
ANSIBLE

# Ansible

## Configuration Management

Configuration Management refers to the process of systematically handling changing to a system rather than manual.

***Ex:***

- Ansible
- Puppet
- Chef
- Saltstack

## Without Configuration Management

- Manual effort (months) to determine which system components should change when requirements change.
- Failed implementations because your project's requirements changed, and you didn’t communicate the changes to all parties.
- Lost productivity by replacing system components with flawed new versions, without the ability to quickly revert to a working state.
- Unexpected outages from incorrectly modifying system components, because you couldn't accurately determine which components were impacted by a change.

## Advantages:

- Increasing up time.
- Improve performance.
- Ensure compliance.
- Prevent errors.
- Cost Reduction and Risks.
- Identify software release levels for the event handling.
- Identify software release levels for the event prevention.
- Reduce the impact of software changes (proactive change impact assessment).
- Reduce downstream/upstream impact (hardware and software).

- Improve relationship identification between a change, incident and problem management.
- Improve communication between units via a common aggregated view into shared data repositories.
- Improve software/hardware ownership knowledge and reliability
- Proactively identify the high-risk failure points
- Execute documentation management solutions for operational systems
- Better understand change windows without the need to research each from scratch

## Ansible History

- Michael DeHaan developed Ansible, and the Ansible project began in February 2012.
- The creator of Cobbler and Func is also the controller of the Fedora Unified network.
- RedHat acquired the Ansible tool in 2015.
- Ansible is included as part of the Fedora distribution of the Linux.
- Ansible is also available for RedHat Enterprise Linux, Debian, CentOS, Oracle Linux, and Scientific Linux via Extra Packages for Enterprise Linux (EPEL) and Ubuntu as well as for other operating systems.

## Ansible

Ansible is a configuration management tool, it's automate the software provision, Configuration Management and Application deployment maintained by Redhat.

- The main components of Ansible are playbooks, configuration management and deployment.
- Ansible uses playbooks to deploy, manage, build, test and configure anything from full server environments to custom compiled source code for applications.
- Ansible was written in Python language.

## Features/Advantages

- Ansible is a simple no need to require any scripting language.
- Ansible is a open source from Redhat.
- Easy to learn and easy to use.

- Ansible is a powerful, it can automate any task and manage thousands of servers.
- Ansible is a agent less (No need to install each machine).
- Ansible relies on SSH communication.
- Built on top of Python and hence provides a lot of Python's functionality.
- Ansible internally uses python language.
- Ansible follows the push mechanism. Push based architecture for sending configurations.
- System requirement also very low.
- Python/YAML based.
- Larger number of ready to use modules for system mgmt.
- Custom modules can be added if needed.

## ***Push Based Vs Pull Based***

Agents on the server periodically checks for the configuration information from central server (Master).

***Ex:***

- Puppet
- Chef

Central server pushes the configuration information on target servers. Control when the changes are made on the servers.

***Ex:***

- Ansible
- Saltstack

## ***How Ansible works?***

- Ansible works on existing SSH connection.
- Ansible doesn't not require to open any network port or no need required any agent installation.
- We need enable the password less authentication (Key exchanging)

## Ansible Vs Chef

| Parameters | Ansible | Chef |
|---|---|---|
| ***Availability*** | Ansible runs with a single active node, called the Primary instance. If the primary goes down, there is a Secondary instance to take its place. | When there is a failure on the primary server, which is a chef server, it has a backup server to take the place of the primary server. |
| ***Easy to setup*** | Ansible has only a master running on the server machine, but no agents running on the client machine. It uses an SSH connection to log in to client systems or the nodes you want to configure. Client machine VM requires no unique setup. That's why it is faster to setup! | Chef has a master-agent architecture. Chef server runs on the master machine, and Chef client runs as an agent on each client machine. And also, there is an extra component called workstation, which contains all the tested configurations and then pushed to the central chef server. That's why it is not that easy. |
| ***Management*** | Easy to manage the configurations as it uses YAML (Yet Another Markup Language). The server pushes configurations to all the nodes. Suitable for real-time application, and there is immediate remote execution. | You need to be a programmer to manage the configurations as it offers configurations in Ruby DSL. The client pulls the configurations from the Server. |
| ***Configuration language*** | Ansible uses YAML (Python). It is quite easy to learn and its administrator oriented. Python is inbuilt | Chef uses Ruby Domain Specific Language (Ruby DSL). It has a Steep Learning Curve and its developer- |

| | | |
|---|---|---|
| | into most Unix and Linux deployments, so setting the tool up and running is quicker. | oriented. |
| ***Pricing*** | The pricing for Ansible Tower for standard IT operations up to 100 nodes is $10,000 per year. This includes 8*5 support, whereas premium offers 24*7 support for $14000 per year. | Chef Automate gives you everything you need to build, deploy in $137 node per year. |
| ***Authoritative configuration*** | Ansible's authoritative configuration comes from its deployed playbooks, which are perfect as source control systems. Or the Ansible method is more accessible and makes more sense. | The chef relies on its server as the authoritative configuration, and those servers require uploaded cookbooks, which means making sure the latter are consistent and identical. |

## ***Ansible Vs Puppet***

| ***Parameters*** | ***Ansible*** | ***Puppet*** |
|---|---|---|
| ***Availability*** | Ansible runs with a single active node, called the Primary instance. If the primary goes down, there is a Secondary instance to take its place. | Puppet has multi-master architecture. If the active master goes down, then the other master takes the active master place. |
| ***Easy to setup*** | Ansible has only a master running on the server machine, but no agents running on the client | Puppet also has a master-agent architecture. Puppet server runs on the master machine, and Puppet |

| | | | |
|---|---|---|---|
| | machine. It uses an SSH connection to log in to client systems or the nodes you want to configure. Client machine VM requires no unique setup. That's why it is faster to setup! | clients run as an agent on the client machine. After that, there is a certificate signing between the agent and the master. That's why it is not that easy to setup. | |
| ***Management*** | Easy to manage the configurations as it uses YAML (Yet Another Markup Language). The server pushes configurations to all the nodes. Suitable for real-time application, and there is immediate remote execution. | Puppet is not easy to manage the configurations as it uses its language called Puppet DSL. The client pulls the configurations from the Server. It is entirely system-administrator oriented, and there is non-immediate remote execution. | |
| ***Configuration language*** | Ansible uses YAML (Python). It is quite easy to learn, and it is administrator oriented. Python is inbuilt into most Unix and Linux deployments, so setting the tool up and running is quicker. | Puppet uses its puppet Domain Specific Language (Puppet DSL). It is not easy to learn, and it is system administrator oriented. | |
| ***Pricing*** | The pricing for Ansible Tower for standard IT operations up to 100 nodes is $10,000 per year. This includes 8*5 support, whereas premium offers 24*7 support for $14000 per year. | The pricing for puppet ranges from $112 node per year with a standard support plan to $199 node per year with the premium plan. | |

| | | |
|---|---|---|
| ***GUI*** | Ansible was the command-line tool only at the time of its inception. Now it has the UI in the enterprise version, but it is not perfect. Sometimes, GUI is not in perfect sync with the Command line and not able to perform the same things like the command-line interface. | Puppet's Graphical User Interface is more interactive than Ansible. It is used to manage, view, and monitor more complex tasks. Otherwise, there is an option of using a command-line interface too when need which is written in Ruby. |

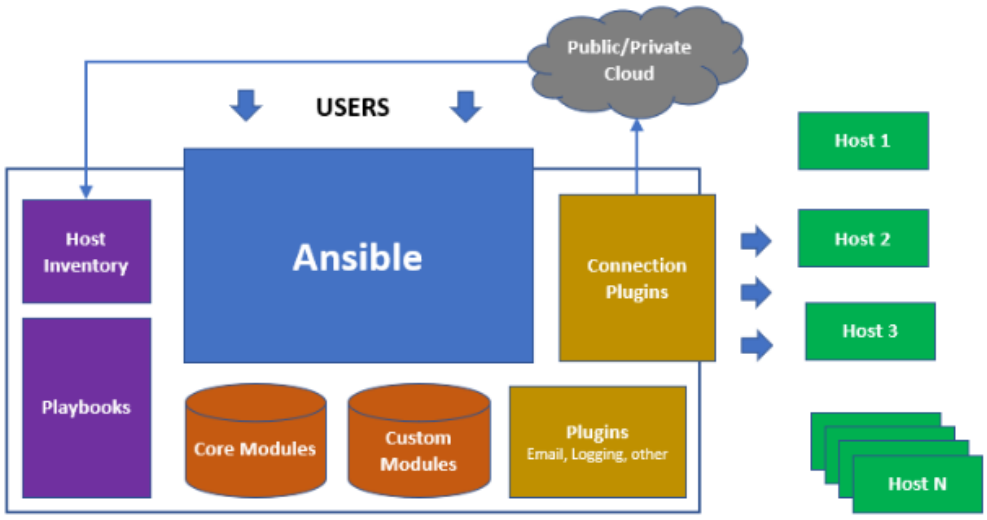
## ***Ansible Control Machine/Engine/Master***

In this machine we can trigger ansible play books and adhoc-commands. This machine must be non-windows and python2.6 OR higher versions. (2.6v)

## ***Ansible Management Machine/node/Clients***

In this machine where we can configure and install the software's.It must be any machine. Contains python 2.6 OR higher versions.

## ***Ansible Architecture***



Let us start with **Public/Private Cloud** which is the Linux server. It can also act as a repository for all IT installation and configurations.

The above architecture has a bunch of **host** machines to which ansible server connects and pushes the playbooks through SSH.

It has **ansible automation engine** using which users can directly run a playbook which gets deployed on the hosts. There are multiple components in the ansible automation engine. The first is a **host inventory**. It's a list of all the IP addresses of all the hosts.

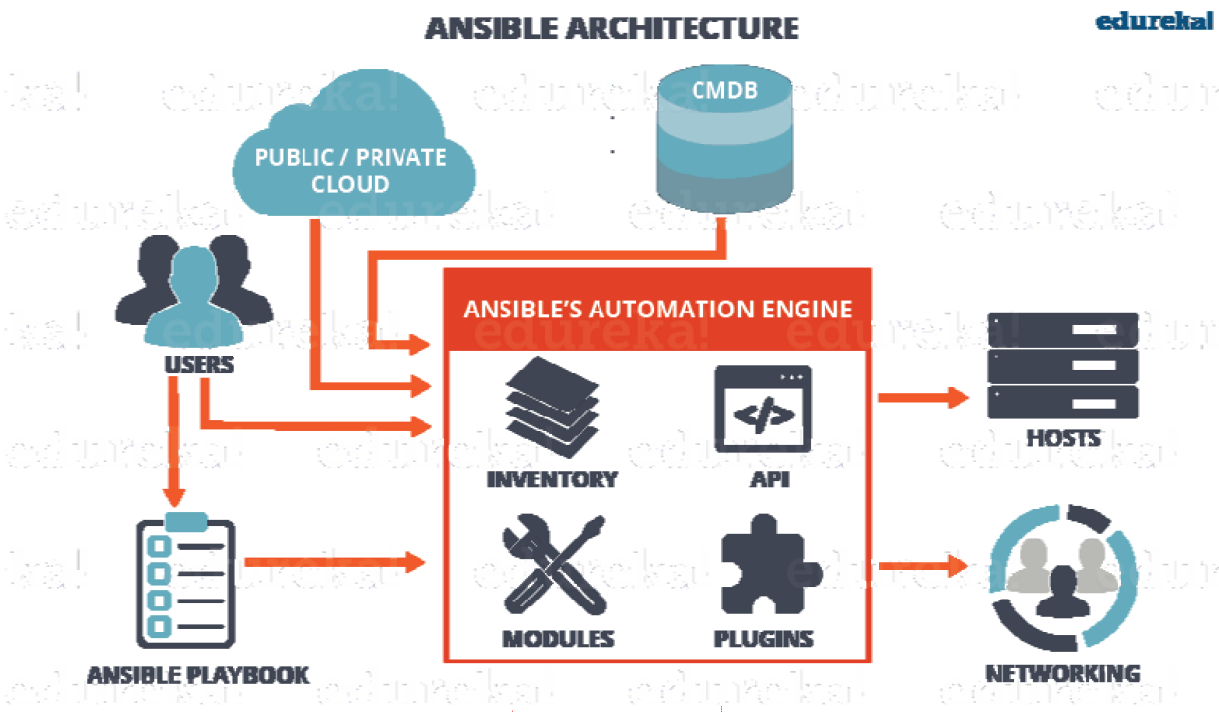
Ansible comes with hundreds of **inbuilt modules** and **modules** are those pieces of code that get executed when you run a playbook. A playbook contains plays, a play contains different tasks, and a task includes modules.

When you run a playbook, it's the modules that get executed on your hosts, and these modules contain action in them. So, when you run a playbook, those action takes place on your host machines. You can make your custom modules also. All you must do is write a few lines of code and make it your module, and you can run it anytime you want.

Then the architecture has **playbooks**. Playbooks here actually define your workflow because whatever tasks that you write in a playbook, it gets executed in the same order that you have written them. For example, if you have written that install a package first and then start, it'll do the same. Playbooks are very simple to write YAML code. YAML code is a very simple data serialization language; it's pretty much like English.

Next, in the architecture are **plugins**. Plugins here are special kind of modules. These plugins get executed before a module is getting executed on the nodes. Plugins get executed on the main control machine for logging purposes. You've got call-back plugins because this enables you to hook into different ansible events for display and logging purposes. Cache plugins are used to keep a cache of facts to avoid costly fact-gathering operations. Ansible also has action plugins, which are front-end modules, and they can execute tasks on the controller machine before calling the modules themselves.

The architecture has connection plugins. It is not always needed to use an SSH for connecting with your host machines; you can also use a connection plug-in. For example, ansible provides you with a docker container connection plugin and using that connection plug-in, you can easily connect to all your docker containers and start configuring right away.




The main component of Ansible is the Ansible automation engine. This engine directly interacts with various cloud services, Configuration Management Database (CMBD) and different users who write various playbooks to execute the Ansible Automation engine.

The Ansible Automation engine consists of the following components:

Inventories: These are a list of nodes containing their respective IP addresses, servers, databases, etc. which needs to be managed.
**APIs:** Just like any other API, the Ansible APIs are used for commuting various Cloud services, public or private services.
**Modules:** The modules are used to manage system resources, packages, libraries, files, etc. Ansible modules can be used to automate a wide range of tasks. Ansible provides around 450 modules that automate nearly every part of your environment.
**Plugins:** If you want to execute Ansible tasks as a job, Ansible Plugins can be used. They simplify the execution of a task by building a job like an environment that basically contains pieces of code corresponding to some specific functionality. There are 100s of Plugins provided by Ansible. An example is the Action plugin, which acts as front ends to modules and can execute tasks on the controller before calling the modules themselves.

**Networking:** Ansible can also be used to automate different networks and services. It can do this by creating a playbook or an Ansible role that easily spans different network hardware.

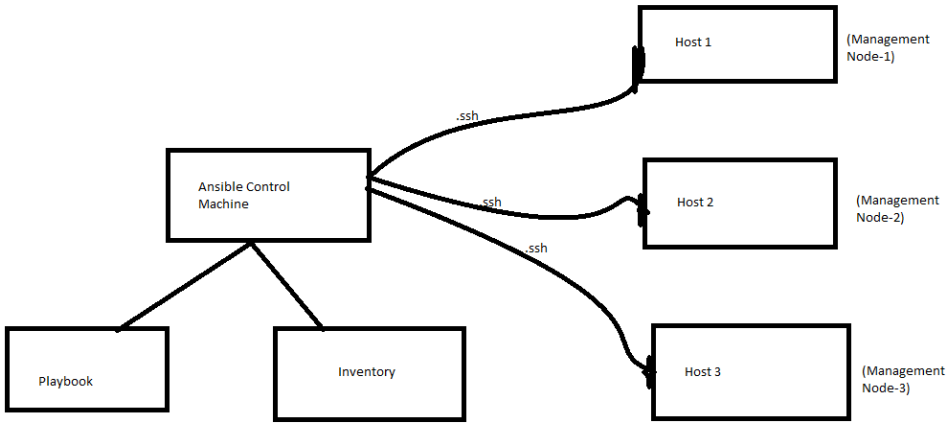
**Hosts:** The Ansible Hosts/ Node systems are machines (Linux, Windows, etc) that are getting automated.

**Playbooks:** Playbooks are simple code files which describe the tasks that need to be executed. The Playbooks are written in YAML format. They can be used to automate tasks, declare configurations, etc.

**CMDB:** It is a database that acts as a storehouse for various IT installations. It holds data about various IT assets (also known as configuration items (CI)) and describes the relationships between such assets.

**Cloud:** It is a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server.

## ***Ansible Workflow***




Ansible works by connecting to your nodes and pushing out a small program called **Ansible modules** to them. Then Ansible executed these modules and removed them after finished. The library of modules can reside on any machine, and there are no daemons, **servers,** or **databases** required.

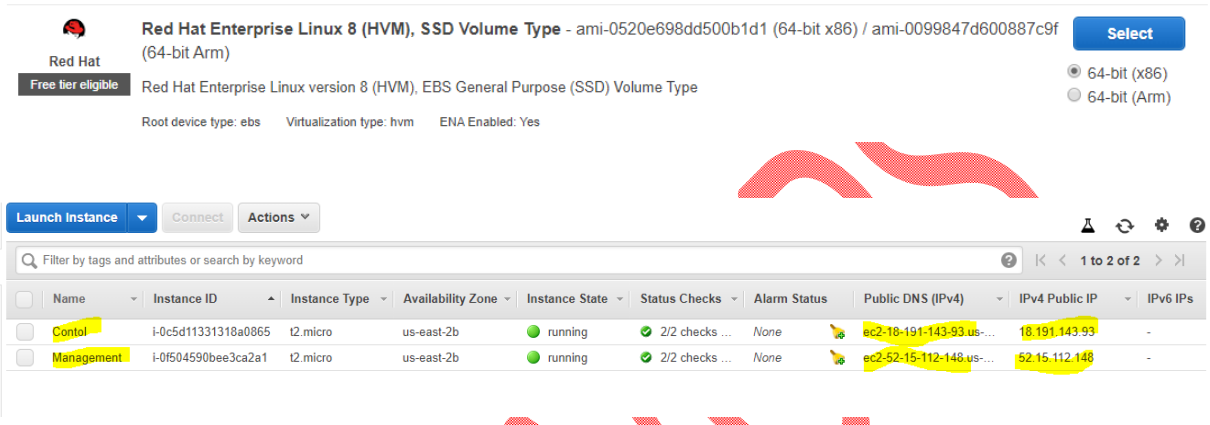
In the above image, The **Control Node** that controls the entire execution of the playbook. The **inventory** file provides the list of hosts where the Ansible modules need to be run. The **Management Node** makes an **SSH** connection and executes the small modules on the host's machine and install the software.

Ansible removes the modules once those are installed so expertly. It connects to the host machine executes the instructions, and if it is successfully installed, then remove that code in which one was copied on the host machine

# Ansible Installation

## Red Hat Enterprise Linux / CENTOS

- Launch two vms on AWS console.




18.191.143.93 **(Control_Machine)**

52.15.112.148 **(Management_Machine)**

- Launch the **Control Machine** and execute below instructions
- Create **ansadmin** user with sufficient privileges.

```
sudo su
useradd ansadmin
passwd ansadmin
echo "ansadmin ALL=(ALL) ALL" >> /etc/sudoers
sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
yum install python3
python3 -V
alternatives --set python /usr/bin/python3
su ansadmin
pip3 install ansible --user
```

```
[ec2-user@ip-172-31-24-193 ~]$ sudo su
[root@ip-172-31-24-193 ec2-user]# useradd ansadmin
[root@ip-172-31-24-193 ec2-user]# passwd ansadmin
Changing password for user ansadmin.
New password:
BAD PASSWORD: The password contains the user name in some form
Retype new password:
passwd: all authentication tokens updated successfully.
[root@ip-172-31-24-193 ec2-user]# echo "ansadmin ALL=(ALL) ALL" >> /etc/sudoers
[root@ip-172-31-24-193 ec2-user]# sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
[root@ip-172-31-24-193 ec2-user]# yum install python3 -y
```

```
[root@ip-172-31-24-193 ec2-user]# python3 -V
Python 3.6.8
[root@ip-172-31-24-193 ec2-user]# alternatives --set python /usr/bin/python3
[root@ip-172-31-24-193 ec2-user]# su ansadmin
[ansadmin@ip-172-31-24-193 ec2-user]$ pip3 install ansible --user
```

- Verify the Ansible version using below command.

ansible –version

```
[ansadmin@ansible-contol ~]$ ansible --version
ansible 2.9.5
  config file = None
  configured module search path = ['/home/ansadmin/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /home/ansadmin/.local/lib/python3.6/site-packages/ansible
  executable location = /home/ansadmin/.local/bin/ansible
  python version = 3.6.8 (default, Apr  3 2019, 17:26:03) [GCC 8.2.1 20180905 (Red Hat 8.2.1-3)]
[ansadmin@ansible-contol ~]$
```

Launch the **management machine** and execute below instructions

Create **ansadmin** user with sufficient privileges.

sudo su

useradd ansadmin

passwd ansadmin

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

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

sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config

```
[root@ansible-management ~]# useradd ansadmin
[root@ansible-management ~]# passwd ansadmin
Changing password for user ansadmin.
New password:
BAD PASSWORD: The password contains the user name in some form
Retype new password:
passwd: all authentication tokens updated successfully.
[root@ansible-management ~]# echo "ansadmin ALL=(ALL) ALL" >> /etc/sudoers
[root@ansible-management ~]# sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
[root@ansible-management ~]# su ansadmin
[ansadmin@ansible-management root]$
```

- Generate and exchange the ***public keys*** from **Control Machine** to **Management Machine.**

ssh-keygen

ssh-copy-id ansadmin@52.15.112.148

```
[ansadmin@ansible-contol ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansadmin/.ssh/id_rsa):
Created directory '/home/ansadmin/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ansadmin/.ssh/id_rsa.
Your public key has been saved in /home/ansadmin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:wlSVxqwC9Y/VcNCukBu+NcQ1a7QN7xw7VW/A0WR8oiE ansadmin@ansible-contol
The key's randomart image is:
+---[RSA 2048]----+
|      .. .+++...=o|
|     .  o  E+B +.=|
|      .. .=.=.X o+|
|      o. =+o * + +|
|       ooS=.o o = |
|        .o +   =  |
|          o .   . |
|         .         |
|                   |
+----[SHA256]-----+
[ansadmin@ansible-contol ~]$ ssh-copy-id ansadmin@52.15.112.148
```

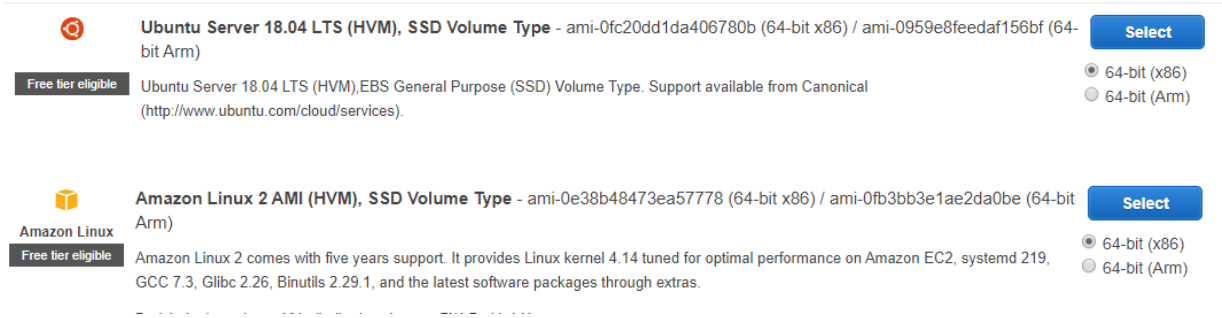
```
[ansadmin@ansible-contol ~]$ ssh-copy-id ansadmin@52.15.112.148
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/ansadmin/.ssh/id_rsa.pub"
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
ansadmin@52.15.112.148's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'ansadmin@52.15.112.148'"
and check to make sure that only the key(s) you wanted were added.

[ansadmin@ansible-contol ~]$
```

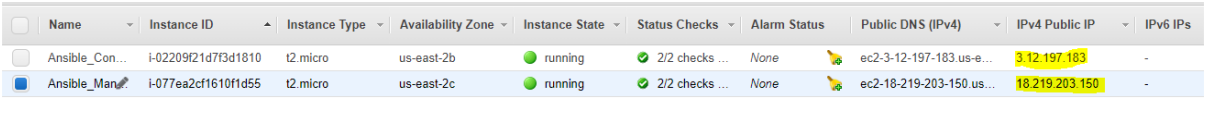
ssh ansadmin@52.15.112.148

```
[ansadmin@ansible-contol ~]$ ssh ansadmin@52.15.112.148
Last login: Sat Feb 22 11:21:45 2020
[ansadmin@ansible-management ~]$
```

## ***Ubuntu-Installation***

- Launch two vms on AWS console.





| Name | Instance ID | Instance Type | Availability Zone | Instance State | Status Checks | Alarm Status | Public DNS (IPv4) | IPv4 Public IP | IPv6 IPs |
|---|---|---|---|---|---|---|---|---|---|
| Ansible_Con... | i-02209f21d7f3d1810 | t2.micro | us-east-2b | running | 2/2 checks ... | None | ec2-3-12-197-183.us-e... | 3.12.197.183 | - |
| Ansible_Man | i-077ea2cf1610f1d55 | t2.micro | us-east-2c | running | 2/2 checks ... | None | ec2-18-219-203-150.us... | 18.219.203.150 | - |

3.12.197.183 **(Control_Machine) - Ubuntu**

18.219.203.150 **(Management_Machine) – Amazon Linux**

apt-get update

adduser ansadmin

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

sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config

service sshd reload

```
ubuntu@ip-172-31-23-129:~$ sudo su
root@ip-172-31-23-129:/home/ubuntu# hostname ansible-control
root@ip-172-31-23-129:/home/ubuntu# adduser ansadmin
Adding user `ansadmin' ...
Adding new group `ansadmin' (1001) ...
Adding new user `ansadmin' (1001) with group `ansadmin' ...
Creating home directory `/home/ansadmin' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for ansadmin
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
root@ip-172-31-23-129:/home/ubuntu# echo "ansadmin ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers
root@ip-172-31-23-129:/home/ubuntu# echo "ansadmin ALL=(ALL:ALL) ALL" >> /etc/sudoers
root@ip-172-31-23-129:/home/ubuntu# sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
root@ip-172-31-23-129:/home/ubuntu# service sshd reload
root@ip-172-31-23-129:/home/ubuntu# apt-get install ansible
```

apt-get install ansible

su ansadmin

cd /home/ansadmin

```
root@ip-172-31-23-129:/home/ubuntu# ansible --version
ansible 2.5.1
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/root/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.17 (default, Nov  7 2019, 10:07:09) [GCC 7.4.0]
root@ip-172-31-23-129:/home/ubuntu# su ansadmin
ansadmin@ansible-control:/home/ubuntu$ cd
ansadmin@ansible-control:~$ pwd
/home/ansadmin
ansadmin@ansible-control:~$ ansible --version
ansible 2.5.1
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/ansadmin/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.17 (default, Nov  7 2019, 10:07:09) [GCC 7.4.0]
ansadmin@ansible-control:~$
```

Launch the **management machine** and execute below instructions

Create **ansadmin** user with sufficient privileges.

sudo su

hostname ansible-management

useradd ansadmin

passwd ansadmin

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

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

sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config

```
[ec2-user@ip-172-31-42-162 ~]$ sudo su
[root@ip-172-31-42-162 ec2-user]# hostname ansible-management
[root@ip-172-31-42-162 ec2-user]# useradd ansadmin
[root@ip-172-31-42-162 ec2-user]# passwd ansadmin
Changing password for user ansadmin.
New password:
BAD PASSWORD: The password contains the user name in some form
Retype new password:
passwd: all authentication tokens updated successfully.
[root@ip-172-31-42-162 ec2-user]# echo "ansadmin ALL=(ALL:ALL) ALL" >> /etc/sudoers
[root@ip-172-31-42-162 ec2-user]# sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
[root@ip-172-31-42-162 ec2-user]# service sshd reload
Redirecting to /bin/systemctl reload sshd.service
[root@ip-172-31-42-162 ec2-user]# su ansadmin
[ansadmin@ansible-management ec2-user]$ cd
[ansadmin@ansible-management ~]$
```

- Generate and exchange the ***public keys*** from **Control Machine** to **Management Machine.**

ssh-keygen

ssh-copy-id ansadmin@18.219.203.150

```
ansadmin@ansible-control:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansadmin/.ssh/id_rsa):
Created directory '/home/ansadmin/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ansadmin/.ssh/id_rsa.
Your public key has been saved in /home/ansadmin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:i121Zx/J4OxcvlWBNC/EsyIUqqdM6EsZTTSX7oZPFJQ ansadmin@ansible-control
The key's randomart image is:
+---[RSA 2048]----+
|     o.oo.. .+   |
|    . oE..  oo+  |
|     ...o   .+oo |
|    + .o . ooo+ o|
|   o ++.S o ooo=.|
|  . =.o* o  ooo.o|
|   + o= o    o .o|
|  . .  .        o|
|   .           . |
+----[SHA256]-----+
ansadmin@ansible-control:~$ ssh-copy-id ansadmin@18.219.203.150
```

```
ansadmin@ansible-control:~$ vi hosts
ansadmin@ansible-control:~$ ansible all -m ping -i ./hosts
18.219.203.150 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
ansadmin@ansible-control:~$
```

## ***Amazon Linux 2 Installation***

- Launch two vms on AWS console.

1 to 40 of 40 AMIs

Amazon Linux

Free tier eligible

**Amazon Linux 2 AMI (HVM), SSD Volume Type** - ami-0e38b48473ea57778 (64-bit x86) / ami-0fb3bb3e1ae2da0be (64-bit Arm)

Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras.

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

Select

64-bit (x86)

64-bit (Arm)

| Name | Instance ID | Instance Type | Availability Zone | Instance State | Status Checks | Alarm Status | Public DNS (IPv4) | IPv4 Public IP | IPv6 IPs |
|---|---|---|---|---|---|---|---|---|---|
| Ansible_Man... | i-077ea2cf1610f1d55 | t2.micro | us-east-2c | running | 2/2 checks ... | None | ec2-18-219-203-150.us... | 18.219.203.150 | - |
| Ansible_Con... | i-0852f1e9fd653d4e7 | t2.micro | us-east-2b | running | 2/2 checks ... | None | ec2-18-223-98-132.us-... | 18.223.98.132 | - |

18.223.98.132 **(Control_Machine) - Amazon Linux**

18.219.203.150 **(Management_Machine) – Amazon Linux**

sudo su

yum update

hostname ansible-control

useradd ansadmin

passwd ansadmin

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

sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config

service sshd reload

amazon-linux-extras install ansible2

```
[ec2-user@ip-172-31-24-122 ~]$ sudo su
[root@ip-172-31-24-122 ec2-user]# hostname ansible-control
[root@ip-172-31-24-122 ec2-user]# useradd ansadmin
[root@ip-172-31-24-122 ec2-user]# passwd ansadmin
Changing password for user ansadmin.
New password:
BAD PASSWORD: The password contains the user name in some form
Retype new password:
passwd: all authentication tokens updated successfully.
[root@ip-172-31-24-122 ec2-user]# echo "ansadmin ALL=(ALL:ALL) ALL" >> /etc/sudoers
[root@ip-172-31-24-122 ec2-user]# sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
[root@ip-172-31-24-122 ec2-user]# service sshd reload
Redirecting to /bin/systemctl reload sshd.service
[root@ip-172-31-24-122 ec2-user]# amazon-linux-extras install ansible2
```

su ansadmin

ansible –version

```
[ansadmin@ansible-control ec2-user]$ ansible --version
ansible 2.8.5
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/ansadmin/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /bin/ansible
  python version = 2.7.16 (default, Dec 12 2019, 23:58:22) [GCC 7.3.1 20180712 (Red Hat 7.3.1-6)]
[ansadmin@ansible-control ec2-user]$
```

Launch the **management machine** and execute below instructions

Create **ansadmin** user with sufficient privileges.

sudo su

hostname ansible-management

useradd ansadmin

passwd ansadmin

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

sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config

```
[ec2-user@ip-172-31-42-162 ~]$ sudo su
[root@ip-172-31-42-162 ec2-user]# hostname ansible-management
[root@ip-172-31-42-162 ec2-user]# useradd ansadmin
[root@ip-172-31-42-162 ec2-user]# passwd ansadmin
Changing password for user ansadmin.
New password:
BAD PASSWORD: The password contains the user name in some form
Retype new password:
passwd: all authentication tokens updated successfully.
[root@ip-172-31-42-162 ec2-user]# echo "ansadmin ALL=(ALL:ALL) ALL" >> /etc/sudoers
[root@ip-172-31-42-162 ec2-user]# sed -ie 's/PasswordAuthentication no/PasswordAuthentication yes/' /etc/ssh/sshd_config
[root@ip-172-31-42-162 ec2-user]# service sshd reload
Redirecting to /bin/systemctl reload sshd.service
[root@ip-172-31-42-162 ec2-user]# su ansadmin
[ansadmin@ansible-management ec2-user]$ cd
[ansadmin@ansible-management ~]$
```

- Generate and exchange the ***public keys*** from **Control Machine** to **Management Machine.**

ssh-keygen

ssh-copy-id ansadmin@18.219.203.150

```
[ansadmin@ansible-control ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ansadmin/.ssh/id_rsa):
Created directory '/home/ansadmin/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/ansadmin/.ssh/id_rsa.
Your public key has been saved in /home/ansadmin/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:7WNF3kSMyrgNNxU7TkIop4IBBstjU9hVe6PWzCTWIng ansadmin@ansible-control
The key's randomart image is:
+---[RSA 2048]----+
|+oo..... .. .+.  |
|ooo.. . =.  oo.  |
|.= + E O B.o= .  |
|. + o + %.** +   |
|      . oSO..+ .  |
|       . ....    |
|          +      |
|         . .     |
|                 |
+----[SHA256]-----+
```

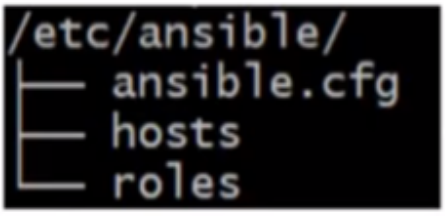
```
+----[SHA256]-----+
[ansadmin@ansible-control ~]$ ssh-copy-id ansadmin@18.219.203.150
/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/ansadmin/.ssh/id_rsa.pub"
The authenticity of host '18.219.203.150 (18.219.203.150)' can't be established.
ECDSA key fingerprint is SHA256:4yhqrcPjXErDWqGu9VLiQv+/dq1HMgWSqlhci/OtgDo.
ECDSA key fingerprint is MD5:b3:5c:8a:eb:d7:14:b2:6a:4b:a0:0a:de:90:88:dd:b4.
Are you sure you want to continue connecting (yes/no)? yes
/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out any that are already installed
/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted now it is to install the new keys
ansadmin@18.219.203.150's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'ansadmin@18.219.203.150'"
and check to make sure that only the key(s) you wanted were added.

[ansadmin@ansible-control ~]$ ssh ansadmin@18.219.203.150
Last login: Sat Feb 22 12:09:41 2020 from 3.12.197.183

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ansadmin@ansible-management ~]$
```

```
[ansadmin@ansible-control ~]$ ansible all -m ping -i ./hosts
 [WARNING]: Platform linux on host 18.219.203.150 is using the discovered Python interpreter at /usr/bin/python, but
future installation of another Python interpreter could change this. See
https://docs.ansible.com/ansible/2.8/reference_appendices/interpreter_discovery.html for more information.

18.219.203.150 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "ping": "pong"
}
[ansadmin@ansible-control ~]$
```

## How to check ansible version?

ansible –version

## Ansible Directory Structure



## *Ansible configuration file(ansible.cfg)*

- If we run any ansible command first it will check the ansible configuration file. The default path is /etc/ansible/ansible.cfg
  Based on requirement we can change/update the ansible configuration file.
- By default all parameters are commented. If you want to change the behavior update uncomment to comment.

```
[ansadmin@ip-172-31-24-122 ansible]$ cat ansible.cfg
# config file for ansible -- https://ansible.com/
# ===============================================

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags. ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]

# some basic default values...

#inventory      = /etc/ansible/hosts
#library        = /usr/share/my_modules/
#module_utils   = /usr/share/my_module_utils/
#remote_tmp     = ~/.ansible/tmp
#local_tmp      = ~/.ansible/tmp
#plugin_filters_cfg = /etc/ansible/plugin_filters.yml
#forks          = 5
#poll_interval  = 15
#sudo_user      = root
#ask_sudo_pass = True
#ask_pass      = True
#transport      = smart
#remote_port    = 22
#module_lang    = C
#module_set_locale = False
```

- The default configuration file contains the parameters like inventory, library, module locations ...etc.
- Before going update the ansible configuration file we need to run the ansible adhoc command below.

```
[ansadmin@ip-172-31-24-122 ~]$ ansible all -m shell -a uptime
 [WARNING]: Platform linux on host 18.222.181.106 is using the discovered Python interpreter at /usr/bin/python, but
future installation of another Python interpreter could change this. See
https://docs.ansible.com/ansible/2.8/reference_appendices/interpreter_discovery.html for more information.

18.222.181.106 | CHANGED | rc=0 >>
 15:38:56 up 35 min,  2 users,  load average: 0.00, 0.00, 0.00

[ansadmin@ip-172-31-24-122 ~]$
```

ansible all -m shell -a uptime

- If we want to update the inventory parameter like below.

```
inventory      = /home/ansadmin/hosts
#library        = /usr/share/my_modules/
#module_utils   = /usr/share/my_module_utils/
```

```
[ansadmin@ip-172-31-24-122 ~]$ ansible all -m shell -a uptime
 [WARNING]: Platform linux on host 3.15.164.103 is using the discovered Python interpreter at /usr/bin/python, but future
installation of another Python interpreter could change this. See
https://docs.ansible.com/ansible/2.8/reference_appendices/interpreter_discovery.html for more information.

3.15.164.103 | CHANGED | rc=0 >>
 15:48:29 up 44 min,  2 users,  load average: 0.08, 0.02, 0.01

 [WARNING]: Platform linux on host 18.222.181.106 is using the discovered Python interpreter at /usr/bin/python, but
future installation of another Python interpreter could change this. See
https://docs.ansible.com/ansible/2.8/reference_appendices/interpreter_discovery.html for more information.

18.222.181.106 | CHANGED | rc=0 >>
 15:48:29 up 44 min,  2 users,  load average: 0.00, 0.00, 0.00
```

## Multiple ansible configuration files

- Ansible has the flexibility to have the multiple configuration files. So changing the single configuration file for every time is not required on the depend on the environment.
- Ansible find the configuration files below as find the order first.
  - Environment variable ANSIBLE_CONFIG set OR not.

    *Ex:*

    ANSIBLE_CONFIG=/home/ansadmin/config/myansible.cfg

    export ANSIBLE_CONFIG
  - ansible.cfg file current running path (Playbook OR adhoc-command)
  - ansible.cfg file home directory
  - ansible.cfg file default path i.e. /etc/ansible/ansible.cfg

## Why do we use multiple ansible configuration files?

Our application may have different web servers, application servers and db servers separately. Each will have different users and different inventory files.

If we want to perform automation for specific servers it's a challenge so in that time we can use multiple configuration files.

## Verifying particular ansible configuration file

Use –v option along with ansible command

***ansible all -m shell -a uptime -v***

```
[ansadmin@ip-172-31-24-122 ec2-user]$ ansible all -m shell -a uptime -v
Using /etc/ansible/ansible.cfg as config file
 [WARNING]: Platform linux on host 18.222.181.106 is using the discovered Python interpreter at /usr/bin/python, but
future installation of another Python interpreter could change this. See
https://docs.ansible.com/ansible/2.8/reference_appendices/interpreter_discovery.html for more information.

18.222.181.106 | CHANGED | rc=0 >>
 16:13:53 up  1:10,  2 users,  load average: 0.00, 0.00, 0.00

[ansadmin@ip-172-31-24-122 ec2-user]$
```

## ***Sample ansible configuration file***

```
# config file for ansible -- https://ansible.com/
# ===============================================

# nearly all parameters can be overridden in ansible-playbook
# or with command line flags. ansible will read ANSIBLE_CONFIG,
# ansible.cfg in the current working directory, .ansible.cfg in
# the home directory or /etc/ansible/ansible.cfg, whichever it
# finds first

[defaults]

# some basic default values...

#inventory      = /etc/ansible/hosts
#library        = /usr/share/my_modules/
#module_utils   = /usr/share/my_module_utils/
#remote_tmp     = ~/.ansible/tmp
#local_tmp      = ~/.ansible/tmp
#plugin_filters_cfg = /etc/ansible/plugin_filters.yml
#forks          = 5
#poll_interval  = 15
#sudo_user      = root
#ask_sudo_pass = True
#ask_pass      = True
#transport      = smart
#remote_port    = 22
#module_lang    = C
#module_set_locale = False

# plays will gather facts by default, which contain information about
# the remote system.
#
# smart - gather by default, but don't regather if already gathered
# implicit - gather by default, turn off with gather_facts: False
# explicit - do not gather by default, must say gather_facts: True
#gathering = implicit

# This only affects the gathering done by a play's gather_facts directive,
# by default gathering retrieves all facts subsets
# all - gather all subsets
# network - gather min and network facts
```

```
# hardware - gather hardware facts (longest facts to retrieve)
# virtual - gather min and virtual facts
# facter - import facts from facter
# ohai - import facts from ohai
# You can combine them using comma (ex: network,virtual)
# You can negate them using ! (ex: !hardware,!facter,!ohai)
# A minimal set of facts is always gathered.
#gather_subset = all

# some hardware related facts are collected
# with a maximum timeout of 10 seconds. This
# option lets you increase or decrease that
# timeout to something more suitable for the
# environment.
# gather_timeout = 10

# Ansible facts are available inside the ansible_facts.* dictionary
# namespace. This setting maintains the behaviour which was the default prior
# to 2.5, duplicating these variables into the main namespace, each with a
# prefix of 'ansible_'.
# This variable is set to True by default for backwards compatibility. It
# will be changed to a default of 'False' in a future release.
# ansible_facts.
# inject_facts_as_vars = True

# additional paths to search for roles in, colon separated
#roles_path    = /etc/ansible/roles

# uncomment this to disable SSH key host checking
host_key_checking = False

# change the default callback, you can only have one 'stdout' type  enabled at a time.
#stdout_callback = skippy


## Ansible ships with some plugins that require whitelisting,
## this is done to avoid running all of a type by default.
## These setting lists those that you want enabled for your system.
## Custom plugins should not need this unless plugin author specifies it.

# enable callback plugins, they can output to stdout but cannot be 'stdout' type.
#callback_whitelist = timer, mail

# Determine whether includes in tasks and handlers are "static" by
# default. As of 2.0, includes are dynamic by default. Setting these
# values to True will make includes behave more like they did in the
# 1.x versions.
#task_includes_static = False
#handler_includes_static = False

# Controls if a missing handler for a notification event is an error or a warning
```

```
#error_on_missing_handler = True

# change this for alternative sudo implementations
#sudo_exe = sudo

# What flags to pass to sudo
# WARNING: leaving out the defaults might create unexpected behaviours
#sudo_flags = -H -S -n

# SSH timeout
#timeout = 10

# default user to use for playbooks if user is not specified
# (/usr/bin/ansible will use current user as default)
#remote_user = root

# logging is off by default unless this path is defined
# if so defined, consider logrotate
#log_path = /var/log/ansible.log

# default module name for /usr/bin/ansible
#module_name = command

# use this shell for commands executed under sudo
# you may need to change this to bin/bash in rare instances
# if sudo is constrained
#executable = /bin/sh

# if inventory variables overlap, does the higher precedence one win
# or are hash values merged together?  The default is 'replace' but
# this can also be set to 'merge'.
#hash_behaviour = replace

# by default, variables from roles will be visible in the global variable
# scope. To prevent this, the following option can be enabled, and only
# tasks and handlers within the role will see the variables there
#private_role_vars = yes

# list any Jinja2 extensions to enable here:
#jinja2_extensions = jinja2.ext.do,jinja2.ext.i18n

# if set, always use this private key file for authentication, same as
# if passing --private-key to ansible or ansible-playbook
#private_key_file = /path/to/file

# If set, configures the path to the Vault password file as an alternative to
# specifying --vault-password-file on the command line.
#vault_password_file = /path/to/vault_password_file

# format of string {{ ansible_managed }} available within Jinja2
# templates indicates to users editing templates files will be replaced.
```

```
# replacing {file}, {host} and {uid} and strftime codes with proper values.
#ansible_managed = Ansible managed: {file} modified on %Y-%m-%d %H:%M:%S by {uid} on {host}
# {file}, {host}, {uid}, and the timestamp can all interfere with idempotence
# in some situations so the default is a static string:
#ansible_managed = Ansible managed

# by default, ansible-playbook will display "Skipping [host]" if it determines a task
# should not be run on a host.  Set this to "False" if you don't want to see these "Skipping"
# messages. NOTE: the task header will still be shown regardless of whether or not the
# task is skipped.
#display_skipped_hosts = True

# by default, if a task in a playbook does not include a name: field then
# ansible-playbook will construct a header that includes the task's action but
# not the task's args.  This is a security feature because ansible cannot know
# if the *module* considers an argument to be no_log at the time that the
# header is printed.  If your environment doesn't have a problem securing
# stdout from ansible-playbook (or you have manually specified no_log in your
# playbook on all of the tasks where you have secret information) then you can
# safely set this to True to get more informative messages.
#display_args_to_stdout = False

# by default (as of 1.3), Ansible will raise errors when attempting to dereference
# Jinja2 variables that are not set in templates or action lines. Uncomment this line
# to revert the behavior to pre-1.3.
#error_on_undefined_vars = False

# by default (as of 1.6), Ansible may display warnings based on the configuration of the
# system running ansible itself. This may include warnings about 3rd party packages or
# other conditions that should be resolved if possible.
# to disable these warnings, set the following value to False:
system_warnings = True

# by default (as of 1.4), Ansible may display deprecation warnings for language
# features that should no longer be used and will be removed in future versions.
# to disable these warnings, set the following value to False:
#deprecation_warnings = True

# (as of 1.8), Ansible can optionally warn when usage of the shell and
# command module appear to be simplified by using a default Ansible module
# instead.  These warnings can be silenced by adjusting the following
# setting or adding warn=yes or warn=no to the end of the command line
# parameter string.  This will for example suggest using the git module
# instead of shelling out to the git command.
# command_warnings = False


# set plugin path directories here, separate with colons
#action_plugins     = /usr/share/ansible/plugins/action
#become_plugins     = /usr/share/ansible/plugins/become
#cache_plugins      = /usr/share/ansible/plugins/cache
```

```
#callback_plugins   = /usr/share/ansible/plugins/callback
#connection_plugins = /usr/share/ansible/plugins/connection
#lookup_plugins     = /usr/share/ansible/plugins/lookup
#inventory_plugins  = /usr/share/ansible/plugins/inventory
#vars_plugins       = /usr/share/ansible/plugins/vars
#filter_plugins     = /usr/share/ansible/plugins/filter
#test_plugins       = /usr/share/ansible/plugins/test
#terminal_plugins   = /usr/share/ansible/plugins/terminal
#strategy_plugins   = /usr/share/ansible/plugins/strategy


# by default, ansible will use the 'linear' strategy but you may want to try
# another one
#strategy = free

# by default callbacks are not loaded for /bin/ansible, enable this if you
# want, for example, a notification or logging callback to also apply to
# /bin/ansible runs
#bin_ansible_callbacks = False


# don't like cows?  that's unfortunate.
# set to 1 if you don't want cowsay support or export ANSIBLE_NOCOWS=1
#nocows = 1

# set which cowsay stencil you'd like to use by default. When set to 'random',
# a random stencil will be selected for each task. The selection will be filtered
# against the `cow_whitelist` option below.
#cow_selection = default
#cow_selection = random

# when using the 'random' option for cowsay, stencils will be restricted to this list.
# it should be formatted as a comma-separated list with no spaces between names.
# NOTE: line continuations here are for formatting purposes only, as the INI parser
#       in python does not support them.
#cow_whitelist=bud-frogs,bunny,cheese,daemon,default,dragon,elephant-in-snake,elephant,eyes,\
#              hellokitty,kitty,luke-koala,meow,milk,moofasa,moose,ren,sheep,small,stegosaurus,\
#              stimpy,supermilker,three-eyes,turkey,turtle,tux,udder,vader-koala,vader,www

# don't like colors either?
# set to 1 if you don't want colors, or export ANSIBLE_NOCOLOR=1
#nocolor = 1

# if set to a persistent type (not 'memory', for example 'redis') fact values
# from previous runs in Ansible will be stored.  This may be useful when
# wanting to use, for example, IP information from one group of servers
# without having to talk to them in the same playbook run to get their
# current IP information.
#fact_caching = memory

#This option tells Ansible where to cache facts. The value is plugin dependent.
```

```
#For the jsonfile plugin, it should be a path to a local directory.
#For the redis plugin, the value is a host:port:database triplet: fact_caching_connection =
localhost:6379:0

#fact_caching_connection=/tmp



# retry files
# When a playbook fails a .retry file can be created that will be placed in ~/
# You can enable this feature by setting retry_files_enabled to True
# and you can change the location of the files by setting retry_files_save_path

#retry_files_enabled = False
#retry_files_save_path = ~/.ansible-retry

# squash actions
# Ansible can optimise actions that call modules with list parameters
# when looping. Instead of calling the module once per with_ item, the
# module is called once with all items at once. Currently this only works
# under limited circumstances, and only with parameters named 'name'.
#squash_actions = apk,apt,dnf,homebrew,pacman,pkgng,yum,zypper

# prevents logging of task data, off by default
#no_log = False

# prevents logging of tasks, but only on the targets, data is still logged on the master/controller
#no_target_syslog = False

# controls whether Ansible will raise an error or warning if a task has no
# choice but to create world readable temporary files to execute a module on
# the remote machine.  This option is False by default for security.  Users may
# turn this on to have behaviour more like Ansible prior to 2.1.x.  See
# https://docs.ansible.com/ansible/become.html#becoming-an-unprivileged-user
# for more secure ways to fix this than enabling this option.
#allow_world_readable_tmpfiles = False

# controls the compression level of variables sent to
# worker processes. At the default of 0, no compression
# is used. This value must be an integer from 0 to 9.
#var_compression_level = 9

# controls what compression method is used for new-style ansible modules when
# they are sent to the remote system.  The compression types depend on having
# support compiled into both the controller's python and the client's python.
# The names should match with the python Zipfile compression types:
# * ZIP_STORED (no compression. available everywhere)
# * ZIP_DEFLATED (uses zlib, the default)
# These values may be set per host via the ansible_module_compression inventory
# variable
#module_compression = 'ZIP_DEFLATED'
```

```
# This controls the cutoff point (in bytes) on --diff for files
# set to 0 for unlimited (RAM may suffer!).
#max_diff_size = 1048576

# This controls how ansible handles multiple --tags and --skip-tags arguments
# on the CLI.  If this is True then multiple arguments are merged together.  If
# it is False, then the last specified argument is used and the others are ignored.
# This option will be removed in 2.8.
#merge_multiple_cli_flags = True

# Controls showing custom stats at the end, off by default
#show_custom_stats = True

# Controls which files to ignore when using a directory as inventory with
# possibly multiple sources (both static and dynamic)
#inventory_ignore_extensions = ~, .orig, .bak, .ini, .cfg, .retry, .pyc, .pyo

# This family of modules use an alternative execution path optimized for network appliances
# only update this setting if you know how this works, otherwise it can break module execution
#network_group_modules=eos, nxos, ios, iosxr, junos, vyos

# When enabled, this option allows lookups (via variables like {{lookup('foo')}} or when used as
# a loop with `with_foo`) to return data that is not marked "unsafe". This means the data may contain
# jinja2 templating language which will be run through the templating engine.
# ENABLING THIS COULD BE A SECURITY RISK
#allow_unsafe_lookups = False

# set default errors for all plays
#any_errors_fatal = False

[inventory]
# enable inventory plugins, default: 'host_list', 'script', 'auto', 'yaml', 'ini', 'toml'
#enable_plugins = host_list, virtualbox, yaml, constructed

# ignore these extensions when parsing a directory as inventory source
#ignore_extensions = .pyc, .pyo, .swp, .bak, ~, .rpm, .md, .txt, ~, .orig, .ini, .cfg, .retry

# ignore files matching these patterns when parsing a directory as inventory source
#ignore_patterns=

# If 'true' unparsed inventory sources become fatal errors, they are warnings otherwise.
#unparsed_is_failed=False

[privilege_escalation]
#become=True
#become_method=sudo
#become_user=root
#become_ask_pass=False

[paramiko_connection]
```

```
# uncomment this line to cause the paramiko connection plugin to not record new host
# keys encountered.  Increases performance on new host additions.  Setting works independently of the
# host key checking setting above.
#record_host_keys=False

# by default, Ansible requests a pseudo-terminal for commands executed under sudo. Uncomment this
# line to disable this behaviour.
#pty=False

# paramiko will default to looking for SSH keys initially when trying to
# authenticate to remote devices.  This is a problem for some network devices
# that close the connection after a key failure.  Uncomment this line to
# disable the Paramiko look for keys function
#look_for_keys = False

# When using persistent connections with Paramiko, the connection runs in a
# background process.  If the host doesn't already have a valid SSH key, by
# default Ansible will prompt to add the host key.  This will cause connections
# running in background processes to fail.  Uncomment this line to have
# Paramiko automatically add host keys.
#host_key_auto_add = True

[ssh_connection]

# ssh arguments to use
# Leaving off ControlPersist will result in poor performance, so use
# paramiko on older platforms rather than removing it, -C controls compression use
#ssh_args = -C -o ControlMaster=auto -o ControlPersist=60s

# The base directory for the ControlPath sockets.
# This is the "%(directory)s" in the control_path option
#
# Example:
# control_path_dir = /tmp/.ansible/cp
#control_path_dir = ~/.ansible/cp

# The path to use for the ControlPath sockets. This defaults to a hashed string of the hostname,
# port and username (empty string in the config). The hash mitigates a common problem users
# found with long hostnames and the conventional %(directory)s/ansible-ssh-%%h-%%p-%%r format.
# In those cases, a "too long for Unix domain socket" ssh error would occur.
#
# Example:
# control_path = %(directory)s/%%h-%%r
#control_path =

# Enabling pipelining reduces the number of SSH operations required to
# execute a module on the remote server. This can result in a significant
# performance improvement when enabled, however when using "sudo:" you must
# first disable 'requiretty' in /etc/sudoers
#
```

```
# By default, this option is disabled to preserve compatibility with
# sudoers configurations that have requiretty (the default on many distros).
#
#pipelining = False

# Control the mechanism for transferring files (old)
#   * smart = try sftp and then try scp [default]
#   * True = use scp only
#   * False = use sftp only
#scp_if_ssh = smart

# Control the mechanism for transferring files (new)
# If set, this will override the scp_if_ssh option
#   * sftp  = use sftp to transfer files
#   * scp   = use scp to transfer files
#   * piped = use 'dd' over SSH to transfer files
#   * smart = try sftp, scp, and piped, in that order [default]
#transfer_method = smart

# if False, sftp will not use batch mode to transfer files. This may cause some
# types of file transfer failures impossible to catch however, and should
# only be disabled if your sftp version has problems with batch mode
#sftp_batch_mode = False

# The -tt argument is passed to ssh when pipelining is not enabled because sudo
# requires a tty by default.
#usetty = True

# Number of times to retry an SSH connection to a host, in case of UNREACHABLE.
# For each retry attempt, there is an exponential backoff,
# so after the first attempt there is 1s wait, then 2s, 4s etc. up to 30s (max).
#retries = 3

[persistent_connection]

# Configures the persistent connection timeout value in seconds.  This value is
# how long the persistent connection will remain idle before it is destroyed.
# If the connection doesn't receive a request before the timeout value
# expires, the connection is shutdown. The default value is 30 seconds.
#connect_timeout = 30

# The command timeout value defines the amount of time to wait for a command
# or RPC call before timing out. The value for the command timeout must
# be less than the value of the persistent connection idle timeout (connect_timeout)
# The default value is 30 second.
#command_timeout = 30

[accelerate]
#accelerate_port = 5099
#accelerate_timeout = 30
#accelerate_connect_timeout = 5.0
```

```
# The daemon timeout is measured in minutes. This time is measured
# from the last activity to the accelerate daemon.
#accelerate_daemon_timeout = 30

# If set to yes, accelerate_multi_key will allow multiple
# private keys to be uploaded to it, though each user must
# have access to the system via SSH to add a new key. The default
# is "no".
#accelerate_multi_key = yes

[selinux]
# file systems that require special treatment when dealing with security context
# the default behaviour that copies the existing context or uses the user default
# needs to be changed to use the file system dependent context.
#special_context_filesystems=nfs,vboxsf,fuse,ramfs,9p

# Set this to yes to allow libvirt_lxc connections to work without SELinux.
#libvirt_lxc_noseclabel = yes

[colors]
#highlight = white
#verbose = blue
#warn = bright purple
#error = red
#debug = dark gray
#deprecate = purple
#skip = cyan
#unreachable = red
#ok = green
#changed = yellow
#diff_add = green
#diff_remove = red
#diff_lines = cyan


[diff]
# Always print diff when running ( same as always running with -D/--diff )
# always = no

# Set how many context lines to show in diff
# context = 3
```

## ***Ansible inventory file(hosts)***

- Once we installed the Ansible, then Ansible to know which servers to connect to and manage.
- Ansible inventory hosts file is used to list and group your servers its default location is ***/etc/ansible/hosts***.
- Servers define in the inventory file as hostnames or IP addresses or FQDN.
- If we run ansible playbook OR adhoc command first it will look the default inventory file located on ***/etc/ansible/hosts.***

  ***ansible all -m ping*** (It will take default inventory file)
- We can specify particular inventory file by using -i option.

  ***ansible all -m ping -i ./hosts***
- Some important points in Inventory file.
  - Comments begin with the '#' character
  - Blank lines are ignored
  - Groups of hosts are delimited by [header] elements
  - You can enter hostnames or IP addresses or FQDN.
  - A hostname/IP addresses can be a member of multiple groups
- For localhost ansible_connection=localhost

## ***List of parameters in Inventory file***

***ansible_connection:***

Connection type to the host. This can be the name of any of ansible's connection plugins. SSH protocol types are smart, ssh or paramiko. The default is smart. Non-SSH based types are described in the next section.

ansible_connection=ssh

For localhost ansible_connection=localhost

***ansible_host:***

The name of the host to connect to,if different from the alias you wish to give to it.

***ansible_port:*** The ssh port number, if not 22

***ansible_user:*** The default ssh user name to use.

***ansible_ssh_pass:***

The ssh password to use (this is insecure, we strongly recommend using --ask-pass or SSH keys)

***ansible_ssh_private_key_file:***

Private key file used by ssh. Useful if using multiple keys and you don't want to use SSH agent.

***ansible_ssh_common_args:***

This setting is always appended to the default command line for sftp, scp, and ssh. Useful to configure a ProxyCommand for a certain host (or group).

***ansible_sftp_extra_args:***

This setting is always appended to the default sftp command line.

***ansible_scp_extra_args:***

This setting is always appended to the default scp command line.

***ansible_ssh_extra_args:***

This setting is always appended to the default ssh command line.

***ansible_ssh_pipelining:***

Determines whether or not to use SSH pipelining. This can override the pipelining setting in ansible.cfg.

***ansible_become:***

Equivalent to ansible_sudo or ansible_su, allows to force privilege escalation

***ansible_become_method:***

Allows to set privilege escalation method

***ansible_become_user:***

Equivalent to ansible_sudo_user or ansible_su_user, allows to set the user you become through privilege escalation

***ansible_become_pass:***

Equivalent to ansible_sudo_pass or ansible_su_pass, allows you to set the privilege escalation password

***ansible_shell_type:***

The shell type of the target system. You should not use this setting unless you have set the ansible_shell_executable to a non-Bourne (sh) compatible shell. By default commands are formatted using sh-style syntax. Setting this to csh or fish will cause commands executed on target systems to follow those shell's syntax instead.

***ansible_python_interpreter:***

The target host python path. This is useful for systems with more than one Python or not located at /usr/bin/python such as *BSD, or where /usr/bin/python is not a 2.X series Python. We do not use the /usr/bin/env mechanism as that requires the

remote user's path to be set right and also assumes the python executable is named python, where the executable might be named something like python2.6.

***ansible_*_interpreter:***

Works for anything such as ruby or perl and works just like ansible_python_interpreter. This replaces shebang of modules which will run on that host.

***ansible_shell_executable:***

This sets the shell the ansible controller will use on the target machine, overrides executable in ansible.cfg which defaults to /bin/sh. You should really only change it if is not possible to use /bin/sh (i.e. /bin/sh is not installed on the target machine or cannot be run from sudo.). New in version 2.1.

## Examples

### 1. Hostname, Username and connection

```
#18.191.209.103   ansible_user=ec2-user      ansible_connection=ssh
18.191.209.103    ansible_user=ansadmin      ansible_connection=ssh
18.219.5.26       ansible_user=ansadmin      ansible_connection=ssh
```

### Testing

```
ansible all -m ping
```

### 2. Alias name, Hostname, Username and connection

```
#a1 ansible_host=18.191.209.103 ansible_user=ubuntu   ansible_connection=ssh
s1 ansible_host=18.191.209.103 ansible_user=ansadmin   ansible_connection=ssh
s2 ansible_host=18.219.5.26      ansible_user=ansadmin   ansible_connection=ssh
s3 ansible_host=18.191.209.103 ansible_user=ansadmin   ansible_connection=ssh
s4 ansible_host=18.219.5.26      ansible_user=ansadmin   ansible_connection=ssh
```

### Testing

```
ansible s1,s2 -m ping
ansible all -m ping
```

### 3. *Passing password without exchanging key*

```
18.191.209.103      ansible_user=ansadmin
18.219.5.26         ansible_user=ansadmin
13.59.124.116       ansible_user=ansadmin   ansible_ssh_pass=ansadmin
```

***Testing***

```
ansible all -m ping
```

### 4. *Passing private key*

```
18.191.209.103      ansible_user=ansadmin
18.219.5.26         ansible_user=ansadmin
13.59.124.116       ansible_user=ansadmin
ansible_ssh_private_key_file=/home/ansadmin/<private-key>
```

***Testing***

```
ansible all -m ping
```

### 5. *Groups*

```
[webservers]
s1 ansible_host=18.191.209.103 ansible_user=ansadmin    ansible_connection=ssh
s2 ansible_host=18.219.5.26    ansible_user=ansadmin    ansible_connection=ssh
s3 ansible_host=18.191.209.103 ansible_user=ansadmin    ansible_connection=ssh
s4 ansible_host=18.219.5.26    ansible_user=ansadmin    ansible_connection=ssh
```

***Testing***

```
ansible all -m ping
ansible webservers -m ping
```

***To run specific server (s3) in the group***

```
ansible webservers[2] -m ping -i ./myhosts
```

***To run range of servers (s1-s3) in the group***

```
ansible webservers[0:2] -m ping -i ./myhosts
```

### *6. Group with Vars*

```
[webservers]
s1 ansible_host=18.191.209.103
s2 ansible_host=18.219.5.26
s3 ansible_host=18.191.209.103
s4 ansible_host=18.219.5.26
[webservers:vars]
ansible_user=ansadmin
ansible_connection=ssh
```

***Testing***

ansible all -m ping

ansible webservers -m ping

***To run specific server (s3) in the group***

ansible webservers[2] -m ping -i ./myhosts

***To run range of servers (s1-s3) in the group***

ansible webservers[0:2] -m ping -i ./myhosts

### *7. Multiple Groups with Vars*

```
[webservers]
s1 ansible_host=18.191.209.103
s2 ansible_host=18.219.5.26
s3 ansible_host=18.191.209.103
s4 ansible_host=18.219.5.26

[webservers:vars]
ansible_user=ansadmin
ansible_connection=ssh

[dbservers]
d1 ansible_host=18.191.209.103
d2 ansible_host=18.219.5.26
d3 ansible_host=18.191.209.103
d4 ansible_host=18.219.5.26
```

[dbservers:vars]
ansible_user=ansadmin
ansible_connection=ssh

***Testing***

ansible all -m ping
ansible webservers -m ping
ansible dbservers -m ping

***To run specific webserver (s3) in the group***

ansible webservers[2] -m ping -i ./myhosts

***To run range of webservers (s1-s3) in the group***

ansible webservers[0:2] -m ping -i ./myhosts

***To run specific webserver (d3) in the group***

ansible dbservers[2] -m ping -i ./myhosts

***To run range of webservers (d1-d3) in the group***

ansible dbservers[0:2] -m ping -i ./myhosts

***8. Merging groups of groups***

[webservers]
s1 ansible_host=18.191.209.103
s2 ansible_host=18.219.5.26
s3 ansible_host=18.191.209.103
s4 ansible_host=18.219.5.26

[dbservers]
d1 ansible_host=18.191.209.103
d2 ansible_host=18.219.5.26
d3 ansible_host=18.191.209.103
d4 ansible_host=18.219.5.26

[webdb:children]
webservers
dbservers

[webdb:vars]

ansible_user=ansadmin
ansible_connection=ssh

## ***Testing***

ansible all -m ping
ansible webservers -m ping
ansible dbservers -m ping

***To run specific webserver (s3) in the group***

ansible webservers[2] -m ping -i ./myhosts

***To run range of webservers (s1-s3) in the group***

ansible webservers[0:2] -m ping -i ./myhosts

***To run specific webserver (d3) in the group***

ansible dbservers[2] -m ping -i ./myhosts

***To run range of webservers (d1-d3 && s1-s3) in the different groups***

ansible dbservers[0:2]: webservers[0:2]  -m ping -i ./myhosts

```
# Consolidation of all groups
[hosts:children]
web-servers
offsite
onsite
backup-servers

[web-servers]
server1 ansible_host=192.168.0.1 ansible_port=1600
server2 ansible_host=192.168.0.2 ansible_port=1800

[offsite]
server3 ansible_host=10.160.40.1 ansible_port=22 ansible_user=root
server4 ansible_host=10.160.40.2 ansible_port=4300 ansible_user=root

# You can make groups of groups
[offsite:children]
backup-servers

[onsite]
server5 ansible_host=10.150.70.1 ansible_ssh_pass=password

[backup-servers]
server6 ansible_host=10.160.40.3 ansible_port=77
```

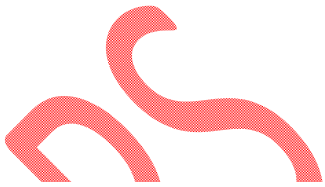
## ***General for all connections***

ansible_host
The name of the host to connect to, if different from the alias you wish to give to it.

ansible_port
The ssh port number, if not 22

ansible_user
The default ssh user name to use.

## Specific to the SSH Connection



ansible_ssh_pass
The ssh password to use (never store this variable in plain text; always use a vault)

ansible_ssh_private_key_file
Private key file used by ssh. Useful if using multiple keys and you don't want to use SSH agent.

ansible_ssh_common_args
This setting is always appended to the default command line for sftp, scp, and ssh. Useful to configure a ProxyCommand for a certain host (or group).

ansible_sftp_extra_args
This setting is always appended to the default sftp command line.

ansible_scp_extra_args
This setting is always appended to the default scp command line.

ansible_ssh_extra_args
This setting is always appended to the default ssh command line.

ansible_ssh_pipelining
Determines whether or not to use SSH pipelining. This can override the pipelining setting in ansible.cfg.

ansible_ssh_executable (added in version 2.2)
This setting overrides the default behavior to use the system ssh. This can override the ssh_executable setting in ansible.cfg.

## Privilege Escalation

ansible_become
Equivalent to ansible_sudo or ansible_su, allows to force privilege escalation

ansible_become_method
Allows to set privilege escalation method

ansible_become_user
Equivalent to ansible_sudo_user or ansible_su_user, allows to set the user you become through privilege escalation

ansible_become_pass
Equivalent to ansible_sudo_pass or ansible_su_pass, allows you to set the privilege escalation password (never store this variable in plain text; always use a vault.)

ansible_become_exe
Equivalent to ansible_sudo_exe or ansible_su_exe, allows you to set the executable for the escalation method selected

ansible_become_flags
Equivalent to ansible_sudo_flags or ansible_su_flags, allows you to set the flags passed to the selected escalation method. This can be also set globally in ansible.cfg in the sudo_flags option

## Ansible ad-hoc/ Arbitary Commands

- Ansible ad-hoc commands are powerful managing complex configurations using ad-hoc commands are critical job rather than we should use play books. Generally we can use ad-hoc commands for one time operation.

***Ex:***

1. Checking server status, getting data one time …..Etc.
2. Just validate the uptime of 1 to 200 remote servers
3. Just get the disk space of remote hosts
4. Ping and validate if the server is alive and responds
5. shutdown multiple remote hosts at a single command

- Ad-hoc commands are one of the simplest ways of using Ansible. These are used when you want to issue some commands on a server or bunch of servers. The ad-hoc commands are not stored for future use, but it represents a fast way to interact with the desired servers.
- The Ansible ad-hoc command uses the ***/usr/bin/ansible*** command-line tool to automate a single task on one or more managed nodes. The Ad-hoc commands

are quick and easy, but they are not re-usable. The Ad-hoc commands demonstrate the simplicity and power of Ansible.

***Syntax:***

```
ansible <hosts> [-m <module_name>] -a <"arguments"> -u <username> [--become]
```

***Hosts:*** It can be an entry in the inventory file. For specifying all hosts in the inventory, use all or "*".

***module_name:*** It is an optional parameter. There are hundreds of modules available in the Ansible, such as **shell, yum, apt, file,** and **copy**. By default, it is the **command**.

***arguments:*** We should pass values that are required by the module. It can change according to the module used.

***username:*** It specifies the user account in which Ansible can execute commands.

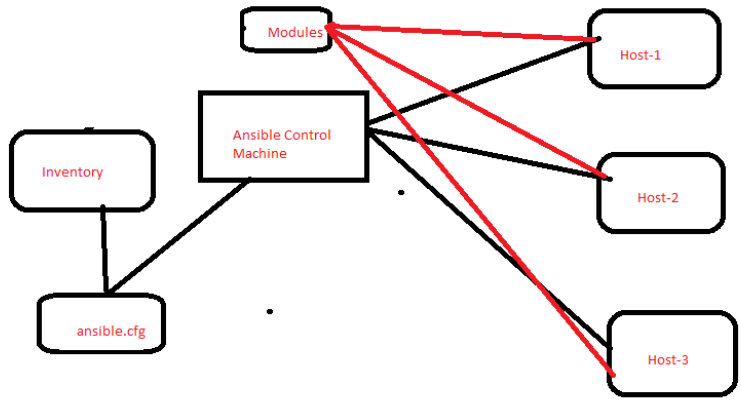
***become:*** It's an optional parameter specified when we want to run operations that need sudo privilege. By default, it becomes false.

## *Examples*

ansible all -m ping -i ./hosts
ansible all -m ping -i ./hosts –o
ansible -i ./hosts all -m shell -a date
ansible -i ./hosts all -m shell -a 'mount'
ansible-doc –l
ansible-doc yum
ansible -i ./hosts all -m shell -a 'uptime'
ansible -i ./hosts all -a 'uptime'
ansible -i ./hosts all -m shell -a 'service sshd status' –b
ansible -i ./hosts all -m shell -a 'uname -a' -b
ansible -i ./hosts all -a "ls /tmp"
ansible -i ./hosts all -a "yum update" -u ansadmin --become -K
ansible -i ./hosts all -a "yum update" -u ansadmin -b
ansible -i ./hosts all -m copy -a "src=test.txt dest=/home/ansadmin"
ansible -i ./hosts all -m synchronize -a "src=test.txt dest=/tmp/host.txt"
ansible -i ./hosts all -m fetch -a "src=/home/ansadmin/ppreddy.txt dest=/home/ansadmin/ppreddy.txt"

ansible -i ./hosts all -m file -a "path=/home/ansadmin/devops state=directory"
ansible -i ./hosts all -m file -a "path=/home/ansadmin/devops state=absent"
ansible -i ./hosts all -m file -a "path=/home/ansadmin/devops/devops.txt state=touch"
ansible -i ./hosts all -m file -a "path=/home/ansadmin/devops/devops.txt state=absent"
ansible -i ./hosts all -m group -a "name=devops state=present" -b
ansible -i ./hosts all -m user -a "name=ppreddy group=devops createhome=yes password=ppreddy" -b
ansible -i ./hosts all -m user -a "name=ppreddy group=devops state=absent" -b
ansible -i ./hosts all -m command -a "free -m"
ansible -i ./hosts all -m git -a "repo=https://github.com/psddevops/sampletest.git dest=/home/ansadmin/pp"
ansible -i ./hosts all -m yum -a "pkg=httpd state=present" -b
ansible -i ./hosts all -m yum -a "pkg=httpd state=latest" -b
ansible -i ./hosts all -m yum -a "pkg=httpd state=absent" -b
ansible -i ./hosts all -m service -a "name=httpd state=started" -b
ansible -i ./hosts all -m service -a "name=httpd state=stopped" -b
ansible -i ./hosts all -m service -a "name=httpd state=restarted" –b
ansible -i ./hosts all -m shell -a "sleep:5; echo 'hi'"
ansible -i ./hosts all -m setup

## ***How ansible executes the commands:***



Ansible Control → ansible.cfg → inventory → ssh connectivity for all servers.
Module  copied to the host machines and executes the respective modules.

Location: <home-path>/.ansible/tmp/<command>

ANSIBLE_KEEP_REMOTE_FILES=1

ANSIBLE_KEEP_REMOTE_FILES=1  ansible -i ./hosts all -m shell -a "sleep:5; echo 'hi'"

## ***How to run one Task Host by Host?***

By default forks=5(five servers at a time) in ansible.cfg file update to forks=1 (execute one by one)

ansible-playbook webserver.yml --forks=1

***Ex***:  22 servers ( 5 servers + 5 servers +5 servers +5 servers +2 servers)

## ***Ansible-Facts/Gathering Facts***

- Ansible gather certain information from remote/managed nodes this process is called ansible-facts OR Gathering facts.
- Ansible uses ***setup module*** to discover this information automatically. Sometime this information is required in playbook as this is dynamic information fetched from remote systems.
- Ansible-facts OR Gathering facts is a time consuming process in Ansible as it has to gather information from all the hosts listed in your inventory file. We can avoid this situation and speed up our play execution by specifying ***gathering_facts*** to false in playbook.
- We can also filter the facts gathering to save some time. This case is mainly useful when you want only hardware or network information that you want to use in your playbook. So rather than asking for all facts, we can minimize this by only asking network or hardware facts to save some time.

There are two types of facts.

1. Default Facts.
2. Custom Facts

Default Facts are nothing but information about Managed Nodes like: os distribution, release, processor, python etc... The task of collecting this remote system information is called as Gathering Facts, and collected/Gathered information is called facts or variable. You can Gather/Collect Facts using setup module in Ad-hoc commands.

Note: Ansible Playbooks call this setup module by default to perform Gathering Facts task.

**Custom Facts**

- To get user defined required tasks
- Need to get the server versions like git, httpd, weblogic …. Etc.
- To identify prod/non-prod/dev/qa servers.

Step-1: Create /etc/ansible/facts.d on Managed Nodes

Step-2: Inside of facts.d place one more custom facts files with extension as .fact

Step-3: The output of fact file should be a json.

Step-4: The fact file should have execution permission.

Ex:

```
#!/bin/bash

git_ver=$(git --version | awk '{ print $3 }')
httpd_ver=$(httpd -version | awk ' NR==1 {print $3}')

cat << EOF
{ "Git_Version": "$git_ver"
  "httpd_version": "$httpd_ver"
}
EOF
```

***Example:***

```
ansible -i ./hosts all -m setup
ansible -i ./hosts all -m setup -a 'filter=*ipv4*'
```

```
ansible -i ./hosts all -m setup -a 'filter=ansible_domain'
ansible -i ./hosts all -m setup -a 'filter=ansible_nodename'
ansible -i ./hosts all -m setup -a 'filter=ansible_mounts'
```

## ***Why Ansible is idempotent?***

If we run commands more than one time it will run but it won't effect. If we perform **'n'** number of operations it won't effect so it's called ansible is idempotent.

***Example***

Installing packages more than one time it's not effect it will run successfully.

## ***Ansible Modules***

- Ansible modules are a small set of programs that perform a specific task. Modules can be used to automate a wide range of tasks. which can be used from the command line or in a playbook task.
- Modules in Ansible are considered to be idempotent or in other words, making multiple identical requests has the same effect as making a single request.
- The modules also referred to as task plugins or library plugins in the Ansible.
- Ansible ships with several modules that are called **module library**, which can be executed directly or remote hosts through the playbook.
- Users can also write their modules. These modules can control like **services, system resources, files,** or **packages,** etc. and handle executing system commands.
- Each module supports taking arguments. Mainly all modules take **key=value** arguments, space delimited.
- Some module takes no arguments, and the shell/command modules take the string of the command which you want to execute.
- Technically, all modules return **JSON** format data, though command line or playbooks, you don't need to know much about that. If you're writing your module, it means you do not have to write modules in any particular language which you get to choose.
- Modules should be idempotent and avoid making any changes if they detect that the current state matches the desired final state. When using Ansible

playbooks, these modules can trigger "**change events**" in the form of notifying "**handlers**" to run additional tasks.

There are 2 types of modules in Ansible:

1. Core modules
2. Extras modules

## ***Core Modules***

These are modules that the core Ansible team maintains and will always ship with Ansible itself. They will also receive a slightly higher priority for all requests than those in the “extras” repos. The source of these modules is hosted by Ansible on GitHub in the Ansible-modules-core.

## ***Extras Modules***

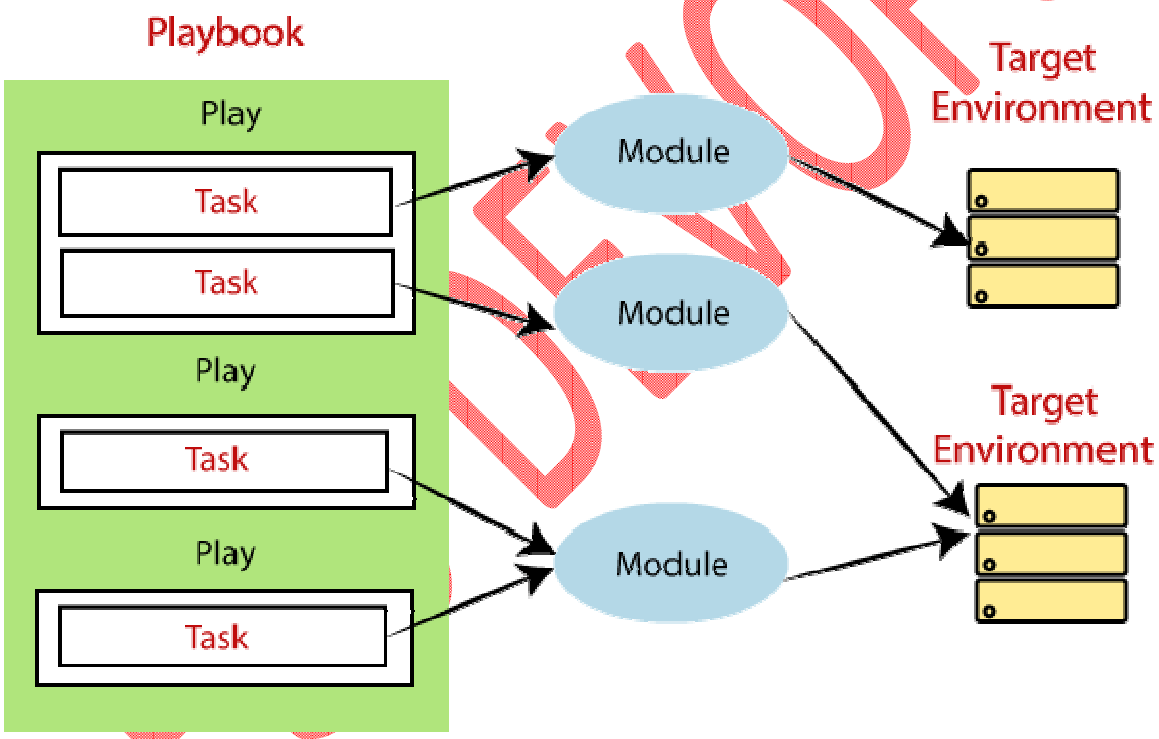
These modules are currently shipped with Ansible but might be shipped separately in the future. They are also mostly maintained by the Ansible Community. Non-core modules are still fully usable but may receive slightly lower response rates for issues and pull requests.Popular “extras” modules may be promoted to core modules over time. The source for these modules is hosted by Ansible on GitHub in the Ansible-modules-extras.

## ***Ansible Playbooks***

- Ansible playbook is YAML (Yet Another Markup Language) document, which as set of plays. A play is specific task. It uses the ***/usr/bin/ansible-playbook*** command to execute the playbooks.
- Ansible playbooks are a way to send commands to remote computers in a scripted way. Instead of using Ansible commands individually to remotely configure computers from the command line, you can configure entire complex environments by passing a script to one or more systems.
- Ansible playbooks are written in the YAML data serialization format. If you don't know what a data serialization format is, think of it as a way to translate a programmatic data structure (lists, arrays, dictionaries, etc) into a format that can be easily stored to disk. The file can then be used to recreate the structure at a later point. JSON is another popular data serialization format, but YAML is much easier to read.

- Each playbook contains one or more plays, which map hosts to a certain function. Ansible does this through something called tasks, which are basically module calls.
- A task can be anything like
  - Creating Linux user OR Directory
  - Updating configuration files.
  - Install/Update the packages
  - Start/Stop the services .....Etc.

## ***Playbook Structure***




- Each playbook is a collection of one or more plays. Playbooks are structured by using Plays. There can be more than one play inside a playbook.
- The function of the play is to map a set of instructions which is defined against a particular host.

## Ad hoc command Vs Playbook

Ansible can be used in either Ad-Hoc or Playbook mode.The ad-hoc mode allows direct management of your hosts by executing single line commands and leveraging Ansible modules. Each Ansible playbook contains one or more plays to help you to perform functions on different host

## Playbook Sections

Ansible playbooks are divided into 4-sections.

1. *Target Section*
2. *Variable Section*
3. *Task Section.*
4. *Handler Section.*

### Target Section:

In this section we have defined hosts against which playbook task has to execute.

### Variable Section:

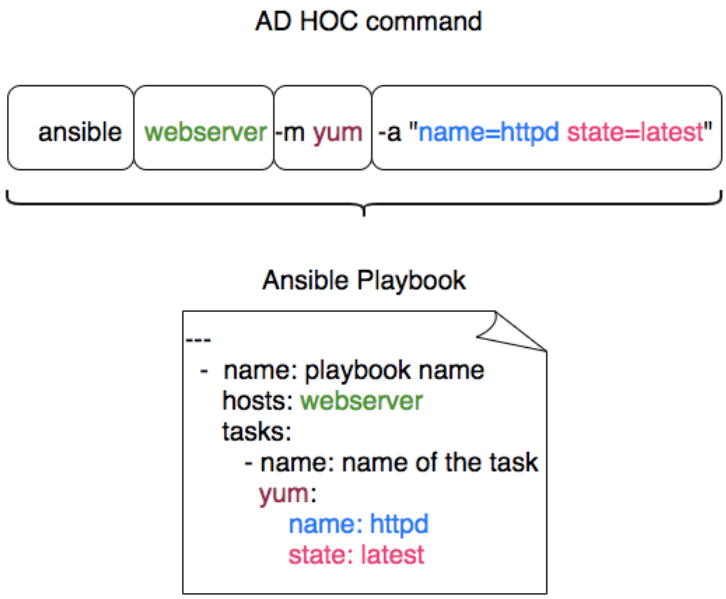
In this section we have defined the variables used by plays.

### Task Section:

In this section we can list of all modules that we need to run in the order.

### Handler Section:

Handlers are ansible plays this are executed only when someone notifies.



```
---
# Target Section
- name: Playbook For Installing httpd service
  hosts: webservers
  become: yes
  become_user: root
  connection: ssh
  gather_facts: False

# Variable Section:
  vars:
    pkg_name: httpd

# Task Section:
  tasks:
    - name: ensure apache is at the latest version
      yum:
        name: ' {{ pkg_name }} '
        state: latest
      notify: restart httpd

# Handler Section:
  handlers:
    - name: restart httpd
      service:
        name: httpd
        state: restarted
```

## ***YAML Scripting***

- YAML, short for Ain't Markup Language.
- YAML is a data serialization language designed to be directly writable and readable by humans.
- It is commonly used for configuration files, but it is used in many applications where data is being stored and transmitted over the network.
- Strictly speaking YAML is a superset of json with additional features like new line and indentation.

- The YAML is a scripting language, means we can communicate with other languages using YAML. Other languages means, it may be OS, any applications or services running on OS.

- There are three editions in YAML scripting.
  - 1.2 -- third edition (Latest Version)
  - 1.1 -- second edition
  - 1.0 -- first edition
- YAML script extension.

  .yaml

  .yml
- YAML is a case sensitive scripting language.
- YAML does not allow the use of tabs for indentation like python.
- But it allows use of space for indentation instead of tabs.

## ***Keys and Data Types in YAML***

### ***Key/Variable***

- In playbooks, the variable is very similar to using the variables in a programming language
- Key/Variable is used to store any value and that value can be change depending on condition in any language.

*Ex:*

Name:Ravi

Age:25

x:25

### ***Data Types***

Before start using variables, it's important to know what valid variable names. Variable names should be letters, numbers,spaces and underscores. The variable should always start with a letter.

- There are different types of keys or data types, based on which type of data we are storing into a key.

*Ex*

```
my_integer: 25
key with space:45
my_string: "yaml scripting"
my_string: yaml scripting
my float: 25.0
my boolean: true
null value: null
```

***Data Collections***

- Generally we will represent data in YAML like key and its value pair (key: value) representation this is also called scalar representation of data. This representation is rarely used in real time.

```
Name:Raja
Age:25
my value: 12
my script: "YAML"
```

Instead of single value for a key we will use more values for a *single key* OR *multiple key value pair* this is called data collection.

They are two type's data collections.

1. Sequence data collection
2. Map data collection

**Sequence representation:**

- It is equivalent to list or arrays in python/Java language

*Ex:*

```
cricket_players:
- sachin
- dhoni
- kohli
- rohit
```

Note: Which is equivalent to python

cricket_players = ['sachin', 'dhoni', 'kohli', 'rohit']

- We can represent any value as scalar

*Ex:*

```
cricket_players:
- sachin: 46
- dhoni: 37
- kohli: 31
- rohit: 32
```

## Map data collection

- It is equivalent to dictionaries in python OR more than one scalar representation.

Map representation:

```
cricket_players:
  sachin: 46
  dhoni: 37
  kohli: 31
  rohit: 32
```

```
cricket_players:
  sachin: 46
  dhoni: 37
  kohli:
      - Captain
      - Age: 31
  rohit: 32
```

Note: Which is equivalent to python

cricket_players={'sachin': 46, `dhoni': 37, `kohli': 31; rohit': 32}

## Ordered/Unordered collections

- Sequences are ordered collections.
- Maps are unordered collections.

Ex:

Team members:

- personl
- person2

Team members:

- person2
- personl

- Sequence collections are not equal in YAML script

Team details:

personl: 27

person2: 33

Team details:

person2: 33

personl: 27

- Maps collections are equal in YAML script

- Ansible playbook is used to send commands to remote servers in a scripted way instead of using ansible commands individually to configure remote servers from command line.
- To understand playbooks:
  - Task
  - Play
  - Playbook

Task:

Install vim editor on server.

yum install vim -y

Play:

Collection of tasks running on one or more servers.

Create a directory under user home.

mkdir -p /home/userl/new_dir r

Create an empty file under newly created directory.

touch /home/userl/new didempty_file

## Playbook:

Composed of one or more plays.

Server-1,2

Create a directory under user home.

mkdir -p /home/userl/new_dir

Create an empty file under newly created directory.

touch /home/userl/new_dir/empty_file

Server-3,4

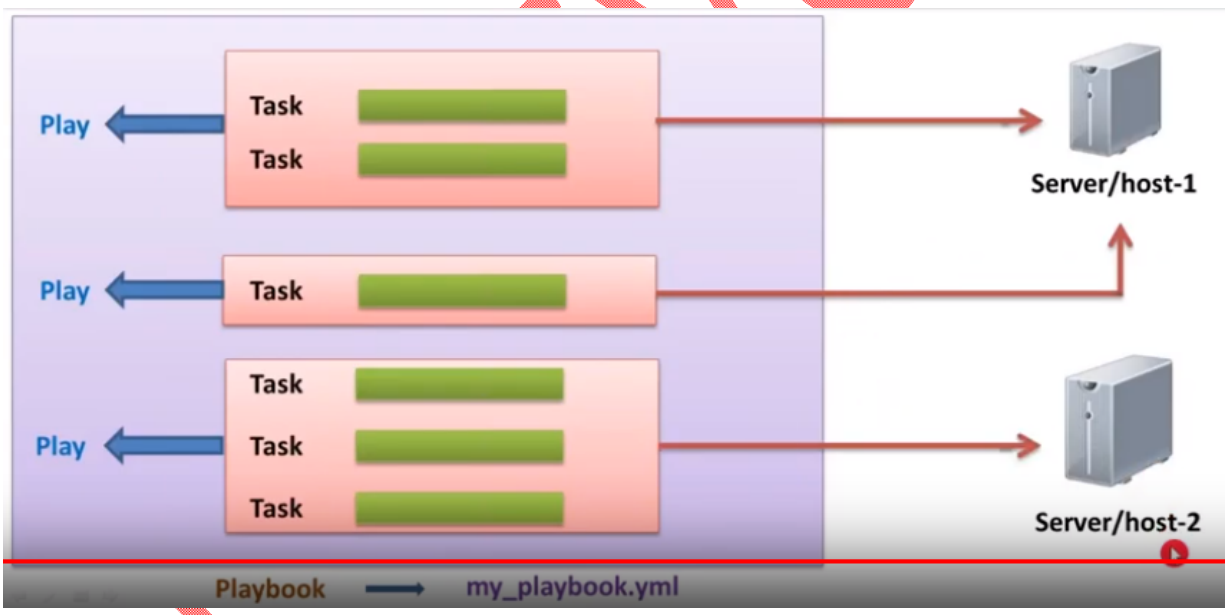
Create a directory under user home.

mkdir -p /home/userl/new_dir

Create an empty file under newly created directory.

touch /home/userl/new_dir/empty_file

## Relation between Task, Play and Playbook




## Basic Steps to write Playbook

Step1: starts with --- Which represents that file is a yaml file or yaml script.

Step2: Target section list (Like hosts, user etc...)

Step3: Variables list (optional)

Step4: Tasks list List all the modules that you run, in the order.

Step5: handlers list (optional)

Note: The file extension should be .yaml or .yml

These are the steps for one play.

Each play is a sequence, and sequence values are maps.

Ex:

```
---
- name: This is play-1
  hosts: webservers
  vars:
    my_content: "This file created using vars concept placing webservers...../n"
  tasks:
    - name: Adding the variable value
      copy:
        dest: /tmp/var_file_web.txt
        content: "{{ my_content }}"

- name: This is play-2
  hosts: appservers
  vars:
    my_content: "This file created using vars concept placing appservers ...../n"
  tasks:
    - name: Adding the variable value
      copy:
        dest: /tmp/var_file_app.txt
        content: "{{ my_content }}"
```

**Write Playbook with variables**

Variable lists are represented with key

Rules to Define Variables in playbook yaml scripting:

Variables are always either sequence(list) or maps (dictionaries)

vars:

varl: "This is first variable"

var2: "This is second variable"

**Ex:**

My task is: create a file in remote location with some content How to write this script

Stepl: Follow the starting syntax of playbook yaml script.

Step2: Start your each play as a sequence or list.

Step3: Each play is a map,So you can define required things like

- hosts
- variables
- tasks
- handlers

Ex:

```
- hosts: app_servers
  vars:
    my_content: "This file created using var concept"
  tasks:
  - copy:
      dest: /tmp/var_file.txt
       content: "{{ my_content }}"
```

## Reading and printing variable value from command line

### Reading

With the help of vars_promt selection

Syntax is:

```
vars_prompt:
  name: varl
  prompt: Enter your value ?
```

### Printing

Printing a value is like a task in playbook.

Use debug module for this task.

Debug module is having one argument that is msg

```
---
 - name: This is for prompting variables
   hosts: all
   vars_prompt:
     name: varl
     prompt: Enter any value ?
   tasks:
     - name: This is used to print a variable value
       debug:
         msg: "The value of your variable is: {{ varl }}"
```

```
---
 - name: This is for prompting variables
   hosts: all
   vars_prompt:
     - name: username
       prompt: "Please enter the username ?"
       private: no

     - name: password
       prompt: "Please enter the password ?"
       private: yes

   tasks:
     - name: username && password is
       debug:
         msg: "The value of Username and Password variable is: {{ username }} and {{
password }}"
```

**Command line variables**

passing variables from command line

```
ansible-playbook <playbook_name> -e "var1=value var2=value"
```

accessing variables from command line

We can access like normal variable script

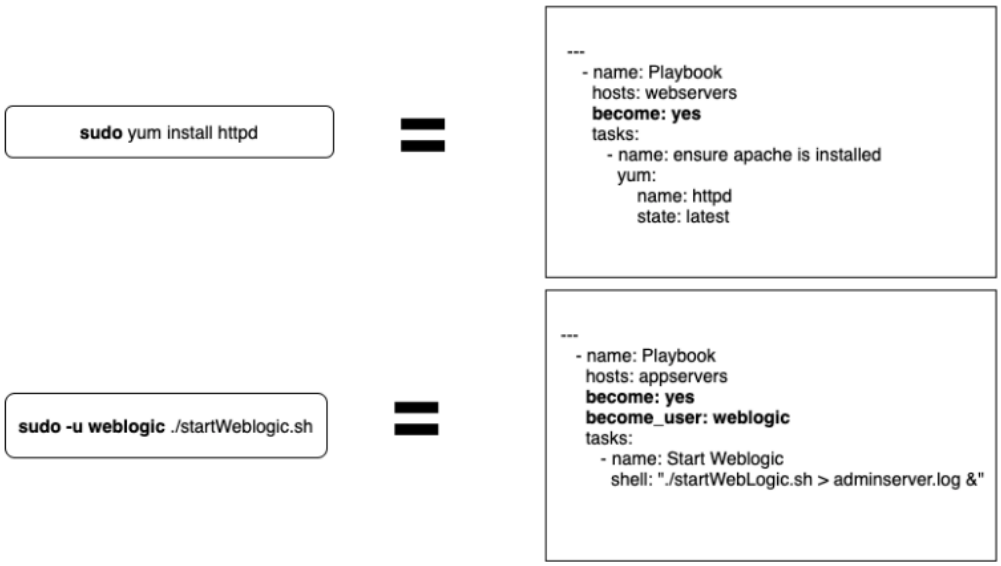
```
- name: This is a play
  hosts: all
  tasks:
    - name: Printing/Reading the values from prompt
      debug:
        msg: The value of varl= {{var1}} and var2={{var2}}"
```

**From different file**

```
- hosts: all
  vars_files:
    - "./vars.yml"
  tasks:
    - name: Adding the variable value
      copy:
        dest: /tmp/var_file1.txt
        content: "{{ my_content }}"
```

## *Ansible sudo – Ansible become*

- Ansible sudo or become is a method to run a particular task in a playbook with Special Privileges like root user or some other user.
- In the earlier versions of ansible there is an option named as **sudo** which is deprecated now, Since ansible 2.0 there are two new options named as **become** and **become_user**
- For Example If you want to run a task on the remote server to install some packages using yum. It's very obvious that you should become root user as Non-Root user cannot install packages, in this case, you can use ansible sudo. to be precise ansible **become** method.
- The requirement where you have to start a WebSphere application server instance on the remote machine which runs as wsadmin user. In this case, you might want to restart the instance as wsadmin only for which you can execute your ansible task with **become-user** method
- become and become_user both have to be used in a playbook in certain cases where you want your remote user to be non-root.it is more like doing sudo -u someuser before running a task.
- When you are not defining the become_user and just use become. Ansible will perform the basic sudo and it will execute the corresponding task as root user




***Examples***

***Host Section Level***

*Run a Task with sudo – Ansible become*

```
---
 - name: Playbook
   hosts: webservers
   become: yes
   become_user: root
   tasks:
    - name: ensure apache is at the latest version
      yum:
        name: httpd
        state: latest
    - name: ensure apache is running
      service:
        name: httpd
        state: started
```

*Run a Task with sudo -u  – Ansible become  user*

```
- name: Start the AdminServer
  tags: startadminserver
  become: yes
  become_user: weblogic
  shell: "./startWebLogic.sh > adminserver.log &"
  args:
    chdir: "{{oracle_home}}/domains/{{domain_name}}/bin/"
  register: startadminserver
  run_once: yes
  when: ansible_hostname == "{{ groups['app'][0] }}" and amprevalidate is failed
  environment:
        USER_MEM_ARGS: "-Djava.security.egd=file:/dev/./urandom"
```

***Task Section Level***

Ansible Become Root OR Ansible Become True

```
---
  - name: Playbook
    hosts: webservers
    tasks:
      - name: ensure apache is at the latest version
        yum:
          name: httpd
          state: latest
      - name: ensure apache is running
        service:
          name: httpd
          state: restarted
        become: yes
```

This task will be executed as the root user. If you mention become: yes in the playbook tasks, the tasks will be executed as the default root user.  Because root is the default user for privilege escalation.

Ansible Become User

```
---
 - name: Playbook
   hosts: webservers
   tasks:
     - name: ensure apache is at the latest version
       yum:
         name: httpd
         state: latest
     - name: ensure apache is running
       service:
         name: httpd
         state: restarted
       become: yes
       become_user: root
```

This task will be executed as user apache because the user is explicitly set. So if you want to run the task as different sudo user mention become yes and become_user. If you mention only become_user in the task, it will not do anything with become_user, because become is not set and become default value is false/no.

## *Ansible Dry Run/Check Mode (Run Playbook in Ansible Check mode)*

- Ansible Dry Run or Ansible Check mode feature is to check your playbook before execution like Ansible's **--syntax-check** feature.
- With Ansible Dry Run feature you can execute the playbook without having to actually make changes on the server. With Ansible Dry Run you can see if the host is getting changed or not.

Syntax:

ansible-playbook <playbook-name> --check or -C

Examples

apache.yml

```
---
 - name: Playbook
   hosts: webservers
   become: yes
   become_user: root
   tasks:
    - name: ensure apache is at the latest version
      yum:
        name: httpd
        state: latest
    - name: ensure apache is running
      service:
        name: httpd
        state: started
```

ansible-playbook apache.yml --check

OR

ansible-playbook apache.yml -C

**Note:**

When you have conditional or result based task execution in the playbook. The dry run would fail. (One pkg depends on other ex: java and tomcat installations)

## *Ansible Debug and Register modules*

- Ansible **debug module** is used to print the message in the log output. The message is nothing but any variable values or output of any task.
- Ansible **register variable** or ansible **register module** is used to capture or store the output of the command or task. By using the register module, you can store that output into any variable.
- Whenever you executed any task or commands ansible will give you some return values like.

  - changed
  - cmd

- delta
- end
- failed
- rc
- start
- stderr
- stderr_lines
- stdout
- stdout_lines

These return values depend on the task you executed.

***Examples***

*Printing Variable Value*

```
- hosts: localhost
  vars:
    my_content: "This file created using var concept"
  tasks:
    - name: Printing the variable vaalue
      debug:
        var: my_content
```

*Printing Variable Value with Adding Some Extra Message*

```
- hosts: localhost
  vars:
    my_content: "This file created using var concept"
  tasks:
    - name: Printing the variable vaalue
      debug:
        msg: "Hi This Is  {{ my_content }}"
```

*Ansible Debug Msg with Examples*

```
---
- hosts: localhost
  gather_facts: yes
    tasks:
   - name: executing sample command
    shell: echo "PSD DevOps"
    register: data

  - name: printing variable
    debug:
      var: data

  - name: printing variable with stdout
    debug:
      var: data.stdout

  - name: printing variable with adding some extra message
    debug:
      msg: "Hi This Is  {{ data.stdout }}"
```

*Ansible register module with an example*

```
---
- hosts: webservers
  gather_facts: no
  tasks:
  - name: starting httpd
    service: name=httpd state=started enabled=yes

  - name: httpd status
    command: service httpd status
    register: httpd_status

  - name: httpd status output
    debug:
      var: httpd_status
```

*Ansible playbook with stdout_lines*

```
---
- hosts: servers
  gather_facts: no
  tasks:
  - name: starting httpd
    service: name=httpd state=started enabled=yes

  - name: httpd status
    command: service httpd status
    register: httpd_status

  - name: httpd status output with stdout
    debug:
      var: httpd_status.stdout_lines
```

## ***Ansible delegate_to and run_once modules***

- If you want to run any task on any particular machine among multiple hosts we can use ansible delegate_to module.
- If you want to run any task on first machine among multiple hosts we can use ansible run_once module.

### ***Examples***

```
- name: delegate module demo
  hosts: webservers
  vars:
    my_content: "sample_test ...."
  tasks:
    - name: Adding the variable value
      copy:
        dest: /tmp/sample.txt
        content: "{{ my_content }}"
      delegate_to: s1
```

## Ansible delegate to localhost

If you want to run the task on local ansible control machine you can use this module. Or you can use ansible local_action module.

```
- name: delegate module demo
  hosts: webservers
  vars:
    my_content: "sample_test"
  tasks:
    - name: Adding the variable value
      copy:
        dest: /tmp/sample.txt
        content: "{{ my_content }}"
      delegate_to: localhost
```

## Run_once example

```
- name: delegate module demo
  hosts: webservers
  vars:
    my_content: "sample_test “
  tasks:
    - name: Adding the variable value
      copy:
        dest: /tmp/sample.txt
        content: "{{ my_content }}"
      run_once: true
```



## ***Ansible yum module***

Ansible has a specific module for managing the Yum packages. You can install, remove, upgrade or downgrade versions and many more by using this module.
The Yum module also requires two parameters for the primary command, like other package management modules in Ansible.

name: provides the name of the package which you want to install.
state: maintains the state of the packages, like what should be the state of the package after the task is completed (present or absent). By default, the value of the parameter is "present".

***Examples***

Installing a Package

```
- hosts: all
  become: yes
  tasks:
  - name: Install yum package in Ansible example
    yum:
      name: git
      state: present
```

Installing the latest Version

```
- hosts: all
  become: yes
  tasks:
  - name: Install yum package in Ansible example
    yum:
      name: git
      state: latest
```

Installing a Specific Version

```
- hosts: all
  become: yes
  tasks:
  - name: Install yum package in Ansible example
    yum:
      name: git  <version-name>
      state: latest
```

## Installing Multiple Packages

```
- hosts: localhost
  become: yes
  tasks:
  - name: Install yum package in Ansible example
    yum:
      name: "{{ item }}"
      state: present
    with_items:
      - git
      - httpd
```

## Update all Packages

```
- hosts: localhost
  become: yes
  tasks:
  - name: Install yum package in Ansible example
    yum:
      name: "*"
      state: latest
```

## Excluding the specific packages

```
- hosts: localhost
  become: yes
  tasks:
  - name: Install yum package in Ansible example
    yum:
      name: "*"
      state: latest
      exclude: httpd*
```

## *Ansible Service Module with Systemd Module*

- Ansible service module or ansible systemd module you can start, stop, restart and enable the services, so this ansible service module and ansible systemd modules are similar to Linux service command or Linux systemctl command.

NOTE:

- We use Linux syetmctl command in Redhat(rhel), centos
- We use Linux service command in ubuntu
- We can use ansible service module in any machine to start or stop the services
- We use ansible systemd in redhat to start or stop the services

service module started/stopped/restarted

```
- name: Make sure a jboss-eap service is running
  service:
    name=httpd
    state=started/stopped/restarted
```

Enabling the service:

```
- name: enable the jboss-eap service
  service: name=httpd enabled=yes
```

systemd module started/stopped/restarted

```
- name: make sure jboss-eap service is running
  systemd: name=jboss-eap state=started/stopped/restarted
```

Enabling the service using systemd

- It will do nothing if the service is already running.
- If the service is not running, it will start it.
- And if the system boots, it will start the service at boot time only

```
- name: enable the jboss-eap
  systemd: name=jboss-eap state=started enabled=yes
```

daemon_reload:

We use daemon-reload command after creating new unit files or modifying existing unit files

```
- name: reload config changes
  systemd: daemon_reload=yes
```

## ***Ansible loops***

- We can execute the same task repeatedly we can go for loops.
- Ansible loop provides a lot of methods to repeat certain tasks until a condition is satisfied.
    - with_list
    - with_items
    - with_indexed_items
    - with_flattened
    - with_together
    - with_dict
    - with_sequence
    - with_subelements
    - with_nested/with_cartesian
    - with_random_choice
- Pls refer examples below link above methods
  https://docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html#with-list

### *Examples*

Installing Multiple Packages

```
- hosts: localhost
  become: yes
  tasks:
  - name: Install yum package in Ansible example
    yum:
      name: "{{ item }}"
      state: present
    with_items:
      - git
      - httpd
```

## Ansible loop with Index

In some scenarios knowing the index value might come in handy. We can use the "with indexed_items" for this. The loop index will be available at item.0 and the value will be available at item.1. index value starts at zero as usual.

```
- hosts: localhost
  tasks:
    - name: Ansible loop with index example
      debug:
        msg: "echo loop index at {{ item.0 }} and value at {{item.1}}"
      with_indexed_items:
        - "hello1"
        - "hello2"
        - "hello3"
```

```
- hosts: localhost
  tasks:
    - name: Ansible loop with index example
      debug:
        msg: "echo loop index at {{ item.0 + 1}} and value at {{item.1}}"
      with_indexed_items:
        - "hello1"
        - "hello2"
        - "hello3"
```

## Ansible loop with conditional

We can also use the "when" conditional statement along with the loop structure. Thus you can control the looping based on a variable or system facts.

```
- hosts: localhost
  vars:
    var1: "India"
  tasks:
    - name: Ansible loop with conditional example
      debug:
        msg: "{{ item }}"
      with_items:
        - "India"
        - "US"
```

```
    - "UK"
  when: item == "{{ var1 }}"
```

## Looping through Dictionaries

Ansible dictionary variable using the **with_dict** parameter. In the following task, I have declared a variable 'States' with 4 key-value pairs. I am using the with_dict to loop through all the values.

```
- hosts: localhost
  vars:
    States:
      AP: 'Visakhapatnam'
      TS: 'Hyderabad'
      KTK: 'Bangalore'
      TN: 'Chennai'
  tasks:
    - name: Ansible dictionary loop Example
      debug:
        msg: "State is {{ item.key}} and Capital is {{item.value}}"
      with_dict: "{{ States }}"
```

## with_list

```
- hosts: localhost
  tasks:
    - name: with_list
      debug:
        msg: "{{ item }}"
      with_list:
        - one
        - two

    - name: with_list -> loop
      debug:
        msg: "{{ item }}"
      loop:
        - one
        - two
```

## *Ansible When Condition Examples*

- Ansible when condition is used to execute the tasks if the conditions that you defined are true.
- If the conditions are not true it will skip the executions of that particular task.

*Examples*

```
---
- hosts: all
  gather_facts: no

  tasks:
    - name: verify httpd version
      command: /usr/sbin/httpd -v
      register: version
      ignore_errors: True

    - name: print httpd version
      debug:
        msg: "{{ version }}"
      when: "version.rc == 0"

    - name: install httpd
      yum: name=httpd state=present
      when: "version.rc != 0 "
```



if the return code is zero

If the return code is zero that means httpd is installed in that system. And we are printing that httpd version in second task. and third task will be skipped.

if the return code is non zero

If the return code is non zero then there is no httpd installed in that system. So the second task will be skipped

and in the third task we are installing httpd.

- With ansible, to verify is the file existed or not, we use ansible stat module with when condition. Stat module is similar to the linux/unix 'stat' command.

We are storing the result of stat module in result variable. In the second task if the file is existed then we are executing copy command. We can use any module in the place of command module depends on our requirement.

```
---
- hosts: localhost
  gather_facts: no
  tasks:
    - name: Ansible check file exists
      stat:
        path: /tmp/devopsaws.txt
      register: result

    - name: Ansible check file exists example.
      commannd: cp /tmp/devopsaws.txt /tmp/devopsaws_backup.txt
      when: result.stat.exists
```

```
---
- hosts: localhost
  gather_facts: no
  tasks:
    - name: Ansible check file exists
      stat:
        path: /tmp/devopsaws1.txt
      register: result

    - name: Ansible create the file
      commannd: touch /tmp/devopsaws.txt
      when: result.stat.exists == false
```

```
---
- hosts: localhost
  tasks:-
    - debug:
        msg: I am redhat flavour
      when: ansible_os_family == "RedHat"
```

```
---
 - name: Find files - Playbook
   hosts: localhost
   tasks:

   # Case1:  when Search String and Modified time is mentioned
     - name:  Find command with *SEARCH STRING* and *MODIFIED TIME*
       shell: "find {{Directory}} -name '{{SearchString}}' -mtime '{{mtime}}'"
       register: case1output
       when: Directory is defined and SearchString is defined and mtime is defined
       ignore_errors: true

   # Case2: when Only Search String is mentioend but NOT Modified time(age)
     - name:  Find command with only with *SEARCH STRING*
       shell: "find {{Directory}} -name '{{SearchString}}' "
       register: case2output
       when: Directory is defined and SearchString is defined and mtime is not
defined
       ignore_errors: true

   # In case of Case1 Success
     - name: Case1 Output -- Output will be displayed only if Case is Success (or) it
will be skipped
       debug: var=case1output.stdout_lines
       when: case1output.stdout_lines is defined

   # In case of Case2 Success
     - name: Case2 Output -- Output will be displayed only if Case is Success (or) it
will be skipped
       debug: var=case2output.stdout_lines
       when: case2output.stdout_lines is defined
```

Command:

ansible-playbook test.yml -e "Directory=/home/ansadmin/myansible/testdir SearchString=*.txt mtime=30"

## *Ansible Async Poll or Ansible Asynchronous Tasks*

- Ansible async module we can put the tasks in background which taking more time and we continue with other tasks. So we can execute other tasks without waiting for the long running task to complete.
- Using ansible async mode we can run the tasks in parallel and we can check this background running tasks later.
  - ✓ async: <seconds to timeout the task>
  - ✓ poll: <seconds to poll for the status of the task>

async:

async indicates the Total time to complete the task or its maximum runtime of the task.

Poll:

poll indicates to Ansible, how often to poll to check if the command has been completed OR how frequently you would like to poll for status. With poll we keep checking whether the job is completed or not. The default poll value is 10 seconds poll: 5 This will poll the results every 5 seconds

poll: 0 Fire and forget

if you do not need to wait on the task to complete, you may run the task asynchronously by specifying a poll value of 0: In this mode ansible will connects to the client system, starts the process and disconnects. we don't need to wait for completion task.

```
tasks:
- name: ansible async poll
  command: /bin/sleep 50
  async: 60
  poll: 10
```

```
---
 # Running tasks parallel
- hosts: webservers
 become: yes
 gather_facts: no

 tasks:
  - name: Install Apache
    yum:
      name: httpd
      state: present
    async: 300
    poll:  10
  - name: restart httpd
    service:
      name: httpd
      state: started
```

## ***Ansible wait_for module***

It's waiting for condition ,If the condition is not available the task will not complete.

```
---
- name: A Playbook to test Async and Poll
 hosts: webservers
 become: yes
 tasks:
  - name: Install httpd
    yum:
      name: httpd
      state: present
  - name: wait for the service
    wait_for:
      timeout: 300

  - name: start the service
    service:
      name: httpd
      state: started
```

## Ansible Group Module

- Ansible group module is used to create and delete a groups in any Linux machine.

### Adding a group

```
- name: Create groups in remote machines
  hosts: webservers
  become: yes
  tasks:
    - name: create a group in target
      group:
        name: cricket
        state: present
```

### Deleting a group

```
- name: Create groups in remote machines
  hosts: webservers
  become: yes
  tasks:
    - name: create a group in target
      group:
        name: cricket
        state: absent
```

## Ansible User Module

- Ansible user module is used to create a user, delete user accounts in any Linux machine.
- Using ansible user module we can add passwords to those users and we can add users to groups.

### *Ansible Add User To Group*

```
- name: Create groups in remote machines
  hosts: webservers
  become: yes
  tasks:
```

```
  - name: create a group in target
    group:
      name: cricket
      state: present
  - name: create a user in cricket group
    user:
      name: rohit
      group: cricket
```

*Change Primary Group of User*

```
- name: Create groups in remote machines
  hosts: webservers
  become: yes
  tasks:
    - name: create a group in target
      group:
        name: ipl
        state: present
    - name: create a user in cricket group
      user:
        name: rohit
        group: ipl
```

*Create User and Primary Group with Groups argument*

```
- name: Create groups in remote machines
  hosts: webservers
  become: yes
  tasks:
   - name: create a user in cricket group
     user:
       name: kohli
       groups:
```

This will create user kohli and it will create primary group, kohli. You can see here we are using argument groups not the group. And we have not mentioned anything in the group's argument.

Adding multiple users to the group

```
- name: Create groups in remote machines
  hosts: webservers
  become: yes
  tasks:
    - name: create a user in cricket group
      user:
        name: "{{ item }}"
        group: ipl
      with_items:
        - bumrha
        - rahul
        - dawan
```

Remove the user

```
- name: Remove the user 'johnd'
  user:
    name: kohli
    state: absent
    remove: yes
```

Removing the multiple users and groups

```
- name: Create groups in remote machines
  hosts: webservers
  become: yes
  tasks:
    - name: create a user in cricket group
      user:
        name: "{{ item }}"
        state: absent
        remove: yes
      with_items:
        - bumrha
        - rahul
        - dawan
```

```
    - rohit
    - kohli
  - name: create aser in cricket group
    group:
      name: "{{ item }}"
      state: absent
    with_items:
      - ipl
      - cricket
      - kohli
```

## ***Ansible File module***

- Ansible file module is used to creating and deleting the file or multiple files in the remote server. You can also create and delete the directories and change the permissions of the data.
- We can also create and delete the soft links (symlinks) as well as hard links. With the help of the Ansible file module, you can set the permission of the files.
- Ansible file module, we have different parameters. We are using **path** and **state** parameters that are must in every file module. In the file parameter, we will mention the path of the file in the remote server. On this path, only the file will be created.

*Create a file in remote server*

```
- name: This is a play
  hosts: localhost

  tasks:
    - name: create file in a remote server
      file:
        path: /home/ansadmin/filemodule/devops.txt
        state: touch
```

**path:** It mentions the path of the file in the remote server.
**state:** It mentions touch, and touch will create file exact like Linux command.

*Delete a file in remote server*

```
- name: This is a play
  hosts: localhost

  tasks:
    - name:  delete file in a remote server
      file:
        path: /home/ansadmin/filemodule/delete.txt
        state: absent
```

*Creating a File with Permissions*

```
- name: This is a play
  hosts: localhost

  tasks:
    - name: Ansible file module to create new file with permissions.
      file:
        path: /home/ansadmin/filemodule/permission.txt
        state: touch
    mode: 0421
    owner: ansadmin

    - name: Ansible file module to create new file with permissions with
symbols
      file:
        path: /home/ansadmin/filemodule/permission1.txt
        state: touch
        mode: "u=rw,g=rw,o=r"
        owner: ansadmin
```

*Creating Multiple Files*

A path parameter: we can create a loop to create multiple files by using "{{item}}".
At with_items parameter: mention file names which you want to create.
By using "{{item}}" and with_items parameter, we can create loop or multiple files.

```
- name: Ansible file module to create multiple files
    file:
      path: "/home/ansadmin/filemodule/{{ item }}"
      state: touch
      mode: 0664
    with_items:
      - devops1.txt
      - devops2.txt
      - devops3.txt
      - devops4.txt
      - devops5.txt
      - devops6.txt
```

*Deleting Multiple Files*

```
- name: Ansible file module to delete multiple files
    file:
      path: "/home/ansadmin/filemodule/{{ item }}"
      state: absent
    with_items:
      - devops1.txt
      - devops2.txt
      - devops3.txt
```

*Recursively change ownership*

```
- name: This is a play
  hosts: localhost

  tasks:
    - name: Recursively change ownership of a directory
      file:
        path: /tmp/recursive/
        state: directory
        recurse: yes
        owner: ansadmin
        group:  ansadmin
```

## ***Ansible Unarchive module***

- Ansible unarchive module helps to unpack or uncompress the files from an archive file such as tar, tar.gz, zip.
- Ansible unarchive module can optionally copy the files to the remote server before uncompressing them.
- Ansible unarchive module uses the basic unzip and tar -xvf command-line tools to perform the operation. So the target server must have these commands installed. Since most of the Unix/Linux distributions are having these tools built-in.
- This module requires zipinfo and gtar/unzip command on the target remote host.
- Can handle .zip files using unzip as well as .tar, .tar.gz, .tar.bz2 and .tar.xz files using gtar.

*Unarchive a file that is already on the remote machine:*

```
- name: This is a play
  hosts: webservers

  tasks:

    - name: Extract foo.tgz into /var/lib/foo
      unarchive:
        src: /home/ansadmin/apache-maven-3.6.3-bin.tar.gz
        dest: /home/ansadmin
        remote_src: yes
```

*unarchive a file that needs to be downloaded:*

```
- name: This is a play
  hosts: webservers
  tasks:
    - name: Unarchive a file that needs to be downloaded
      unarchive:
        src: https://www-eu.apache.org/dist/maven/maven-3/3.6.3/binaries/apache-
maven-3.6.3-bin.tar.gz
        dest: /home/ansadmin
        remote_src: yes
```

```
---
- name: Playbook to download and install tomcat8
  hosts: webservers
  become: yes

  tasks:
    - name: install Java
      become: yes
      yum:
        name: java-1.8.0-openjdk
        state: present
      register: java_output
    - debug:
        var: java_output.stdout_lines

    - name: crate a directory
      become: yes
      file:
        path: "/opt/tomcat8"
        state: directory
        mode: 0755

    - name: Download and install tomcat
      become: yes
      unarchive:
        src:      "https://downloads.apache.org/tomcat/tomcat-8/v8.5.51/bin/apache-
tomcat-8.5.51.tar.gz"
        dest: "/opt/tomcat8/"
        mode: 0755
        remote_src: yes
      register: tcinstall
    - debug:
        var: tcinstall.stdout_lines

    - name: create a link
      file:
        src:  /opt/tomcat8/apache-tomcat-8.5.51
        dest: /usr/share/tomcat8
        state: link
        force: yes
```

```
  - name: Change ownership of Tomcat installation
    file: path=/usr/share/tomcat8/ owner=tomcat group=tomcat state=directory recurse=yes

  - name: Configure Tomcat as a service
    copy: src=tomcat-initscript.sh dest=/etc/init.d/tomcat mode=0755

  - name: Start the tomcat
    service:
      name: tomcat
      state: started
      enabled: yes
```

tomcat-initscript.sh

```
#!/bin/bash
#
# chkconfig: 345 99 28
# description: Starts/Stops Apache Tomcat
#
# Tomcat 7 start/stop/status script
# Forked from: https://gist.github.com/valotas/1000094
# @author: Miglen Evlogiev <bash@miglen.com>
#
# Release updates:
# Updated method for gathering pid of the current proccess
# Added usage of CATALINA_BASE
# Added coloring and additional status
# Added check for existence of the tomcat user
#

#Location of JAVA_HOME (bin files)
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.242.b08-0.amzn2.0.1.x86_64/jre

#Add Java binary files to PATH
export PATH=$JAVA_HOME/bin:$PATH

#CATALINA_HOME is the location of the bin files of Tomcat
export CATALINA_HOME=/opt/tomcat8/apache-tomcat-8.5.51

#CATALINA_BASE is the location of the configuration files of this instance of Tomcat
export CATALINA_BASE=/usr/share/tomcat8

#TOMCAT_USER is the default user of tomcat
export TOMCAT_USER=tomcat

#TOMCAT_USAGE is the message if this script is called without any options
```

```
TOMCAT_USAGE="Usage: $0 {\e[00;32mstart\e[00m|\e[00;31mstop\e[00m|\e[00;32mstatus\e[00m|\e[00;31mrestart\e[00m}"

#SHUTDOWN_WAIT is wait time in seconds for java proccess to stop
SHUTDOWN_WAIT=20

tomcat_pid() {
      echo `ps -fe | grep $CATALINA_BASE | grep -v grep | tr -s " "|cut -d" " -f2`
}

start() {
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
  then
    echo -e "\e[00;31mTomcat is already running (pid: $pid)\e[00m"
  else
    # Start tomcat
    echo -e "\e[00;32mStarting tomcat\e[00m"
    #ulimit -n 100000
    #umask 007
    #/bin/su -p -s /bin/sh tomcat
       if [ `user_exists $TOMCAT_USER` = "1" ]
       then
             su $TOMCAT_USER -c $CATALINA_HOME/bin/startup.sh
       else
             sh $CATALINA_HOME/bin/startup.sh
       fi
       status
  fi
  return 0
}

status(){
        pid=$(tomcat_pid)
        if [ -n "$pid" ]; then echo -e "\e[00;32mTomcat is running with pid: $pid\e[00m"
        else echo -e "\e[00;31mTomcat is not running\e[00m"
        fi
}

stop() {
  pid=$(tomcat_pid)
  if [ -n "$pid" ]
  then
    echo -e "\e[00;31mStoping Tomcat\e[00m"
    #/bin/su -p -s /bin/sh tomcat
       sh $CATALINA_HOME/bin/shutdown.sh

    let kwait=$SHUTDOWN_WAIT
    count=0;
    until [ `ps -p $pid | grep -c $pid` = '0' ] || [ $count -gt $kwait ]
    do
      echo -n -e "\n\e[00;31mwaiting for processes to exit\e[00m";
      sleep 1
      let count=$count+1;
    done
```

```
   if [ $count -gt $kwait ]; then
     echo -n -e "\n\e[00;31mkilling processes which didn't stop after $SHUTDOWN_WAIT seconds\e[00m"
     kill -9 $pid
   fi
  else
    echo -e "\e[00;31mTomcat is not running\e[00m"
  fi

  return 0
}

user_exists(){
       if id -u $1 >/dev/null 2>&1; then
       echo "1"
       else
              echo "0"
       fi
}

case $1 in

       start)
         start
       ;;

       stop)
         stop
       ;;

       restart)
         stop
         start
       ;;

       status)
              status

       ;;

       *)
              echo -e $TOMCAT_USAGE
       ;;
esac
exit 0
```

## Ansible Git Module

- Ansible git module is used to checkout or download the code from your github or bitbucket or gitlab account and archive the code in zip or any format

```
---
- hosts: localhost
  gather_facts: no
  vars:
    username: psddevops
    token: 3aba2f5c10b95c15a7bb3388402504a99ec7dd
    repo_name: sampletest

  tasks:
    - name: Checkout The Code From Github Using Ansible.
      git:
        repo: 'https://{{ token }}@github.com/{{ username }}/{{ repo_name }}.git'
        dest: /home/ansadmin/git-checkout/
```

## Ansible Command module

- Ansible command module is used to run any *Linux (or) shell* commands or run any scripts in the remote target machine OR used to execute commands on a remote node.

Examples

```
---
- hosts: localhost
  gather_facts: no
  become: yes
  tasks:
    - name: Executing a command using the command module
      command: cat /home/ansadmin/helloworld.txt
      register: data

    - debug:
        var: data.stdout_lines
```

```
---
- hosts: localhost
  gather_facts: no
  become: yes
  tasks:
    - name: ansible command with chdir and executable parameters
      command: ls -lrt
      args:
        chdir: /home/ansadmin/myansible
        executable: /bin/bash
      register: data

    - debug:
        var: data.stdout_lines
```

## Executing Multiple Commands

```
---
- hosts: localhost
  gather_facts: no
  become: yes
  tasks:
     - name: Ansible command module multiple commands
       command: "touch {{ item }}"
       with_items:
         - hello.txt
        - hello1.txt
         - hello2.txt
     args:
       chdir: /tmp
```

## Checking uptime,username and Disk Usage of Remote server

```
---
  - name: Check the remote host servers
    hosts: webservers
    tasks:
      - name: Execute the Uptime command over Command module
```

```
    register: uptimeoutput
    command: "uptime"
  - debug:
      var: uptimeoutput.stdout_lines

  - name: Execute the UNAME command
    register: unameout
    command: "uname -a"
  - debug:
      var: unameout.stdout_lines

  - name: Execute the df command
    register: dfout
    command: "df -h"
  - debug:
      var: dfout.stdout_lines
```

## ***Ansible shell module***

- Ansible shell module is designed to execute the shell commands against the target UNIX based hosts. Ansible can run except any high complexes commands with pipes, redirection. And you can also perform the shell scripts using the Ansible shell module.
- The Advantage of Ansible Shell module of supporting highly complexed commands with pipes and semicolons can also be a disadvantage from the security perspective as a single mistake could cost a lot and break the system integrity.

Examples

*Execute a Single Command with Ansible Shell*

```
tasks:
   - name: Check Date with Shell command
     shell:
       "date"
     register: datecmd
   - debug: msg="{{datecmd.stdout}}"
```

## 1. *Execute a Command with Pipe and Redirection*

```
---
 - name: Shell Examples
   hosts: localhost
   tasks:
    - name: Check Date with Shell command
      shell: "date"
      register: datecmd
    - debug: msg="{{datecmd.stdout}}"

    - name: Dir list and write to file
      shell: " ls -lrt /home/ansadmin/myansible awk '{print $9}'|sed '/^$/d' >
/tmp/dirlist.txt "
      register: lsout
    - debug: msg="{{lsout.stdout_lines}}"

    - name: Display the file
      shell: cat /tmp/dirlist.txt
      register: displaylist
    - debug: msg="{{displaylist.stdout_lines}}"
```

## 2. *Execute multiple commands in a Single Shell – Play*

```
---
- name: Playbook to download and install tomcat8
 hosts: webservers
 become: yes
 vars:
  username: psddevops
  token: 3aba2f5c10b95c15a7bb3388402504a99ec7dd5a
  repo_name: sampletest

 tasks:
  - name: install Git
    become: yes
    yum:
      name: git
      state: present
```

```
    register: git_output
  - debug:
      var: git_output.stdout

  - name: Checkout The Code From Github Using Ansible.
    git:
      repo: 'https://{{ token }}@github.com/{{ username }}/{{ repo_name }}.git'
      dest: /home/ansadmin/psddevops
    register: git_checkout
  - debug:
      var: git_checkout.stdout_lines
  - name: download and install java rpm
    yum:
      name: https://corretto.aws/downloads/latest/amazon-corretto-8-x64-linux-
jdk.rpm
      state: present
    register: java_output
  - debug:
      var: java_output.stdout_lines

  - name: crate a directory
    become: yes
    file:
      path: "/opt/tomcat8"
      state: directory
      mode: 0755

  - name: Download and install tomcat
    unarchive:
      src: "https://downloads.apache.org/tomcat/tomcat-8/v8.5.51/bin/apache-
tomcat-8.5.51.tar.gz"
      dest: "/opt/tomcat8/"
      mode: 0755
      remote_src: yes
    register: tcinstall
  - debug:
      var: tcinstall.stdout_lines

  - name: Download and install maven
    unarchive:
```

```
      src: "https://downloads.apache.org/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz"
      dest: "/opt/"
      mode: 0755
      remote_src: yes
    register: maveninstall
  - debug:
      var: maveninstall.stdout_lines

  - name: maven configuration and compilateion
    shell: |
      echo -e "\n rename the maven"
      cd /opt
      mv apache-maven-3.6.3 maven
      echo -e "\n setting the path for maven"
      PATH=$PATH:/opt/maven/bin:/usr/lib/jvm/java-1.8.0-amazon-corretto/bin
      export PATH
      cd /home/ansadmin/psddevops
      mvn clean package
      cp ./target/psdapp.war /opt/tomcat8/apache-tomcat-8.5.51/webapps/

  - name: crete group "tomcat"
    group:
      name: tomcat
      state: present
    ignore_errors: true
  - name: Add the user 'tomcat'
    user:
      name: tomcat
      group: tomcat
    ignore_errors: true

  - name: create a link
    file:
      src:  /opt/tomcat8/apache-tomcat-8.5.51
      dest: /usr/share/tomcat8
      state: link
      force: yes

  - name: Change ownership of Tomcat installation
```

```
    file: path=/usr/share/tomcat8/ owner=tomcat group=tomcat state=directory
recurse=yes

  - name: Configure Tomcat as a service
    copy: src=tomcat-initscript.sh dest=/etc/init.d/tomcat mode=0755

  - name: Start the tomcat
    service:
      name: tomcat
      state: started
      enabled: yes
```

## Ansible Command Vs Shell modules

A typical example is the Ansible modules Shell and Command. In the most use cases both modules lead to the same goal. Here are the main differences between these modules.

- With the Command module the command will be executed without being proceeded through a shell. As a consequence some variables like $HOME are not available. And also stream operations like <, >, | and & will not work.
- The Shell module runs a command through a shell, by default /bin/sh. This can be changed with the option executable.
- The command module is more secure, because it will not be affected by the user's environment.

## Ansible get_url module

- Ansible get_url module is being used to download files from **HTTPS, HTTP** and **FTP** servers (websites/URLs)
- By Default use the default proxy configuration of the node, You can use custom proxy and you can change the proxy address/url by setting environment variables such as **http_proxy** or **https_proxy** or by using the ansible built-in **use_proxy** option.
- We can pass through or handle the Basic Authentication. In other words, you can access the web sites/pages which are secure with Basic Authentication.

```
---
- name: get_url module demo
  hosts: localhost
  vars_prompt:
```

```
  - name: "username"
    prompt: "Please enter the username ?"
    private: no

  - name: "password"
    prompt: "Please enter the password ?"
    private: yes

 tasks:
  - name: download the zip file from remote repo
    get_url:
      url:
https://github.com/psddevops/jenkins_pipelines/raw/master/shell_script.zip
      url_username: "{{ username }}"
      url_password: "{{ password }}"
      dest: /tmp

  - name: Extract shell_script.zip file
    unarchive:
      src: /tmp/shell_script.zip
      dest: /tmp
      remote_src: yes

  - name: Run the shell script
    command: sh /tmp/shell_script/sample_script.sh 60
    register: script_out
  - debug:
      var: script_out.stdout_lines
```

Deployment through ansible

```
- name: get_url module demo
 hosts: all
 become: yes
 vars_prompt:
  - name: "username"
    prompt: "Please enter the username ?"
    private: no
```

```
  - name: "password"
    prompt: "Please enter the password ?"
    private: yes

 tasks:
  - name: download the zip file from remote repo
    get_url:
      url:
https://github.com/psddevops/jenkins_pipelines/raw/master/psdapp.war
      url_username: "{{ username }}"
      url_password: "{{ password }}"
      dest: /tmp
  - name: Copy the war file from temp to tomcat webapps
    command: "cp /tmp/psdapp.war /opt/tomcat8/apache-tomcat-
8.5.51/webapps/"
  - name: Start the tomcat
    service:
      name: tomcat
      state: started
      enabled: yes
  - name: Remove the war file from temp to tomcat webapps
    command: "rm /tmp/psdapp.war "
```

## ***Ansible copy module***

- Ansible copy module is one of the modules in file modules in Ansible. Ansible copy module is used for copy the file from ansible machine to the remote servers.

Copy the file with force yes

```
---
- name: copy module demo
  hosts: all
  tasks:
   - name: Copy file to a remote machine
     copy:
     src: /home/ansadmin/sample.txt
     dest: /tmp
```

```
---
- name: copy module demo
  hosts: all
  tasks:
    - name: Copy file to a remote machine
      copy:
      src: /home/ansadmin/sample.txt
      dest: /tmp
      force: yes
```

In above example test.txt file in the ansible machine will be copied to the destination location in the remote server but if the same file(with the same name) already exists in destination location in the remote server, it will replace with the file from ansible machine. It will not take care of the file is existed or not existed in a remote location. Blindly it will copy the file from ansible machine to the remote server.By defaut force is yes.

So both above codes are work like the same. Mentioning force: yes or removing force: yes it depends on you.

Copy the file with force no

```
---
- name: copy module demo
  hosts: all
  tasks:
    - name: Copy file to a remote machine
      copy:
      src: /home/ansadmin/sample.txt
      dest: /tmp
      force: no
```

In above example if the file is already existed in destination location it will not replace the file. That means we are not forcing to copy the file from ansible machine to remote server.

## Copy a directory from ansible machine to remote destination

```
---
- name: copy module demo
  hosts: all
  tasks:
    - name:  Copy a directory to a remote machine
      copy:
      src: /home/ansadmin/data
      dest: /tmp
```

In above example code can send or copy data directory from ansible machine to a remote server location.
See at the end of src: /home/ansadmin/data we have not mentioned (/) so that's why data directory will copy from ansible machine to remote.
If you mention (/) at the end of src, the files in data directory will copy to the remote.

```
---
- name: copy module demo
  hosts: all
  tasks:
    - name:  Copy all files inside a directory to a remote machine
      copy:
      src: /home/ansadmin/data/
      dest: /tmp
```

## Copy the file with permissions

```
---
- name: copy module demo
  hosts: all
  become: yes
  tasks:
    - name:  copy a file with permissions
      copy:
        src: /home/ansadmin/sample.txt
        dest: /etc/sample.conf
```

```
      owner: ansadmin
      group: ansadmin
      mode: 0644

  - name:  copy a file with permissions
    copy:
      src: /home/ansadmin/sample.txt
      dest: /tmp/sample.conf
      owner: ansadmin
      group: ansadmin
      mode: "u=rw,g=r,o=r"
```

Copy Content to Remote server

```
---
- name: copy module demo
  hosts: all
  become: yes
  tasks:
    - name:  create a file with content
      copy:
      content: "Hello devops"
      dest: /tmp/copydemo.txt
```

Copying multiple files

```
---
- name: copy module demo
  hosts: all
  become: yes
  tasks:
    - name: copy rundeck files
      copy: src=/home/ansadmin/{{ item }} dest=/home/ansadmin
      with_items:
        - samplefile1.txt
        - samplefile2.txt
        - samplefile3.txt

    - name: ansible copy multiple files
```

```
    copy: src={{ item.src }} dest={{ item.dest }}
    with_items:
      - { src: '/home/ansadmin/testfile1.txt', dest: '/home' }
      - { src: '/home/ansadmin/testfile2.txt', dest: '/home/ansadmin' }
      - { src: '/home/ansadmin/testfile3.txt', dest: '/etc' }
```

## ***Ansible fetch  module***

- Copy a file from remote machine to local/control machine in ansible we use ansible fetch module.
- Fetch module is used to fetch the files from remote to local machine.

```
---
- hosts: all
  gather_facts: no
  tasks:
    - name: ansible copy file from remote to local.
      fetch:
        src: /home/ansadmin/fetchfile.txt
        dest: /home/ansadmin/
```

```
---
- hosts: all
  gather_facts: no
  tasks:
    - name: ansible copy file from remote to local.
      fetch:
        src: /home/ansadmin/fetchfile.txt
        dest: /home/ansadmin
        flat: yes
```

If you execute above playbook, you will get the file in as you expected path i.e in /home/ansadmin.If you mention flat as yes.

## ***Ansible Synchronize  module***

Ansible Synchronize is used to copy the files between remote servers (or) target hosts.  This is more like performing RSYNC with help of Ansible.

```
---
- hosts: all
  gather_facts: no
  tasks:
    - name: ansible copy file using synchronize module
      synchronize:
        src: /home/ansadmin/synctest.txt
        dest: /home/ansadmin
```

Copy from s1 to d1 using fetch module

```
---
 - hosts: all
   become: yes
   tasks:
     - name: Fetch the file from the webservers to control machine
       fetch: src=/home/ansadmin/fetchdemo.txt dest=buffer/ flat=yes
       when: "{{ inventory_hostname == 's1' }}"

     - name: Copy the file from control machine to appservers
       copy: src=buffer/fetchdemo.txt dest=/tmp/
       when: "{{ inventory_hostname == 'd1' }}"
```

Copy from S1 to S2 using synchronize module pull mechanism

```
---
- name: Sync Pull task
  hosts: "{{groups['webservers'][0]}}"
  user: ansadmin
  tasks:
    - name: Copy the file from S1 to S2 using Method Pull
      tags: sync-pull
      synchronize:
```

```
      src: "{{ item }}"
      dest: "{{ item }}"
      mode: pull
    delegate_to: "{{groups['webservers'][1]}}"
    register: syncfile
    run_once: true
    with_items:
     - "/home/ansadmin/syncpull.txt"
```

Copy from S1 to S2 using synchronize module push mechanism

```
- name: Sync Push task
  hosts: "{{groups['webservers'][1]}}"
  user: ansadmin
  tasks:
    - name: Copy the file from S1 to S2 using Method Push
      synchronize:
        src: "{{ item }}"
        dest: "{{ item }}"
        mode: push
      delegate_to: "{{groups['webservers'][0]}}"
      register: syncfile
      with_items:
       - "/home/ansadmin/syncpush.txt"
```

## ***Ansible Tags***

- By using ansible tags we can execute the specific task in the ansible playbook.
- When we execute a playbook, we can filter tasks based on the tags in 2 ways.
  1. Command line, with the -tags or -skip-tags options.
  2. Ansible configuration settings, with the TAGS_RUN and TAGS_SKIP options.

```
---
 - name: Tags - Playbook
   hosts: localhost

   vars:
    dev_var: "This script is going deploy DEV environment"
    uat_var: "This script is going deploy UAT environment"
```

```
  perf_var: "This script is going deploy PERF environment"
  prod_var: "This script is going deploy PROD environment"
  common_var: "This script is execute on all environments"

 tasks:
  - name: executing the COMMON script
    debug:
     var: common_var

  - name: executing the DEV script
    debug:
      var: dev_var
    tags:
      - dev

  - name: executing the UAT script
    debug:
      var: uat_var
    tags:
      - uat

  - name: executing the PERF script
    debug:
      var: perf_var
    tags:
      - perf

  - name: executing the PROD script
    debug:
      var: prod_var
    tags:
      - prod
```



ansible-playbook test.yml --tags="dev,prod"
ansible-playbook test.yml --skip-tags="prod,dev"

*Tag Reuse*

We can apply the same tag to more than one task. By using the "--tags" command line options, all tasks with that tag name will be run.

*Special Tags*

"always" is a unique tag that will always run a task, unless specifically skipped (--skip-tags always)

ansible-playbook test.yml --skip-tags="prod,always"

## *Ansible Blockinfile module*

Ansible blockinfile module is used to insert/update/remove a block of lines in a file. This module will insert/update/remove a block of multi-line text surrounded by customizable marker lines.

```
---
- hosts: all
  gather_facts: no
  tasks:
    - name: add block of lines to a file.
      blockinfile:
        path: /home/ansadmin/users.txt
        block: |
          Sami
          Bumraha
        backup: yes
```

path: here we have to give the path of the file which you want to modify.

block: here you have to write your lines of code(content) which you want to add to the file

backup: if you mention backup yes, it will create one file in the current directory with timestamp. it will have the content of your old file. in this way we can take backup of your file.

By default, the block will be inserted at the end of the file.

## *Ansible Lineinfile Module*

- Ansible lineinfile module is used to insert new lines at the end of the file or anywhere in the file. With ansible lineinfile module we can remove existed lines from the file and we can replace the lines.

```
---
- name: lineinfile demo
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Ansible lineinfile module.
      lineinfile:
        path: /home/ansadmin/myansible/lineinfile.txt
        line: This is sample line
        state: present
        create: yes
```

path parameter tells the lineinfile to which file it has to modify So in the path parameter we should mention path of the file which we are going to modify. In line parameter we have to add the line which is going to add in the file. To add the line to the file, we should mention state parameter as present. Create parameter with yes, we used to create the file if the file is not existed.

```
---
- name: lineinfile demo
  hosts: localhost
  gather_facts: no

  tasks:
  - name: Ansible check directory.
    lineinfile:
      path: /home/ansadmin/myansible/lineinfile1.txt
      regexp: '^Amaravathi'
      line: 'Visakhapatnam is the capital of Andhrapradesh'
      state: present
      backrefs: yes
```

Here we mentioned in the regexp parameter as '^Amaravathi'. that means the line started with Amaravathi has to modify with the line 'Visakhapatnam is the capital of Andhrapradesh' in the file.

We should mention backrefs as yes. Backrefs yes will replace the line with 'Visakhapatnam is the capital of Andhrapradesh' only if it is find regexp

line(^Amaravathi) in file. If there is no line started with Amaravathi it won't change anything in the file, the file will be left unchanged.
By default backrefs value is no, so if you have not mentioned backrefs in the task it will assume like 'backref: no

Add a line after/before a particular line:

```
---
- name: lineinfile demo
  hosts: localhost
  gather_facts: no
  tasks:
  - name: Ansible check directory.
    lineinfile:
     path: /home/ansadmin/myansible/lineinfile1.txt
     insertafter: '^Visakhapatnam'
     line: '================================='
```

```
---
- name: lineinfile demo
  hosts: localhost
  gather_facts: no
  tasks:
  - name: Ansible check directory.
    lineinfile:
     path: /home/ansadmin/myansble/lineinfile1.txt
     insertbefore: '^Visakhapatnam'
     line: '#================================='
```

Removing a line in particular file

```
---
- name: lineinfile demo
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Ansible lineinfile remove the line
      lineinfile:
```

```
    dest: /home/ansadmin/myansible/lineinfile.txt
    line: This is third line
    state: absent
```

Commenting a line with Ansible lineinfile backrefs

```
---
- name: lineinfile demo
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Ansible lineinfile regexp comment
      lineinfile:
        dest: /home/ansadmin/myansible/lineinfile.txt
        regexp: '(This is second line)'
        line: '#\1'
        backrefs: yes
```

Uncommenting the line with lineinfile regexp

```
---
- name: lineinfile demo
  hosts: localhost
  gather_facts: no
  tasks:
    - name: Ansible lineinfile backrefs uncommenting example
      lineinfile:
        dest: /home/ansadmin/myansible/lineinfile.txt
        regexp: '#(This is second line)'
        line: '\1'
        backrefs: yes
```

## ***Ansible APT Module***

Ansible apt module is used to install or remove the packages from ubuntu/debian and update the installed packages and to upgrade the system.

```
---
- name: aptmodule demo
  hosts: ubuntu
  become: yes
```

```
 gather_facts: no
 tasks:
  - name: Update cache and install "apache2" package
    apt:
      update_cache: yes
      name: apache2
      force_apt_get: yes
```

It will update the cache i,e it will update all the packges in /etc/apt/sources.list to new versions. And install the latest version apache2 from sources.list file.
Ansible apt module supports apt-get commands as well and you could use instruct ansible apt module to use apt-get using a parameter named force_apt_get

```
---
- name: aptmodule demo
  hosts: ubuntu
  become: yes
  gather_facts: no
  tasks:
    - name: Install apache2 httpd
      apt:
        name: apache2
        state: present
        force_apt_get: yes
```

To install any package using ansible apt module we use state argument as present. Present represents,install that particular package.

Install multiple packages

```
---
- name: aptmodule demo
  hosts: ubuntu
  become: yes
  gather_facts: no
  tasks:
    - name: Install a list of packages
      apt:
```

```
      name: "{{ item }}"
      update_cache: yes
      state: present
      force_apt_get: yes
    with_items:
      - 'git'
      - 'apache2'
      - 'docker'
```

Uninstall multiple packages

```
---
- name: aptmodule demo
  hosts: ubuntu
  become: yes
  gather_facts: no
  tasks:
    - name: Install a list of packages
      apt:
        name: "{{ item }}"
        update_cache: yes
        state: absent
        force_apt_get: yes
      with_items:
        - 'git'
        - 'apache2'
        - 'docker'
```

Updates the list of available packages

```
---
- name: aptmodule demo
  hosts: ubuntu
  become: yes
  gather_facts: no
  tasks:
    - name: Only run "update_cache=yes" if the last one is more than 1 hour ago
      apt:
        update_cache: yes
```

```
    cache_valid_time: 3600
```

Here Ansible will not update the cache if it's been updated in the last 3600 seconds. Ansible will compare last updated time with this task executed time. If the time gap is less than 3600 seconds it will skip this task. If the time gap is greater than 3600 seconds it will update the cache.

update_cache will updates the list of available packages and their versions, i,e it will update apt's database /etc/apt/sources.ist. Fetches the list of available updates.

Update & Upgrade All the packages installed in OneGo

```
---
 - name: Ansible apt module examples
   hosts: ubuntu
   become: yes
   tasks:
    - name: Ansible Update Cache and Upgrade all Packages
      register: updatesys
      apt:
        name: "*"
        state: latest
        update_cache: yes

    - name: Display the last line of the previous task to check the stats
      debug:
        msg:  "{{updatesys.stdout_lines}}"
```

Install a .deb file or package using ansible apt

```
---
 - name: Ansible apt module examples
   hosts: ubuntu
   become: yes
   tasks:
    - name: Ansible install filebeat deb file from url
      apt:
        deb:        https://artifacts.elastic.co/downloads/beats/filebeat/filebeat-7.6.0-amd64.deb
```

ansible apt module can be used to install the .deb (Debian Software Package file) packages as well additionally ansible apt can download the file from remote URL and install it as well.

## How to validate if a package is installed – Ansible apt

```
---
 - name: Ansible validate if the packages are installed
   hosts: ubuntu
   become: yes
   tasks:

    - name: Gather Package facts
      package_facts:
        manager: auto

    - name: Validating if the package is installed or not
      debug:
        msg: "{{item}} is installed"
      when: '"{{item}}" in ansible_facts.packages'
      with_items:
        - apache2
        - filebeat
```

## Clean up the useless Packages using ansible apt

```
---
 - name: Ansible remove useless packages
   hosts: ubuntu
   become: yes
   tasks:
    - name: Remove useless packages from the cache
      apt:
        autoclean: yes
    - name: Remove dependencies that are no longer required
      apt:
        autoremove: yes
```

## ***Ansible Vault/encryption***

- Ansible Vault is a feature that allows you to keep all your secrets safe and you can encrypt the secret files.
- Ansible Vault is primarily useful when you want to store confidential data to encrypt your secret files in ansible we use a utility called ansible-vault.

### Creating New Encrypted Files

To create a new encrypted file with ansible Vault, use ansible-vault create command. it will ask you vault password two times, enter the password two times, this password we will use in the future to run the playbooks so remember this vault password and after entering vault password two times you will enter into vi(vim) editor of that file.

***ansible-vault create sample.yaml***

### Encrypting Existing Files

To encrypt the existing files we can use ansible-vault encrypt command. it will ask you vault password two times enter it.

***ansible-vault encrypt cred.yml***

### Display (view) the content of encrypted files

To display or to view the content of encrypted files we can use ansible-vault view command. we can not cat or vi of any encrypted file since if you use these commands to see the content it will show you some encrypted code. So to view the data or content of encrypted files we use ansible-vault view command.

***ansible-vault view cred.yml***

It will ask you the vault password, enter the password and then you can see the content of the file.

## Editing Encrypted Files

If you want to add extra data or remove the data from the encrypted file, we can not directly edit or vi(vim) of the encrypted file. for this, we use the ansible-vault edit command. using this command we can edit the encrypted files.

***ansible-vault edit cred.yml***

It will ask you ansible vault password, enter the password, after enting the password file will be opened in vi(vim)editor, edit the file and save it.

## Changing the Password of Encrypted Files

To change the vault password of encrypted files we use ansible-vault rekey command.

***ansible-vault rekey cred.yml***

When you enter the command, you will first be prompted with the file's current password. After entering it, you will be asked to select and confirm a new vault password. enter the new vault password, that's it in this way we can change the vault password of an encrypted file.

## Decrypt the Encrypted Files

To remove encryption from the encrypted files, we use ansible-vault decrypt command.

***ansible-vault decrypt cred.yml***

It will ask you vault password, enter the password, encryption will be removed from the file or file will be decrypted.

## Encrypt Specific Variables in Ansible

Ansible vault grants you the ability to encrypt certain variables. This is done using the ansible-vault encrypt_string command as shown.

***ansible-vault encrypt_string 'string' --name 'ppreddy'***

## ***Method-1***

To run any playbook in ansible we use ansible-playbook command here also we use the same command to run the playbook, but we have to pass one new extra argument when you are running encrypted file and that is –ask-vault-pass.

```
- hosts: localhost
  become: yes
  gather_facts: no
  vars_files:
    - "./cred.yml"
  tasks:
    - name: Adding the username value
      copy:
        dest: /tmp/files_11.txt
        content: "{{ username }}"
    - name: Adding the password value
      copy:
        dest: /tmp/files_12.txt
        content: "{{ password }}"
    - name: Printing/Reading the values from prompt
      debug:
        msg: "The capital of {{ state }} is {{ capital }}"
```

***ansible-playbook test.yml --ask-vault-pass -b -K***

It will ask you vault password enter it, that's it in this way you can run the encrypted playbook

***Method-2***

Password prompts some times can get annoying to avoid this, we can one ansible feature called "password file" which references to a file containing the password. You can then just pass this password file during runtime.

***ansible-playbook test.yml --vault-password-file ./mypassword.txt -b -K***

## ansible ignore_errors=True

In ansible if anyone of the task fails then it will stop the the entire execution of playbook or role. So to avoid this problem we use ignore_errors=true

```
---
- hosts: localhost
  gather_facts: no

  tasks:
    - name: deleting a file
      command: "rm /home/ansadmin/ppreddy.txt"
      ignore_errors: true
```

```
---
- hosts: localhost
  tasks:
  - name: Stop httpd if already running
    service: name=httd state=stopped
    ignore_errors: true

  - name: Stop httpd if already running
    service: name=httpd state=stopped
```

```
---
 - name: Ansible ignore_errors examples
   hosts: linux
   become: yes
   tasks:
    - name: list out the files from cricket
      command: "ls -lrt /home/ansadmin/cricket"
      register: cricket_out
      ignore_errors: true

    - debug:
        var: cricket_out
```

```
- name: list out the files from tennis
  command: "ls -lrt /home/ansadmin/tennis"
  register: tennis_out
  ignore_errors: true
- debug:
    var: tennis_out


- name: list out the files from hockey
  command: "ls -lrt /home/ansadmin/hockey"
  register: hockey_out
  ignore_errors: true
- debug:
    var: hockey_out


- name: list out the files from kabaddi
  command: "ls -lrt /home/ansadmin/kabaddi"
  register: kabaddi_out
  ignore_errors: true
- debug:
    var: tennis_out
```



## ***Ansible block***

- Ansible stops playbook execution on a task failure and we can choose to ignore_erros to continue with remaining tasks.
- Blocks allow for logical grouping of tasks and in play error handling. Most of what you can apply to a single task (with the exception of loops) can be applied at the block level, which also makes it much easier to set data or directives common to the tasks.
- If there is any error block the rescue section would get executed with whatever you need to do to recover from the previous error.
- The always section runs no matter what previous error did or did not occur in the block and rescue sections.

```
---
 - name: Ansible block module examples
   hosts: linux
   become: yes
   tasks:
     - block:

         - name: list out the files from cricket
           command: "ls -lrt /home/ansadmin/cricket"
           register: cricket_out

         - name: list out the files from tennis
           command: "ls -lrt /home/ansadmin/tennis"
           register: tennis_out

         - name: list out the files from hockey
           command: "ls -lrt /home/ansadmin/hockey"
           register: hockey_out

         - name: list out the files from kabaddi
           command: "ls -lrt /home/ansadmin/kabaddi"
           register: kabaddi_out

       ignore_errors: true

     - debug:
         var: tennis_out
     - debug:
         var: cricket_out
     - debug:
         var: hockey_out
```

```
---
 - name: Ansible block module examples
   hosts: linux
   tasks:
     - block:
         - name: Install httpd on Redhat family
```

```
        yum:
          name: httpd
          state: present
      - name: start the httpd service on Redhat family
        service:
          name: httpd
          state: started
    when: ansible_os_family =='RedHat'
    become: yes
  - block:
    - name: Install apache2 on Debian family
      yum:
        name: apache2
        state: present
    - name: start the apache2 service on Debian family
      service:
        name: apache2
        state: started
    when: ansible_os_family =='Debian'
    become: yes

  - debug:
      msg: "playbook execution completed"
```

```
---
 - name: Ansible block module examples
   hosts: linux
   become: yes
   tasks:
    - block:
      - name: list out the files from cricket
        command: "ls -lrt /home/ansadmin/cricket"
        register: cricket_out

      rescue:
       - debug:
          msg: "There is no directory /home/ansadmin/cricket"

      always:
       - debug:
```

```
        msg: "Task successfully completed ...."
```

```
---
 - name: Ansible block module examples
   hosts: linux
   become: yes
   tasks:
    - block:
        - name: list out the files from cricket
          command: "ls -lrt /home/ansadmin/hockey"
          register: hockey_out
        - debug:
            var: hockey_out

      rescue:
        - debug:
            msg: "There is no directory /home/ansadmin/hockey"

      always:
        - debug:
            msg: "Task successfully completed ...."
```



## ***Ansible reusable concepts***

### Import Vs Include

- All import* statements are pre-processed at the time playbooks are parsed.
- All include* statements are processed as they encountered during the execution of the playbook.
- So import is static, include is dynamic.

### Example

Include/import tasks

```
---
 - name: Ansible import/include module examples
```

```
  hosts: linux:ubuntu
  become: yes
  tasks:
   - name: Install httpd on Redhat family
     yum:
       name: httpd
       state: present
     when: ansible_os_family =='RedHat'

   - name: Install open java rpm on Redhat family
     yum:
       name:    https://corretto.aws/downloads/latest/amazon-corretto-8-x64-linux-jdk.rpm
       state: present
     when: ansible_os_family =='RedHat'

   - name: Install apache2 on Debian family
     apt:
       update_cache: yes
       name: apache2
       state: present
       force_apt_get: yes
     when: ansible_os_family =='Debian'

   - name: Install jdk package on Debian server
     apt:
       update_cache: yes
       name: openjdk-8-jdk
```

```
---
 - name: Install httpd on Redhat family
   yum:
     name: httpd
     state: present
 - name: Install open java rpm on Redhat family
   yum:
     name:       https://corretto.aws/downloads/latest/amazon-corretto-8-x64-linux-jdk.rpm
     state: present
```

```
---
 - name: Install apache2 on Debian family
   apt:
     update_cache: yes
     name: apache2
     force_apt_get: yes

 - name: Install jdk package on Debian server
   apt:
     update_cache: yes
     name: openjdk-8-jdk
     force_apt_get: yes
```

```
---
 - name: Ansible import/include module examples
   hosts: linux:ubuntu
   become: yes
   tasks:
     - import_tasks: install_pkgs_RedHat.yml
       when: ansible_os_family =='RedHat'

     - include_tasks: install_pkgs_Debian.yml
       when: ansible_os_family =='Debian'
```

```
---
 - name: Ansible import/include module examples
   hosts: linux:ubuntu
   become: yes
   tasks:
     - include_tasks: install_pkgs_{{ ansible_os_family }}.yml
       when: ansible_os_family =='RedHat'

     - include_tasks: install_pkgs_{{ ansible_os_family }}.yml
       when: ansible_os_family =='Debian'
```

import_vars and include_vars:

```
---
 - name: Ansible import/include module examples
```

```
  hosts: linux:ubuntu
  gather_facts: true
  become: yes
  tasks:
    - include_vars: required_vars_{{ ansible_os_family}}.yml
    - debug:
        msg: "Pkg name= {{ pkg }}"
```

```
---
 pkg: httpd
```

```
---
 pkg: apache2
```

## ***Ansible Template module***

Ansible Template module is little Different form ***COPY*** module. Instead of copy file from control machine to destination machine, it will supply the variable values first to the file, after that it will copy the file to destination. So using ansible template module we can copy dynamic files to the target server. Whenever you have dynamic values, ansible has a templating system. That templating system is Jinja. Jinja templates are the files with extension '.j2'. Most of the time we use them to replace configuration files or place some documents on the remote server. We can also perform conditional statements, loops, filters for transforming the data using ansible templates.

Examples

Hostdetails.txt.j2

```
Host Details
                              ============



Ansible system version is              ================>>>    {{ ansible_system }}
```

```
Ansible OS family is ===============>>>  {{ ansible_os_family }}

Ansible server FDQN is ===============>>>  {{ ansible_fqdn }}

Ansible hostname is ===============>>>  {{ ansible_hostname }}

Ansible domain is ===============>>>  {{ ansible_domain }}

Ansible distribution is ===============>>>  {{ ansible_distribution }}

Ansible BIOS date is ===============>>>  {{ ansible_bios_date }}

Ansible BIOS version is ===============>>>  {{ ansible_bios_version }}

Ansible architecture is ===============>>>  {{ ansible_architecture }}

ansible_managed is ===============>>>  {{ ansible_managed }}

local_ip is ===============>>> {{ ansible_default_ipv4["address"] }}

local_user is ===============>>>  {{ ansible_user }}

template_host is ===============>>>  {{ template_host }}

template_uid is ===============>>>  {{ template_uid }}

template_path is ===============>>>  {{ template_path }}

template_fullpath is ===============>>>  {{ template_fullpath }}

template_run_date is ===============>>>  {{ template_run_date }}
```

```
---
 - name: Ansible block module examples
   hosts: webservers
```

```
tasks:
 - block:
    - name: Install httpd on Redhat family
      yum:
        name: httpd
        state: present

    - name: copy the jinja template
      template:
        src: hostdetails.txt.j2
        dest: /home/ansadmin/hostdetails.txt

    - name: start the httpd service on Redhat family
      service:
        name: httpd
        state: started
  when: ansible_os_family =='RedHat'
  become: yes
 - block:
    - name: Install apache2 on Debian family
      yum:
        name: apache2
        state: present
    - name: copy the jinja template
      template:
        src: hostdetails.txt.j2
        dest: /home/ansadmin/hostdetails.txt
    - name: start the apache2 service on Debian family
      service:
        name: apache2
        state: started
  when: ansible_os_family =='Debian'
  become: yes
 - debug:
    msg: "playbook execution completed"
```

## Handlers

Handlers are special task that run at the end of a play if notified by another task. If a configuration file gets changed notify a service restart task it needs to run.

```
hosts: localhost
become: true
tasks:
- name: install httpd
yum:
name=httpd
update_cache=yes
state=latest
notify:
- start httpd
handlers:
- name: start httpd
service:
name: httpd
state: restarted
```

## Ansible Roles

- Ansible roles are consists of many playbooks, which is similar to modules in puppet and cook books in chef. We term the same in ansible as roles.
- An Ansible role is composed of multiple folders, each of which contain several YAML files.
- By default, they have a main.yml file, but they can have more than one when needed.
- There is a standardized structure for all Ansible roles, which allows Ansible playbooks to automatically load predefined variables, tasks, handlers, templates, and default values located in separate YAML files.
- It can be easily reuse the codes by anyone if the role is suitable to someone.
- It can be easily modify and will reduce the syntax errors.
- Each Ansible role should contain at least one of the following directories, if not all of them.
    - defaults
    - vars

- files
- templates
- meta
- tasks
- handlers
- tests

***Directory Structure***

tasks - contains the main list of tasks to be executed by the role.
handlers - contains handlers, which may be used by this role or even anywhere outside this role.
defaults - default variables for the role.
vars - other variables for the role. Vars has the higher priority than defaults.
files - contains files required to transfer or deployed to the target machines via this role.
templates - contains templates which can be deployed via this role.
meta - defines some data / information about this role (author, dependency, versions, examples, etc,.)

defaults folder

- Contains default variables for the role. Variables in default have the lowest priority so they are easy to override.
- defaults/main.yml is a configuration file that you can use to define default values for variables used in your role.
- It allows for a centralized management of the default values of the variable of the role.
- Default values are always vulnerable because they change a lot depending on the needs and policies of the user.
- Having this solution allows one file to change all the values.

vars folder

- Contains variables for the role. Variables in vars have higher priority than variables in defaults directory.

- This is where the role variables get stored.
- Usually, it is used for a permanent variable that does not require any changes between environments.

files folder

- Contains files which we want to be copied to the remote host. We don't need to specify a path of resources stored in this directory.
- This folder holds all extra files that are required to achieve the role task.
- These files usually get dispatched to remote hosts as part of certain tasks.
- They are usually static, and they do not contain any variables to change, be copied, extracted, or compressed to the remote host.

templates folder

- This folder contains the template files used by the role to create the actual configuration files.
- These are then deployed by the role to the remote hosts.
- They are Jinja2 template engine scripts that enable loops and other features.

tasks folder

- Contains the main list of steps to be executed by the role.
- tasks/main.yml is where you'll spend most of your time.
- It contains the main YAML files.
- The code within those files executes the main role tasks by calling all the other defined elements of the role.
- Any actions defined in this file will be executed by Ansible when your role runs.

handlers folder

- Contains handlers which can be invoked by "notify" directives and are associated with service.
- handlers/main.yml is where you will define handlers like restart nginx/apache/tomcat.
- It is mainly used for service management to apply a configuration change performed by another task.
- Handlers can be called in the same role, from other roles, and from the calling playbook

meta folder

- meta/main.yml is the metadata file for your role.
- These folders contain the role metadata, which includes information about authors, licenses, compatibilities, and dependencies.
- The main use for this option is to declare dependencies, more specifically, roles.
- If the current role relies on another role, that gets declared in a meta folder.

tests folder

- This folder contains a test environment with an inventory file and a playbook script to test the role.
- This is primarily used when testing your role automatically using a continuous integration system, such as Jenkins or Travis CI.

*Note:*

- We have discussed eight folders were just discussed, not all of them are mandatory.
- Your role can be very useful, even if you only provide a tasks file. Most of the time, you'll find that you're working in the tasks folder, with supporting files found in handlers and templates.

*Different location for Ansible roles*

Roles can be stored at default location and from there can be easily used from playbooks

- ./roles has the highest precedence (current directory)
- ~/.ansible/roles is checked after that
- /etc/ansible/roles is checked next
- /usr/share/ansible/roles

```
/home/ansadmin
mkdir myroles/
mkdir roles
cd /home/ansadmin/myroles/roles
ansible-galaxy init samplerole
```

```
====================================

defaults/main.yml

---
# defaults file for samplerole
  system_manager: admin@psddevops.com

files/files_demo.txt

cat files_demo.txt
This is sample files module demo

handlers/main.yml
cat main.yml

# handlers file for samplerole
  - name: handler section execution
    command: " ls -lrt /home/ansadmin/ "

meta/main.yml
cat main.yml
galaxy_info:
  author: Pushpanath
  description: Technical Lead
  company: HCL Technologies Pvt Ltd.

templates/sample_templates.txt.j2

cat sample_templates.txt.j2
Welcome to {{ ansible_hostname }}

This file was created on {{ ansible_date_time.date }}
Stay home if you don't have any work

Contact {{ system_manager }} if anything is wrong

vars/main.yml
cat main.yml
```

```
---
# vars file for samplerole

my_content: "This file created using roles  concept,Created by PSD Devops"
name: "Sachin Tendulkar"
game: "Cricket"
place: "Mumbai"
country: "India"
```

tasks/main.yml

```
---
# tasks file for samplerole
- name: Copy the static file from remote to mgmt machines
  copy: src=files_demo.txt dest=/home/ansadmin/

- name: Copy the template file from remote to mgmt machines
  copy: src=files_demo.txt dest=/home/ansadmin/

- name: copy the jinja template
  template:
    src: sample_templates.txt.j2
    dest: /home/ansadmin/sample_templates.txt

- debug:
    msg: " I am from {{ name }} coming from {{ place }} located in {{ country }} and
playing {{ game }}"

- name: Executing the handlers task
  command: " ls -lrt /home/ "
  notify: handler section execution

==================================================
---
- hosts: webservers
  become: yes
  roles:
    - samplerole
```

```
/home/ansadmin/myroles >> ansible-playbook samplerole.yml -i ./hosts
/home/ansadmin/myroles
ansible-playbook /home/ansadmin/myroles/samplerole.yml -i ./hosts

================================**********==========

cd /home/ansadmin/myroles/roles
ansible-galaxy init myapache

defaults/main.yml
=================
cat main.yml
---
# defaults file for myapache
class: Ansible
institute: PSD Devops

files/mystatic.html
===================
cat mystatic.html
<html>
<head>
<center>
<h1> Ansible Roles_Files Module Demo </h1>
</center>
</head>
<h2> Ansible Roles_Files Module Demo example file </h2>
</html>

handlers/main.yml
=================
cat main.yml

cat main.yml
---
# handlers file for myapache
- name: restart httpd
  service:
    name: httpd
    state: restarted
```

```
  when: ansible_os_family =='RedHat'


- name: restart apache2
  service:
    name: apache2
    state: started
  when: ansible_os_family =='Debian'


tasks/main.yml
==============

cat main.yml
============

---
# tasks file for myapache
- name: Install httpd on Redhat family
  yum:
    name: "{{ pkg_redhat }}"
    state: present
  notify: restart httpd
  when: ansible_os_family =='RedHat'

- name: Install apache2 on Debian family
  apt:
    update_cache: yes
    name: "{{ pkg_ubuntu }}"
    force_apt_get: yes
  notify: restart apache2
  when: ansible_os_family =='Debian'

- name: copy the jinja template
  template:
    src: mydynamic.html.j2
    dest: /var/www/html/mydynamic.html

- name: copy the file to remote servers
  copy:
```

```
   src: mystatic.html
   dest: /var/www/html/mystatic.html

templates/mydynamic.html

cat mydynamic.html.j2
====================

<html>
<head>
<center>
<h1> Ansible Roles template Module Demo </h1>
</center>
</head>
<h2> We are conducting {{ class }} on {{ institute }} technologies </h2>
<h2> The hostname of this webserver is {{ ansible_hostname }} </h2>
<h2> It is running on {{ ansible_os_family }} system </h2>
</html>


vars/main.yml

pkg_ubuntu: "apache2"
pkg_redhat: "httpd"
---
- hosts: webservers
  become: yes
  roles:
    - myapache
```



## ***Ansible Galaxy***

Ansible galaxy is a community website where we can share ansible roles and we can download ansible roles from it freely. You can find thousands of ansible roles in ansible galaxy website. https://galaxy.ansible.com/ To download ansible roles or to create a Ansible role structure, we use the ansible-galaxy command.

## Search ansible roles with ansible galaxy command

We can search available ansible roles with ansible galaxy search command. It will list all the available roles in galaxy website in your shell.

***ansible-galaxy search Jenkins***

## Find more information about role

After listing of all available roles if you want to know more information about the role like author of the role,github path, number of downloads of that role,..etc. You can use ansible-galaxy info command. ansible-galaxy info command will give you all the information about that particular role.

***ansible-galaxy info akry.jenkins***

## Download roles from ansible galaxy website

Download ansible roles from ansible galaxy website we use ansible-galaxy install command.

***ansible-galaxy install akry.jenkins***

## Create a role with ansible galaxy command

Using ansible-galaxy init command we can create the role. It will create a role template or directory structure for your role.

***ansible-galaxy init mycustomrole***

## List Downloaded roles

To list installed/downloaded roles from ansible galaxy website we can use ansible-galaxy list command. it will list only installed/downloaded roles. It will not display any role user created manually or role that created with init command

***ansible-galaxy list***

## Delete the role

To delete the installed/downloaded ansible role from your system we can use ansible-galaxy remove command.

***ansible-galaxy remove akry.jenkins***

## ***Ansible Serial/Rolling Deployment***

By using serial/rolling deployment we can achieve zero down time deployment.

```
- hosts: all
  become: yes
  serial: 4
  tasks:
    - name: Disable the server {{ inventory_hostname }}
      debug:
        msg: "Disable the server {{ inventory_hostname }}"

    - name: Update the server {{ inventory_hostname }}
      debug:
        msg: "Update the server {{ inventory_hostname }}"

    - name: Enable the server {{ inventory_hostname }}
      debug:
        msg: "Enable the server {{ inventory_hostname }}"
```

## ***GIT URL FOR PLAYBOOKS***

https://github.com/psddevops/jenkins_pipelines/blob/master/Playbooks