Inventory :

Inventory is a text file where you define the host information that you want to manage with ansible. The default inventory file location is /etc/ansible/hosts. You can specify a different inventory file using the -i <path> option on the command line.

For this exercise we need two Linux servers, you can spin two centos vm or ec2 instance for practice.

Hosts and Groups.

Create a file named inventory-dev(name can be anything) and add below mentioned entry

- Go to Control server (login with user id and password)
- Login with root user
- mkdir ansible_dir
- cd ansible_dir
- vi my_invt

```
root@control:~# mkdir ansible_dir
root@control:~# cd ansible_dir
root@control:~/ansible_dir# vi my_invt
root@control:~/ansible_dir# cat my_invt
web01 ansible_ssh_host=172.16.0.184 ansible_ssh_user=vagrant ansible_ssh_pass=vagrant
db01 ansible_ssh_host=172.16.0.185 ansible_ssh_user=vagrant ansible_ssh_pass=vagrant
local ansible_ssh_host=localhost ansible_ssh_user=vagrant ansible_ssh_pass=vagrant


[webservers]
web01

[dbservers]
db01
root@control:~/ansible_dir#
```

We can create variables at group level as well like below,

We can create variables at group level as well like below

```
root@control:~/ansible_dir# vi my_invt
root@control:~/ansible_dir# cat my_invt
web01 ansible_ssh_host=192.168.0.103
db01 ansible_ssh_host=192.168.0.105


[webservers]
web01

[dbservers]
db01

[webdbgrp:children]
webservers
dbservers


[webdbgrp:vars]
ansible_ssh_user=vagrant
ansible_ssh_pass=vagrant
root@control:~/ansible_dir#
```

Explanation

- web01 and db01 are the names that we have given to the hosts.
- ansible_ssh_host is the variable and its value is the IP address of the server.
- ansible_ssh_user variable holds the username
- ansible_ssh_password holds the password
- [webserver] & [dbservers] is the name of the group which can contain n number hosts. Groupnames are enclosed in square brackets [] .

Note: Mentioning password in the inventory file is not recommended, it's just for initial learning later we will do ssh key exchange.

```
root@control:~/ansible_dir# ansible -i my_invt -m ping web01
web01 | FAILED! => {
    "msg": "Using a SSH password instead of a key is not possible because
ss does not support this.  Please add this host's fingerprint to your kno
}
root@control:~/ansible_dir# vi /etc/ansible/ansible.cfg
root@control:~/ansible_dir# ansible -i my_invt -m ping web01
web01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
root@control:~/ansible_dir# ansible -i my_invt -m ping all
web01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
root@control:~/ansible_dir#
```

Keys Exchange between the Ansible server and connected Nodes :

As we have seen above we put the password in clear text and IP address information also in the inventory. This is a real concern for security, you cannot share this inventory with anyone and also cannot track it in VCS like git. We have better ways to deal with this situation.

1. Since ansible uses SSH, its always recommended to do SSH key exchange and authorize ansible server login to the nodes its managing. Note: Refer Bash Scripting chapter to learn SSH key exchange.

This way we don't need to mention username and password in the inventory file.

2. Next thing is the IP address, we can manage that with the /etc/hosts file. Map IP to hostname in /etc/hosts file and you can then mention the hostname directly in the inventory.

Now we need to create same user in all systems and we have to exchange the keys.

Login web server and db server do the following steps

useradd devops

passwd devops

visudo

and update user details like below in sudoers file. So that created user wil have access to root permission.

```
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
devops  ALL=(ALL)       NOPASSWD: ALL
```

Now go to control server and create key for new user.

```
root@control:~# cd .ssh
root@control:~/.ssh# ls -a
.  ..  authorized_keys  known_hosts
root@control:~/.ssh# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
04:09:eb:ca:34:23:c3:fd:68:ab:04:e4:06:f9:bb:43 root@control
The key's randomart image is:
+--[ RSA 2048]----+
|     ....        |
| .    ...        |
|o.   .   .       |
|=...     .       |
|++=..    S       |
|o=E=o            |
| o+o .           |
|. o..            |
| .oo             |
+-----------------+
root@control:~/.ssh# ls -a
.  ..  authorized_keys  id_rsa  id_rsa.pub  known_hosts
root@control:~/.ssh#
```

Now key has been generate for user called root and we have to copy .pub key to all connected nodes under authorized_keys folder like below.

```
root@control:~/.ssh# ssh-copy-id devops@192.168.0.103
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to f
ed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are p
keys
devops@192.168.0.103's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'devops@192.168.0.103'"
and check to make sure that only the key(s) you wanted were added.

root@control:~/.ssh#
```

```
root@control:~/.ssh# ssh-copy-id devops@192.168.0.105
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to fil
ed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are pr
keys
devops@192.168.0.105's password:

Number of key(s) added: 1

Now try logging into the machine, with:   "ssh 'devops@192.168.0.105'"
and check to make sure that only the key(s) you wanted were added.

root@control:~/.ssh#
```

So we have copied to 2 servers like web and db with devops user from root user on control server.

```
root@control:~/.ssh# ssh devops@192.168.0.103
Welcome to your Vagrant-built virtual machine.
[devops@web ~]$ exit
logout
Connection to 192.168.0.103 closed.
root@control:~/.ssh# ssh devops@192.168.0.105
Welcome to your Vagrant-built virtual machine.
[devops@db ~]$
```

Now we are able to login without password to all connected nodes.

Now we are going to remove password variable from host inventory and changing user name with newly created user like below

```
web01 ansible_ssh_host=192.168.0.103
db01 ansible_ssh_host=192.168.0.105


[webservers]
web01

[dbservers]
db01

[webdbgrp:children]
webservers
dbservers


[webdbgrp:vars]
ansible_ssh_user=devops
```

```
root@control:~/ansible_dir# vi my_invt
root@control:~/ansible_dir# ansible -i my_invt -m ping all
web01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
root@control:~/ansible_dir#
```

Success with keys as password less.

About Modules

Modules (also referred to as “task plugins” or “library plugins”) are the ones that do the actual work in ansible, they are what gets executed in each playbook task. But you can also run a single one using the ‘ansible’ command.

List of all the modules

https://docs.ansible.com/ansible/list_of_all_modules.html

Now we are going to install nginx on web server group

```
root@control:~/ansible_dir# ansible -i my_invt -m yum -a "name=nginx state=installed" --become webservers
web01 | FAILED! => {
    "changed": false,
    "msg": "No package matching 'nginx' found available, installed or updated",
    "rc": 126,
    "results": [
        "No package matching 'nginx' found available, installed or updated"
    ]
}
root@control:~/ansible_dir# ansible -i my_invt -m yum -a "name=epel-release state=installed" --become webserver
s
web01 | SUCCESS => {
    "changed": true,
    "msg": "",
    "rc": 0,
    "results": [
root@control:~/ansible_dir# ansible -i my_invt -m yum -a "name=nginx state=installed" --become webservers
web01 | SUCCESS => {
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "nginx-1.10.2-1.el6.x86_64 providing nginx is already installed"
    ]
}
```

It has been installed now, Am going to start this service using service module like below

Explanation :

- “yum” is a ansible module that manages packages on red hat based systems, for Debian based we use module named “apt”.
- -a is used to provide arguments for the module like name=httpd(key=value). Majority of the modules will have arguments, some arguments are mandatory like “name” argument for “yum”.
- $ ansible-doc yum will show you list of all the arguments for yum module.

- --sudo or - -become tells ansible to execute the module with root privileges, user should have the sudo privileges or else the module will fail.

Output of adhoc commands

ansible command return output is json format

- web1 is the name of the hosts on which module got executed.
- Status is SUCCESS that means it got executed successfully.
- changed: true means that the module execution made some changes in web1.
- changed: false means that the system is in the same desired state as shown below.
- httpd service was already running on web1 so even executing the adhoc command again will not make any changes this is called the IDEMPOTENT behavior in below screen shot.

```
root@control:~/ansible_dir# ansible -i my_invt -m service -a "name=nginx state=started enabled=yes" --become we
bservers
web01 | SUCCESS => {
    "changed": true,
    "enabled": true,
    "name": "nginx",
    "state": "started"
}
root@control:~/ansible_dir# ansible -i my_invt -m service -a "name=nginx state=started enabled=yes" --become we
bservers
web01 | SUCCESS => {
    "changed": false,
    "enabled": true,
    "name": "nginx",
    "state": "started"
}
root@control:~/ansible_dir#
```

How to stop the nginx service.

```
root@control:~/ansible_dir# ansible -i my_invt -m service -a "name=nginx state=stopped enabled=no" --become web
servers
web01 | SUCCESS => {
    "changed": true,
    "enabled": false,
    "name": "nginx",
    "state": "stopped"
}
root@control:~/ansible_dir# ansible -i my_invt -m service -a "name=nginx state=stopped enabled=no" --become web
servers
web01 | SUCCESS => {
    "changed": false,
    "enabled": false,
    "name": "nginx",
    "state": "stopped"
}
root@control:~/ansible_dir#
```

Install httpd now on web server

```
   5/6 \n\r  Verifying  : apr-util-1.3.9-3.el6_0.1.x86_64                                  6/6 \n\nInstalled:\
pd.x86_64 0:2.2.15-60.el6.centos.6                                          \n\nDependency Installed:\n  ap
64 0:1.3.9-5.el6_9.1                                                  \n  apr-util.x86_64 0:1.3.9-3.el6_0.
                                  \n  apr-util-ldap.x86_64 0:1.3.9-3.el6_0.1
          \n  httpd-tools.x86_64 0:2.2.15-60.el6.centos.6                                  \n  mailcap.no
:2.1.31-2.el6                                                \n\nComplete!\n"
    ]
}
root@control:~/ansible_dir# ansible -i my_invt -m yum -a "name=httpd state=installed" --become webservers
web01 | SUCCESS => {
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "httpd-2.2.15-60.el6.centos.6.x86_64 providing httpd is already installed"
    ]
}
root@control:~/ansible_dir#
```

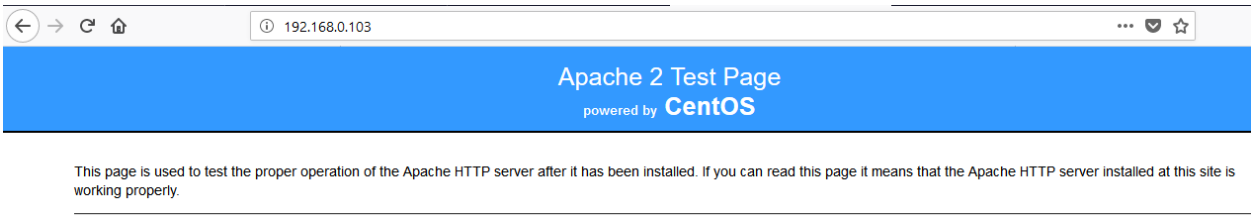
Starting the httpd service now

```
root@control:~/ansible_dir# ansible -i my_invt -m service -a "name=httpd state=started enabled=yes" --become we
bservers
web01 | SUCCESS => {
    "changed": true,
    "enabled": true,
    "name": "httpd",
    "state": "started"
}
root@control:~/ansible_dir# ansible -i my_invt -m service -a "name=httpd state=started enabled=yes" --become we
bservers
web01 | SUCCESS => {
    "changed": false,
    "enabled": true,
    "name": "httpd",
    "state": "started"
}
root@control:~/ansible_dir#
```

Go to browser and type ip address apache2 web page should display if not we have disable iptables on webserver.

```
root@control:~/ansible_dir# ansible -i my_invt -m service -a "name=iptables state=stopped enabled=no" --become
webservers
web01 | SUCCESS => {
    "changed": true,
    "enabled": false,
    "name": "iptables",
    "state": "stopped"
}
root@control:~/ansible_dir# ansible -i my_invt -m service -a "name=iptables state=stopped enabled=no" --become
webservers
web01 | SUCCESS => {
    "changed": false,
    "enabled": false,
    "name": "iptables",
    "state": "stopped"
}
root@control:~/ansible_dir#
```

Now go to browser and refresh the screen



How to copy a file to all connected nodes,

Need to create one file called index.html

```
root@control:~/ansible_dir# vi index.html
root@control:~/ansible_dir# cat index.html
Hello Ansible !!!!!
root@control:~/ansible_dir#
```

```
root@control:~/ansible_dir# ansible -i my_invt -m copy -a "src=index.html dest=/var/www/html/index.html" --beco
me webservers
web01 | SUCCESS => {
    "changed": true,
    "checksum": "c49a8c9d5f6c6b7cb90069298bf29a6e15f673d8",
    "dest": "/var/www/html/index.html",
    "gid": 0,
    "group": "root",
    "md5sum": "973e25f978282bed6c99287ce095a63a",
    "mode": "0644",
    "owner": "root",
    "size": 20,
    "src": "/home/devops/.ansible/tmp/ansible-tmp-1525226614.58-231954198070932/source",
    "state": "file",
    "uid": 0
}
root@control:~/ansible_dir#
```

Again push same file without modifications

```
root@control:~/ansible_dir# ansible -i my_invt -m copy -a "src=index.html dest=/var/www/html/index.html" --beco
me webservers
web01 | SUCCESS => {
    "changed": true,
    "checksum": "c49a8c9d5f6c6b7cb90069298bf29a6e15f673d8",
    "dest": "/var/www/html/index.html",
    "gid": 0,
    "group": "root",
    "md5sum": "973e25f978282bed6c99287ce095a63a",
    "mode": "0644",
    "owner": "root",
    "size": 20,
    "src": "/home/devops/.ansible/tmp/ansible-tmp-1525226614.58-231954198070932/source",
    "state": "file",
    "uid": 0
}
root@control:~/ansible_dir# ansible -i my_invt -m copy -a "src=index.html dest=/var/www/html/index.html" --beco
me webservers
web01 | SUCCESS => {
    "changed": false,
    "checksum": "c49a8c9d5f6c6b7cb90069298bf29a6e15f673d8",
    "dest": "/var/www/html/index.html",
    "gid": 0,
    "group": "root",
    "mode": "0644",
    "owner": "root",
    "path": "/var/www/html/index.html",
    "size": 20,
    "state": "file",
    "uid": 0
}
root@control:~/ansible_dir#
```

Now verify on browser

Few more sample modules with adhoc commands.

To transfer a file directly to many servers:

$ ansible -i inventory-dev -m copy -a "src=/etc/hosts dest=/tmp/hosts" datacenter

The file module allows changing ownership and permissions on files. These same options can be passed directly to the copy module as well:

$ ansible webservers -m file -a "dest=/opt/info.txt mode=600"

$ ansible webservers -m file -a "dest=/opt/info.txt mode=600 owner=devops group=devops"

Ensure a package is installed, but don't update it:

$ ansible webservers -m yum -a "name=acme state=present"

Ensure a package is installed to a specific version:

$ ansible webservers -m yum -a "name=acme-1.5 state=present"

Ensure a package is at the latest version:

$ ansible webservers -m yum -a "name=acme state=latest"

Ensure a package is not installed:

$ ansible webservers -m yum -a "name=acme state=absent"

Ansible Configuration

- Certain settings in Ansible are adjustable via a configuration file. The stock configuration should be sufficient for most users, but there may be reasons you would want to change them**.**
- Changes can be made in global ansible.cfg file /etc/ansible/ansible.cfg or you can create your own ansible.cfg (current working directory) which will have higher precedence over the global file.

$ cat ansible.cfg

[defaults]

hostfile = inventory_prod

host_key_checking=False

#ask_sudo_pass = True

Now I want to create my own config file like below

```
root@control:~/ansible_dir# vi ansible.cfg
root@control:~/ansible_dir# cat ansible.cfg
[defaults]
host_key_checking = False
inventory = my_invt
root@control:~/ansible_dir#
```

This ansible.cfg file is located at the same place where you have the inventory file.

Explanation

- [defaults] is the main section of ansible.cfg
- hostfile will have the value where inventory file is located if it's in the current working directory specify the name or else complete path of the file should be specified. After mentioning inventory path in ansible.cfg its not required to pass the inventory path with -i option.
- host_key_checking=False tells ansible to not check the host fingerprints before doing ssh to the host.
- There long list of ansible config parameters that you can choose from. List of ansible configuration is specified in ansible documentation.

https://docs.ansible.com/ansible/intro_configuration.html

setup - Gathers facts about remote hosts

It can be executed directly by /usr/bin/ansible to check what variables are available to a host. Ansible provides many facts about the system, automatically. Run below the command

$ ansible -m setup web1

```
web1 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.0.2.15",
            "192.168.1.13"
],
        "ansible_all_ipv6_addresses": [
            "fe80::a00:27ff:fe15:d519",
```

```
            "fe80::a00:27ff:febf:5936"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "12/01/2006",
        "ansible_bios_version": "VirtualBox",
        "ansible_cmdline": {
output trimmed
"ohai_uptime": "1 hours 38 minutes 26 seconds",
        "ohai_uptime_seconds": 5906,
        "ohai_virtualization": {
            "role": "guest",
            "system": "vbox"
        }
    },
    "changed": false
```


 # Display facts from all hosts and store them indexed by I(hostname) at C(/tmp/facts).

```
# ansible all -m setup --tree /tmp/facts
```

 # Display only facts regarding memory found by ansible on all hosts and output them.

```
# ansible all -m setup -a 'filter=ansible_*_mb'
```

 # Display only facts returned by facter.

```
# ansible all -m setup -a 'filter=facter_*'
```

 # Display only facts about certain interfaces.

ansible all -m setup -a 'filter=ansible_eth[0-2]'

 # Restrict additional gathered facts to network and virtual.

ansible all -m setup -a 'gather_subset=network,virtual'

 # Do not call puppet facter or ohai even if present.

ansible all -m setup -a 'gather_subset=!facter,!ohai'

 # Only collect the minimum amount of facts:

ansible all -m setup -a 'gather_subset=!all'

 # Display facts from Windows hosts with custom facts stored in C(C:\custom_facts).

ansible windows -m setup -a "fact_path='c:\custom_facts'"

Now if you want to do lot of tasks means we have to type lot adhoc commands so here we are going to create Playbooks.

Playbooks

Playbooks are Ansible's configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process. If Ansible modules are the tools in your workshop, playbooks are your instruction manuals, and your inventory of hosts are your raw material.

At a basic level, playbooks can be used to manage configurations of and deployments to remote machines. At a more advanced level, they can sequence multi-tier rollouts involving rolling updates, and can delegate actions to other hosts, interacting with monitoring servers and load balancers along the way.

Playbooks are a completely different way to use ansible than in adhoc task execution mode, and are particularly powerful.

While you might run the main /usr/bin/ansible program for ad-hoc tasks, playbooks are more likely to be kept in source control and used to push out your configuration or assure the configurations of your remote systems are in spec.

Playbook Language Example

Playbooks are expressed in YAML format and have a minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process.

YAML is data representation language.

YAML Basics

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”. So, we need to know how to write lists and dictionaries in YAML.

There’s another small quirk to YAML. All YAML files (regardless of their association with Ansible or not) can optionally begin with --- and end with .... This is part of the YAML format and indicates the start and end of a document.

All members of a list are lines beginning at the same indentation level starting with a "- " (a dash and a space): ---

Each playbook is composed of one or more ‘plays’ in a list.

The goal of a play is to map a group of hosts to some well-defined roles, represented by things ansible calls tasks. At a basic level, a task is nothing more than a call to an ansible module.

```
root@control:~/ansible_dir# ansible -m ping all
web01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
db01 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
root@control:~/ansible_dir#
```

Just go to Ansible documentation and check about our configuration settings like fork and logs to write in your own config file.

When ever you want to change the default behavior of ansible we have to update the configuration file that simple it is.

Writing a playbook now

```
---
- hosts: websrvs
  become: yes
  tasks:
    - name: Ensure apache is installed
      yum: name=htppd state=installed

    - name: Start httpd
      service: name=httpd state=started enabled=yes


- hosts: dbsrvs
  become: yes
  tasks:
    - name: Install mysql
      yum: name=mysql-server state=installed

    - name: Start mysql
      service: name=mysqld state=started enabled=yes
```

We have to execute that using below command

ansible-playbook myplaybook.yml

One More Playbook example :

$ cat web_db.yaml

- hosts: webservers

  become: yes

  tasks:

  - name: Ensure Apache installed

    yum: name=httpd state=present

  - name: Creates directory

    file: path=/var/www/html/ansible state=directory

  - name: Deploy webpage to path=/var/www/html/ansible

```
    copy: src=index.html dest=/var/www/html/ansible/ mode=0644

  - name: Ensure Apache is running
    service: name=httpd enabled=yes state=started

  - name: Flush all temporary rules
    service: name=iptables state=restarted

  - name: Allow port 80/http access from anywhere
    iptables:
      action: insert
      chain: INPUT
      protocol: tcp
      destination_port: 80
      state: present
      source: 0.0.0.0/0
      jump: ACCEPT



- hosts: dbservers
  become: yes
tasks:
  - name: Ensure mysql server installed
    yum: name=mysql-server state=present
```

```
- name: Ensure mysql running

  service: name=mysqld state=started


- name: Ensure MySQL-python is installed

  yum: name=MySQL-python state=present


- name: Create Database

  mysql_db: name=devops state=present


- name: Create user named mint

  mysql_user: name=mint  password=12345 priv='*.*:ALL' state=present
```

Explanation :

- --- is not mandatory but specifies a start of YAML file
- - hosts: webservers represent the host/group name where tasks will get executed. It is the start of a play.
- become: yes tell ansible to execute all the tasks with sudo privileges older version of ansible has sudo:yes
- tasks: specifies the list of task or modules that will be executed against webserver group.
- - name: is not a mandatory option but will always help us read the output of task when executed. If -name option is not specified then the module name should start with a " - " for example "– yum:"
- yum: is the name of the module that will get executed along with the arguments "name=httpd state=present" . If -name option is not provided then yum will begin with a "– " that represents the element in the YAML list as specified below.

```
tasks:

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

- iptables module modifies the systems firewall rule, knowledge of iptables is required to understand its options. We are allowing port 80/http access from everywhere.
- - hosts:dbservers is the start of a next play that will get executed on dbservers group. As mentioned earlier playbook is the list of Plays, os -hosts: dbservers is just the second play in the playbook. Likewise we can have multiple plays in the same playbook.
- mysql_db module is used to create/delete databases in mysql db service.

- https://docs.ansible.com/ansible/mysql_db_module.html
- mysql_user module is used to create user in mysql db service.
- https://docs.ansible.com/ansible/mysql_user_module.html NOTE: Its highly recommended to read ansible module documentation to understand more about modules used in the playbook.

Running Playbook :

ansible-playbook playbookname.yml

Explanation

- Playbook gets executed in top to down order. First play is PLAY [webservers].
- TASK [Gathering Facts] is a default task that runs the setup module for every host in the play. We can disable gathering facts by specifying gather_facts: False in playbook as shown below.
- If the -name option is used in the task then it will display the content from it.
- changed: [web1] is a status message for the task, it means that the task made changes to the target host.
- ok: [web1] means that the task has not made any changes on the target host. It could be due to the nature of the module which could be for information gathering like setup module. It could also mean that the system is in the same state for example httpd is already installed so it will return ok:
- PLAY RECAP Display the summary of all the tasks got executed on all the hosts. unreachable display the number of hosts that does not have proper connectivity with ansible server. failed displays number of failed tasks.

Ansible documentation gives very nice description of every module with examples.

https://docs.ansible.com/ansible/list_of_all_modules.html

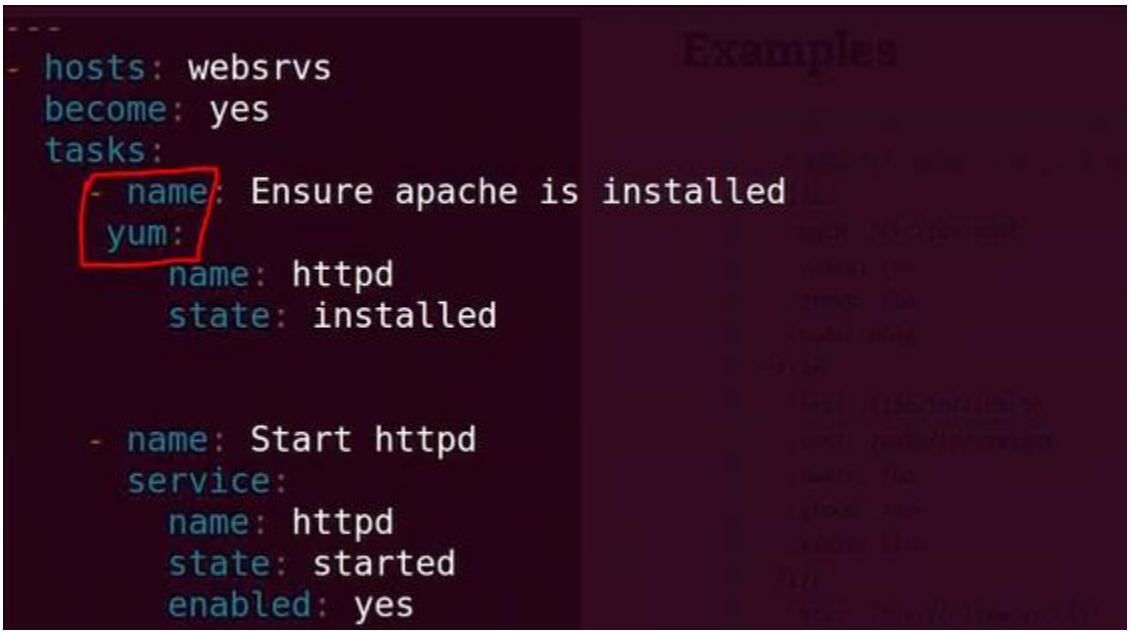
Setting up web server yml file :

```
---
- hosts: websrvs
  become: yes
  tasks:
    - name: Ensure apache is installed
      yum:
        name: httpd
        state: installed


    - name: Start httpd
      service:
        name: httpd
        state: started
        enabled: yes

    - name: Create ansible directory in apache http document root directory
      file:
        path: /var/www/html/ansible
        state: directory
        mode: 0755

    - copy:
        src: index.html
        dest: /var/www/html/index.html
        mode: 0644
```

Run this playbook

The above playbook has been in a new style. Here we are having some syntax issues with white spaces. Always error will come white spaces.

Like below



Now run the playbook we are able to see the failure.

Now we are writing second section for second group in latest style.

```
    - name: Create ansible directory in apache http document root directory
      file:
        path: /var/www/html/ansible
        state: directory
        mode: 0755

    - copy:
        src: index.html
        dest: /var/www/html/index.html
        mode: 0644



- hosts: dbsrvs
  become: yes
  tasks:
    - yum:
        name: mysql-server
        state: installed
~
~
:wq
```

Always go with modules and always go with ansible documentation to create playbooks for your tasks.

If something not matching with your requirement we have to go with module called shell module.

Examples

```
- name: Execute the command in remote shell; stdout goes to the specified file on the remote.
  shell: somescript.sh >> somelog.txt

- name: Change the working directory to somedir/ before executing the command.
  shell: somescript.sh >> somelog.txt
  args:
    chdir: somedir/
```

Shell

Cmd

Script

These 3 modules we can go evenever our requirement is not matching with any module we have to go with these 3 modules but always avoid to use these since these three are not idempotent.

New and Old argument style

Old Style

tasks:

- yum: name=httpd state=present

New Style

tasks:

 - yum:

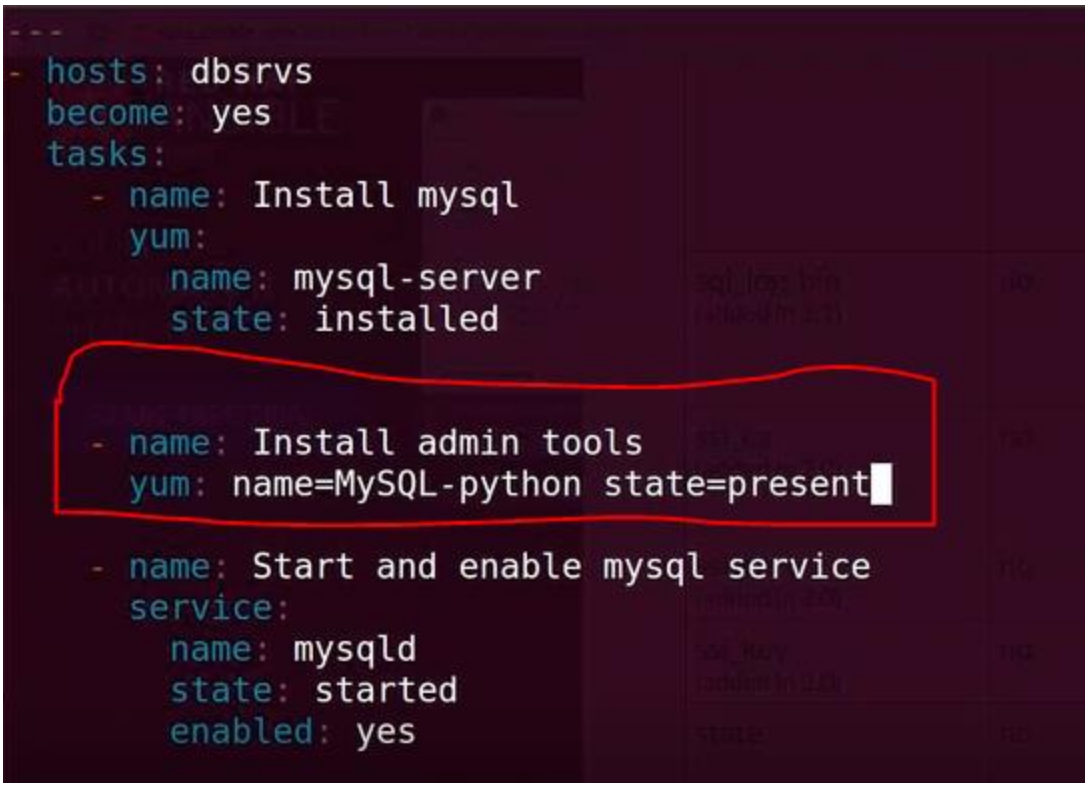
    name: httpd

    state: present

Creating DataBase YML file :

Here we wil get this below error since we will not have required dependency package so we have to install this package **MySQL-python** first.

We are going to modify again our database.yml file



Now successfully Database server database and user has been created if we want to check about all creation we can test now via terminal like below,

mysql –h db01 –u admin –p

enter password

Few commands can be executed here to check about database configuration.

show databases;

use accounts;

show tables;

Looping statements :

If I keep add like this all required packages if I need 10 or 15 packages my playbook size and length wil get increased so here am going to do in a different way with multiple package options.

```
tasks:
  - name: Install mysql
    yum:
      name: mysql-server
      state: installed


  - name: Install admin tools
    yum: name={{item}} state=present
    with_items:
      - MySQL-python
      - wget
      - git

  - name: Starte list of services
    service: name={{item}} state=restarted
    with_items:
      - httpd
      - snmp
      - mysqld
```

Above changes will not work here variable is not in quotes so always variables should in quotes and am going to to write in new style.

```
tasks:
  - name: Install mysql
    yum:
      name: mysql-server
      state: installed


  - name: Install admin tools
    yum:
      name: "{{item}}"
      state: present
    with_items:
      - MySQL-python
      - wget
      - git
```

Item is a variable here.

- mysql_db module is used to create/delete databases in mysql db service.

  https://docs.ansible.com/ansible/mysql_db_module.html

- mysql_user module is used to create user in mysql db service.

  https://docs.ansible.com/ansible/mysql_user_module.html

- Playbook gets executed in top to down order. First play is PLAY [webservers].
- TASK [Gathering Facts] is a default task that runs the setup module for every host in the play. We can disable gathering facts by specifying gather_facts: False in playbook as shown below.

Variables Defined in a Playbook :

Variables can be defined with some custom value in playbook as shown below. These variables can be used in the playbook by enclosing variable name in {{varname}}.

After Host section we have to create variable section like below and in playbook where ever we need these variable values we can replace with name with quotes. "{{varname}}"

The benefit of using variables is in playbook the playbooks can be re usable for our all projects need easily.

```
- hosts: dbservers
  become: yes
  vars:
   dbpackage: mysql-server
   dbname: mysqldb
   dbuser: admin
   dbpass: 12345
```

```
---
- hosts: dbsrvs
  become: yes
  vars:
    dbpackage: mysql-server
  tasks:
    - name: Install mysql
      yum:
        name: "{{dbpackage}}"
        state: installed

- name: Create a new database with name accounts
  mysql_db:
    name: "{{dbname}}"
    state: present


- name: Create user admin with all priveleges
  mysql_user:
    name: "{{dbadmin}}"
    password: 12345
    priv: '*.*:ALL'
    state: present
```

Variables can be write there are several places in ansible. First place is jst write variables in playbook itself. And it can be defined outside of the playbook as well like below

Variables in group_vars & host_vars

These variables are inventory specific and can only be accessed by host & groups from the inventory located in current directory. You can have multiple inventory in separate directory structure as shown below. Every inventory may have its own group_vars & host_vars directory where we store variables.

Variables can be defined into a directory structure. group_vars holds variables that can be used by all the groups. host_vars holds variable specific to the hostname.

- group_vars/all will contain variables that can be used by all the hosts from the inventory file.
- group_vars/dbservers variables will be only accessible for the dbservers group and not any other host or group from inventory.
- host_vars/web1 variables will be only accessible for the web1 host and not any other host from the inventory file.

Just create 2 directories here

mkdir group_vars

mkdir host_vars

creating a variable file called all in group_vars directory.

vi group_vars/all ---- All groups can be accessed these variables in your playbook

```
# Variables accessible by all groups
dbname: accounts
dbpass: 12345
```

vi group_vars/<Group Name>

example : vi group_vars/webservers ---- Only this group can be used variables which were defined here but make sure that group name should be correct inside of host inventory file.

```
# Variables accessible by websrvs group only
webpackage: httpd
webfile: index.html
destwebfile: /var/www/html/
```

Here we can create new file for dbservers group as well

And we can defined variables for hostfile as well like for each host like below

vi host_vars/web01

you can mention how many variables you want to add here.

Variable s can be used in a order if you are having same name of variable in multiple places. The order of execution is below

playbook

host_vars/hostname

group_vars/groupname

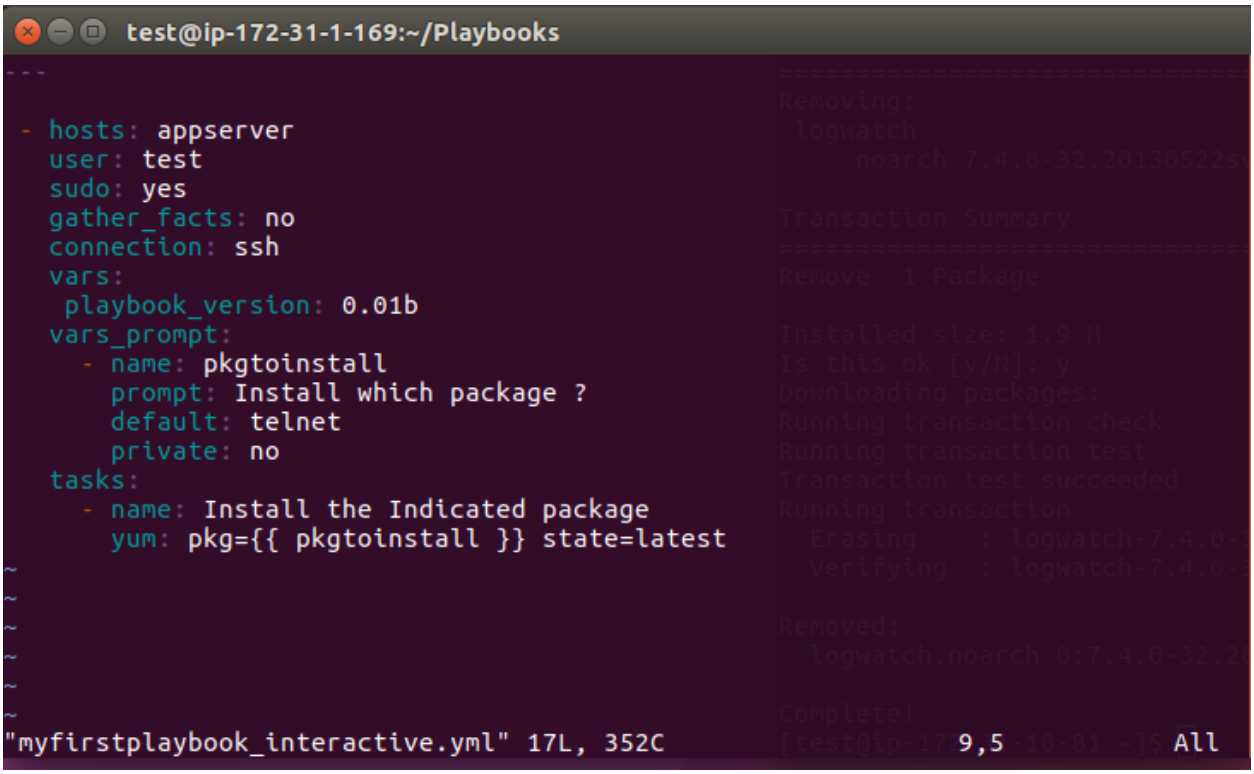
group_vars/all

if you don't find anywhere here it is going to give an error finally.

Prompt Variables :

Writing a playbook with Interactive mode

Now that we have seen a playbook, let's step back and develop a more complex one and explaining it a section at a time, Now how to use the variables we have defined.



Including Playbooks :

In one yml , we call other playbooks. Note this is SUPER short, because it's just including some other playbooks. Remember, playbooks are nothing more than lists of plays:

- include: webservers.yml

- include: dbservers.yml

We have to write like above in other playbook.

Store Output of a command.

Register module is used to store output of any module/command and store it into a variable

```
---
- hosts: all
  become: yes
  tasks:
    - name: Execute linux command
      shell: /usr/bin/whoami

    - name: Execute linux command
      shell: mkdir /tmp/test1
```

Just run for 2 times this yml file for second time it is going to give an error why means shell module is not idempotent.

```
---
- hosts: all
  become: yes
  gather_facts: False
  tasks:
    - name: Execute linux command
      shell: /usr/bin/whoami
      register: usrnm

    - debug:
        msg: "{{usrnm}}"

#     - name: Execute linux command
#       shell: mkdir /tmp/test1
```

Debug module is used to display the output.

Debug module is used to print messages or variable values while playbook execution. It helps finding the problem if the variable values are not properly assigned or accessed.

```
TASK [debug] ********************************************************
ok: [web01] => {
    "msg": {
        "changed": true,
        "cmd": "/usr/bin/whoami",
        "delta": "0:00:00.002655",
        "end": "2018-01-29 03:34:38.490264",
        "failed": false,
        "rc": 0,
        "start": "2018-01-29 03:34:38.487609",
        "stderr": "",
        "stderr_lines": [],
        "stdout": "root",
        "stdout_lines": [
            "root"
        ]
    }
```

Here the output is all about the data. But I ned only user name means I have to use stdout variable like below.

```
---
- hosts: all
  become: yes
  gather_facts: False
  tasks:
    - name: Execute linux command
      shell: /usr/bin/whoami
      register: usrnm

    - debug:
        msg: "{{usrnm.stdout}}"

#    - name: Execute linux command
#      shell: mkdir /tmp/test1
```

```
TASK [debug] ********************************
ok: [db01] => {
    "msg": "root"
}
ok: [web01] => {
    "msg": "root"
}
```

Fact Variables :

Fact variables we are getting using setup module means getting the data about the system, that is Gather_facts target section variable

```
ansible -m setup web01
```

```
ansible -m setup web01 | grep ansible_os_family
 "ansible_os_family": "RedHat",
```

Now writing a playbook to install httpd service and apache2 service on single redhat system in a single playbook like below.

```
---
- hosts: websrvs
  become: yes
  gather_facts: False
  tasks:
    - name: Install httpd on cento
      yum: name=httpd state=present

    - name: Install apache2 on ubuntu
      apt: name=apache2 state=present
```

Output

```
TASK [Install httpd on cento] ***********************************************
ok: [web01]

TASK [Install apache2 on ubuntu] ********************************************
fatal: [web01]: FAILED! => {"changed": false, "cmd": "apt-get update", "msg":
 file or directory", "rc": 2}
        to retry, use: --limit @/root/ansible-repo/exercise5/whenos.retry

PLAY RECAP ******************************************************************
web01                      : ok=1    changed=0    unreachable=0    failed=1
```

But in centos system apt modle can't work so here we have to get gather_facts and verify the host and then apply conditional statements using when module. That's the advantage of fact variables.

```
---
- hosts: websrvs
  become: yes
#  gather_facts: False
  tasks:
    - name: Install httpd on cento
      yum: name=httpd state=present
      when: ansible_os_family=="RedHat"

    - name: Install apache2 on ubuntu
      apt: name=apache2 state=present
      when: ansible_os_family=="Debian"
```

Interactive playbook using Promt Variables :

Take user input while executing playbook with vars_prompt and store into a variable as shown below.

```
---

- hosts: dbservers

  vars:

   http_port: 8087

   username: cassini

  vars_prompt:

   - name: "dbpass"
```

prompt: "Enter password for database."

tasks:

- debug: msg="DB password is {{dbpass}}"

```
---
- hosts: dbservers
  vars:
   http_port: 8087
   username: cassini
  vars_prompt:
   - name: "dbpass"
     prompt: "Enter password for database."
  tasks:
     - debug: msg="DB password is {{dbpass}}"
```

```
---
- hosts: dbservers
  vars:
   http_port: 8087
   username: cassini
  vars_prompt:
   - name: "dbpass"
     prompt: "Enter password for database."
  tasks:
     - debug: msg="DB password is {{dbpass}}"
~
~
```

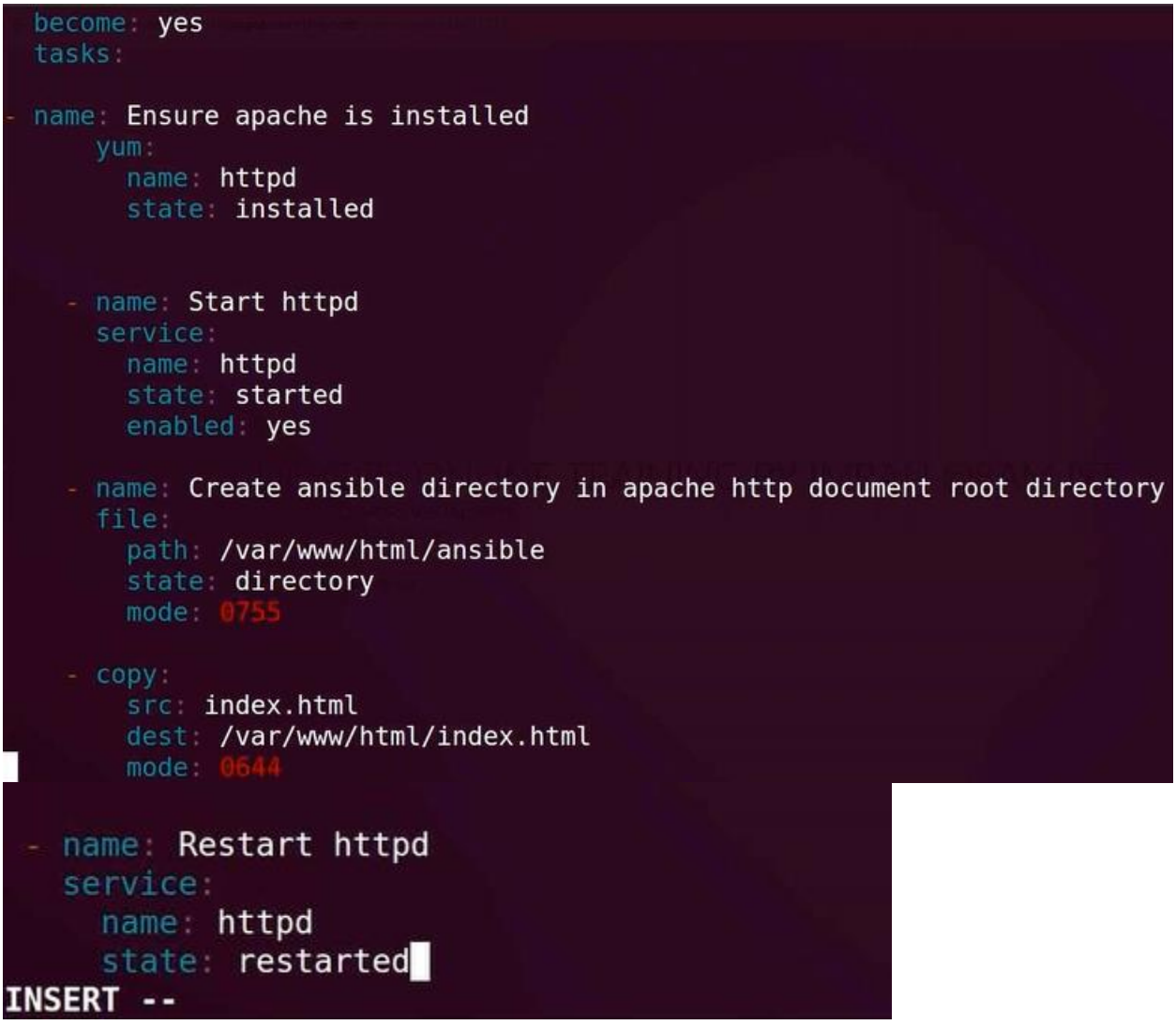
Handlers :

Handlers are special kind of tasks. In first glimpse, it will look exactly like any other task but the difference is in the execution. Tasks as we have seen so far gets executed as we run our playbook but handlers being in same playbook will only get executed when it gets notified. Notification would be sent from the task if the state of the task is changed: true.

For example, if we copy a file using copy module it gets copied if the destination file is different or not present. In this case the state of the task is changed: true but if the destination file is same as source then the file does not get overwritten and state would be changed: false.

So, if we are notifying a handler from such task the handler will get notified only if the file is copied or else it will not send a notification.

- hosts: webservers



Am going to run the playbook now.

ansible-playbook webserver.yml

We can restart is successs once copied index.html file to nodes. Then again we wanted to add one more task at the end now.

```
- name: Restart httpd
  service:
    name: httpd
    state: restarted

- name:
  file:
    path: /tmp/testingsomething
    state: directory
    mode: 0777
```

Again re run the playbook what wil happen in the series of sequence even if you don't want to restart now since our index.html not changed but it is going to restart let see

ansible-playbook webserver.yml

```
TASK [copy] ********************************************************************
ok: [web01]

TASK [Restart httpd] ***********************************************************
changed: [web01]

TASK [file] ********************************************************************
changed: [web01]

PLAY RECAP *********************************************************************
web01                      : ok=7    changed=2    unreachable=0    failed=0
```

It has been restarted again but I need restart only when I change my index.html config file only so for that we need to go Handlers section. Only need restart when comething changed into config file like index.html.

Handlers are again same like task we have to write.

```
    - name: Deploy website
      copy:
        src: index.html
        dest: /var/www/html/index.html
        mode: 0644
      notify:
        - Restart httpd

#     - name: Restart httpd
#       service:
#         name: httpd
#         state: restarted
```

```
  - name:
    file:
      path: /tmp/testingsomething
      state: directory
      mode: 0777

handlers:
  - name: Restart httpd
    service:
      name: httpd
      state: restarted
```

Now just some change into index.html and run playbook that time only it is going to restart. When ever we are having tasks like restarts we have to do in handlers section.

Templates :

Templates are similar to copy module, it copies the source file to the target hosts destination.

But here we don't have plain static files, we have template file which contains pre-defined variables.

These variables could be defined in playbook or host_vars or group_vars etc.

While the template module gets executed it will read the template file and change all the variables to its value and copy the file to the target host.

Template file ends with .j2 extension which stands for Jinja2 templates.

Copy module and Template module both can be used for pushing the files connected hosts. But only the difference is Template moduleis going to read the content of a template file is there any variable to

change to its value and then push the file. Now variable in that it could be fact variable it could be you have self defined variable of a playbook.

```
---
- hosts: websrvs
  become: yes
  gather_facts: False
  vars:
    webserver: APACHE HTTPD
  tasks:
    - name: Ensure apache is installed
      yum:
        name: httpd
        state: installed
```

```
- name: Deploy website
  template:
    src: index.j2
    dest: /var/www/html/index.html
    mode: 0644
  notify:
    - Restart httpd
```

The above playbook having one self defined variable and using Template module we are pushing index.j2 template file

mv index.html index.j2 first

then vi index.j2

index.j2 is not a regular file it can be called as a template file notation.

```
Testing service restarts
Testing handlers now
OS architecture is {{ansible_architecture}}
We are deploying {{webserver}}
```

And here gather_facts should not be false case.

Run the playbook now.

And to verify that we need to login connected system and go to location and vefiy the changes like below,

```
[vagrant@web ~]$ sudo -i
[root@web ~]# cd /var/www/html/
[root@web html]# ls
ansible  index.html
[root@web html]# cat index.html
Testing service restarts
Testing handlers now
OS architecture is x86_64
We are deploying APACHE HTTPD
[root@web html]#
```

Just go to browser and verify the text now.

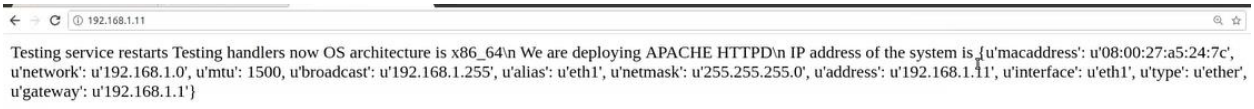
Ok if I want to verify what is the Ip address of node we can add one more fact variable into index.j2 template

```
Testing service restarts
Testing handlers now
OS architecture is {{ansible_architecture}}\n
We are deploying {{webserver}}\n
IP address of the system is {{ansible_default_ipv4}}
~
```

Now re run the playbook and go to browser and verify the things.

ansible-playbook webserver.yml

When you go to browser it going to key list of key values of fact variable.



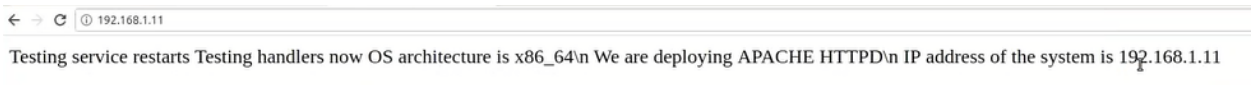
vi index.j2

```
Testing service restarts
Testing handlers now
OS architecture is {{ansible_architecture}}\n
We are deploying {{webserver}}\n
IP address of the system is {{ansible_default_ipv4.address}}
```

Re run the same playbook

Sometimes the fact variables will be having a value going to be normal text or string but some times wil have a value that will be again in nested so to access that time we have to givem them “.” And need to access the value.

Go to browser now and verify the text



Here he can identify all fact variables and you don't need to remember when ever we need we will just go to setup module and we will take that fact variable.

Execrcise3 :

Here we can create our http conf file and our own index.html file using Templates. Here we are having web.yml httpd.j2 and index.j2 files using those we can do one exercise here.

mkdir templates

cd templates

vi httpd.j2

vi index.j2

copy content from our examples codes

vi web.yml under our exercise folder itself

```
root@control:~/excercise1# ls
ansible.cfg  my_invt  templates  web.retry  web.yml
root@control:~/excercise1#
```

```
root@control:~/excercise1# cd templates/
root@control:~/excercise1/templates# ls
httpd.j2  index.j2
root@control:~/excercise1/templates#
```

Then Run Playbook.

Serial Execution : vi serial.yml

```
---
- hosts: webservers
  serial: 1
  tasks:
    - name: sleep for 5 seconds
      shell: /bin/sleep 5
~
```

```
Ansible-master#ansible-playbook serial.yaml

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

TASK [Gathering Facts] **********************************************************

ok: [agent]

TASK [sleep for 5 seconds] ******************************************************

changed: [agent]

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

TASK [Gathering Facts] **********************************************************

ok: [agent2]

TASK [sleep for 5 seconds] ******************************************************

changed: [agent2]

PLAY RECAP **********************************************************************

agent                      : ok=2    changed=1    unreachable=0    failed=0
agent2                     : ok=2    changed=1    unreachable=0    failed=0

Ansible-master#
```

Parallel Execution : By default playbook will execute paralally.

vi parallel.yml

```
---
- hosts: webservers
  tasks:
    - name: sleep for 5 seconds
      shell: echo hello
~
```

```
Ansible-master#ansible-playbook parallel.yaml

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

TASK [Gathering Facts] **********************************************************************

ok: [agent2]
ok: [agent]

TASK [sleep for 5 seconds] ******************************************************************

changed: [agent2]
changed: [agent]

PLAY RECAP **********************************************************************************

agent                      : ok=2    changed=1    unreachable=0    failed=0
agent2                     : ok=2    changed=1    unreachable=0    failed=0

Ansible-master#
```

parallel execution

serial: 1 ==> on each machine execute serially

4 hosts
serial:1 ==>

4 hosts
1 and2 - parallelly
3 and 4 - parallelly
serial: 2 ==>

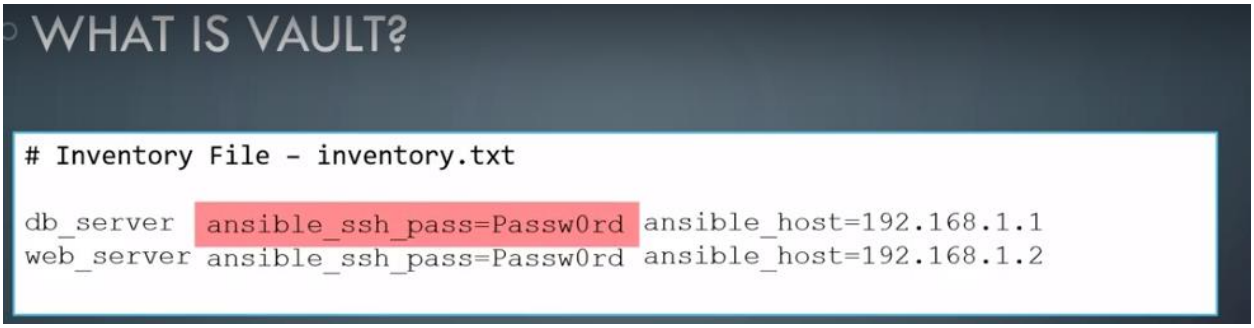
Ansible Vault

Managing secrets with Ansible vault.

The vault feature can encrypt any structured data file used by Ansible. This can include “group_vars/” or “host_vars/” inventory variables, variables loaded by “include_vars” or “vars_files”, or variable files passed on the ansible-playbook command line with “-e @file.yml” or “-e @file.json”. Role variables and defaults are also included!

We can store our passwords/secrets encrypted in the ansible vault.

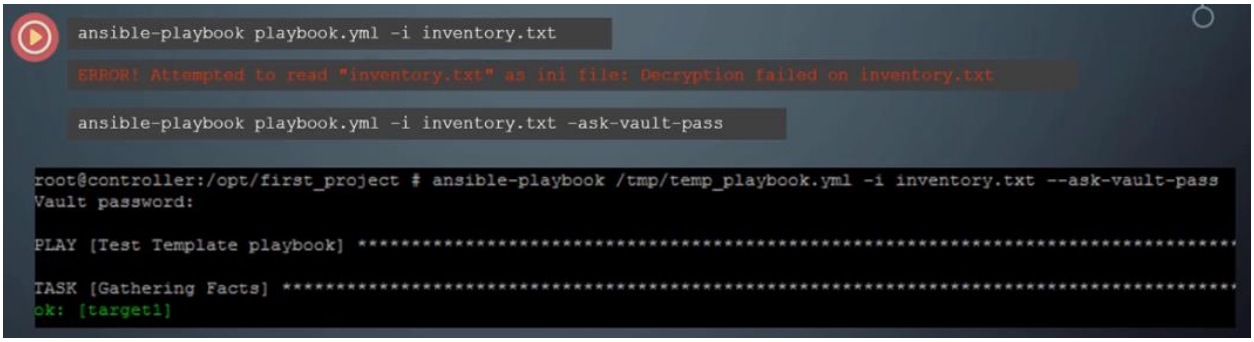
Ansible vault can be stored data in a encrypted format.

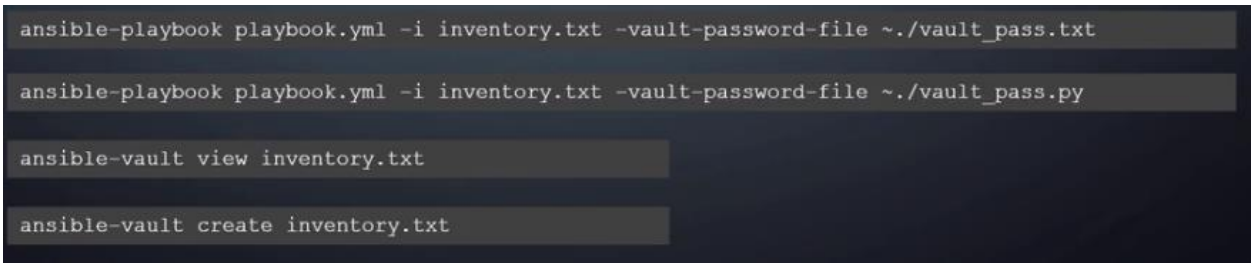


ansible-vault encrypt inventory.txt

```
$ANSIBLE_VAULT;1.1;AES256
61383464383939633238383239356239666432313565333463636435326462363863323263636261
64326238643130326364346139313162626465346331651340a32366433366132396166636132643 0
62636562333738636638376631326233646130386133646438633739623362646238626438356265
65346633353861383 70a623133653339356138623831306638383838363839303866303031643038
33373061653863303664383935316662623065316137343361313435313761303332633637333932
64623362623565396665393237356430653966616339643666393832346333636632663136306633
61343865376362643166356466665383661393766623662623564613063323839336139663361316 2
65633033386663383638323265646365363465366533313161313166323133633830306263663039
```

It looks like above.





- ansible-vault encrypt <> - to encrypt the file
- ansible-valut edit <> - to edit the file
- ansible-vault rekey – to change password

Summary:

- Configuration Management tools have majorly replaced the scripting languages in DevOps for automation.
- It's easy to manage and automate infrastructure from a centralised place compared to scripts.
- Ansible is the current hot favourite automation tool and is very widely adopted in DevOps life cycles.
- It's easy to setup. Easy to read and write, does not need any programming knowledge.
- Ansible gives it little but very powerful AdHoc commands which is used to execute single tasks on multiple servers.
- Playbooks are automation scripts of Ansible and includes series of tasks that should be executed on selected hosts.
- Ansible Roles gives us a very modular structure to our code and make the code sharable with other team or projects. It focuses on reusability of the code.
- There are so many Sample Playbook in this chapter which you can use in your day to day DevOps Automation.
- Ansible is very innovative and releases new modules every now and then. Always refer Ansible documentation, because that's the single source of truth for everything in Ansible.

https://docs.ansible.com/