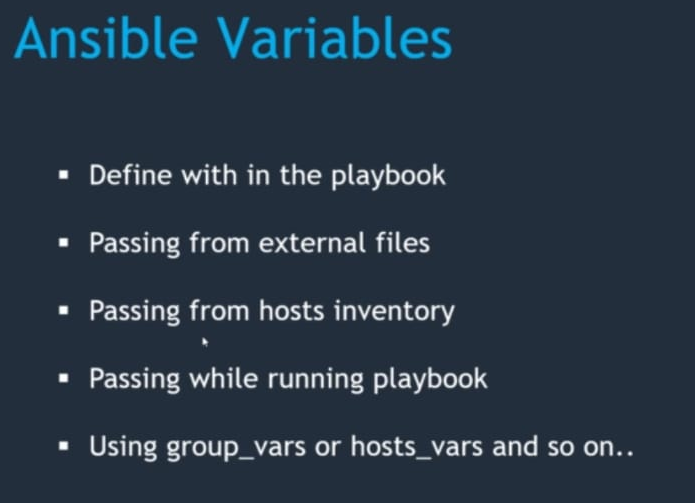
- gcc

- docker

- gzip

- wget

**Ansible Variables:**

****