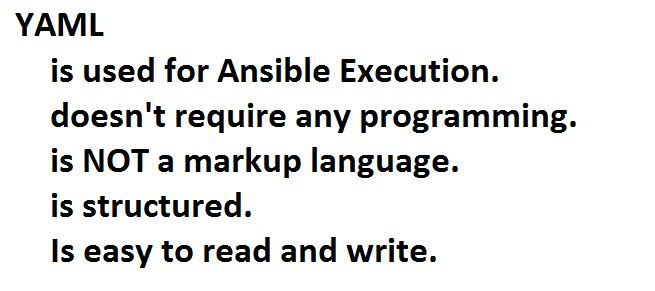
**Ansible**

**Ansible** is simple open source IT engine which automates application deployment, intra service orchestration, cloud provisioning and many other IT tools.

Ansible is easy to deploy because it does not use any agents or custom security infrastructure.

Ansible uses playbook to describe automation jobs, and playbook uses very simple language i.e. **YAML** (It’s a human-readable data serialization language & is commonly used for configuration files but could be used in many applications where data is being stored) which is very easy for humans to understand, read and write. Hence the advantage is that even the IT infrastructure support guys can read and understand the playbook and debug if needed (YAML – It is in human readable form).



Ansible is designed for multi-tier deployment. Ansible does not manage one system at time, it models IT infrastructure by describing all of your systems are interrelated. Ansible is completely agentless which means Ansible works by connecting your nodes through ssh(by default). But if you want other method for connection like Kerberos, Ansible gives that option to you.

After connecting to your nodes, Ansible pushes small programs called as “Ansible Modules”. Ansible runs that modules on your nodes and removes them when finished. Ansible manages your inventory in simple text files (These are the hosts file). Ansible uses the hosts file where one can group the hosts and can control the actions on a specific group in the playbooks.

What is Configuration Management

Configuration management in terms of Ansible means that it maintains configuration of the product performance by keeping a record and updating detailed information which describes an enterprise’s hardware and software.

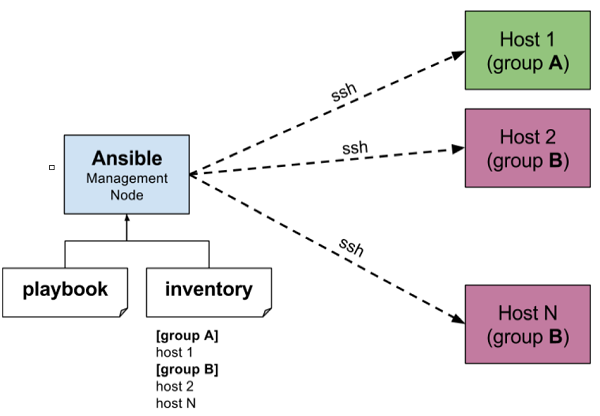
Such information typically includes the exact versions and updates that have been applied to installed software packages and the locations and network addresses of hardware devices. For e.g. If you want to install the new version of **WebLogic/WebSphere** server on all of the machines present in your enterprise, it is not feasible for you to manually go and update each and every machine.

You can install WebLogic/WebSphere in one go on all of your machines with Ansible playbooks and inventory written in the simplest way. All you have to do is list out the IP addresses of your nodes in the inventory and write a playbook to install WebLogic/WebSphere. Run the playbook from your control machine & it will be installed on all your nodes.

How Ansible Works?

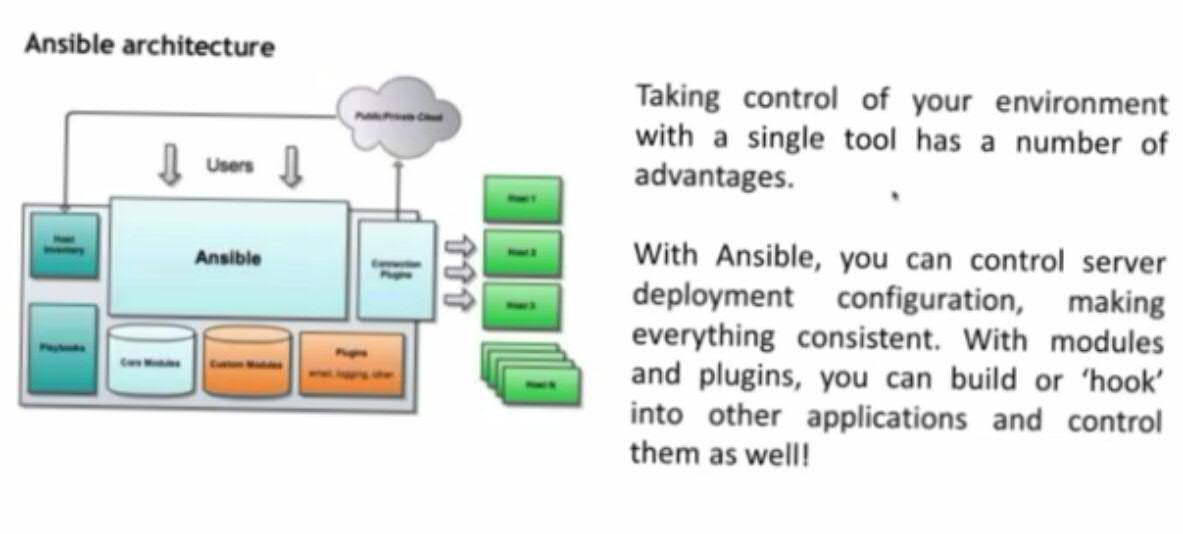
The picture given below shows the working of Ansible.

**Ansible works** by connecting to your nodes and pushing out small programs, called "**Ansible** **modules**" to them. **Ansible** then executes these modules (over SSH by default) and removes them when finished. Your library of modules can reside on any machine, and there are no servers, daemons, or databases required.



**Inventory:** Which maintains the list of nodes which we want to manage from Ansible.

**Playbook:** It consists of plays. Each play is a statement, like command line actions.

The management node in the above picture is the controlling node (managing node) which controls the entire execution of the playbook. It’s the node from which you are running the installation. The inventory file provides the list of hosts where the Ansible modules needs to be run and the management node does a SSH connection and executes the small modules on the hosts machine and installs the product/software.

**Beauty** of Ansible is that it removes the modules once those are installed so effectively it connects to host machine, executes the instructions and if it’s successfully installed removes the code which was copied on the host machine which was executed.

Ansible internally uses python.

## Installation Process

Mainly, there are two types of machines when we talk about deployment −

* **Control machine** − Machine from where we can manage other machines.
* **Remote machine** − Machines which are handled/controlled by control machine.

There can be multiple remote machines which are handled by one control machine. So, for managing remote machines we have to install Ansible on control machine.

### **Control Machine Requirements**

Ansible can be run from any machine with Python 2 (versions 2.6 or 2.7) or Python 3 (versions 3.5 and higher) installed.

**Note** − Windows does not support control machine.

By default, Ansible uses **ssh** to manage remote machine.

Ansible does not add any database. It does not require any daemons to start or keep it running. While managing remote machines, Ansible **does not** leave any software installed or running on them. Hence, there is no question of how to upgrade it when moving to a new version.

Ansible can be installed on control machine which have above mentioned requirements in different ways. You can install the latest release through Apt, yum, pkg, pip, OpenCSW, pacman, etc.

### **Installation through Apt on Ubuntu Machine**

For installing Ansible you have to configure PPA on your machine. For this, you have to run the following line of code −

$ sudo apt-get update

$ sudo apt-get install software-properties-common

$ sudo apt-add-repository ppa:ansible/ansible $ sudo apt-get update

$ sudo apt-get install ansible

After running the above line of code, you are ready to manage remote machines through Ansible. Just run ansible --version to check the version and just to check whether Ansible was installed properly or not.

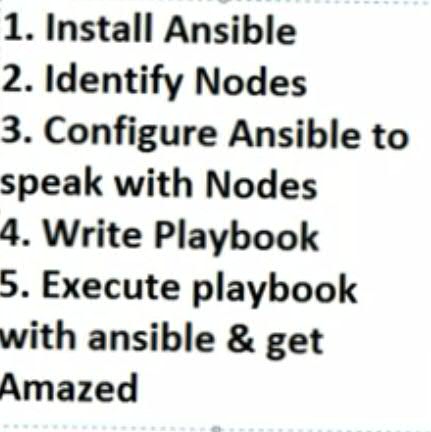
**Ansible is idempotent:** How many times you run the script, it gives the same result.  
  
Ansible and Salt are push based were Chef and Puppet are pull based.

Ansible is from RedHat.

When using Ansible, Node can be of any OS, but Master should be Linux OS of any Flavor.

Ansible server can be any normal machine, no needs to be a high configuration OS.

**Ansible workflow:**



Steps 4 and 5 can be replaced with **Adhoc** commands.

**Another way of installing Ansible (RehHat)**

Create Redhat7 machine (Master) on EC2 instance, then switch as a root user (**sudo -i**) and then run **yum update**

Then run, **yum install wget** and then say yes to confirm.

Then run, **wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm**

Check whether it is downloaded or not by using **ls** command.

Then run, **rpm -i epel-release-latest-7.noarch.rpm**

Then run, **yum install ansible -y**

So Ansible is installed and it called manager or controller machine. To verify, just verify by version **ansible --version**

Create a user with any name (ex: ansible) and the same user should exist in all nodes and needs permissions to execute.

Then run, **adduser ansible**

Then run**, passwd ansible** and then give password

Give sudo access to created ansible user

Then type, **visudo**

Then add below line after **root** user to stop allowing for password the ansible user,

**root** **ALL=(ALL) ALL**  
**ansible ALL=(ALL) NOPASSWD: ALL**

Then go to: **vi /etc/ssh/sshd\_config**

Uncomment **PasswordAuthentication yes**

And comment **PasswordAuthentication no**

Then**: service sshd restart**

Then come out of root user with **exit** command and switch to **ansible** user using **su ansible** command.  
Then generate ssh key using: **ssh-keygen**

**This key needs to be copied from all the Nodes master**. But here for testing, we are copying to local machine itself.Then **ssh-copy-id localhost**

**ssh-copy-id localhost** to be executed on master node only with each worker node public ipaddress or hostname

**Ex: ssh-copy-id workernodehostname**  
Then give the ansible user password.

Then try to connect to local machine without password: **ssh localhost**If it is success, then you no need of password connection with that user.

Then **exit** from it.

Then go to the inventory file: **cd /etc/ansible/** which has **ansible.config,** **hosts** and **roles**.  
Here, **ansible.config** is the main configuration file of Ansible.

Then investigate the contents of hosts**, less hosts**

Rename the original file for backup (use root user).

Then create a simple host file, **vi hosts** and write **localhost**, to check the connection for same machine.  
Then run this command with **ansible** user**, ansible -m ping all,** then confirm yes for authenticity.

Now we are ready with Ansible master.

In AWS, we can see public and private IPAddress. Private is used to communicate when both machines are in same network.

Public is used to communicate when machines are in separate networks.

After every restart of instance, we will get different public IPs.   
As our Ansible master and Nodes are under same network, we use private IP to communicate between Ansible master and Nodes.

Private DNS is the machine name(hostname). You can also check the hostname using Linux command **hostname -f**

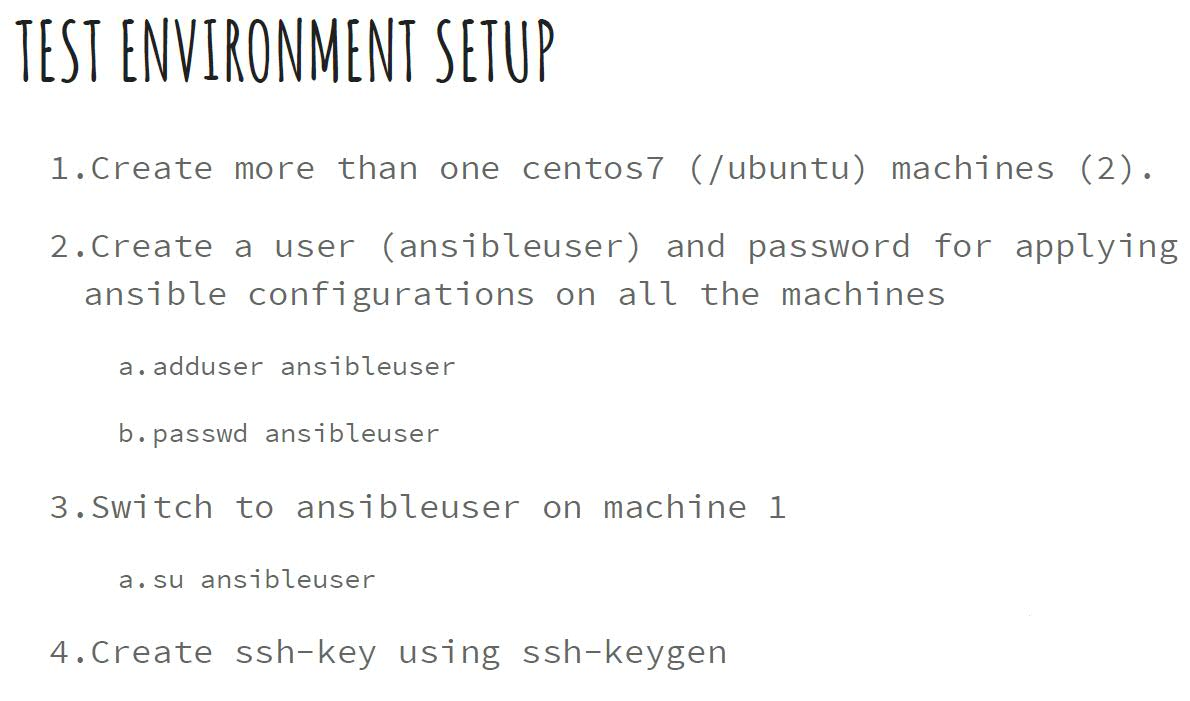
**Now try to install Nodes**

Create more that 1 ubuntu EC2 instances and switch as a root user.

To communicate between Ansible Master and Nodes, both master and nodes should work with ping, so they should stay in same network.

Node should have the same **ansible** user to communicate from master to Node.

Repeat below steps on each instance.



5. Copy the ssh-key to amchine2 using ssh-copy-id and repeat the steps 4,5 by loginas ansibleuser on machine2.

a. Test the steps 4,5 using ssh <machine\_host\_name/ipaddress> should not ask any password.

Run, **apt-get update**

Then run, **adduser ansible**

Then run**, passwd ansible** and then give password

Then type, **visudo**

Then add below line after root user to stop allowing for password the ansible user,

**root** **ALL=(ALL) ALL**  
**ansible ALL=(ALL) NOPASSWD: ALL**

Then go to: **vi /etc/ssh/sshd\_config**

Uncomment **PasswordAuthentication yes**

And comment **PasswordAuthentication no**

Then**: service sshd restart**

Then copy worker node ssh keys to master.Then **ssh-copy-id workernodeip**  
Then give the ansible user password.

Then try to connect to local machine without password: **ssh workernodeip**If it is success, then you no need of password connection with that user.

Now, go to Master, edit hosts using **vi hosts**., as root user. Then add the new machine name like below:  
**localhost**

**ip-172-------.compute.internal**Then save the file

Now switch as ansible user: **su ansible**

Now copy the ssh key of master to Node.  
**ssh-copy-id nodename**

Ex: **ssh-copy-id ip-172-------.compute.internal**

Now check, whether you are able to login from master to Node or not using:

**ssh ip-172-------.compute.internal**

Then **exit** from node.  
Then try, **ansible -m ping all**

Now, you will be able to communicate to all the nodes configured in inventory file.

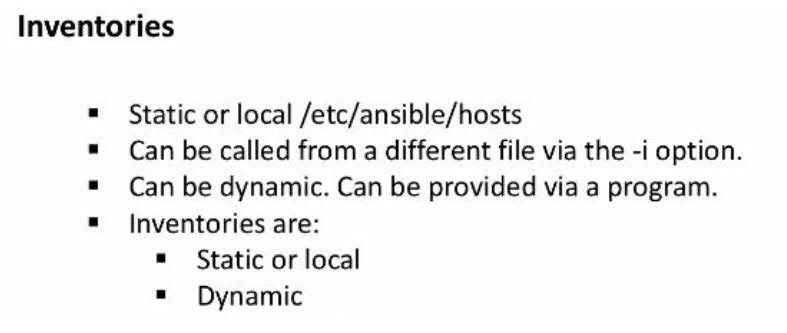
**ansible -m ping all,** this way of executing Ansible is called Adhoc command. This can be achieved using YAM file also. Adhoc commands are used when there is not repetitive work. We go for Adhoc commands when the task is not repetitive.

**Core components of Ansible:**

**Core components of ansible**:

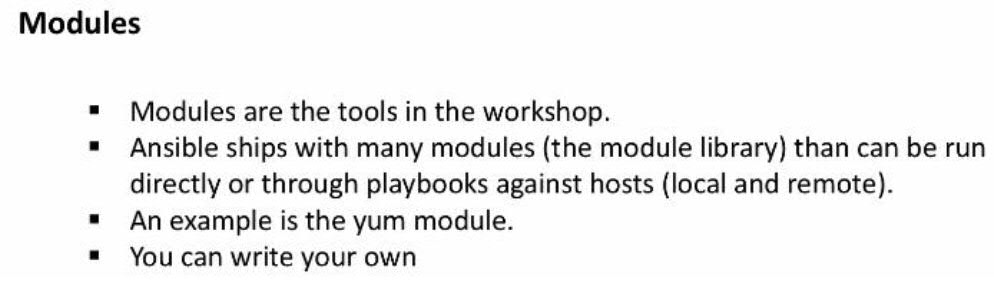


**Inventories:**



To maintain list of Nodes. Hardcoding the Node names is called static inventory. We have dynamic inventories too. We can write dynamic inventory in shell or python which returns in json format.

**Modules:**



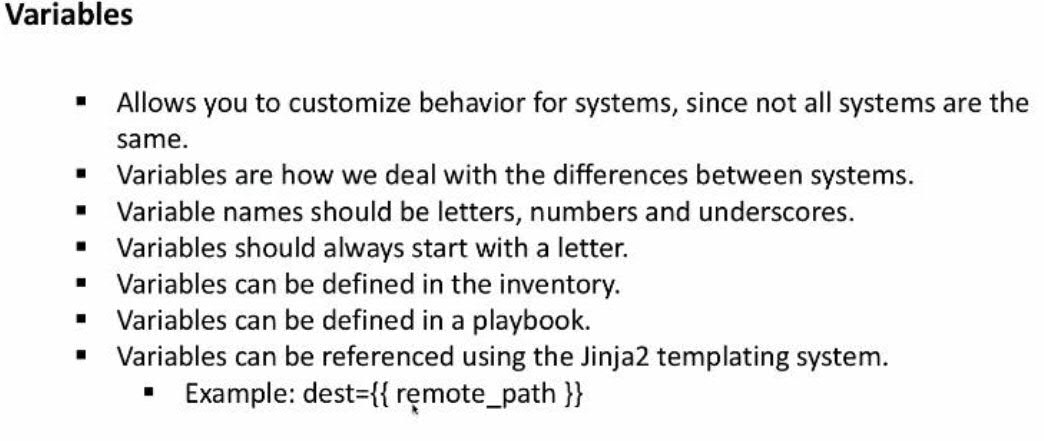
**Ex: ansible -m ping**

For each individual activity which we do, we can find an equivalent ansible module for it.

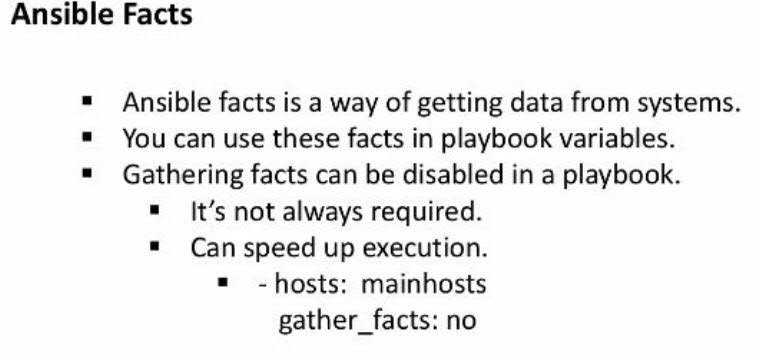
Here -m stands for module.  
We write these modules in YML file.

If any module is not found, then we have as **bash** module to execute bash script Or we can create our own module using python. And these modules are called custom modules.

**Variables:**

Used to declare some values in YML file. These are same like other programming languages.

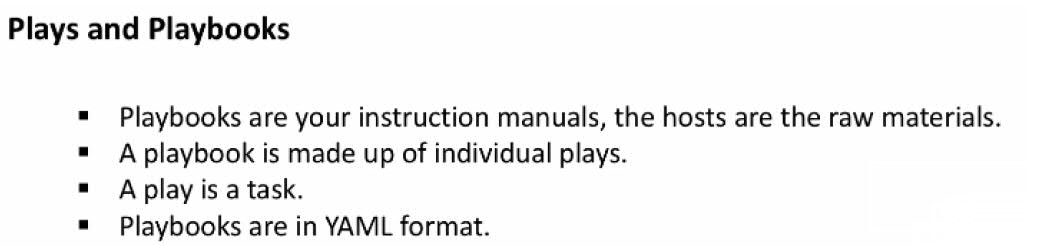
**Ansible facts:**



Tries to pull some information of the nodes like node name, ip, available memory etc.  
Those pulled values can be used as variables in subsequent statements in YML file.

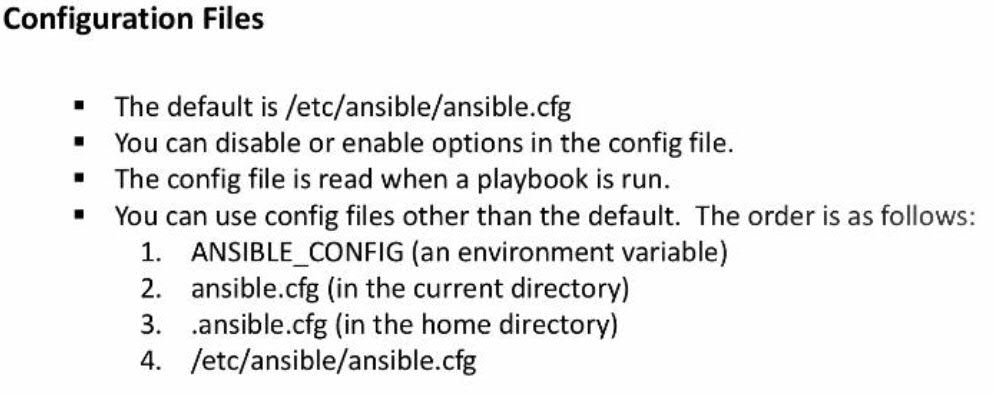
By default, facts are enabled. But we can disable facts so that Ansible will be faster in performance.

**Plays and Playbooks:**

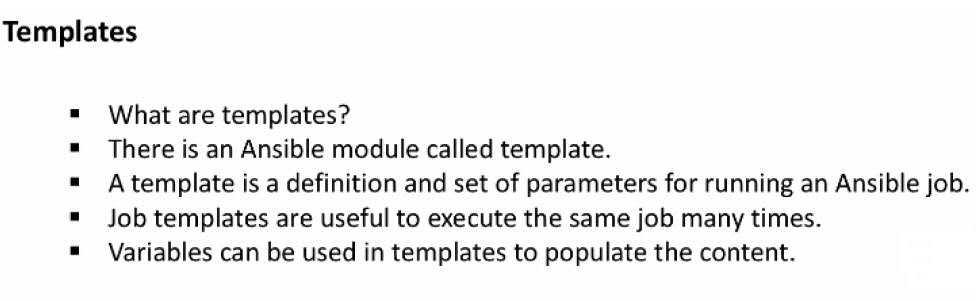


Playbook is collection of plays. Each play is a task. Play is like a module we write, and it can be re-used. Playbook is a YAML file

**Configuration files:**

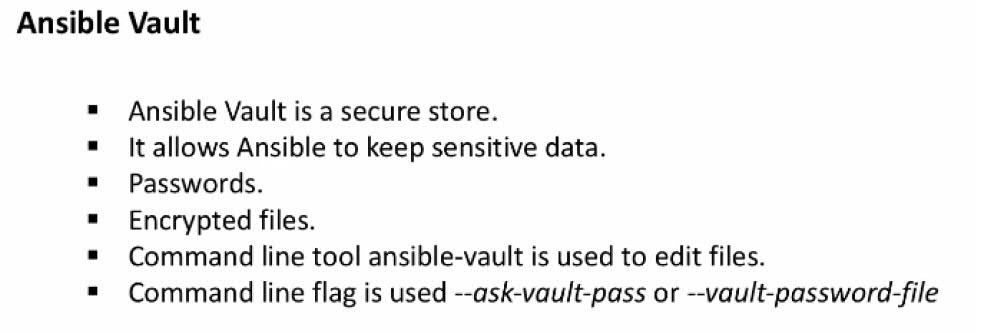


It has 4 ways to get the configuration and it has some order.  
  
**Templates:**

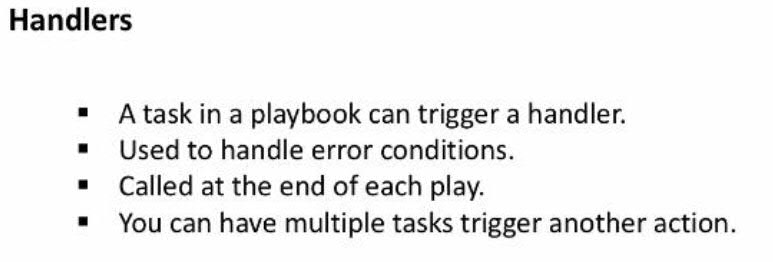


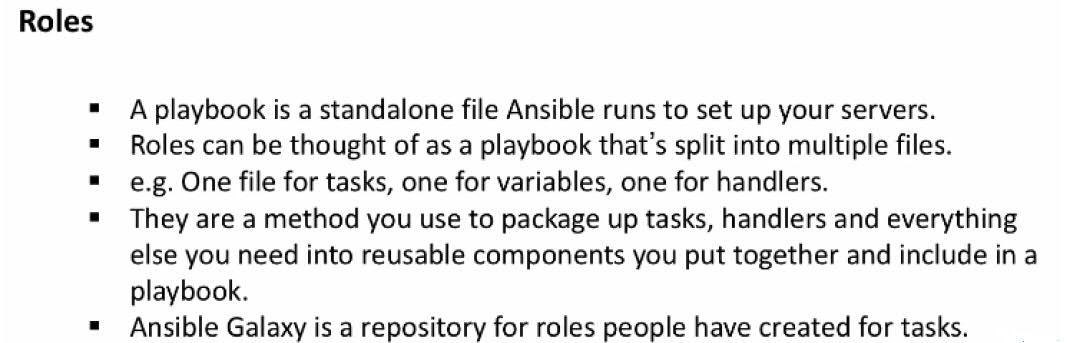
Here we are speaking about Jinja templates.

**Ansible Vault:**



Whenever we are doing environments, then we need to deal with passwords.  
Vault we store passwords. Those passwords will be stored in encrypted format and those can be given by decrypting. We use Vault for securing passwords.  
  
JSON and YML came into picture to solve the problems of XML. SML is good to understand by machine but not by humans. XML consumes more bandwidth than Json.





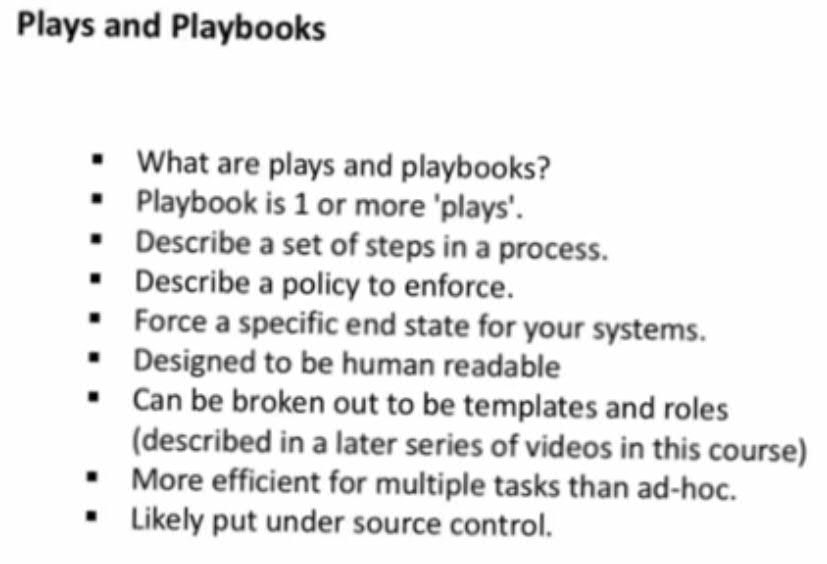
**YAML basics:**

<https://docs.ansible.com/ansible/latest/reference_appendices/YAMLSyntax.html> ---> **Refer this site for ansible basics.**

**Play and Playbooks:**

This is the file where we write the Ansible automation.

Playbook is about writing the stuff in the format of a file, so that we can repeat it as many times as we want, and can run by anybody.  
In inventory we put the host machines information, and we write the tasks in playbook which are going to execute on host machines machined in inventory.  
We tell Ansible what to do, so ansible will take care of how to do it.



Roles makes playbooks reusable.

Create a file with any name(ping.yml) and write below playbook text.

**Sample play book:**

---

- hosts: all

tasks:

#ansible all -m ping

* name: This task will ping

ping:

Here, **hosts: all** indicates to run this playbook file on all the hosts defined in inventory file.

In tasks we write individual tasks. Means child’s to tasks.

**#ansible all -m ping**, indicates comment.  
**- name** indicates the name of the task.  
  
To find the suitable module for **ping** command, search in Ansible document with the word **ansible module ping**

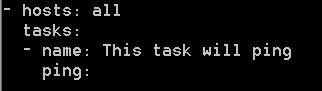
ping module will not have any arguments.

**ping:** indicates the module**.** It is ansible module replacement of ansible all -m ping Linux command.

Add the hostname in inventory file.  
Ex: cd /etc/ansible/hosts  
Mention the hostnames in this host file. Ex:

localhost

Ip-172--------------compute.internal

  
To run any playbook, we use the command: ansible-playbook playbookName.  
  
We no need to give the inventory file name as it is default location. If the inventory file is in different location, then we give -i and the path.

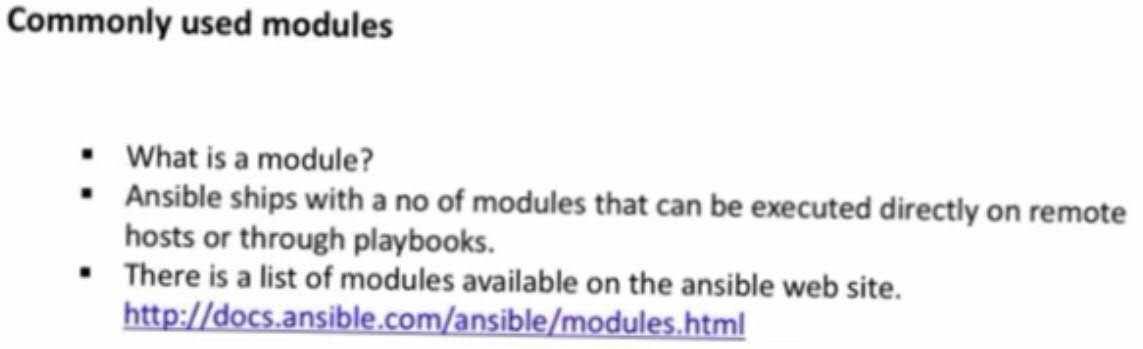
Run the above created playbook using this command: **ansible-playbook ping.yml**

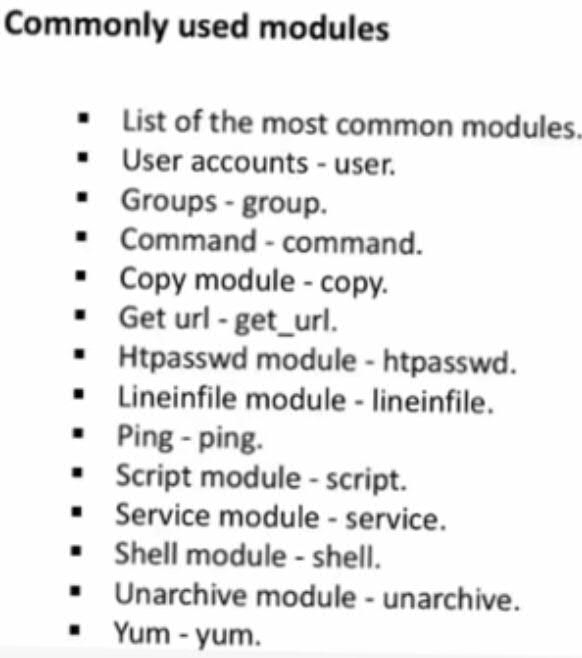
In the console you can see the **ok** status for each host mentioned in the inventory file.

|  |
| --- |
|  |
|  |
| [$1F1BE94888956A2C.jpg] |
|  |  |
| [$5366AE219A81C9F4.jpg] | 12:41:56 AM |

Idempotent mean, as many times we run, it gives the same result.

If we don’t mention, what user Ansible has to use to run the playbooks, then it uses the logged in user to execute the playbooks.





get\_url is for wget.  
Two modules, Command and Shell are slightly different to other modules.  
These two gives an option to execute Linux statements. Ex: ls -la  
Shell also does the same thing, but we can pass Script to it.  
  
**Command module example:**

Search in google for command module as: ansible command module.  
This has arguments and those will be listed in a table, and command module has few required fields.  
For Ansible adhoc commands you can search in google: **ansible adhoc commands**

Try, **ansible all -m command -a “whoami”**

Here, **all** says to execute this command on all the inventory hosts.  
**-a “whoami”** is the command to execute on inventories.  
  
Create a file with name whoami.yaml  
- hosts: all

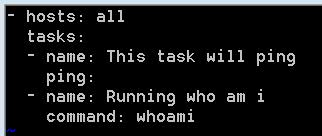
tasks:

* name: Find who is xecuting

command: whoami

* name: ping all

ping:



Here, we are giving 2 tasks.

Then run this file: **ansible-playbook whoami.yaml**

Command and Shell module are not idempotent. Because we are executing raw Linux commands and those Linux commands can be different to each OS. And, we write a command to create a directory, and we run the playbook twice. This is why they both ate not idempotent.

Create one more file: whoamitest.yml

Create a file with name whoami.yaml  
- hosts: all

tasks:

* name: Find who is executing

command: whoami

* name: ping all

ping:

* name: findout current directory

command: pwd

Then run this file: **ansible-playbook whoamitest.yml**

The **pwd** will not give any response.  
Try to run the same from command prompt: **ansible all -m command -a “pwd”**

This will print the current directory on console.  
This is because, in the file, pwd response is a return of our task. We have to store the result in a variable and have to print it. We can’t directly print it.

By default, Ansible will execute Linux commands from user home directory.

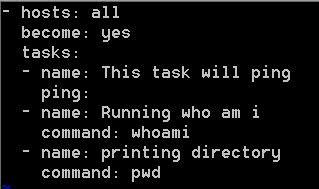
Ex: ansible all -m command -a “yum install tree -y”

Note: It is very important to use -y to stop asking for human interventions. Otherwise your script will be a failure.  
Above command will fail, because

Now try: Ex: ansible all -b -m command -a “yum install tree -y”

-b is **become user with permissions**. If not given any user name, it becomes user with permissions.  
This command is not idempotent.

Whenever we do installations, we have to give the **become: yes.** It is like executing commands with sudo permissions.



We can give the become at playbook root level or at each task level. Above example is for root level.

To use the same yum command as idempotent, then we can use equivalent **yum** module. For this search in google: **ansible module yum**  
  
Ex: Create a file with uninstalltree.yml:  
- hosts: all

become: yes

tasks:

* name: Find who is executing

command: whoami

* name: ping all

ping:

* name: findout current directory

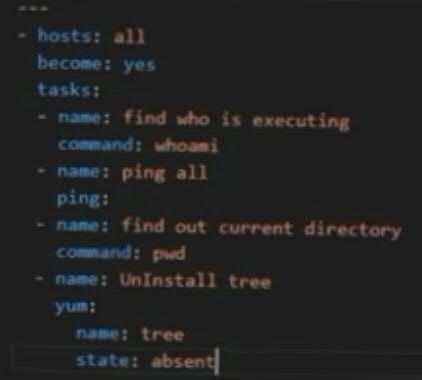
command: pwd

* name: uninstall tree

yum:

name: tree

state: absent



By default, state is present.  
Here using **state** we are saying, tree has to be uninstalled only when it is present.   
If we try to run the file twice, then it will not be uninstalled for the second time.   
So, **yum** has idempotent. But **Command** and **Shell** are not idempotent.  
So, Ansible is not 100% idempotent. When we use shell and command, then its our responsibility to take care of idempotent.  
  
Run the playbook: **ansible-playbook uninstalltree.yml**

Here, For the task Uninstall tree, first time it shows yellow color which indicates the uninstalled.  
Next time it shows in green color as it is not executed, as tree is not present.

**Example playbook to install apache server:**

Create a file(apache.yml) and add below text.

---

* hosts: all

become: yes

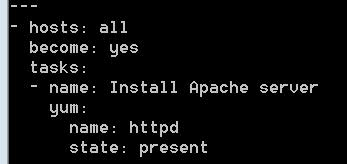
tasks:

- name: Install Apache server

yum:

name: httpd

state: present



Here writing **state: present** is optional, as the state is present by default.

Run the file: **ansible-playbook apache.yml**

Try to re run this command and check the idempotent.

**Service module:**

Search in google “ansible module service”

Check the **required** arguments, **enabled** and **state**

Create a file(apache.yml) and add below text.

---

* hosts: all

become: yes

tasks:

- name: Install Apache server

yum:

name: httpd

state: present

- name: Ensure Apache is running

service:

name: httpd

state: started

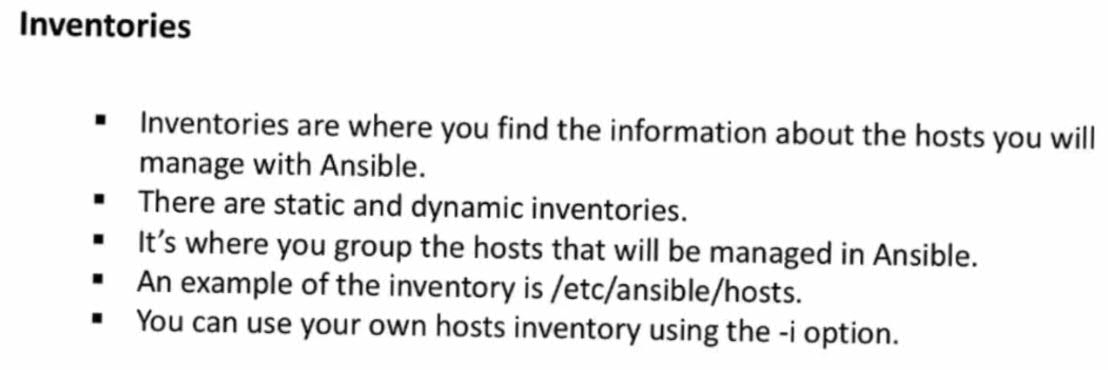
enabled: yes

Run the file: **ansible-playbook apache.yml**

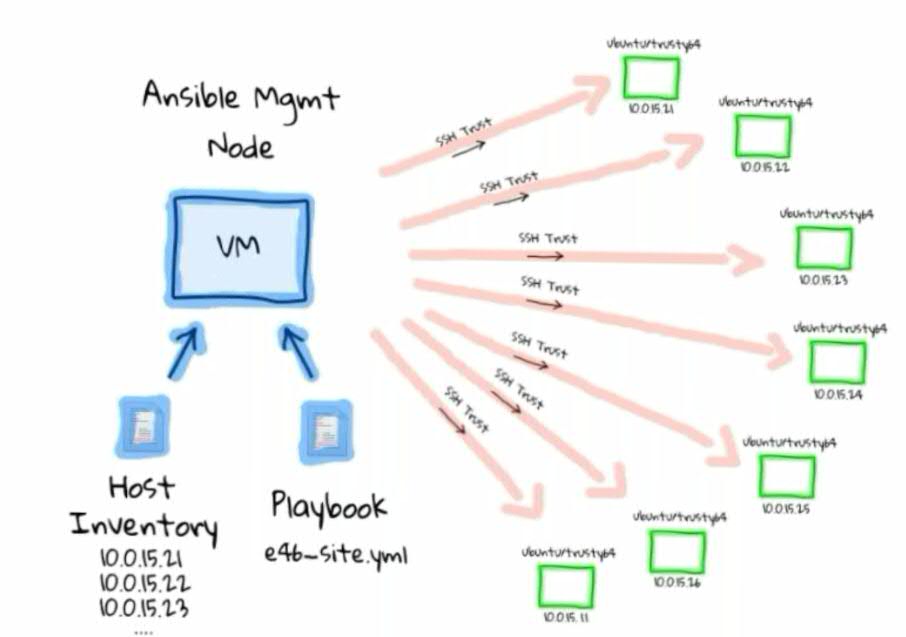
Now you can observe the services state has changes. Now, rerun the command and you will not see anything related to service.

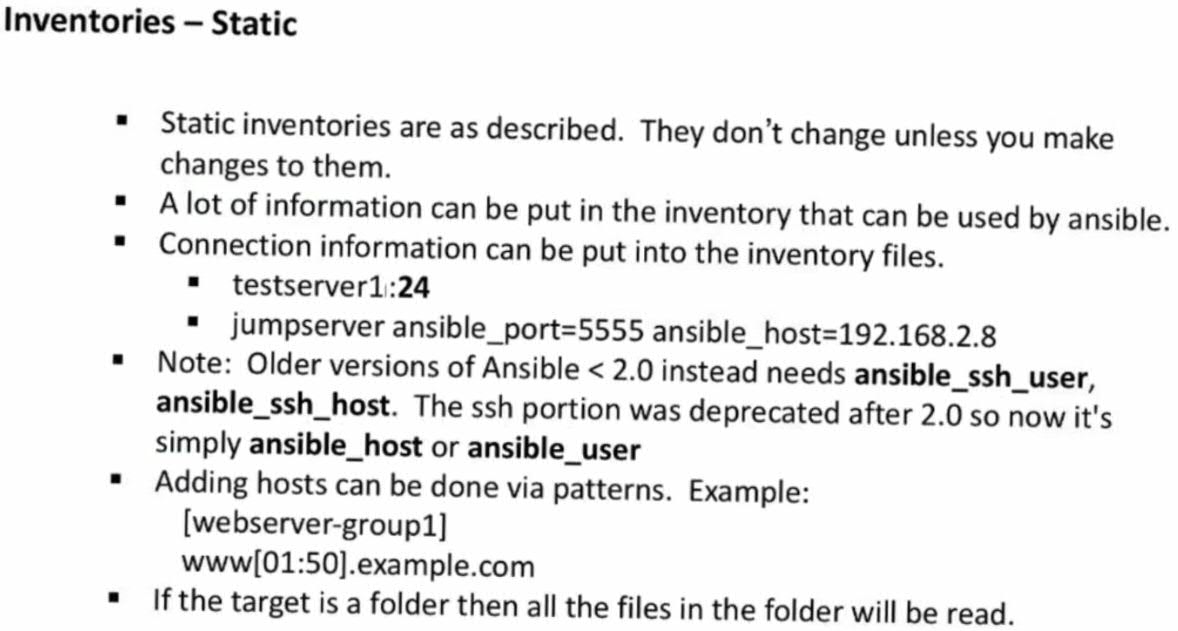
Now go to browser and try to access apache server: **http://ipaddress:8080**

**Inventories**



You may need dynamic inventories. When there is a auto scaling to increase or decrease servers size, then we need to configure those dynamically.  
For this we can write any kind of script like, python, groovy, shell etc.

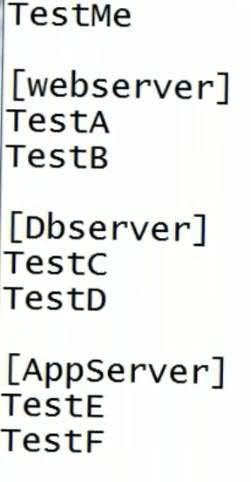




**Grouping hosts in the inventory**

We can group hosts in the inventory.  
Example, If there are 2 DB servers, 3 Web servers, 4 Application servers.  
Then you can group each type with some name.

Ex:



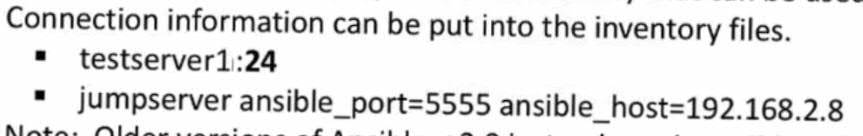
If we want to ping only web servers, then we execute the command as:

**ansible webserver -m ping** . here webserver is a group.  
For all hosts we use **ansible all -m ping .** here all indicates all hosts.

Here, **TestMe** doesn’t belongs to any group.

If you want to ping only to the **TestMe,** which has no group, then you can mention as: **ansible ungroup -m ping**

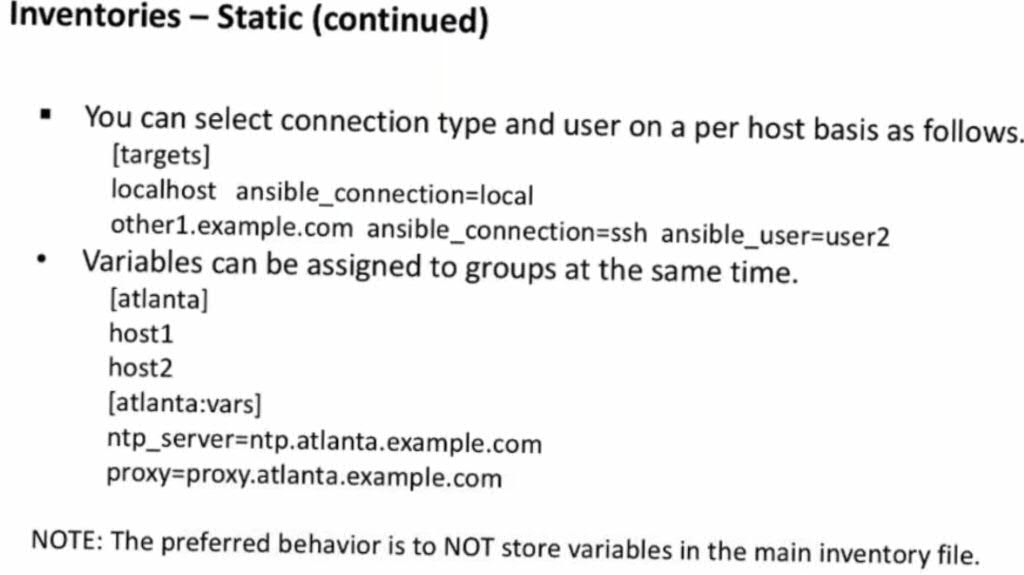
In the inventory file, if we want to specify anything for the host, then we can specify like below

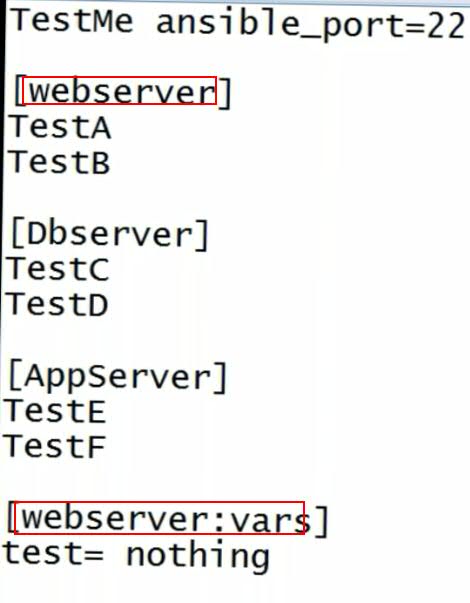


Here ansible\_port indicates the ssh port to connect. By default, port number is 22. ansible\_host indicates the ipaddress.

If we use ansible before 2.0, then the same variables must be used as ansible\_ssh\_port and ansible\_ssh\_host.

If there are many variable has to be declared for a group, then we can declare separately as a group. Bu that is not a recommended way to use in inventory file. Ansible recommends to create those variables in a separate folder.

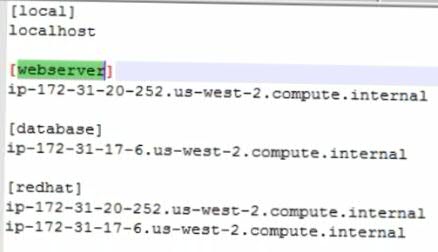
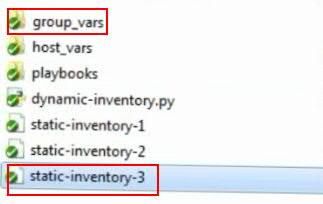




Here the servers group name and variables group name should match.

**Note**: same host can exist in multiple groups.

Supper if the inventory file name is static-inventory-3, then create a folder called group-vars at the same location where our inventory file exists.



Above indicates the static inventory file, group-vars folder location and content of the static-inventory-3 file.  
For example, if we want to create variables for the **database** group in the inventory file, then create a file with the same group name (**database**) under **groups-vars** directory. This file can have YAML or YML extension or none. But the format we write in this file is Uniformat and not YAML format.



Create required variables in that file in YAML format.

[$71B18407C7440558.jpg]

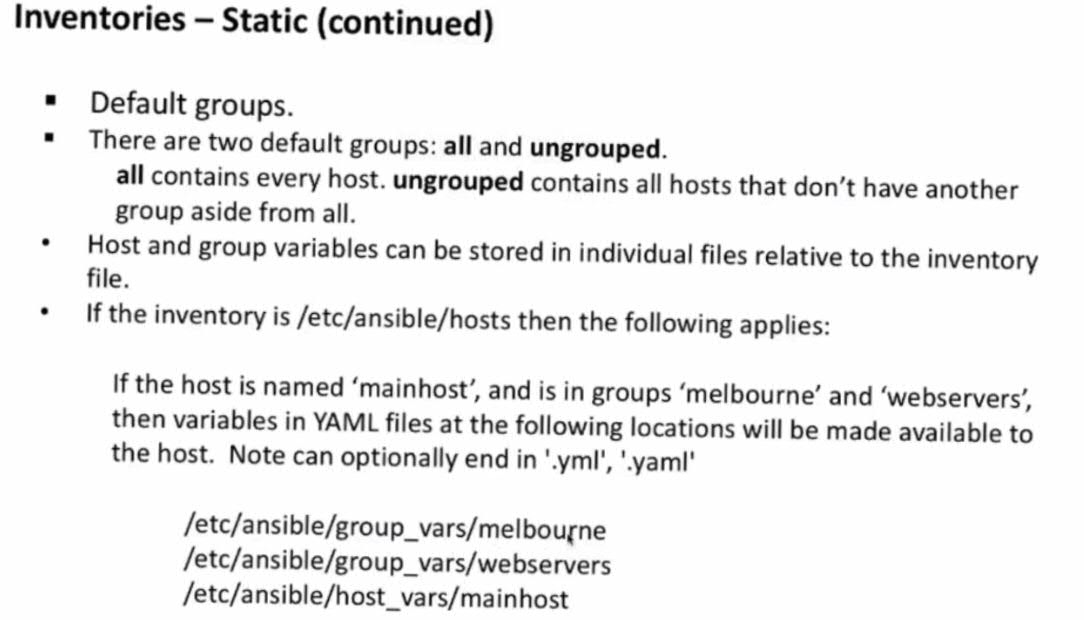
Similarly, we can create variables for hostnames, which are specific to a hostname.  
For this we need to create a folder with the name **host\_vars** and create a file with the hostname and write variables inside it. Below is the example of

Filename under **host\_vars** directory. This file name should be same as hostname inside inventory file.  

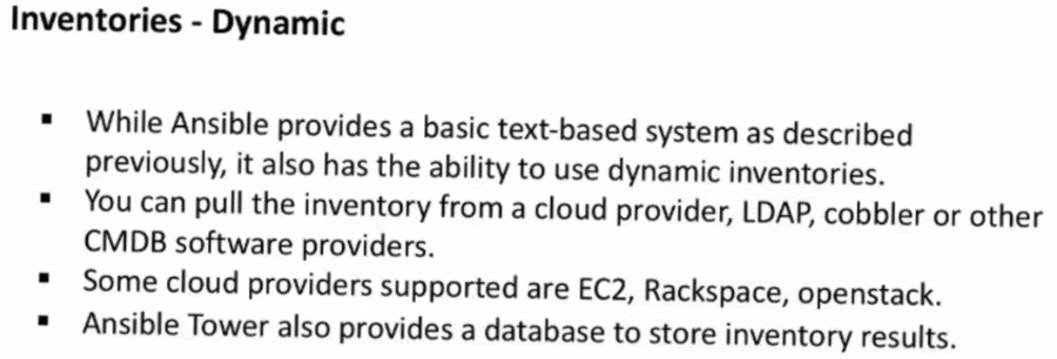

Below is variables content inside host file.

[$5B741C0366A863B8.jpg]

/etc/ansible/hosts is the default inventory.  
If we want to change the default inventory file location, then change it in /etc/ansible/ansible.cfg file.



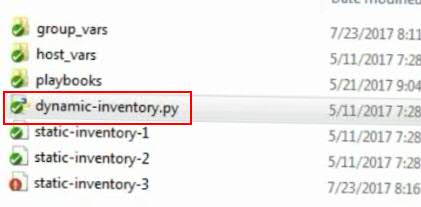
**Dynamic inventories:**



For dynamic inventories we can scripts like python or shell script. It can be of any language. That script can be called with 2 command line arguments.  
1) --list. When script is called with --list, then it will give the list of all the servers.  
2) --host. When script is called with --host, then it will give the info of all the host.

For our example, we use python script.

Dynamic inventories are used only when we work in cloud.



Connect to EC2 instance.  
Then create a folder (**Hosts**). Then create a file with the name **myhosts** and write below text in that file.

[own]

localhost

Then run the file using, **ansible own -i myhosts -m ping**

Here, own says the group name, -I indicates the host file location and -m ping is the command. This will be success

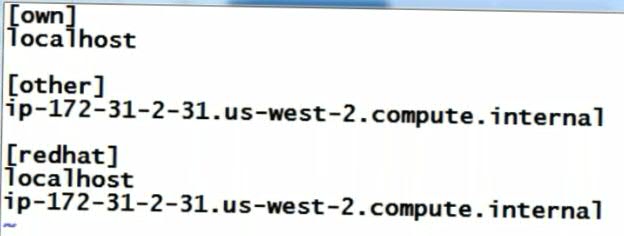
Then run the file using, **ansible all -i myhosts -m ping**

This will be success.

Then run the file using, **ansible all -m ping**

As we have not given -i, his will get the inventories from default inventory files(/etc/ansible/hosts).

Now modify the same host file(**myhosts**) with multiple groups with duplicate host, like



Now try to run the file, **ansible all -i myhosts -m ping**

As we have added the same host multiple times, but ansible will not execute multiple times.

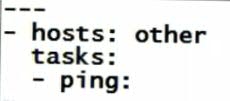
Now try to run the file, **ansible other -i myhosts -m ping**

This will ping only other group.

Now try to run the file, **ansible -b all -i myhosts -m command -a “ifconfig”**

Here, as the logged in use don’t have permissions, it will ask to become, so we gave -b.

Now, we will write a playbook file for this (**myplaybook.yaml**)

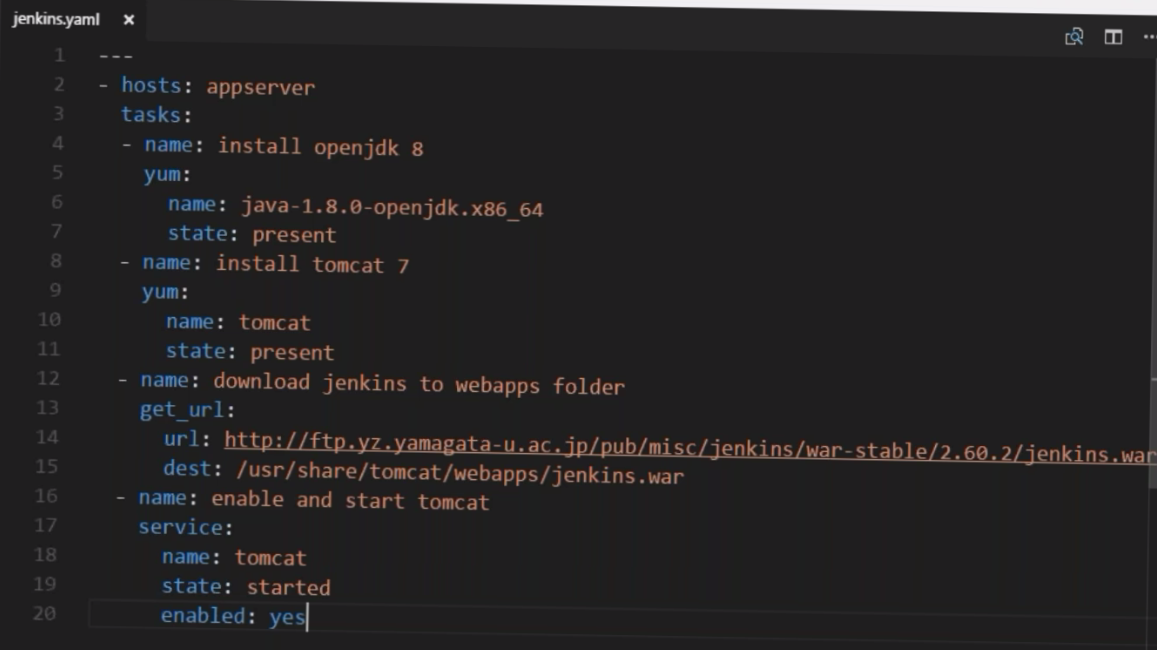


Now run this file, **ansible-playbook -i myhosts myplaybook.yaml**

**Note: ansible app -m shutdown**: to shutdown all servers which are configured in inventory file

**Now, we will try to write a playbook to install Jenkins.**  
For this, the manual steps are:  
1) Install Java 8, (yum install java-1.8.0-openjdk.x86\_64)  
2) Install Tomcat 7, (yum install tomcat)  
3) Copy Jenkins war from internet to tomcat webapps folder. (copy downloadurl).   
For downloading the war, we use get\_url ansible module.

Create a file Jenkins.yaml



---

- hosts: all

become: yes

tasks:

- name: Install Java

yum:

name: java-1.8.0-openjdk.x86\_64

state: present

- name: Install tomcat

yum:

name: tomcat

state: present

- name: Download jenkins war

get\_url:

url: http://ftp-chi.osuosl.org/pub/jenkins/war-stable/2.204.1/jenkins.war

dest: /usr/share/tomcat/webapps/jenkins.war

- name: Start tomcat

service:

name: tomcat

state: started

enabled: yes

As these are installations, use **become: yes** after and parallel to **hosts** key.

Here we have **state: present,** we can get this by searching in google for **ansible module yum**

**Now create inventory file(myhosts)**



Try to install git on EC2 instance using ansible adhoc commands:

**ansible -b localhost -m yum -a “name=git”**

**To uninstall git:**

**ansible -b localhost -m yum -a “name=git” “state=offset”**

Then we can clone the ansible code from out git repo.

**git clone giturl**

Now, go to the path where our playbook exists in git location and run the playbook.

**ansible-playbook -i myhosts Jenkins.yml**

Now, try to re run the command and observe. get\_url will not download is the file has same content. So, even get\_url module is also idempotent.

Now, to find the Nodes, use this command: **ansible-playbook -i myhosts --list-hosts Jenkins.yml**

This will list all the hosts mentioned in file myhosts and Jenkins is running.

Now take any one of the host and connect to it using: **ssh hostname**

And check the status of tomcat: **service tomcat status**

Now try to access Jenkins from browser. So for that, first enable the port number 8080 on EC2 instance Inbound rules. Then take the public ip address of the instance and try to access it from browser.

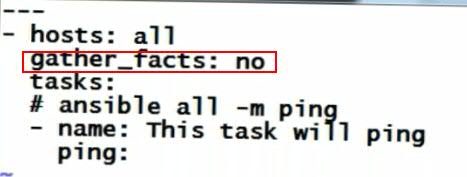
<http://ipaddress:8080> or http://ipaddress:8080/jenkins

If you see any error, then go to tomcat logs folder using: cd /usr/share/tomcat/logs

Then, less Catalina---.log

Note: From master, if we run any playbooks, in the console we can see one line of entry called “Gathering facts”. It means, master is fetching all the details of all Nodes. And also facts are fetched when we run any adhoc commands. Ex: **ansible all -m setup -a ‘filter=ansible\_default+ipv4\_address’**

Fetching of facts is not required in few cases and fetching may consume more time. So if we want to block fetching of facts, then in the playbook we write one entry called: **gather\_facts: no.** By defaults its value is yes.



Now, run the file and check the console: **ansible-playbook ping.yml**

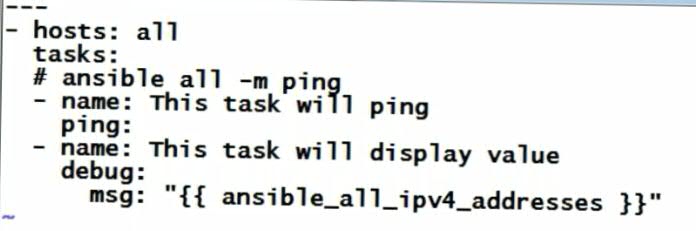
But this will not execute anything.

We can execute in another way using -v: **ansible-playbook -v ping.yml**

V is called verbosity and max we can use 3 vs.

**If we want to use these facts results as variables in our playbook:**

For this we use ansible module called **‘debug’.** This debug has a value called **msg**



Now, try to run this file: ansible-playbook pingallwithfacts.yml

Now, in the console we can see the ipv4 values printed.

Like, this we can use any system variables like facts variables, free memory etc etc using **msg** and its format.

To see the complete details of system variables, use this command:

**ansible all -m setup | more**

So, you see variables like OS distribution version, release etc etc.

For example, if we want to run few executions based on type of OD distribution, then we can use these variables.

**Creating custom variables:**

1 way) We can create **vars:** inside playbook and write. But this is not a recommended process.

Ex:

**become: yes**

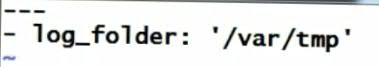
**vars:**

**test: hi**

2) using conventions.

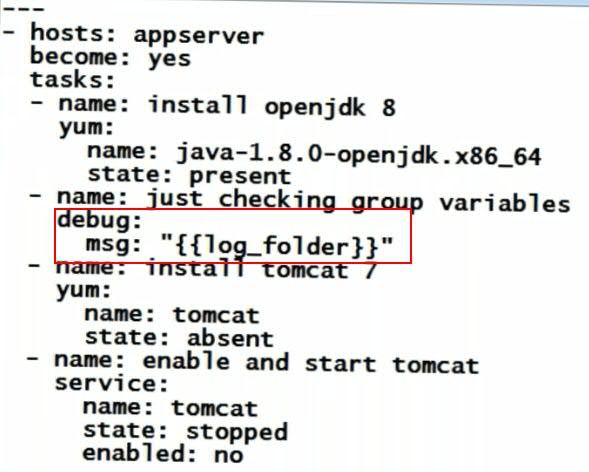
Using this we can create variables for separate groups too.

Suppose, we want to create variables for group “appserver”.  
Then we need to go to the inventory file location. Then create a folder there as **“group\_vars”.** Under that folder, create a file with the name of group, Ex: appserver

Now, open that file and write the variables.  


Why we are writing variables in a separate file is, we no need to modify the playbook always.

We can use these variables in playbook like below.



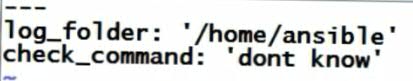
Now run the file: **ansible-playbook -b -i myhosts Jenkins.yaml**

In the console, you can see the variable value printed.

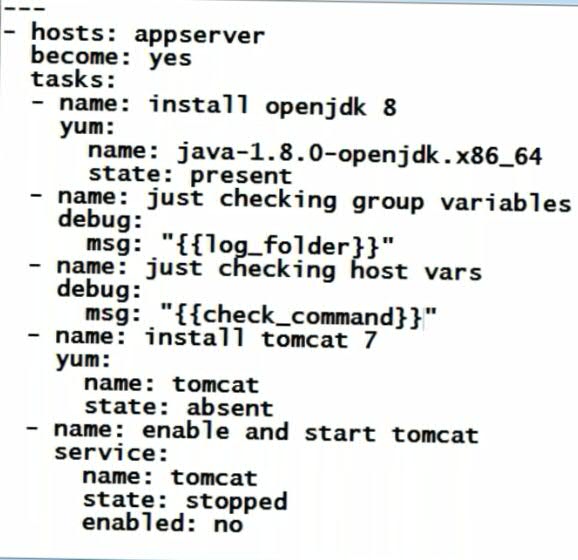
Now, if we modify the **log\_folder** variable in the variables file and run the playbook, then we can see the new variable value. By maintaining separate file, we no need to touch the playbook always.  
To know the list of file: **ansible-playbook -i myhosts --lists-hosts jenkins.yaml**

To create variables for each host, then we create a folder called **host\_vars** at the place where our inventory file exists and inside that we create a file with the hostname and we create variables in that file. So, those variables can be used by that host only.

For example, if our host file name is **ip-173-31-6-93,** then we will create a file with his name under **host\_vars** folder. Then, we will write few variables inside it like below.



Now, try to use these variables in Jenkins.yaml file, like below:



Here, we have same variable in group vars and host vars.

Now, try to run Jenkins.yaml:  
**ansible-playbook -i myhosts jenkins.yaml**

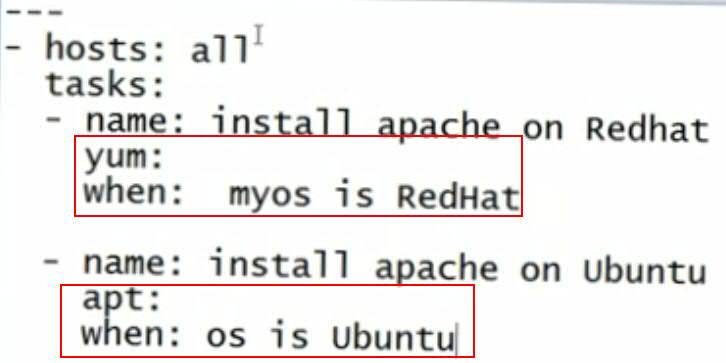
Here, we will get the host variable value here, as host variable will have

**For example, if we have both group and host variables, then host variable value over rides the group variables.**

For example, we need a same variable value for all the hosts except one. Then we create group variable and then we create a host variable for only one host to whom we need separate value.

**Conditional statements in Ansible**

Suppose we have a playbook which has steps to install apache on Ubuntu and Redhat. But ubuntu commands should not run Redhat and Redhat commands should not run on Ubuntu.  
In this case we can use conditional statements (Ex: when)



But, using below command, you can find the OS distribution variables and their values: **ansible localhost -m setup | more**

When you run this command, at the end you can find the “ansible\_distribution”: “RedHat” and

“ansible\_os\_family”: ”RedHat”

We can use any of these variables in the playbook. Similar way you can find for other Systems as well.

Create a playbook. Ex: apache.yml and add below stuff.



---

- hosts: all

become: yes

tasks:

- name: installing httpd server

yum:

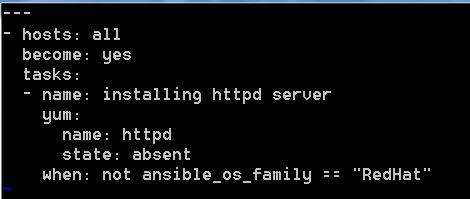
name: httpd

state: present

when: ansible\_os\_family == "RedHat"

Now run the file using **ansible-playbook apache.yml**

We can use not equal to by using **not** keyword before variable.



Below is the playbook to install apache webserver on Ubuntu and Redhat wil different commands:



Now run the file using **ansible-playbook apache.yml**

---

- hosts: all

become: yes

tasks:

- name: installing httpd server on Redhat

yum:

name: httpd

state: present

when: ansible\_os\_family == "RedHat"

- name: installing apache2 server on Ubuntu

apt:

name: apache2

state: present

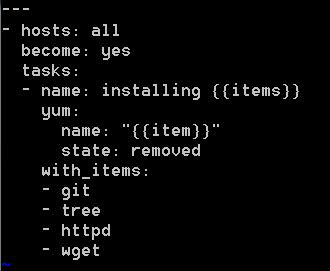
when: ansible\_os\_family == "Ubuntu"

- name: installing httpd server

debug:

msg: "Free memory is {{ansible\_memory\_mb.real.free}}"

**Loops and lists**

If we want to perform multiple installations, then we can declare those in a loop and can write a loop to install one by one. For this we can write like below. Create a playbook with the name loop.yaml  


**with\_items** is going to deprecate from 2.11 version. So use below format.

---

- hosts: all

become: yes

tasks:

- name: Installing "{{item}}" using loop

yum:

name:

- git

- tree

- httpd

- wget

state: absent

Now, run this file using **ansible-playbook loop.yml**

If we want to have a custom file to store variable and use those in playbook, then we must read that file in playbook using **vars\_file:** variable.

If we want to run only few lines in a big playbook, then we achieve this by using **tags** and below is the example.

Save the file as tags.yaml

|  |
| --- |
| --- |
|  | - hosts: localhost |
|  | become: yes |
|  | tasks: |
|  | - name: install apache on RedHat Server |
|  | yum: |
|  | name: httpd |
|  | state: present |
|  | when: ansible\_os\_family == "RedHat" |
|  | tags: |
|  | - install |
|  | - name: install apache on Ubuntu server |
|  | apt: |
|  | name: apache2 |
|  | state: present |
|  | when: ansible\_os\_family == "Debian" |
|  | tags: |
|  | - install |
|  | - service |
|  | - name: print free memory |
|  | debug: |
|  | msg: "free memory is {{ansible\_memory\_mb.real.free}}" |
|  | tags: |
|  | - debug |

To run only one specific tag, then run the above file using: **ansible-playbook tags.yaml --tags “debug”**

To run multiple tags at a time: **ansible-playbook tags.yaml --tags ubuntu, redhat**

We can use multiple tags.

If we want to skip only a specific tag, then run the above file using:

**ansible-playbook tags.yaml --skip-tags “install”**

We can have same tag at multiple places. Actually, tags are module.

If we want to run a playbook command based on the response of previous command, then we use option called “**register**”. Below is the example.

Create a file with the name register.yaml

|  |
| --- |
|  |
| --- |
|  | - hosts: localhost |
|  | tasks: |
|  | - name: print contents of file |
|  | command: cat /home/ansible/test |
|  | register: cat\_contents |
|  | ignore\_errors: yes |
|  | - name: display contents |
|  | debug: |
|  | msg: "{{cat\_contents}}" |
|  | - name: display content |
|  | debug: |
|  | msg: "{{cat\_contents.stdout}}" |
|  | when: cat\_contents.stderr == "" |
|  | - name: display error |
|  | debug: |
|  | msg: "{{cat\_contents.stderr}}" |
|  | when: not cat\_contents.stderr == "" |

In the above example, register contains the returned response of previous command i.e **command**

To run only file using: **ansible-playbook register.yaml**

We can use **when** with && and || operators**.**

**Templates:**

Ansible has adopted templates from Python Jinja templates.

Definition:

For Example, if we want to print the Server OS. If We deploy it on Ubuntu then it should print Ubuntu, on RedHat it should print RedHat. And same for IP as well. Here the jinja template comes into picture in Ansible. These files are saved with **.j2** extension.

Ex: index.html.j2

There are a few kinds of delimiters. The default Jinja delimiters are configured as follows:

* {% ... %} for [Statements](http://jinja.pocoo.org/docs/2.10/templates/#list-of-control-structures)
* {{ ... }} for [Expressions](http://jinja.pocoo.org/docs/2.10/templates/#expressions) to print to the template output
* {# ... #} for [Comments](http://jinja.pocoo.org/docs/2.10/templates/#comments) not included in the template output
* #  ... ## for [Line Statements](http://jinja.pocoo.org/docs/2.10/templates/#line-statements)

Refer below sites for more details regarding Jinja templates:

<http://jinja.pocoo.org/docs/2.10/templates/>

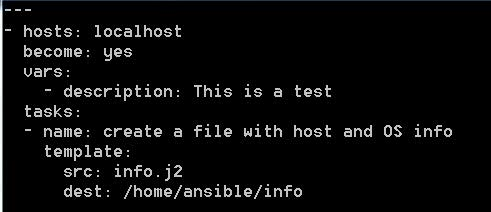
<https://docs.ansible.com/ansible-container/container_yml/template.html>

Ex: hostname {{ ansible\_hostname }}  
Here ansible\_hostname is a fact. Whenever we execute this template, then the value will be replaced. To use this, we need use a module called **Templates**.

Will try with an Example:

Create a simple playbook Ex:template.yaml and write the below content.

|  |
| --- |
| --- |
|  | - hosts: localhost |
|  | become: yes |
|  | vars: |
|  | - description: This is a test |
|  | tasks: |
|  | - name: create a file with host and OS info |
|  |
|  |
|  |
|  |
|  |
|  | template: |
|  | src: info.j2 |
|  | dest: /home/ansible/info |



To verify the Host name and its variable, run this command: ansible localhost -m setup -a ‘filter=\*host\*’

Create one more file info.j2 and add below content to it.

Hostname is {{ansible\_hostname}}

Operating System family is {{ansible\_os\_family}}

Ip Address is {{ansible\_default\_ipv4.address}}

Description is {{description}}

Make sure template.yaml and info.j2 resides in same directory. If not, then we have to give the relative path of info.j2 for **src:** value in template.yaml

Here in template.yaml, **src** is the location where our template resides.   
**dest** is the destination where the destination should exist.

In info.j2, what are all inside {{}} are variables, which are returned by facts or custom variable values.

Now run the playbook: **ansible-playbook template.yaml**

Then go to the file path /home/ansible/info

There you can see the result with dynamic values.

We can have multiple j2 files in a playbook like below:

|  |
| --- |
| --- |
|  | - hosts: all |
|  | become: yes |
|  | vars: |
|  | - myname: template examples |
|  | tasks: |
|  | - name: Create a file with static content |
|  | copy: |
|  | src: hello.txt |
|  | dest: /home/ansible/hello |
|  |  |
|  | - name: Create a file with dynamic content |
|  | template: |
|  | src: message.j2 |
|  | dest: /home/ansible/message |

In real-time, we can use these templates in scenarios like to add tomcat username passwords dynamically.

Search in google for Jinja filters. You can see all the list of jinja filters and you can use those in your examples as references.

Search in google for Jinja2 templates and there you can see many examples for if, if else conditions, loops, date formats etc.

Now, lets try to create a playbook and template to install and start a httpd server and then get a home page. For that create a playbook with below content. Ex: apache.yaml

---

- hosts: localhost

become: yes

vars:

inst\_name: KC

tasks:

- name: install apache on RedHat Server

yum:

name: httpd

state: present

when: ansible\_os\_family == "RedHat"

tags:

- install

- name: install apache on Ubuntu server

apt:

name: apache2

state: present

when: ansible\_os\_family == "Debian"

tags:

- install

- service

- name: print free memory

debug:

msg: "free memory is {{ansible\_memory\_mb.real.free}}"

tags:

- debug

- name: start and enable server

service:

name: httpd

state: started

when: ansible\_os\_family == "RedHat"

- name: Copy home page

template:

src: index.html.j2

dest: /var/html/index.html

notify: restart httpd

handlers:

- name: restart httpd

service:

name: httpd

state: restarted

Handlers gets called whenever we do any operation and do notify. For example, we have to restart httpd for every update on change on it. So in that case we have to write the restart stuff multiple times. But using handlers we can write the restart stuff only once and we can execute the restart stuff whenever requires using **notify**.

Here, the handler name and notify name should be same. Whenever any task is executed, and it has a notify, then whatever the handler matches to that notify name will be executed.

So, handler is something which gets called whenever it sees notify and this is identified using name.

Now create a template, index.html.j2 and copy below content.

|  |
| --- |
| <html> |
|  | <body> |
|  | <h1> Welcome to {{inst\_name}} </h1> |
|  |  |
|  | <div> |
|  | Server os family= {{ansible\_os\_family}} |
|  | Server ip = {{ansible\_default\_ipv4.address}} |
|  | </div> |
|  | </body> |
|  | </html> |

Now let’s run the playbook **ansible-playbook apche.yaml**

If everything goes fine, then we can access apache server from browser.  
<http://ipaddress>

Also, we can verify the server status using: **service httpd status**

Now, go and check the index.html in the destination. There you can see the dynamic data updated.

**Modulers:**

For example, if the content of the playbook is too big, in this case we can separate the content into separate files, and then we can call those files from the main file using **include** module**.**

For Example, if we have 2 files, main.yaml and tasks.yaml

main.yaml contains the actual data and the tasks.yaml contains the tasks data.

**main.yaml**

---

- hosts: localhost

vars:

description: This is sample description

tasks:

- include: tasks.yaml

**tasks.yaml**

---

- name: create a file with hostname and os info

template:

src: info.j2

dest: /home/ansible/info

But, along with tasks we have many things like variables, templates, handlers etc. With this approach it gets again complex. So Ansible has introduced a concept called Roles.

**Roles:**

Role is a modular way of writing playbook. It means splitting playbook into multiple files, so that it looks simple and can be reusable.

Main advantage of role is reusability.

Using roles, we split the playbook into multiple files with a proper directory structure.

To do this with sample, just create one directory. Ex: **Roles**

Now we will try to write a role for the apache playbook what we worked so far. Also, we can use the existing roles created by someone else. Also, the role we created can be made use by others. For this we have one concept called Galaxy.

Ansible galaxy is a place where we can get the roles created and pushed by someone. We can create a role and we can push that to Ansible galaxy, so others can use it.

For example, search in google for **ansible galaxy** and then search for the role what we want. For example, if we want a role for java, then search with the word java. So, there you can see the roles created and pushed by someone to galaxy. There you can see the command to install java in your local. Also, you can see one option called README, which has the explanation to use the role and the platforms and versions it supports.

Ansible galaxy is for sharing roles, which can be reused.

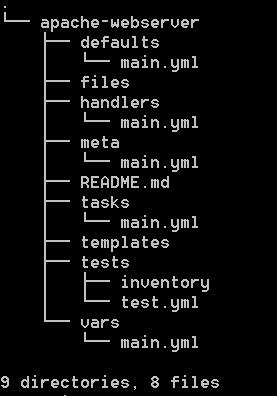
Let’s, create our sample role. Roles creation is nothing but creating folder structure. But the problem is, we have to remember the folder names and structure.

But, for this ansible has given an easiest way. This can be done with a command. Syntax: **ansible-galaxy init rolename**

Ex: **ansible-galaxy init apache-webserver**

Here, apache-webserver is the role name for apache.   
After running this command, a folder structure will be created for us with different folders and files. You can see the structure using tree command.

Ex: **tree**



There you can see main.yml file in all the folders. Then roles, in every folder there should be a main.yml file.

Lets, see the use of each directory.

**defaults:** Is the folder where we place all the default variables.

**handlers:** Is the folder where we put all the handlers.

**meta:** is the folder place where we write about dependencies. If your role depends on other roles, then we write the dependency description here.

README.md: It is a documentation file.

**tasks:** where we write all our tasks.

**tests:** Is where we write our test scripts and we can execute the test scripts to test.

**Vars:** where we place all the variables.

Now observe, we have scattered one playbook into multiple files.

Now, lets try to install java.

EX: **sudo ansible-galaxy install geerlingguy.java**

When you install any role, then all its related info will be created under **/etc/ansible/roles** directory

By default roles will be created under **/etc/ansible/roles** directory.

If we want to create roles on custom path then we have to give the path

EX: **sudo ansible-galaxy install geerlingguy.java -p .**

Here -p and . indicates the current directory as custom path.

If we want to change the default location then in ansible.cfg file, we have to change the default roles path to your custom path. So all the roles will be created under this custom path.

After running the command, you will get a success message.

Then after running above command, you can see a directory created with the role name(**geerlingguy.java**) and its directory structure. You can see the structure using tree command.

If we create a role and want to push it to ansible galaxy, then it is recommended to follow some naming convention syntax like: companyname or creatorName.roleName

Ex: myname.myrole

For our apache server we can create role like below.  
**sudo ansible-galaxy init mycompany.apache-webserver**

When running a playbook on multiple servers and we want to run on each server parallelly. This concept in Ansible is called as Forking. By default fork value is 5. But we can change it and our system should of capable of doing it.

Ex: **ansible all -f 2 -m setup** or

If we don’t want parallel execution, then use Ex: **ansible all -f 1 -m setup**

Now, lets create our own role to install Apache server on multiple Linux distributions. So, go to the **Roles** directory we created.

Then create our role with: **ansible-galaxy init myrole.apache**

So the directory **myrole.apache** will be created which has role structure.

Now, go to **vars** directory and create few variables. Place below text.

|  |
| --- |
| --- |
|  | # vars file for myrole.apache |
|  | apache\_package\_deb: apache2 |
|  | apache\_package\_rhel: httpd |
|  | apache\_homepage\_path: /var/www/html/index.html |

Here, we created 3 variables.

Then go to **main.yml** file under **tasks** directory. Then write what we have to do exactly. Copy below text for our example.

---

# tasks file for myrole.apache

- name: install apache on debian

apt:

name: "{{apache\_package\_deb}}"

state: present

when: ansible\_os\_family == "Debian"

- name: install apache on rhel

yum:

name: "{{apache\_package\_rhel}}"

state: present

when: ansible\_os\_family == "RedHat"

- name: set service name for Debian

set\_fact:

apache\_package: "{{apache\_package\_deb}}"

when: ansible\_os\_family == "Debian"

- name: set service name for RedHat

set\_fact:

apache\_package: "{{apache\_package\_rhel}}"

when: ansible\_os\_family == "RedHat"

- name: Service enable and start

service:

name: "{{apache\_package}}"

enabled: yes

state: started

- name: test by print

debug:

msg: firing notification for redhat

notify: restart rhel apache

when: ansible\_os\_family == "RedHat"

We created the last step (**name: test by print**) only for RedHat machine. If we want this for Ubuntu machine, then we have to repeat the steps and do changes for Ubuntu.

We can add **become: yes** at task level too. But its recommended to add at playbook.

If wewant to separate the tasks info different files, then we can create multiple file and add each task in each file. Then call all the tasks in the main.yml file using **include** option.

**Ex: include: apache.yml**

**when: ansible\_os\_family == "RedHat"**

Here, **set\_fact** is the module, where you can create a new variable and assign a value to it. You can search for this in google, ansible module set\_fact.

Now, go to **main.yml** under **handlers** and write the required handlers. For our example, use the below text.

---

# handlers file for myrole.apache

- name: restart debian apache

service:

name: "{{apache\_package\_deb}}"

state: restarted

- name: restart rhel apache

service:

name: "{{apache\_package\_rhel}}"

state: restarted

Create a **templates** folder and copy the **index.html.j2** file created earlier.

Now, Under **Roles**(root) directory and parallel to **myrole.apache**, create few files with these name: apache.yml, dev, qa, prod

Here, **apache.yml** is a playbook and other 3 are files are inventory files, i.e we can maintain different inventory files for each environment.

Now, add the required content for playbook. For our example, use below text:

---

- hosts: all

become: yes

roles:

- myrole.apache

Here, we can also write the 3rd party roles here.

As we are not tight coupling the role to playbook, we can reuse this role for many playbooks, by just calling the role in the playbook.

As of now, we successfully created a role. Now let’s try to execute it.

Before running the playbook, make sure apache is not running: sudo service httpd status.

If running, then uninstall it.

Go to **Roles** folder where playbook and roles exists.

Use this command to copy the **hosts** file to current location: **sudo cp /etc/ansible/hosts hosts**

Then run the playbook: **ansible-playbook apache.yml**

To run the playbook with dev inventory files: **ansible-playbook -i dev apache.yml**

**Ansible Tower**

**What is Ansible Tower?**

It is an enterprise version of Ansible which makes it easy, so that we can control whatever we have been doing in Ansible.

It will not write playbooks, roles. It is just an UI application where we can execute everything from UI and no need to use command line.

We use Ansible tower when we have many nodes (50 or 60) and not for less number(5 or 6) of Nodes.

**Install Ansible Tower**

Ansible tower can be downloaded as a tar file and that can be executed.

Or Ansible has predefined images presented in multiple cloud environments.

We can create an EC2 machine, which starts tower and then we can configure it.

Below are steps to install Ansible tower free trail on EC2 machine:

Go to <https://www.ansible.com/tower-trial>

Then expand “LAUNCH TOWER IN AMAZON EC2”

Then select the region where our region where our EC2 exists.

Then it will redirect to Amazon. Minimum 2 GB Ram is required for Ansible Tower to work.

So select a machine with 2 GB RAM or more(t2 medium) and then launch the instance with default ports.

Now connect to the EC2 instance from your local using pem file and switch to root user.

Now open 2 port numbers 443 and 80 for both http and https and as tower works with https.

Now, go back to the instance, select your machine, take the ipaddress and try to access from browser. Ex: <https://ipaddress>

Now, from command prompt, exit as root user and again try to switch as root user. There you can see a Welcome message for Ansible tower.

Copy the username and password printed there and login from browser using those credentials.

Then click on request license and select “free tower trial – limited features up to 10 nodes”. Then give the details and click on Submit.

Then go to your email inbox and download the received text file. Then you have to browse the downloaded file from browser option which is next to “Request license”. Then agree the agreement and click on Submit.

So that Ansible will be created and you can see Tower dashboard on browser. There you can see many options.

Jobs: It is periodic execution of playbook. If we can to run a playbook on some time interval, then we can create a Job.

Projects: Which contains details of our playbook and its related stuff.

For example for our project, if we can create a playbook and push it to git and we want to configure the git url(playbook) and jobs of tower.

Inventories: We can write an yml file or simply can add inventory by clicking on ADD. We can also add variables and groups too.

Now, lets do one example with Ansible tower by installing tomcat and deploying OpenMrs in it.

For this, first try to know the manual steps to install tomcat. Then write a role.

**Ansible Vault:**

Ansible Vault is a feature of ansible that allows you to keep sensitive data such as passwords or keys in encrypted files, rather than as plaintext in playbooks or roles. These vault files can then be distributed or placed in source control.

Commands:

To create secret encryption file: **ansible-vault create myencrypt.yml**

To edit encryption file: **ansible-vault edit myencrypt.yml**

To make secret encryption file to normal text file: **ansible-vault decrypt myencrypt.yml**

To make back text file to secret encryption file: **ansible-vault encrypt myencrypt.yml**

If we encrypt an playbook, then you cannot run the playbook normally using **ansible-playbook myencrypt.yml** command.

To run the encrypted playbook, we have to use the **--ask-vault-pass** after the playbook execute command.

Ex: **ansible-playbook myencrypt.yml --ask-vault-pass**

Then it will ask for the encrypted file password. Playbook will be executed after giving the password.

**Note**: In real-time, mostly we encrypt variables in roles.

How to check any ansible file syntax validation:  
Ans: ansible-playbook rds\_prod.yml --syntax-check

Ansible Dynamic inventory:

<https://www.youtube.com/watch?v=9z67s1t8UgM>

<https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.py>

<https://raw.githubusercontent.com/ansible/ansible/devel/contrib/inventory/ec2.ini>