Ansible Up and Running

What is Configuration management

The process of **standardizing** and **administering resource** configurations and entire IT infrastructure in an automated way is Configuration Management. It is the concept where you put your server infrastructure as code.

Pull based and Push Base Configuration

Configuration Management tools implement one (or both) of these models of management. **Push-based CM and Pull-based CM** are the ways in which a CM tool performs actions, like installing packages or writing files.

Pull Model : Good scalability but difficult management.

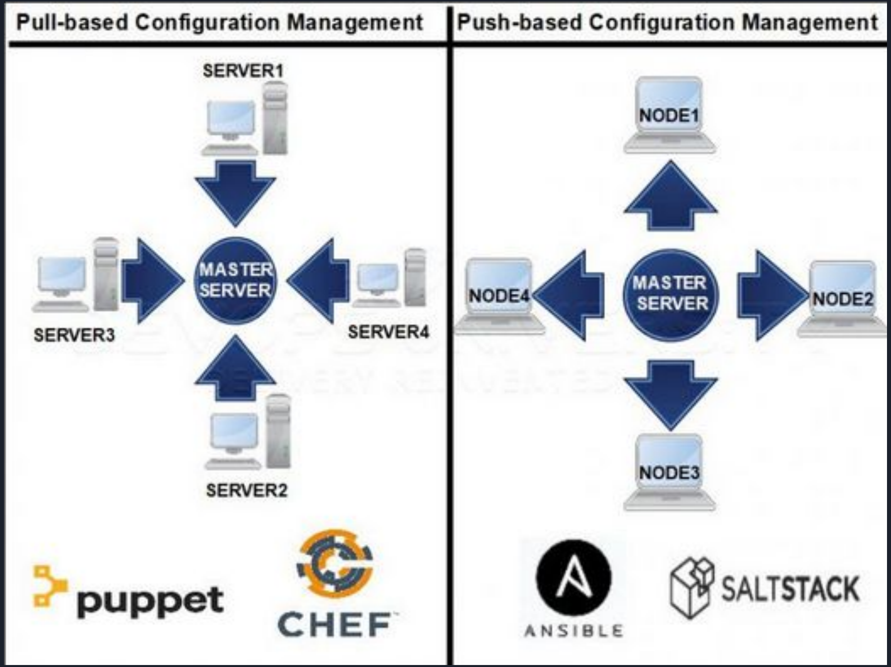
- ➔ The server nodes run an agent daemon that periodically checks from the master node if/when there are any updates to be pulled and applied.
- ➔ A daemon needs to be installed on all machines and a setup of the central authority is required.

Push Model : Simple management and easy setup but poor scalability.

→ Here, it is the central server or the master node which takes the responsibility to contact the server nodes to send updates as and when they occur.

→ Whenever a change is made to the infrastructure (code), each node is informed of the update and they run the changes.

Configuration management tools facilitate faster, repeatable, scalable and predictable deployments and help in maintaining the desired state



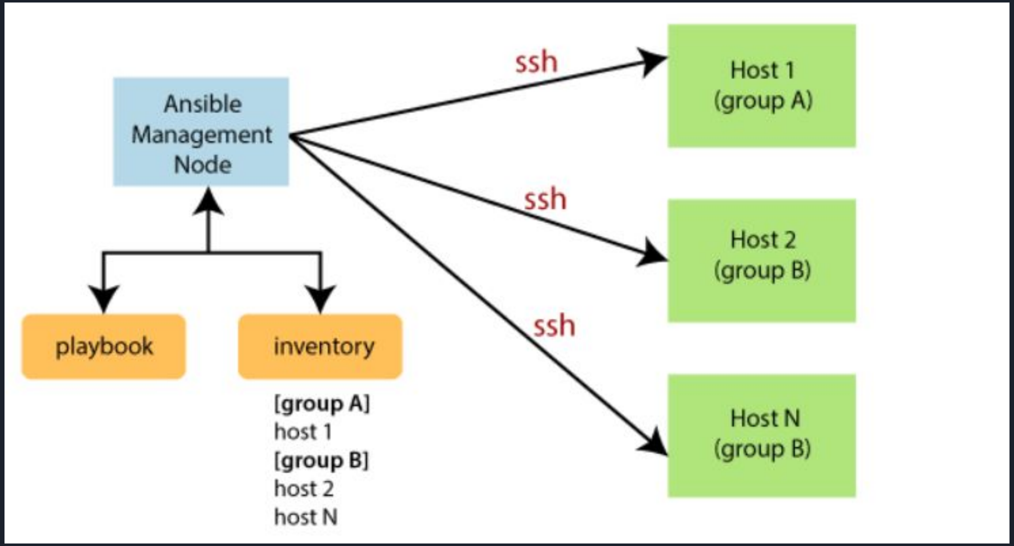
What is Ansible ?

Ansible is an open-source IT engine that automates application deployment, cloud provisioning, intra service orchestration, and other IT tools.

- It can easily connect to clients using **SSH-Keys,** simplifying though the whole process. Client details, such as **hostnames** or **IP addresses** and **SSH ports,** are stored in the files, which are called inventory files. If you created an inventory file and populated it, then Ansible can use it.
- It is very Simple tool to use yet powerful enough to **automate Complex IT applications and infrastructures**

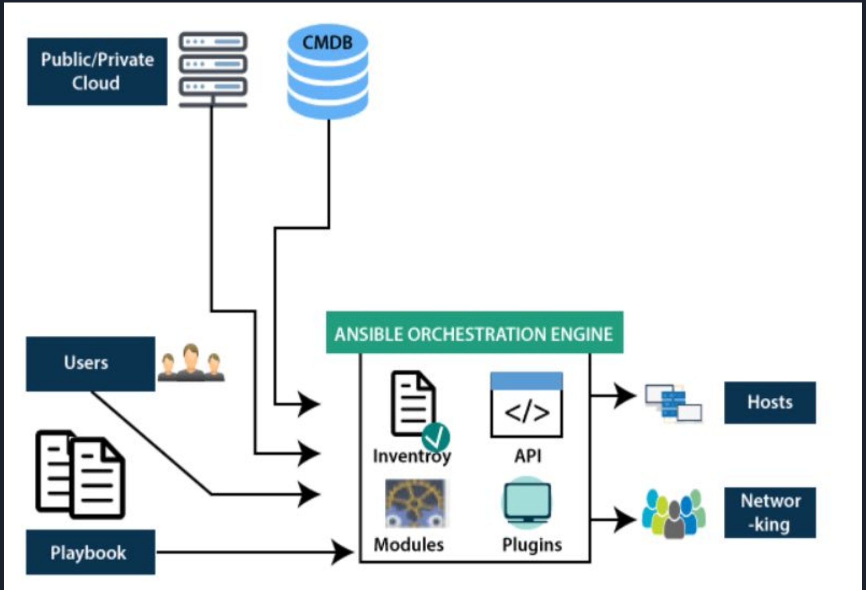
Ansible Workflow

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.



Ansible Architecture

The **Ansible orchestration** engine interacts with a user who is writing the Ansible playbook to execute the Ansible orchestration and interact along with the services of private or public cloud and configuration management database. You can show in the below diagram, such as:



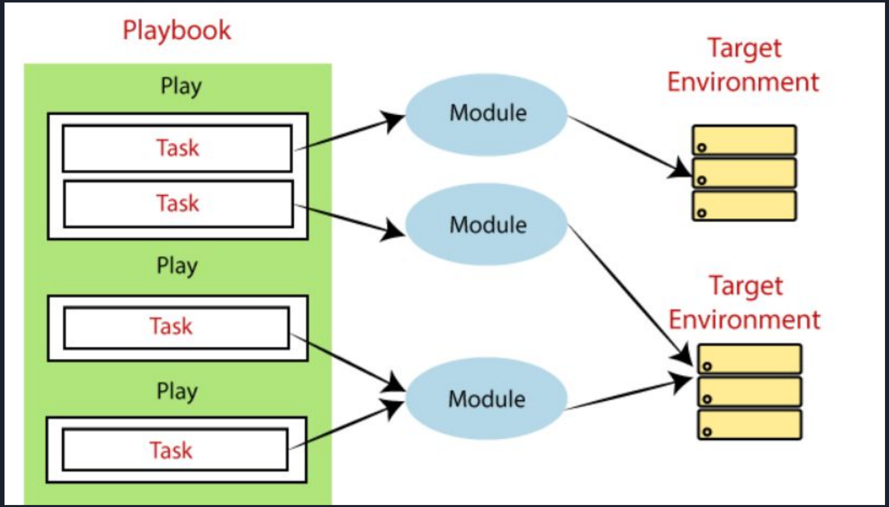
Architecture	Description
Inventory	Inventory is lists of nodes or hosts having their IP addresses, databases, servers, etc. which are need to be managed.
API's	APIs are used to transport content for Cloud services, public or private
Modules	The modules can control system resources, like services, packages, or files (anything really),
Plugins	Plugins is a piece of code that expends the core functionality of Ansible. There are many useful plugins, and you also can write your own.
Playbooks	Playbooks consist of your written code, and they are written in YAML format, which describes the tasks and executes through the Ansible. Also, you can launch the tasks synchronously and asynchronously with playbooks.
Hosts	In the Ansible architecture, hosts are the node systems, which are automated by Ansible, and any machine such as RedHat, Linux, Windows, etc
Networking	Ansible is used to automate different networks, and it uses the simple, secure, and powerful agentless automation framework for IT operations and development
CMDB	CMDB is a type of repository which acts as a data warehouse for the IT installations.

Ansible Playbook

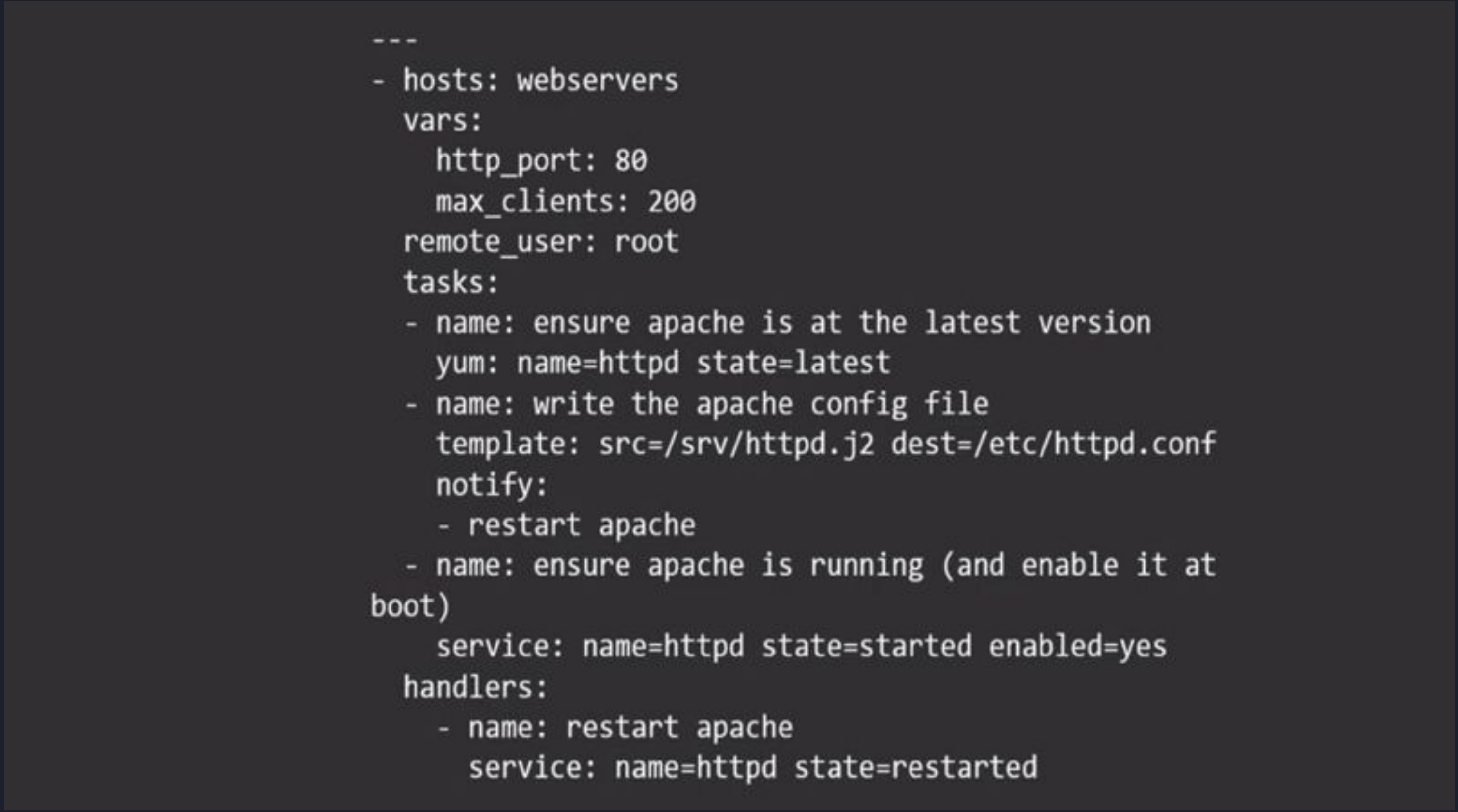
Playbooks are the files where Ansible code is written. **Playbooks are written in YAML format.** YAML stands for Yet Another Markup Language. Playbooks are one of the core features of Ansible and tell Ansible what to execute. They are like a to-do list for Ansible that contains a list of tasks.

Playbooks contain the steps which the user wants to execute on a particular machine. And playbooks are run sequentially. Playbooks are the building blocks for all the use cases of Ansible.

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.



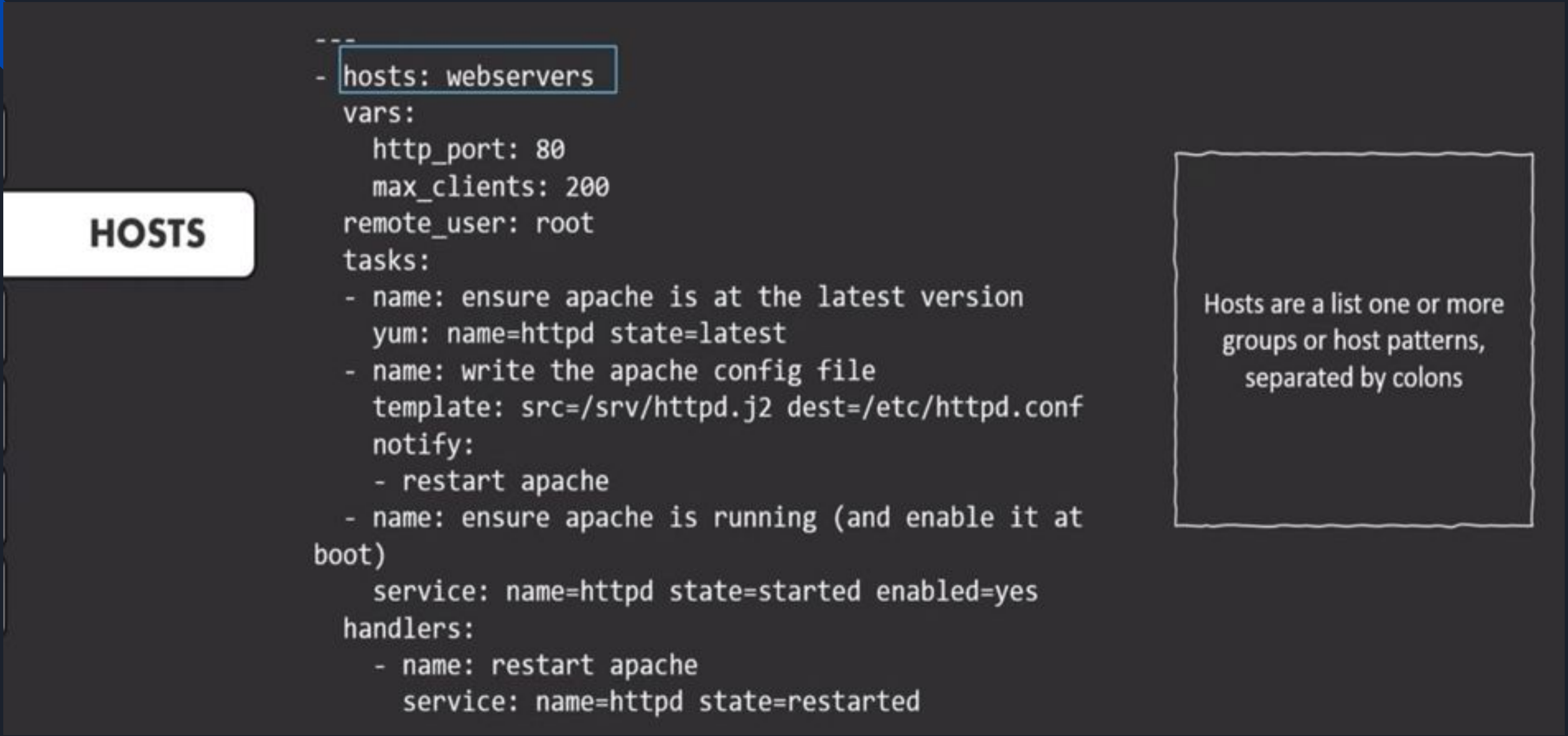
How to create PlayBook ?



```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at
boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at
boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Every YAML file starts with 3 dashes [---]

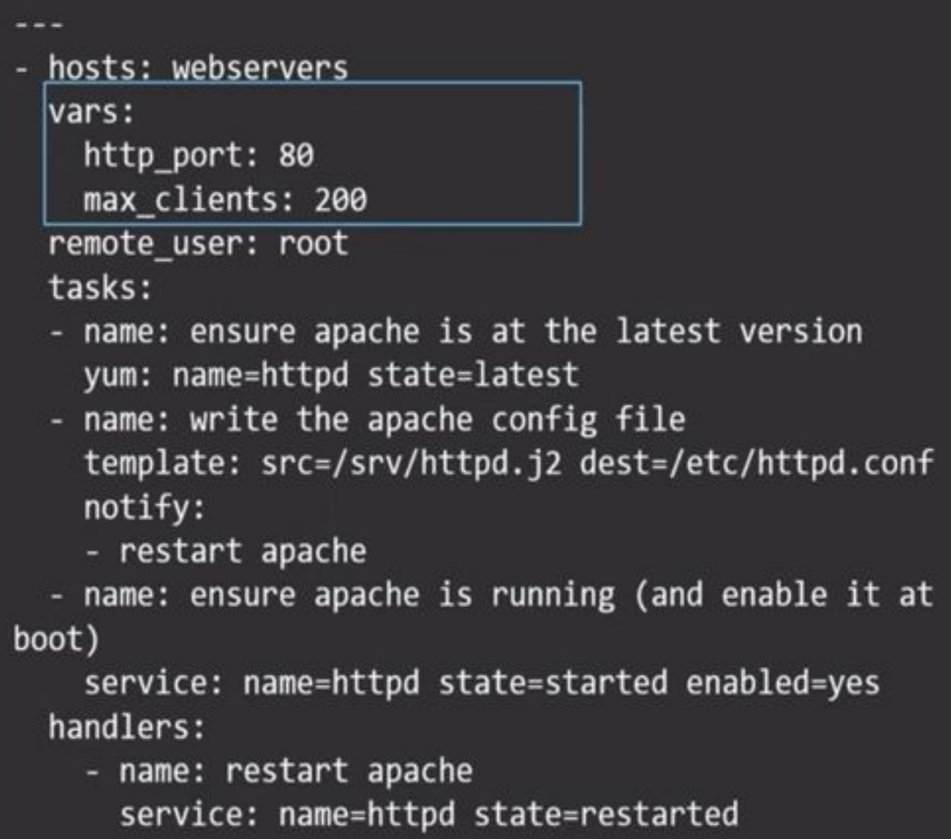


HOSTS

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at
boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Hosts are a list one or more groups or host patterns, separated by colons

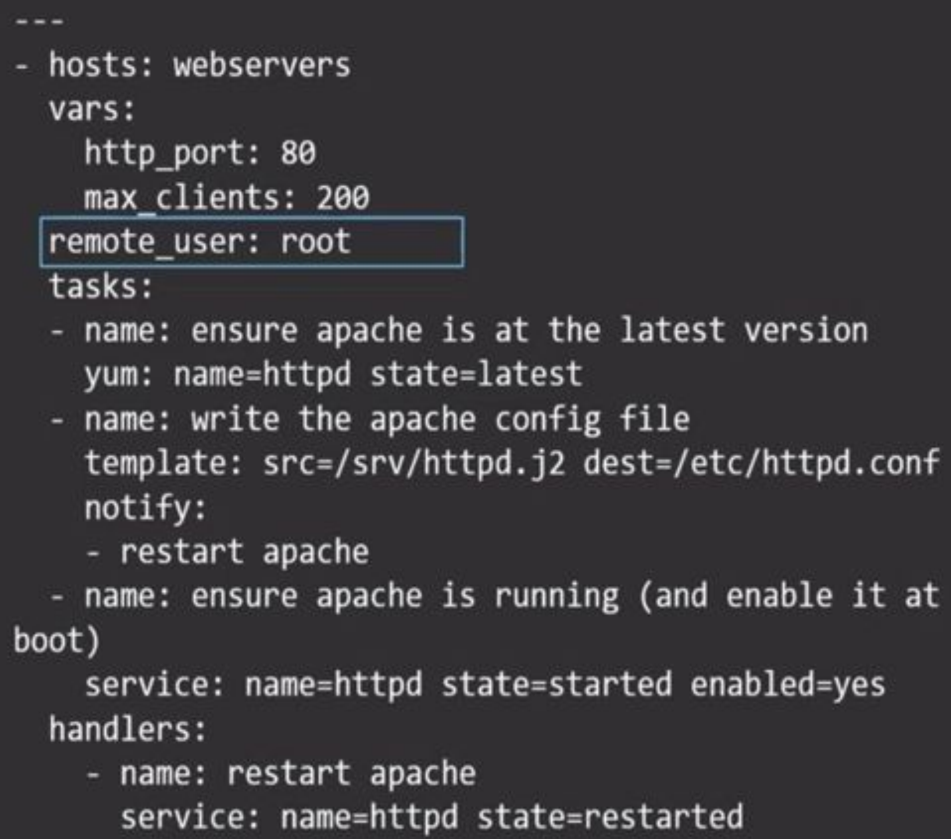
VARIABLES



```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at
boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Used to enable more flexibility in playbooks and roles. They are also used to loop through a set of given values, access various information and replace certain strings in templates.

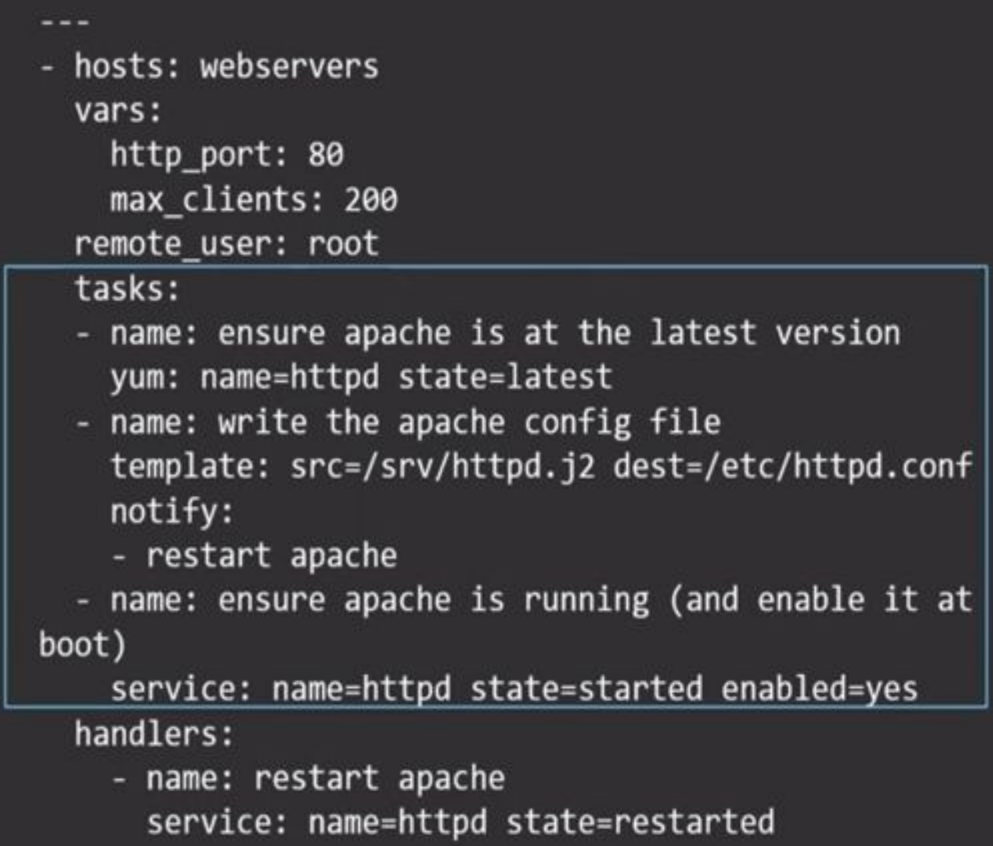
USERS



```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at
boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

User as the name suggests is the name of the user account. Here it is the root user.

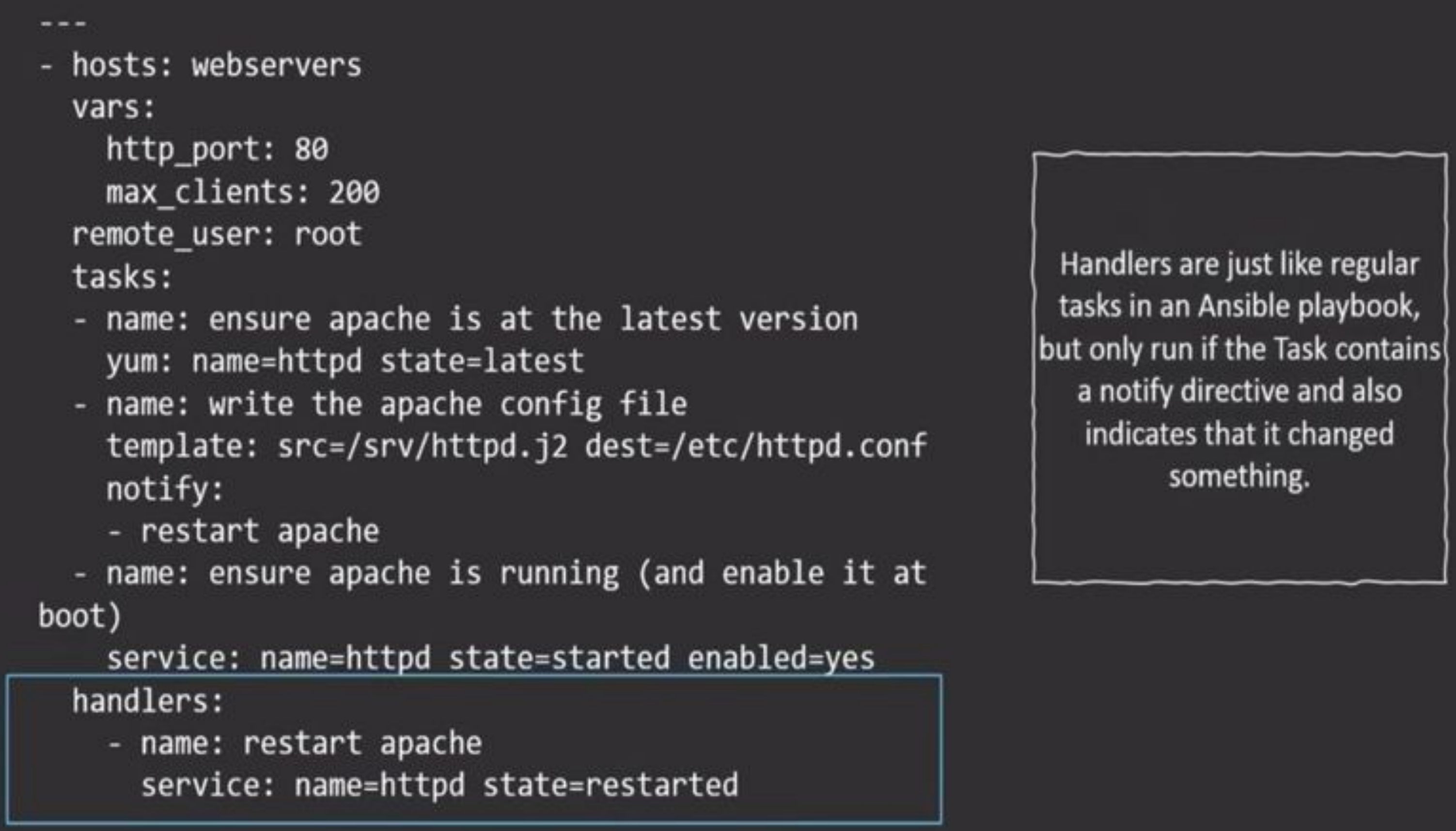
TASKS



```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at
boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Tasks allow you to break up bits of configuration policy into smaller files. Task includes pull from other files.

HANDLERS

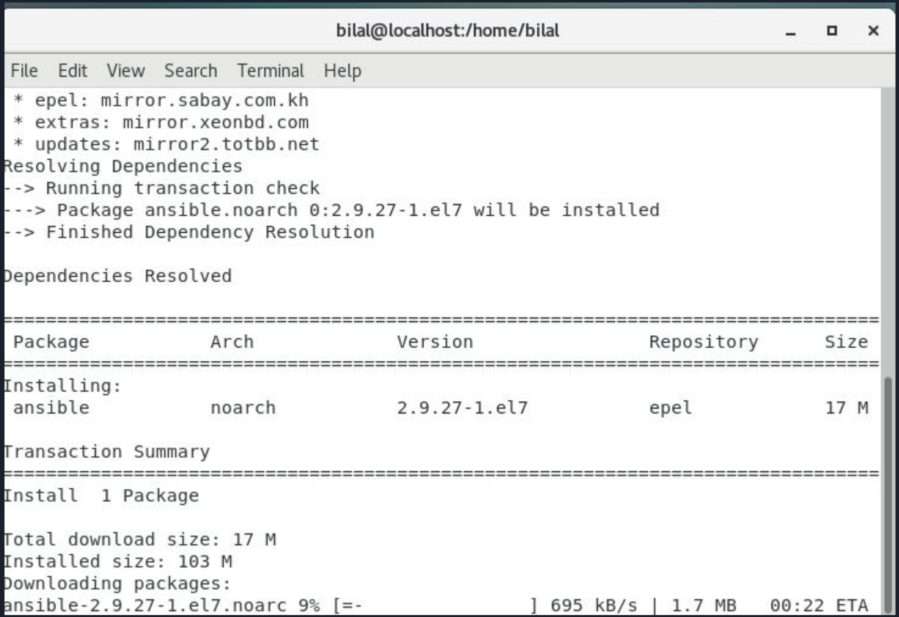


```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at
boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Handlers are just like regular tasks in an Ansible playbook, but only run if the Task contains a notify directive and also indicates that it changed something.

Installation

→ sudo yum install -y ansible

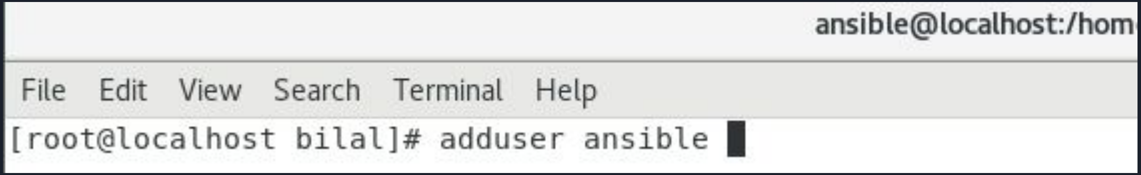


bilal@localhost:/home/bilal
File Edit View Search Terminal Help

```
 * epel: mirror.sabay.com.kh
 * extras: mirror.xeonbd.com
 * updates: mirror2.totbb.net
Resolving Dependencies
--> Running transaction check
---> Package ansible.noarch 0:2.9.27-1.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package            Arch               Version                 Repository  Size
================================================================================
Installing:
 ansible            noarch             2.9.27-1.el7            epel       17 M

Transaction Summary
================================================================================
Install  1 Package

Total download size: 17 M
Installed size: 103 M
Downloading packages:
ansible-2.9.27-1.el7.noarc 9% [=-               ] 695 kB/s | 1.7 MB   00:22 ETA
```

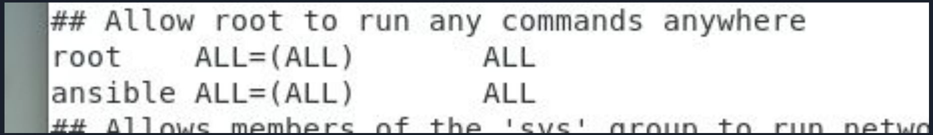
Lab-Setup

→ Adduser ansible on both Machines [**Ansible and Node**]



```
[root@localhost bilal]# adduser ansible
```

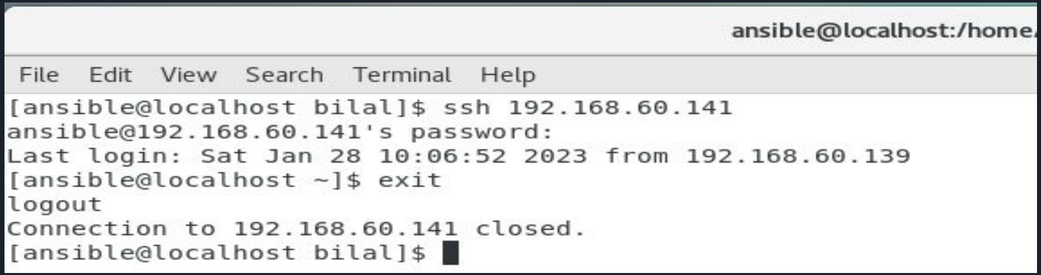
→ Assign Sudo rights via “ **visudo** “ on both Machines [**Ansible and Node**]



```
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
ansible ALL=(ALL)       ALL
## Allows members of the 'sys' group to run netwo
```

Lab-Setup

→ ssh login from ansible to node

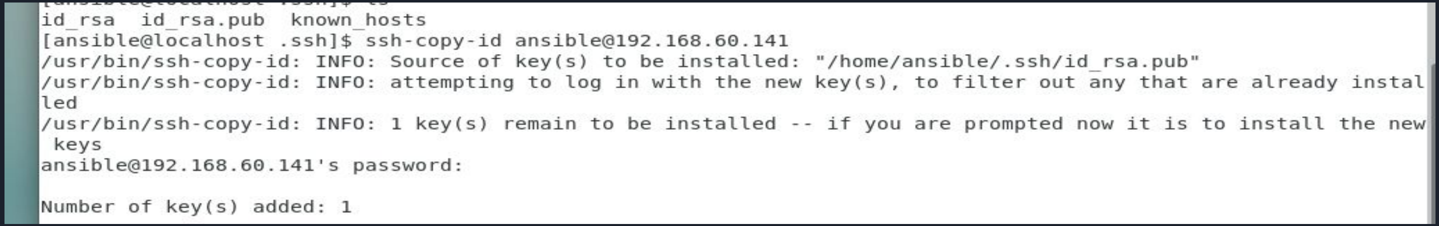


```
[ansible@localhost bilal]$ ssh 192.168.60.141
ansible@192.168.60.141's password:
Last login: Sat Jan 28 10:06:52 2023 from 192.168.60.139
[ansible@localhost ~]$ exit
logout
Connection to 192.168.60.141 closed.
[ansible@localhost bilal]$ 
```

→ Create Password-less ssh logins from ansible to node

→ generate public key via “ ssh-keygen

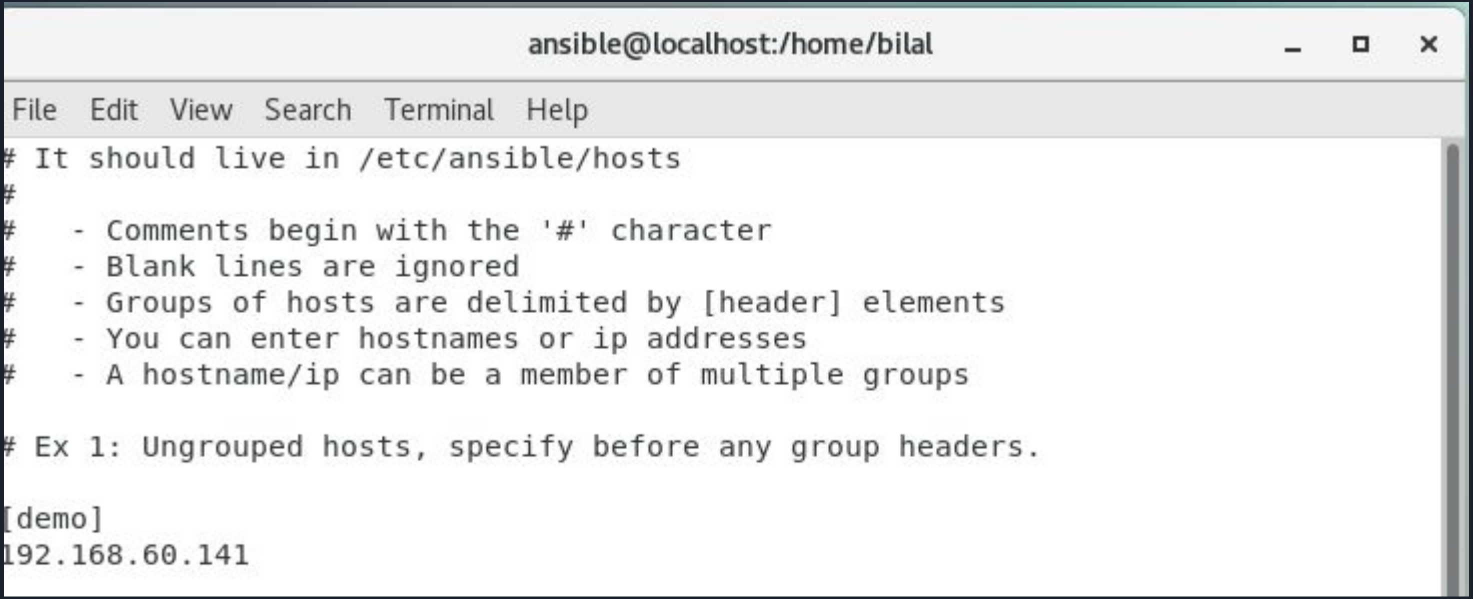
→ copy public key to the nore



```
[ansible@localhost .ssh]$ ls
id_rsa  id_rsa.pub  known_hosts
[ansible@localhost .ssh]$ ssh-copy-id ansible@192.168.60.141
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/ansible/.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 instal
led
/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
ansible@192.168.60.141's password:

Number of key(s) added: 1
```

Host - Configuration in ansible

→ Host can be found : “ **vi /etc/ansible/hosts**”

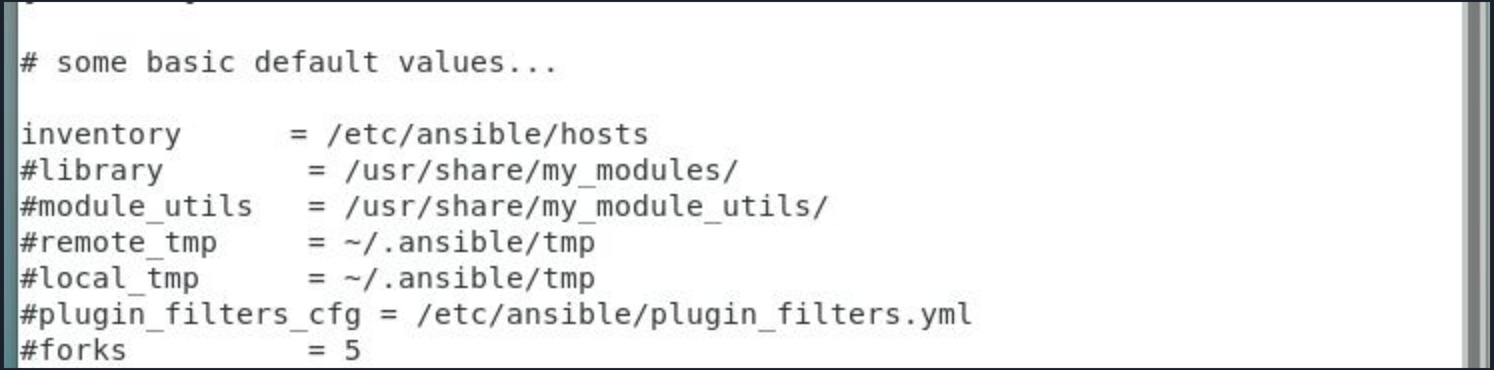


ansible@localhost:/home/bilal
File Edit View Search Terminal Help

```
# It should live in /etc/ansible/hosts
#
#   - Comments begin with the '#' character
#   - Blank lines are ignored
#   - Groups of hosts are delimited by [header] elements
#   - You can enter hostnames or ip addresses
#   - A hostname/ip can be a member of multiple groups

# Ex 1: Ungrouped hosts, specify before any group headers.

[demo]
192.168.60.141
```

Host - Configuration in ansible

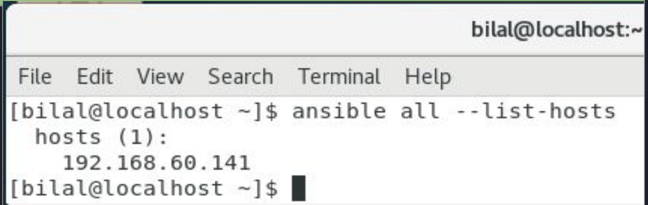
→ inventory configuration file can be found “ **sudo vi /etc/ansible/ansible.cfg** “

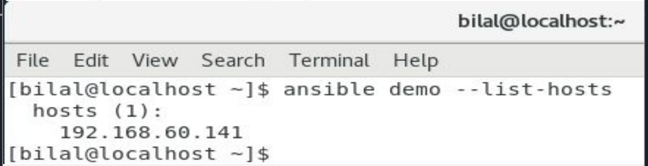


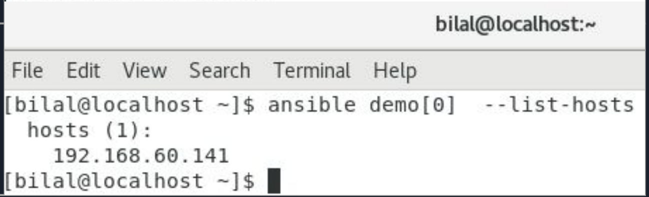
```
# some basic default values...

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
```

Host Patterns

Patterns	Description	POC

ansible all
--list-hosts
	" all " refer to all machines in an inventory	 [bilal@localhost ~]$ ansible all --list-hosts
		  hosts (1):
		    192.168.60.141
		[bilal@localhost ~]$

ansible
<group_name>
--list-hosts
	Specific group list in the inventory	 [bilal@localhost ~]$ ansible demo --list-hosts
		  hosts (1):
		    192.168.60.141
		[bilal@localhost ~]$

ansible
<group_name>
[0] --list-hosts
	Group specific machine list in the inventory	 [bilal@localhost ~]$ ansible demo[0] --list-hosts
		  hosts (1):
		    192.168.60.141
		[bilal@localhost ~]$

Ad-hoc Commands , Module and Playbooks

→ Since we have are done with all the prerequisites [Installation , ssh between ansible and nodes]

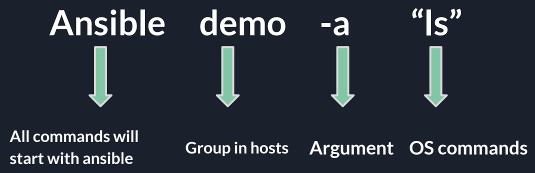
→ **There are three ways to push the configuration through ansible**

- Ad-hoc Commands
- Modules
- PlayBooks

Name	Description
Ad-hoc Commands	Ad hoc commands are commands which can be run individually to perform quick functions. These commands need not be performed later
Modules	Single commands which meant to executed on client side called as module.
PlayBooks	Playbooks are the files where Ansible code is written. Playbooks are written in YAML format. (**More than one module to be executed will be called Playbook**)

What are Ad-hoc Commands ?

- Ad-hoc commands are commands which can be run individually to perform quick functions
- These ad-hoc commands are not used for configuration management and deployments because these commands are for one time usage.
- The ansible ad-hoc commands uses the **/usr/bin/ansible** command line tool to automate a single task



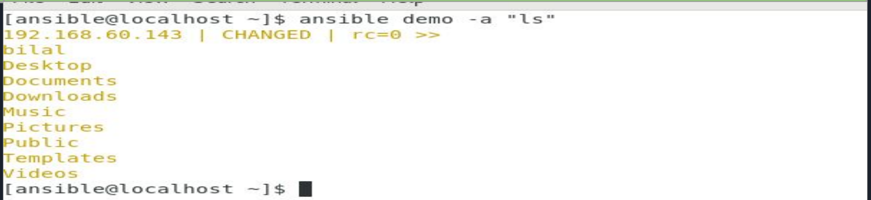
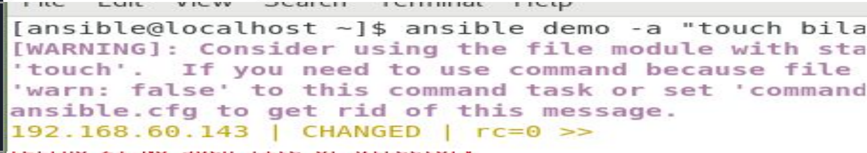
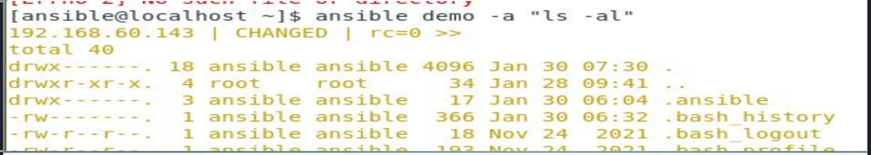
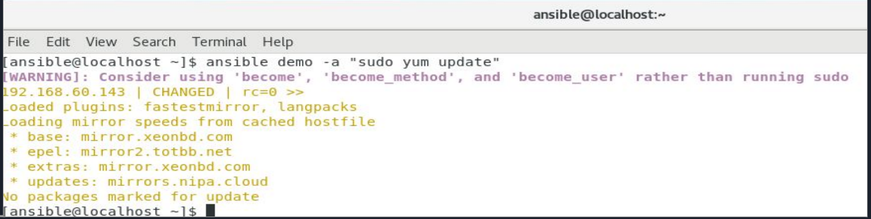
Ansible demo -a “ls”

All commands will start with ansible

Group in hosts

Argument

OS commands

Commands	POC
ansible demo -a “ls”	
ansible demo [0] -a”touch file “	
ansible demo -a “ls -al”	
ansible demo -a “sudo yum update “	

```
[ansible@localhost ~]$ ansible demo -a "ls"
192.168.60.143 | CHANGED | rc=0 >>
bilal
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
[ansible@localhost ~]$ █
```

```
[ansible@localhost ~]$ ansible demo -a "touch bila
[WARNING]: Consider using the file module with sta
'touch'.  If you need to use command because file
'warn: false' to this command task or set 'command
ansible.cfg to get rid of this message.
192.168.60.143 | CHANGED | rc=0 >>
```

```
[ansible@localhost ~]$ ansible demo -a "ls -al"
192.168.60.143 | CHANGED | rc=0 >>
total 40
drwx------. 18 ansible ansible 4096 Jan 30 07:30 .
drwxr-xr-x.  4 root    root      34 Jan 28 09:41 ..
drwx------.  3 ansible ansible   17 Jan 30 06:04 .ansible
-rw-------.  1 ansible ansible  366 Jan 30 06:32 .bash_history
-rw-r--r--.  1 ansible ansible   18 Nov 24  2021 .bash_logout
```

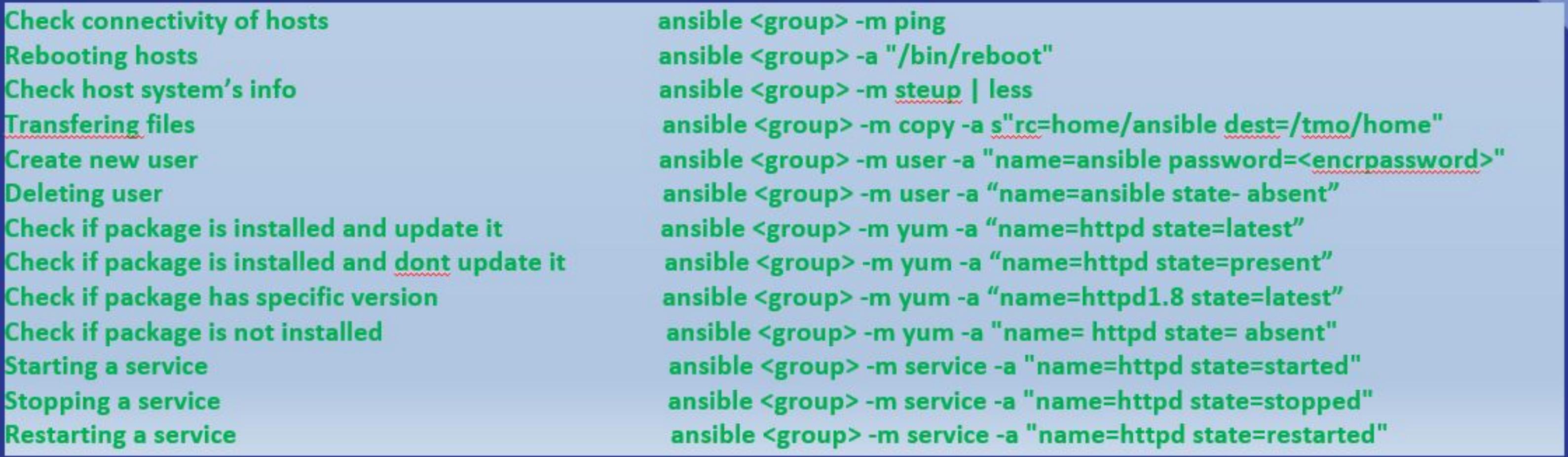
ansible@localhost:~

```
[ansible@localhost ~]$ ansible demo -a "sudo yum update"
[WARNING]: Consider using 'become', 'become_method', and 'become_user' rather than running sudo
192.168.60.143 | CHANGED | rc=0 >>
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirror.xeonbd.com
 * epel: mirror2.totbb.net
 * extras: mirror.xeonbd.com
 * updates: mirrors.nipa.cloud
No packages marked for update
[ansible@localhost ~]$ █
```

Home Task - 1

→ Setup Home Lab

→ Practice ad-hoc commands



Check connectivity of hosts	ansible <group> -m ping
Rebooting hosts	ansible <group> -a "/bin/reboot"
Check host system’s info	ansible <group> -m steup | less
Transfering files	ansible <group> -m copy -a s"rc=home/ansible dest=/tmo/home"
Create new user	ansible <group> -m user -a "name=ansible password=<encrpassword>"
Deleting user	ansible <group> -m user -a “name=ansible state- absent”
Check if package is installed and update it	ansible <group> -m yum -a “name=httpd state=latest”
Check if package is installed and dont update it	ansible <group> -m yum -a “name=httpd state=present”
Check if package has specific version	ansible <group> -m yum -a “name=httpd1.8 state=latest”
Check if package is not installed	ansible <group> -m yum -a "name= httpd state= absent"
Starting a service	ansible <group> -m service -a "name=httpd state=started"
Stopping a service	ansible <group> -m service -a "name=httpd state=stopped"
Restarting a service	ansible <group> -m service -a "name=httpd state=restarted"

Ansible Module

→ Ansible ships with a number of modules called “ **Module library** “ that can be executed directly on remote host or through playbook.

→ Library of module can be reside on any machine and these no server , daemons and database require , Generally can be found in libraries “ **/etc/ansible/hosts** “

→ Module can be write in YAML format

What is the difference between ad-hoc commands and module ?

Module	Ad-hoc commands
demo -b -m yum -a “ pkg=httpd state=present “	demo -a “yum install httpd”
Module always declared as “ -m “	Ad hoc does have any parameter

Ansible modules

- - m module_name
- Update = latest

→ ansible demo -b -m yum -a “pkg = httpd state=latest“

- Present = install

→ ansible demo -b -m yum -a “ pkg = httpd state=present “

- Absent = uninstall

-- ansible demo -b -m yum -a “ pkg = httpd state = absent “

- Httpd service can be restart

→ ansible demo -b -m service -a “ pkg = httpd state = started “

Ansible modules

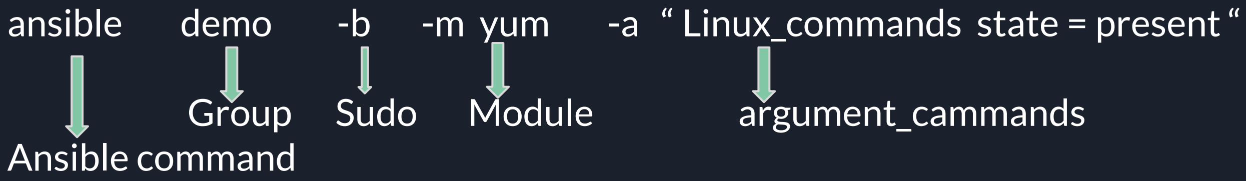
- User Module

→ ansible demo -b -m user -a “ name=bilal mazhar “

- Copy module

→ ansible demo -b -m copy -a “ src = file_name dest= /temp

Module command structure



ansible demo -b -m yum -a “ Linux_commands state = present “

Ansible command

Group

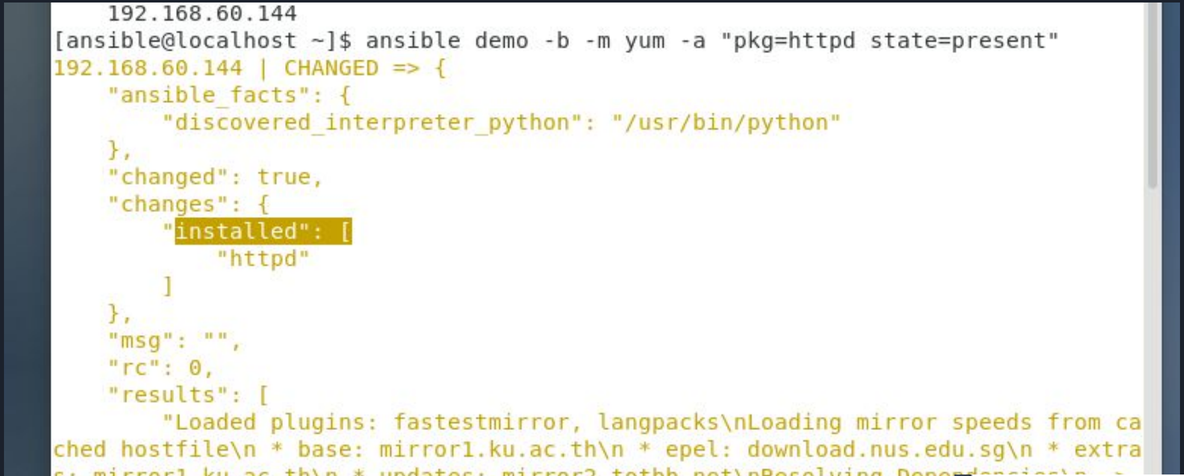
Sudo

Module

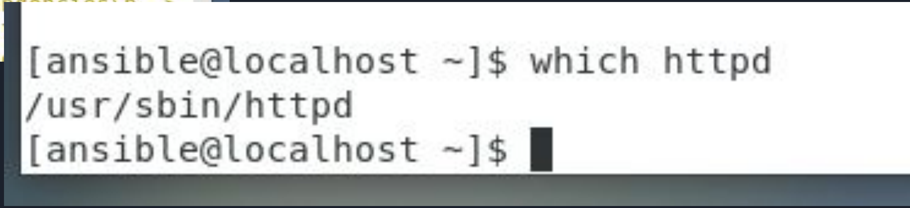
argument_cammands

Example : 1

→ Install httpd via module command



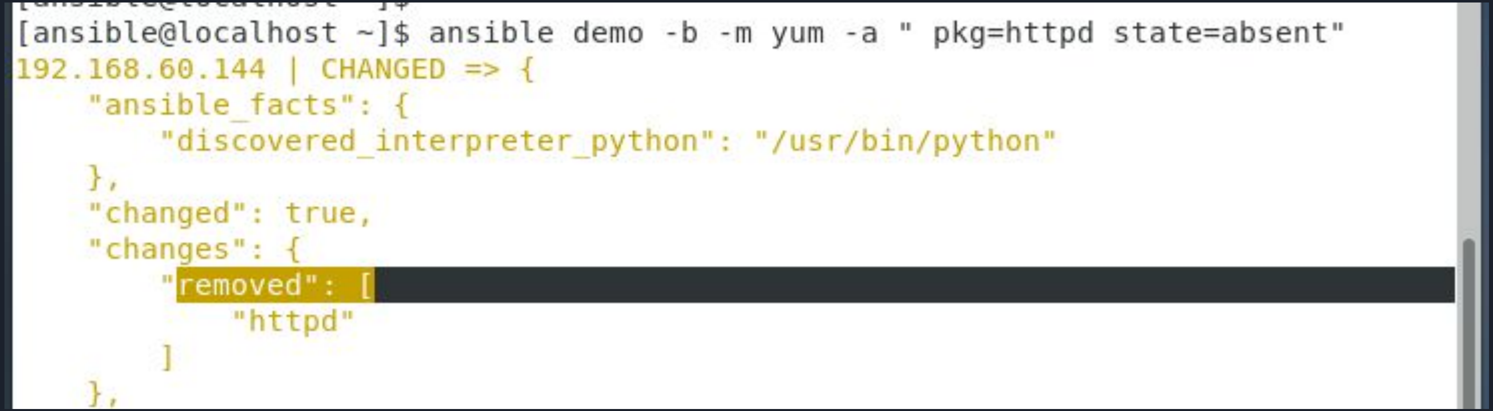
```
    192.168.60.144
[ansible@localhost ~]$ ansible demo -b -m yum -a "pkg=httpd state=present"
192.168.60.144 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "changes": {
        "installed": [
            "httpd"
        ]
    },
    "msg": "",
    "rc": 0,
    "results": [
        "Loaded plugins: fastestmirror, langpacks\nLoading mirror speeds from ca
ched hostfile\n * base: mirror1.ku.ac.th\n * epel: download.nus.edu.sg\n * extra
s: mirror1.ku.ac.th\n * updates: mirror2.totbb.net\nResolving Depe
Running transaction check\n---> Package httpd.x86_64 0:2.4.6-98.el
l be installed\n--> Processing Dependency: httpd-tools = 2.4.6-98
```



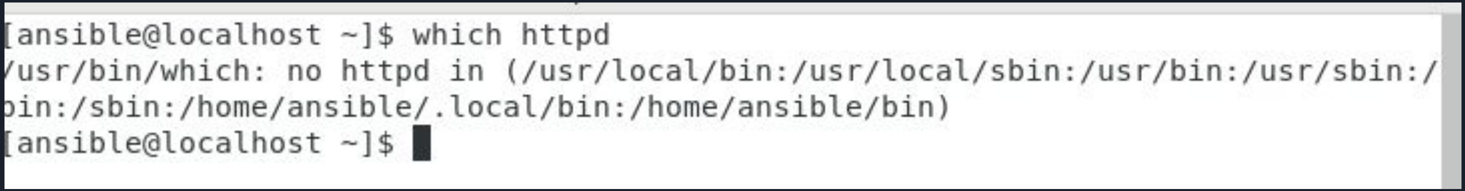
```
[ansible@localhost ~]$ which httpd
/usr/sbin/httpd
[ansible@localhost ~]$ 
```

Example : 2

→ Uninstall httpd using module command



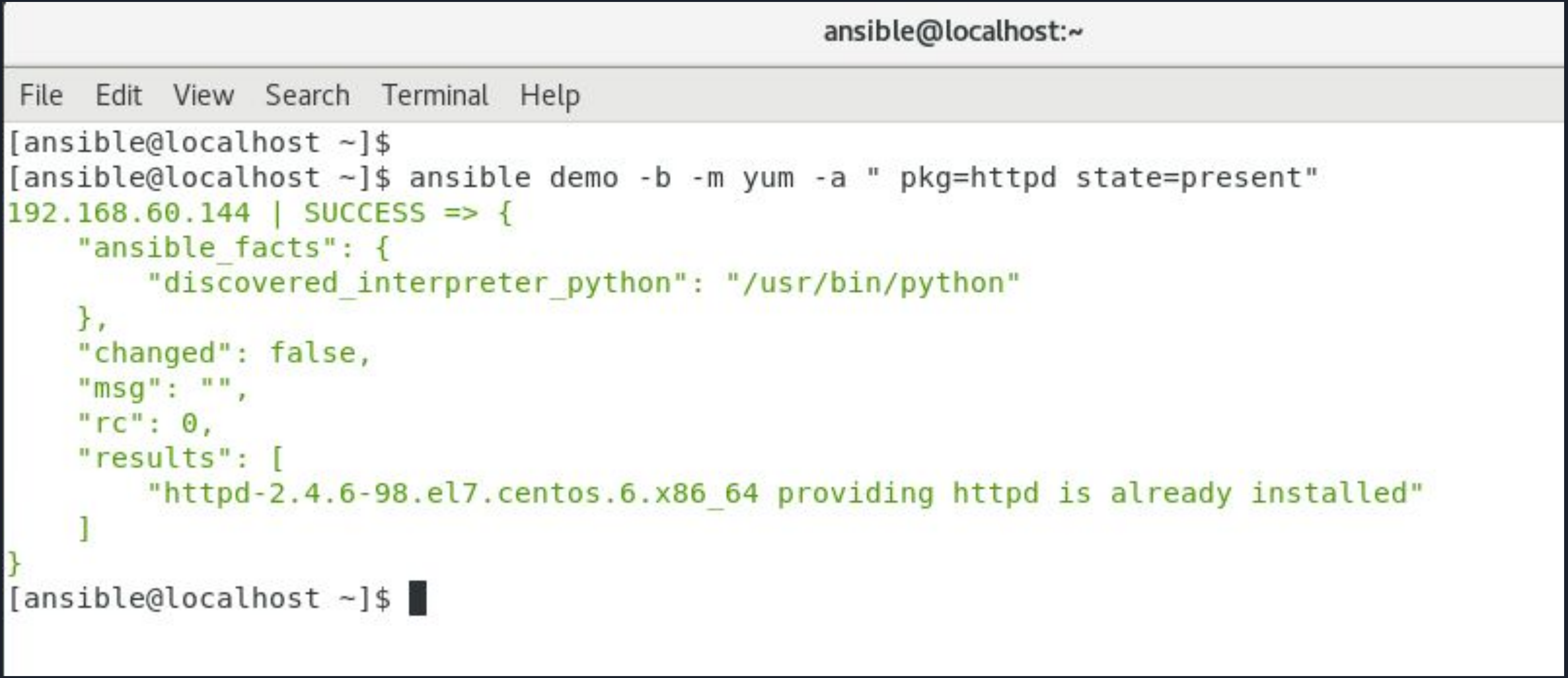
```
[ansible@localhost ~]$ ansible demo -b -m yum -a " pkg=httpd state=absent"
192.168.60.144 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "changes": {
        "removed": [
            "httpd"
        ]
    },
```



```
[ansible@localhost ~]$ which httpd
/usr/bin/which: no httpd in (/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/
bin:/sbin:/home/ansible/.local/bin:/home/ansible/bin)
[ansible@localhost ~]$
```

Example : 3

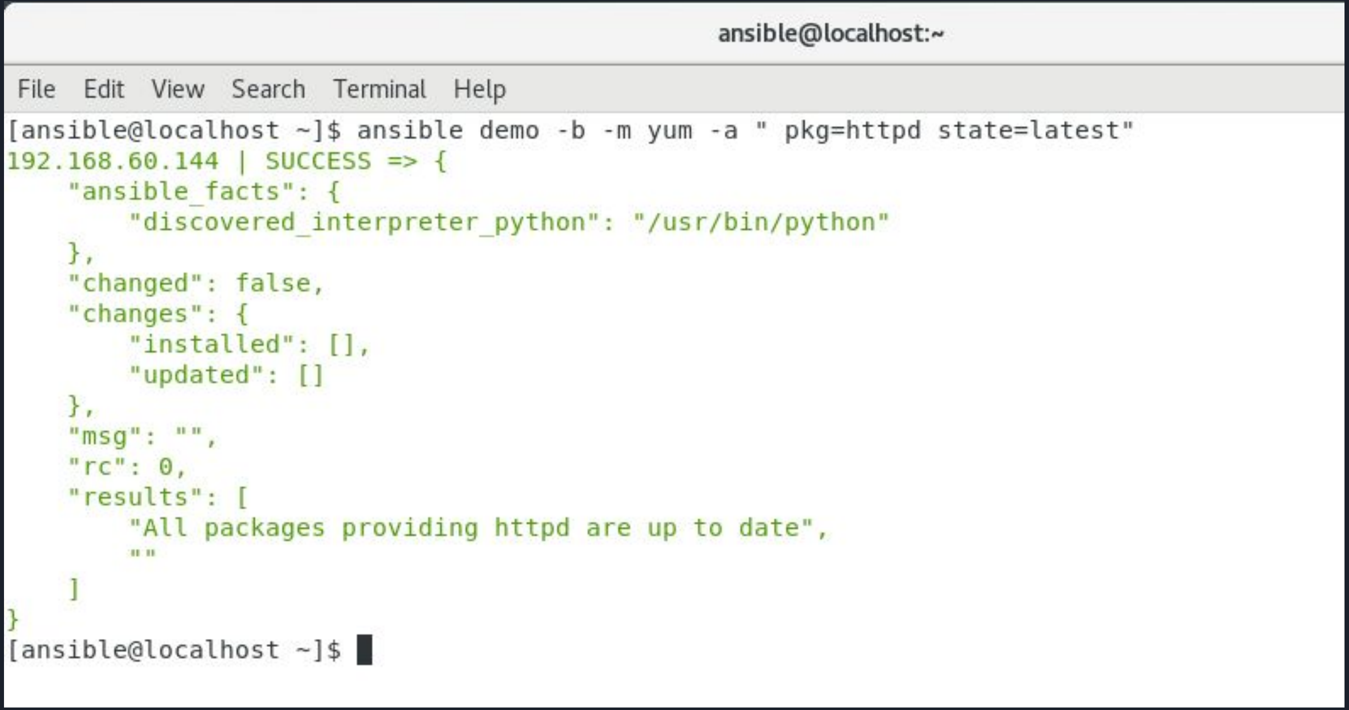
→ Module always verify before installations



```
[ansible@localhost ~]$
[ansible@localhost ~]$ ansible demo -b -m yum -a " pkg=httpd state=present"
192.168.60.144 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "msg": "",
    "rc": 0,
    "results": [
        "httpd-2.4.6-98.el7.centos.6.x86_64 providing httpd is already installed"
    ]
}
[ansible@localhost ~]$
```

Example : 4

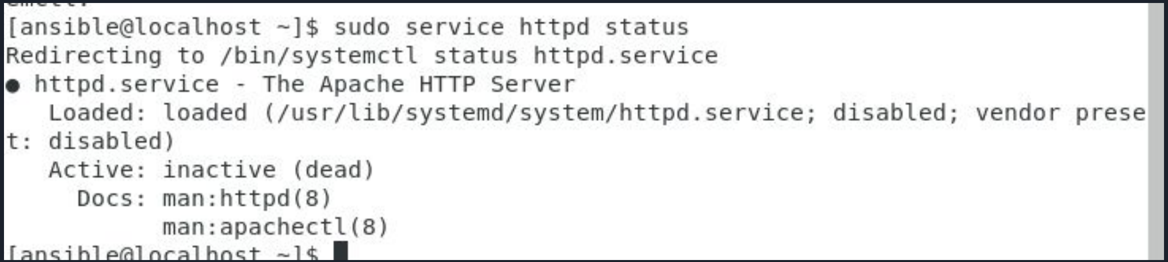
→ Update to the latest version



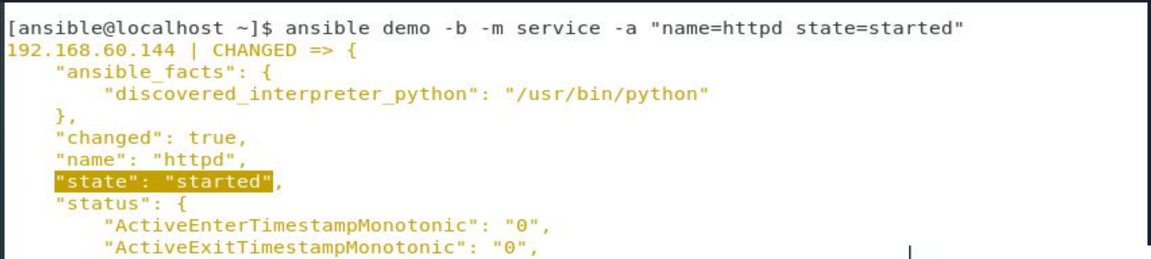
```
[ansible@localhost ~]$ ansible demo -b -m yum -a " pkg=httpd state=latest"
192.168.60.144 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false,
    "changes": {
        "installed": [],
        "updated": []
    },
    "msg": "",
    "rc": 0,
    "results": [
        "All packages providing httpd are up to date",
        ""
    ]
}
[ansible@localhost ~]$ 
```

Example : 5

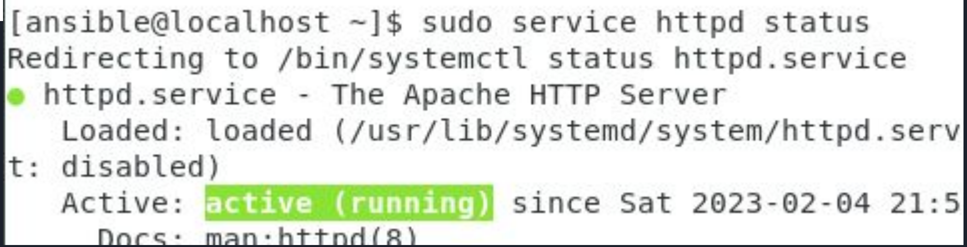
→ Enable the httpd service



```
[ansible@localhost ~]$ sudo service httpd status
Redirecting to /bin/systemctl status httpd.service
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor prese
t: disabled)
   Active: inactive (dead)
     Docs: man:httpd(8)
           man:apachectl(8)
[ansible@localhost ~]$
```



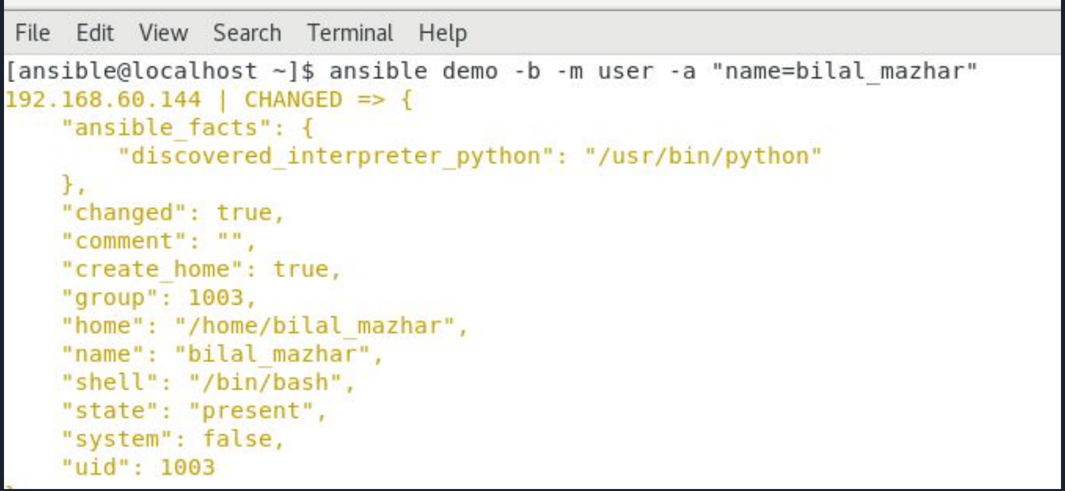
```
[ansible@localhost ~]$ ansible demo -b -m service -a "name=httpd state=started"
192.168.60.144 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "name": "httpd",
    "state": "started",
    "status": {
        "ActiveEnterTimestampMonotonic": "0",
        "ActiveExitTimestampMonotonic": "0",
        "ActiveState": "inactive",
```



```
[ansible@localhost ~]$ sudo service httpd status
Redirecting to /bin/systemctl status httpd.service
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.serv
t: disabled)
   Active: active (running) since Sat 2023-02-04 21:5
     Docs: man:httpd(8)
```

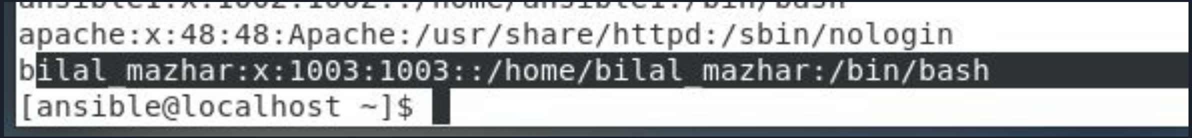
Example : 6

→ create a user using module command



```
[ansible@localhost ~]$ ansible demo -b -m user -a "name=bilal_mazhar"
192.168.60.144 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "comment": "",
    "create_home": true,
    "group": 1003,
    "home": "/home/bilal_mazhar",
    "name": "bilal_mazhar",
    "shell": "/bin/bash",
    "state": "present",
    "system": false,
    "uid": 1003
```

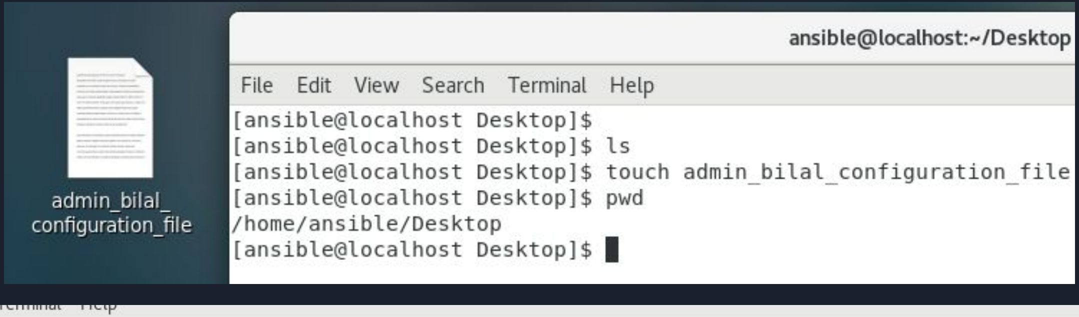
→ cat /etc/passwd



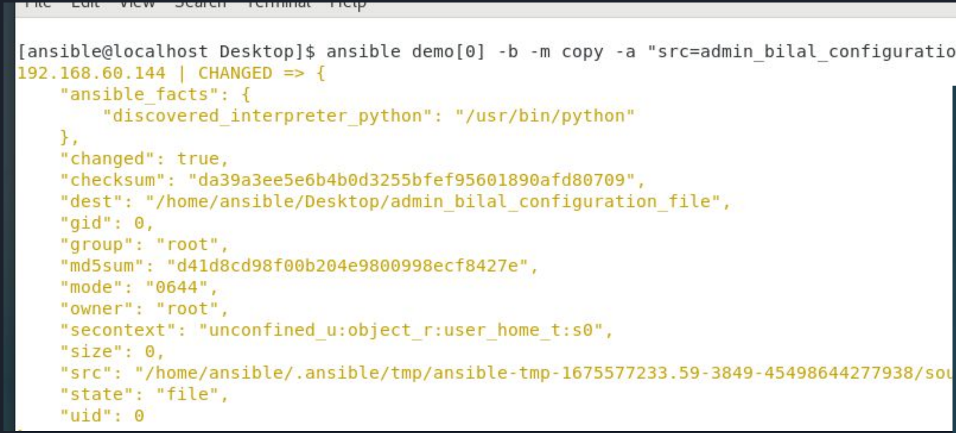
```
ansible1:x:1002:1002::/home/ansible1:/bin/bash
apache:x:48:48:Apache:/usr/share/httpd:/sbin/nologin
bilal_mazhar:x:1003:1003::/home/bilal_mazhar:/bin/bash
[ansible@localhost ~]$
```

Example : 7

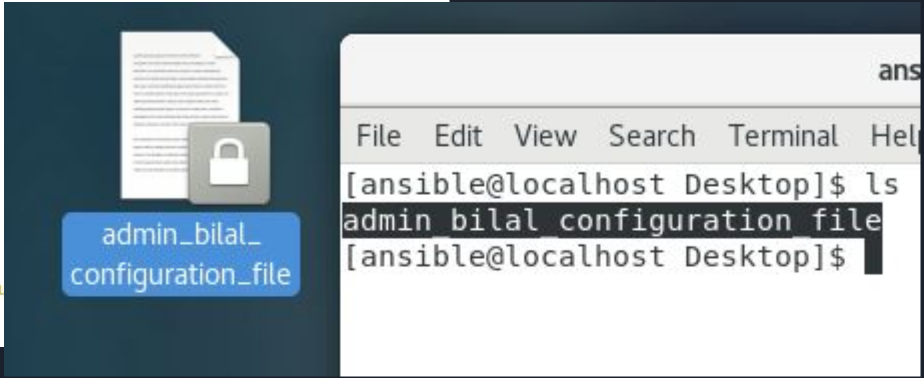
→ Copy file using module


admin_bilal_
configuration_file
ansible@localhost:~/Desktop
File Edit View Search Terminal Help

```
[ansible@localhost Desktop]$
[ansible@localhost Desktop]$ ls
[ansible@localhost Desktop]$ touch admin_bilal_configuration_file
[ansible@localhost Desktop]$ pwd
/home/ansible/Desktop
[ansible@localhost Desktop]$ 
```



```
[ansible@localhost Desktop]$ ansible demo[0] -b -m copy -a "src=admin_bilal_configuration_file dest=/home/ansible/Desktop"
192.168.60.144 | CHANGED => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": true,
    "checksum": "da39a3ee5e6b4b0d3255bfef95601890afd80709",
    "dest": "/home/ansible/Desktop/admin_bilal_configuration_file",
    "gid": 0,
    "group": "root",
    "md5sum": "d41d8cd98f00b204e9800998ecf8427e",
    "mode": "0644",
    "owner": "root",
    "secontext": "unconfined_u:object_r:user_home_t:s0",
    "size": 0,
    "src": "/home/ansible/.ansible/tmp/ansible-tmp-1675577233.59-3849-45498644277938/sou
    "state": "file",
    "uid": 0
```

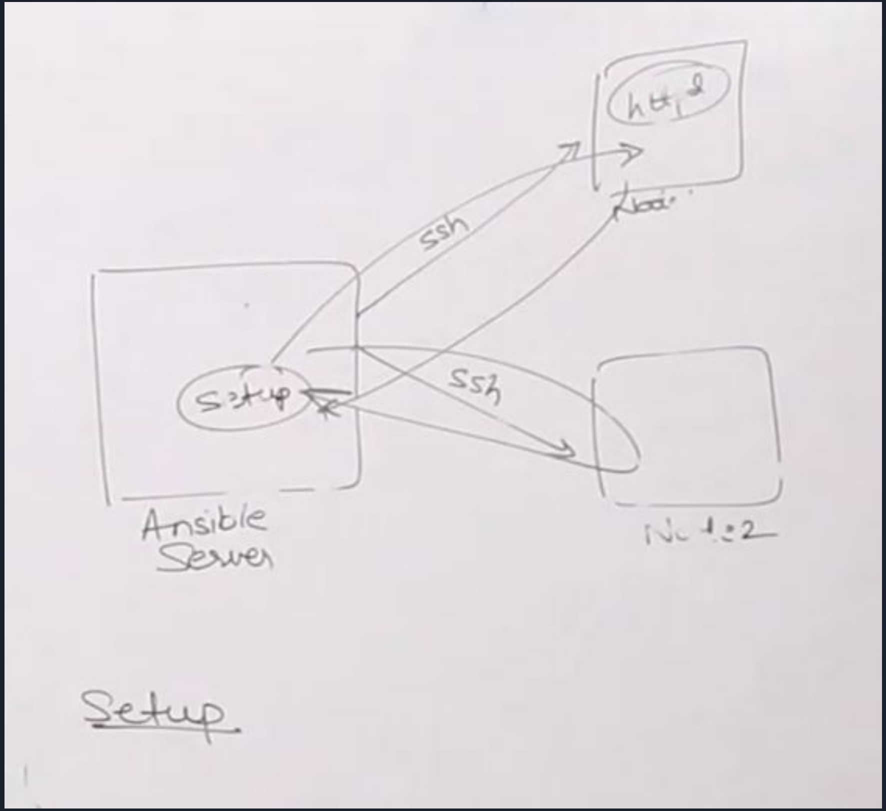


```
[ansible@localhost Desktop]$ ls
admin_bilal_configuration_file
[ansible@localhost Desktop]$ 
```

Ansible Module idempotency

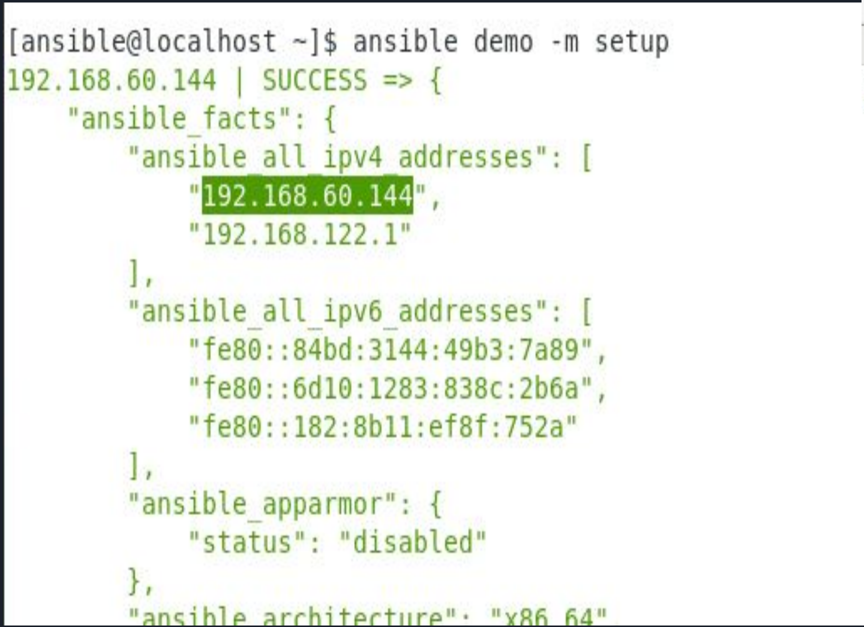
→ Setup module is responsible to check for idempotency

→ setup module always run before the module commands to check for the latest configuration of the remote node.



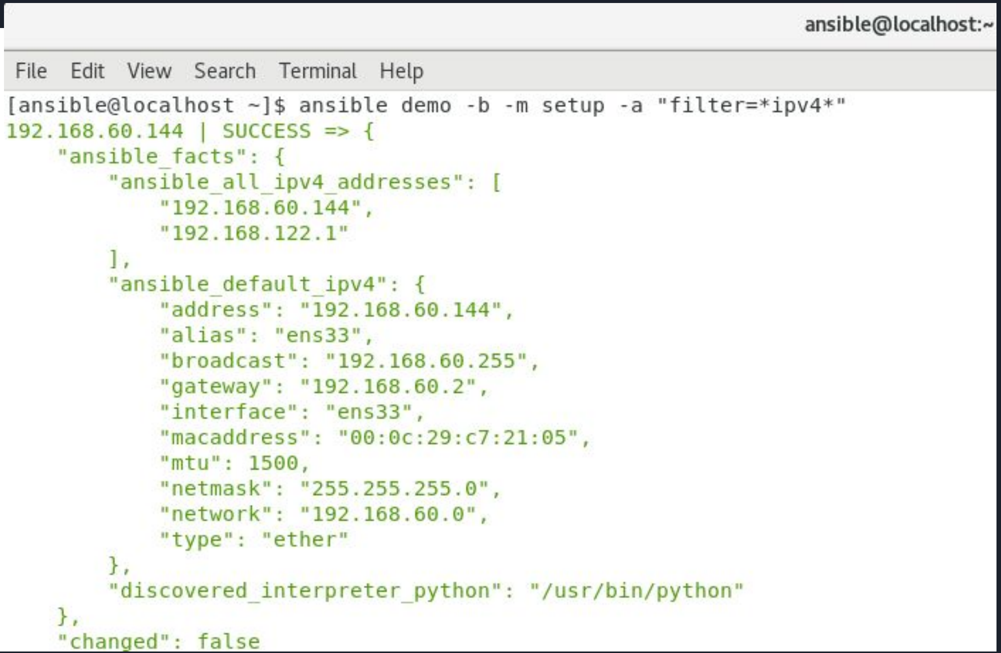
How we check the current configurations and ip address of remote hosts ?

configuration of host



```
[ansible@localhost ~]$ ansible demo -m setup
192.168.60.144 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.60.144",
            "192.168.122.1"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::84bd:3144:49b3:7a89",
            "fe80::6d10:1283:838c:2b6a",
            "fe80::182:8b11:ef8f:752a"
        ],
        "ansible_apparmor": {
            "status": "disabled"
        },
        "ansible_architecture": "x86_64"
```

filter only IP address

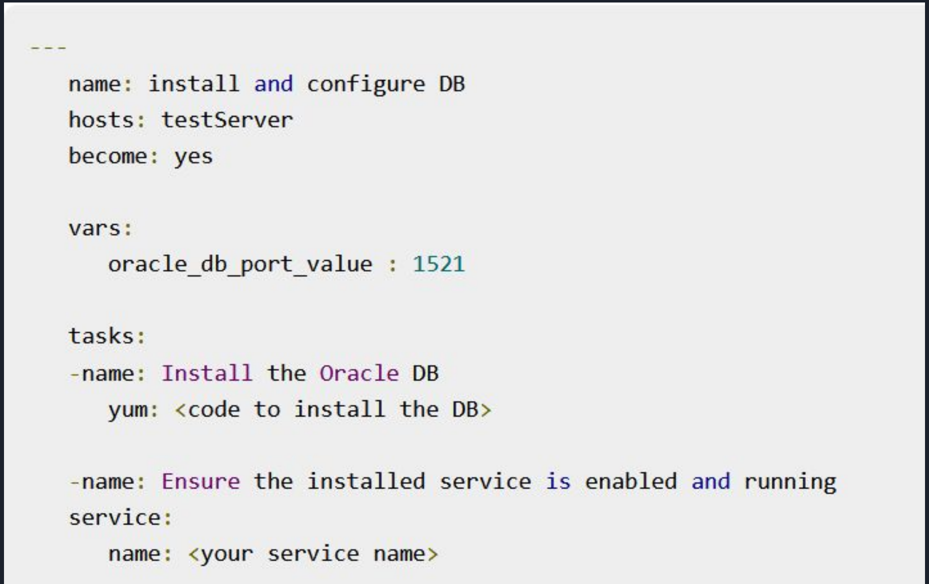


```
[ansible@localhost ~]$ ansible demo -b -m setup -a "filter=*ipv4*"
192.168.60.144 | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "192.168.60.144",
            "192.168.122.1"
        ],
        "ansible_default_ipv4": {
            "address": "192.168.60.144",
            "alias": "ens33",
            "broadcast": "192.168.60.255",
            "gateway": "192.168.60.2",
            "interface": "ens33",
            "macaddress": "00:0c:29:c7:21:05",
            "mtu": 1500,
            "netmask": "255.255.255.0",
            "network": "192.168.60.0",
            "type": "ether"
        },
        "discovered_interpreter_python": "/usr/bin/python"
    },
    "changed": false
```

Playbook

→ Playbooks are the files where Ansible code is written. Playbooks are written in **YAML format**. YAML stands for **Yet Another Markup Language**. Playbooks are one of the core features of Ansible and tell Ansible what to execute. They are like a to-do list for Ansible that contains a list of tasks. “

Each playbook is composed of one or more module.



```
---
   name: install and configure DB
   hosts: testServer
   become: yes

   vars:
      oracle_db_port_value : 1521

   tasks:
   -name: Install the Oracle DB
      yum: <code to install the DB>

   -name: Ensure the installed service is enabled and running
   service:
      name: <your service name>
```

YAML Tags

Tags	Description
Name	This tag specifies the name of the Ansible playbook. As in what this playbook will be doing. Any logical name can be given to the playbook.
hosts	This tag specifies the lists of hosts or host group against which we want to run the task. The hosts field/tag is mandatory. It tells Ansible on which hosts to run the listed tasks.
vars	Vars tag lets you define the variables which you can use in your playbook. Usage is similar to variables in any programming language.
tasks	All playbooks should contain tasks or a list of tasks to be executed. Tasks are a list of actions one needs to perform. A tasks field contains the name of the task

YAML Basics

→ For ansible nearly every YAML files start with the a “ **list** “ ,

→ Each item in the list is a list of “ **key-value pair** “ commonly called as “ **Dictionary** “ ,

→ All YAML files will start with “ **---** “ and end with “ … “ but not mandatory

```
--- # list of customer
        Name:  Bilal Mazhar
        Job:  Trainer
        Exp:  9 Years
```

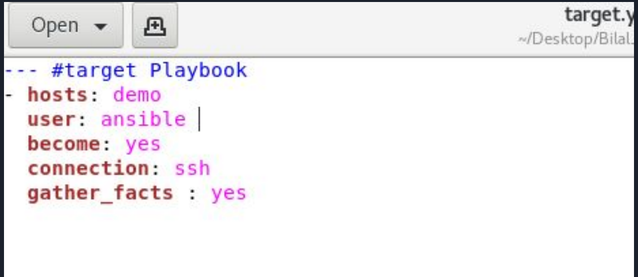
→ can be save as .yml extension,

→ each member of the list have to be using indentation “ -” ,

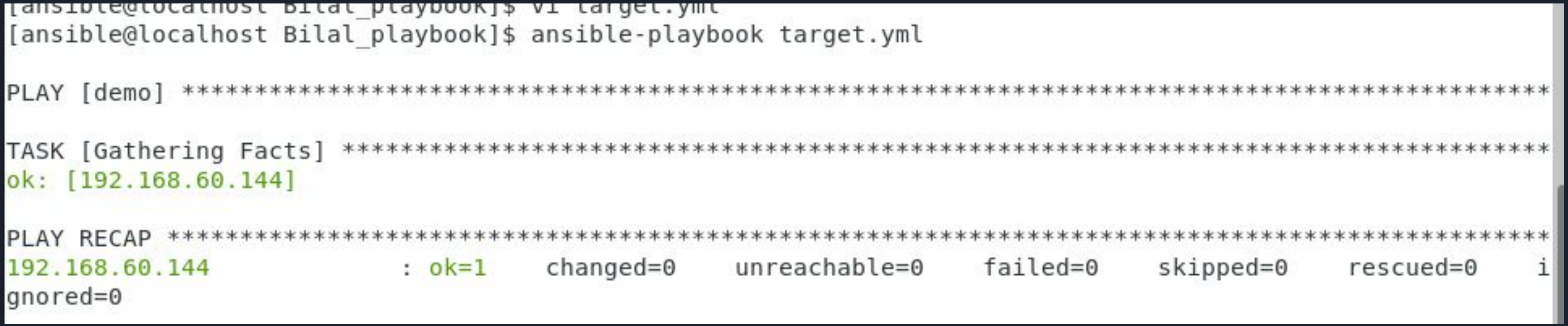
→ there should be space between : and space value

Playbook - Example : 1

→ **To create Playbook** : vi bilal_playbook-1.yml



```yaml
--- #target Playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  gather_facts : yes
```

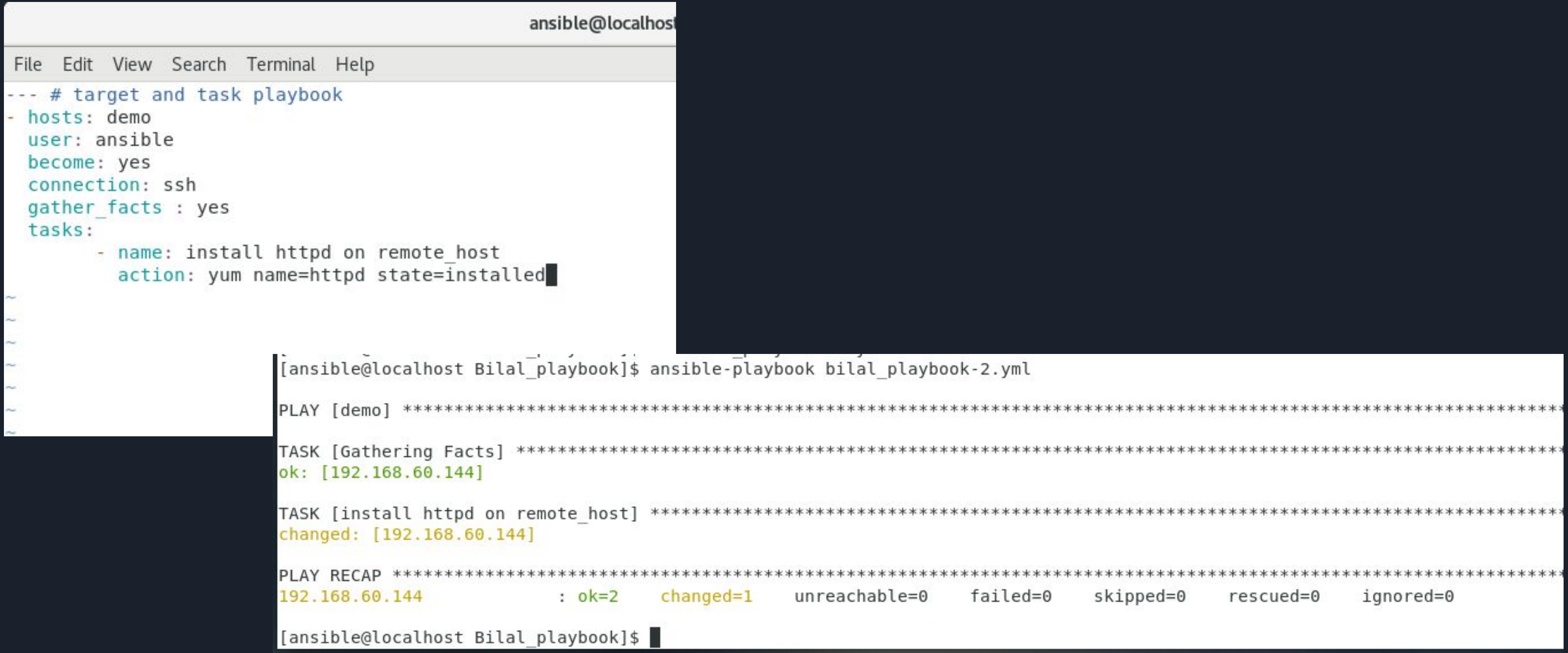


```
[ansible@localhost Bilal_playbook]$ vi target.yml
[ansible@localhost Bilal_playbook]$ ansible-playbook target.yml

PLAY [demo] *********************************************************************************

TASK [Gathering Facts] **********************************************************************
ok: [192.168.60.144]

PLAY RECAP **********************************************************************************
192.168.60.144             : ok=1    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    i
gnored=0
```

Playbook - Example : 2

→ **To create Playbook** : vi bilal_playbook-2.yml

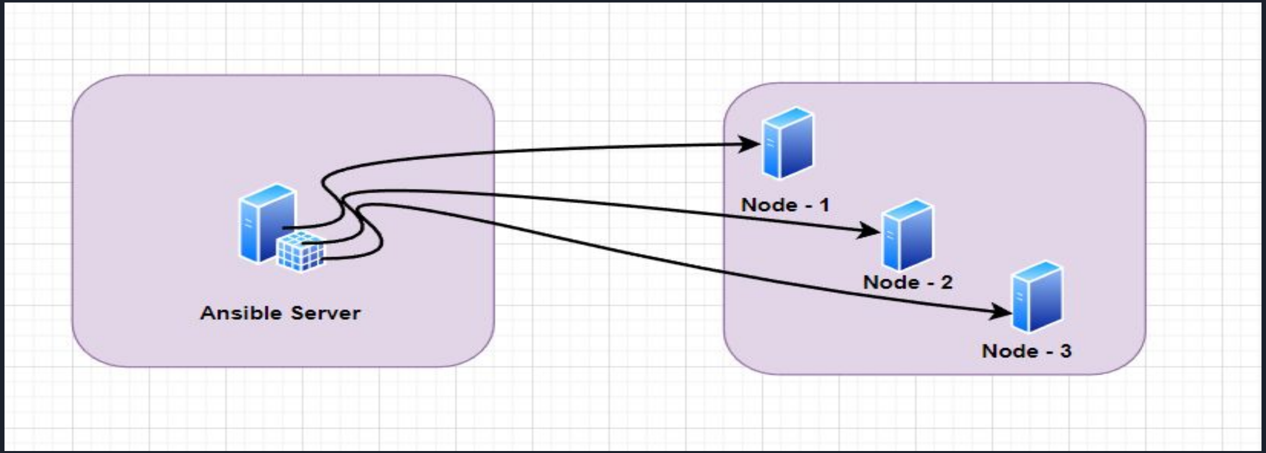
→ objective is to install the httpd



```
--- # target and task playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  gather_facts : yes
  tasks:
        - name: install httpd on remote_host
          action: yum name=httpd state=installed
~
~
~
~
~
~
~
```

```
[ansible@localhost Bilal_playbook]$ ansible-playbook bilal_playbook-2.yml

PLAY [demo] ****************************************************************************************************************

TASK [Gathering Facts] *****************************************************************************************************
ok: [192.168.60.144]

TASK [install httpd on remote_host] ****************************************************************************************
changed: [192.168.60.144]

PLAY RECAP *****************************************************************************************************************
192.168.60.144             : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ansible@localhost Bilal_playbook]$ 
```

Home Task - 2



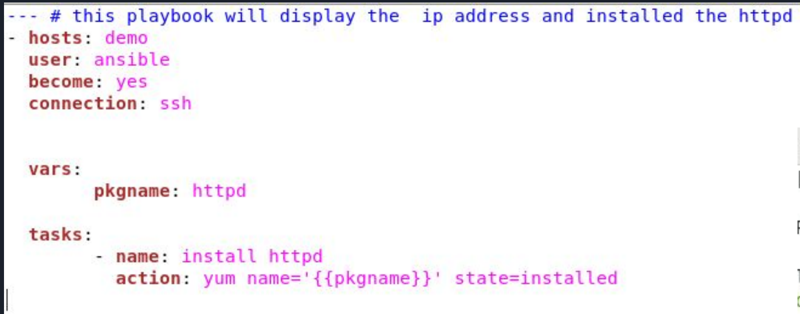
→ Task 1 : Install httpd using module and playbook in last node,
→ Task 2 : list down the current configuration of first node only,
→ Task 3 : Update && upgrade using module and playbook all nodes
→ task 4 : uninstall httpd using playbook all nodes

Variables

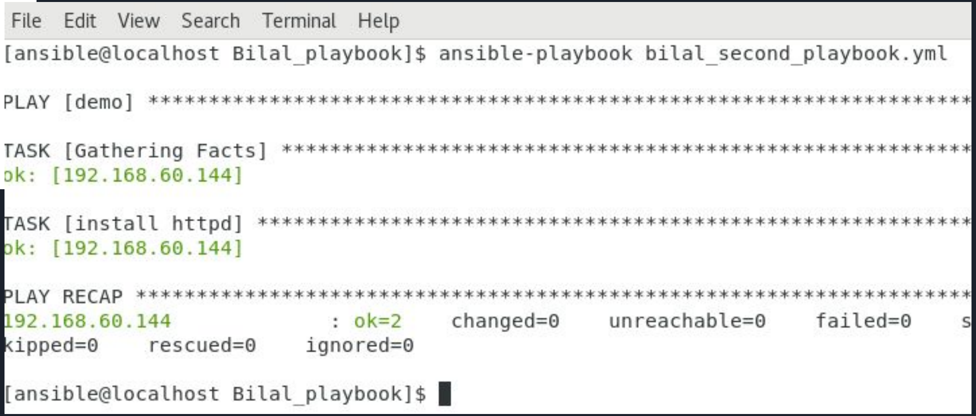
→ Ansible uses variables which are defined previously to enable more flexibility in playbook and roles.

→ They can be , access various information like the host name of the system and replace certain strings in template

→ Alway define var before tasks



```
--- # this playbook will display the  ip address and installed the httpd
- hosts: demo
  user: ansible
  become: yes
  connection: ssh


  vars:
        pkgname: httpd

  tasks:
        - name: install httpd
          action: yum name='{{pkgname}}' state=installed
```

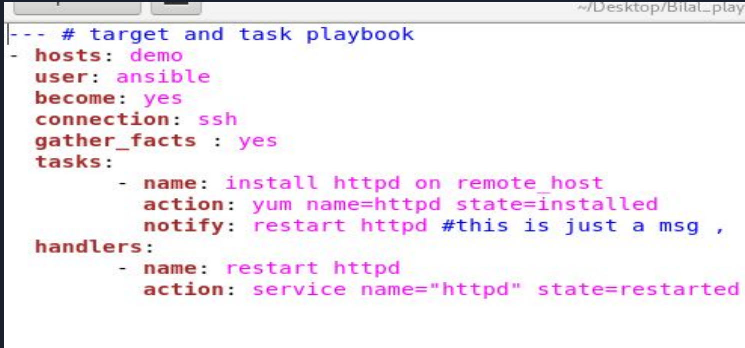


```
[ansible@localhost Bilal_playbook]$ ansible-playbook bilal_second_playbook.yml

PLAY [demo] **************************************************************

TASK [Gathering Facts] ***************************************************
ok: [192.168.60.144]

TASK [install httpd] *****************************************************
ok: [192.168.60.144]

PLAY RECAP ***************************************************************
192.168.60.144             : ok=2    changed=0    unreachable=0    failed=0    s
kipped=0    rescued=0    ignored=0

[ansible@localhost Bilal_playbook]$ █
```

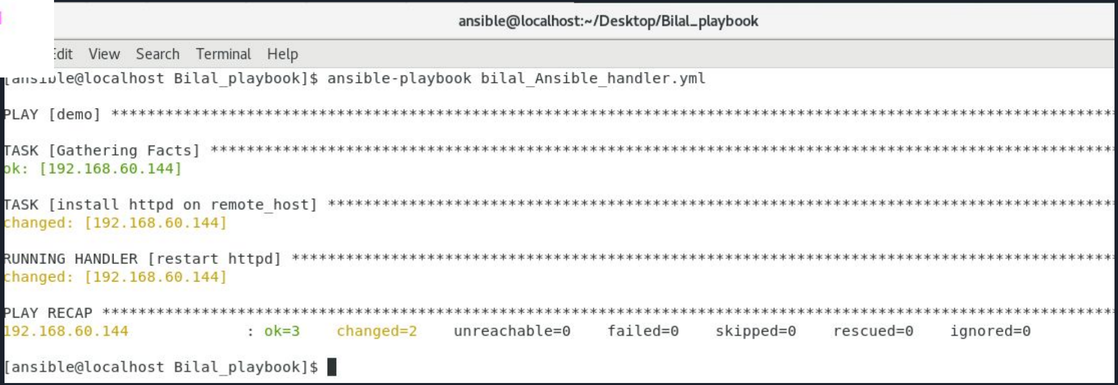
Handlers

→ A handler is exactly the same as a task, but i will run when called by another task,

→ Handlers are just like regular task in an ansible playbook , but only run if the task taken notify directive and also indicate some thing is **change**



```
--- # target and task playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  gather_facts : yes
  tasks:
        - name: install httpd on remote_host
          action: yum name=httpd state=installed
          notify: restart httpd #this is just a msg ,
  handlers:
        - name: restart httpd
          action: service name="httpd" state=restarted
```

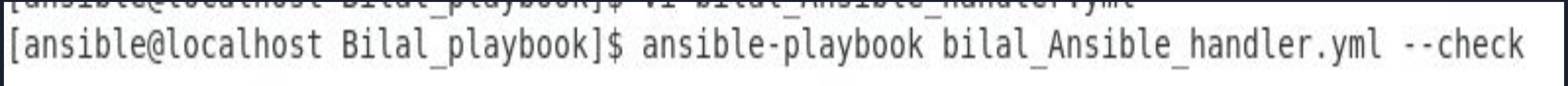


ansible@localhost:~/Desktop/Bilal_playbook
Edit View Search Terminal Help

```
[ansible@localhost Bilal_playbook]$ ansible-playbook bilal_Ansible_handler.yml

PLAY [demo] ************************************************************************************************

TASK [Gathering Facts] *************************************************************************************
ok: [192.168.60.144]

TASK [install httpd on remote_host] ************************************************************************
changed: [192.168.60.144]

RUNNING HANDLER [restart httpd] ****************************************************************************
changed: [192.168.60.144]

PLAY RECAP *************************************************************************************************
192.168.60.144             : ok=3    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ansible@localhost Bilal_playbook]$ 
```

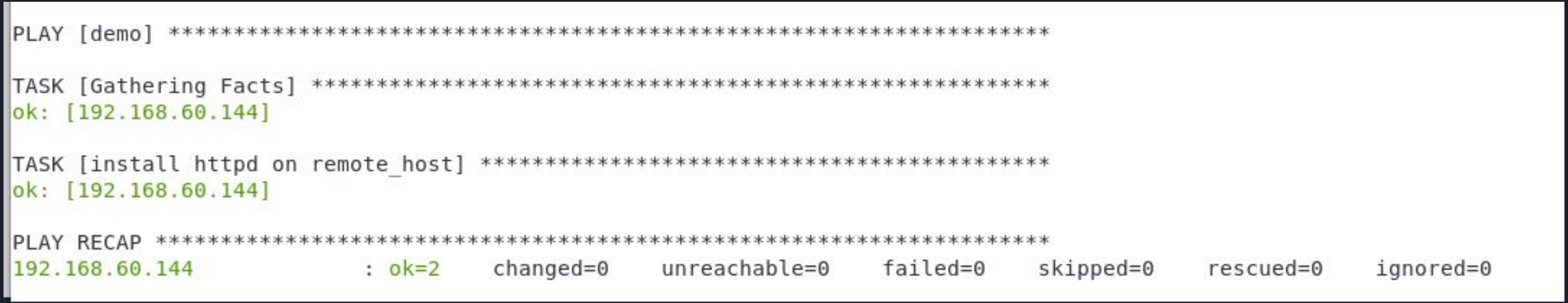
Dry Run

→ To check whether my playbook is working fine , Ansible has an features called “ Dry Run “



```
[ansible@localhost Bilal_playbook]$ vi bilal_Ansible_handler.yml
[ansible@localhost Bilal_playbook]$ ansible-playbook bilal_Ansible_handler.yml --check
```

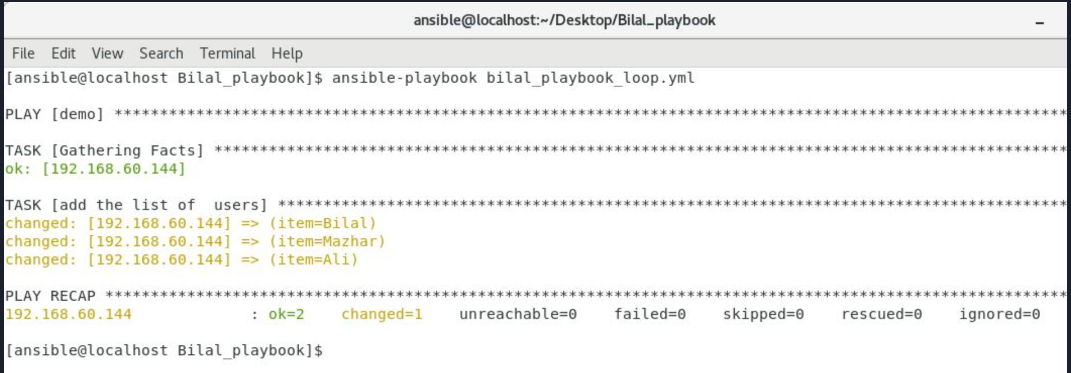
→ Output will be like this :



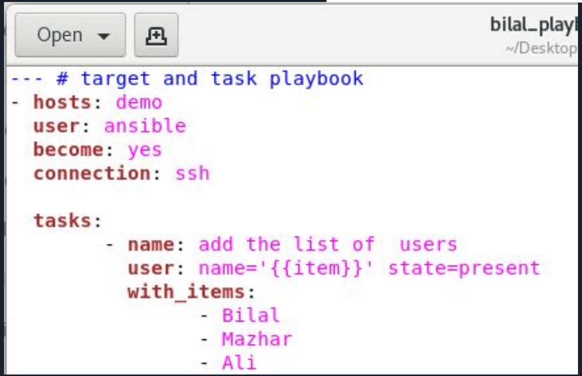
```
PLAY [demo] *********************************************************************

TASK [Gathering Facts] **********************************************************
ok: [192.168.60.144]

TASK [install httpd on remote_host] *********************************************
ok: [192.168.60.144]

PLAY RECAP **********************************************************************
192.168.60.144             : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Loop

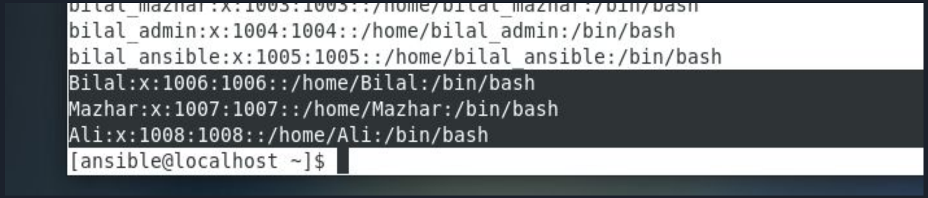
→ Sometimes you want to repeat the tasks on multiple times , so ansible has this loop feature in ansible playbook


ansible@localhost:~/Desktop/Bilal_playbook
File Edit View Search Terminal Help

```
[ansible@localhost Bilal_playbook]$ ansible-playbook bilal_playbook_loop.yml

PLAY [demo] *************************************************************************************

TASK [Gathering Facts] **************************************************************************
ok: [192.168.60.144]

TASK [add the list of  users] *******************************************************************
changed: [192.168.60.144] => (item=Bilal)
changed: [192.168.60.144] => (item=Mazhar)
changed: [192.168.60.144] => (item=Ali)

PLAY RECAP **************************************************************************************
192.168.60.144             : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ansible@localhost Bilal_playbook]$
```



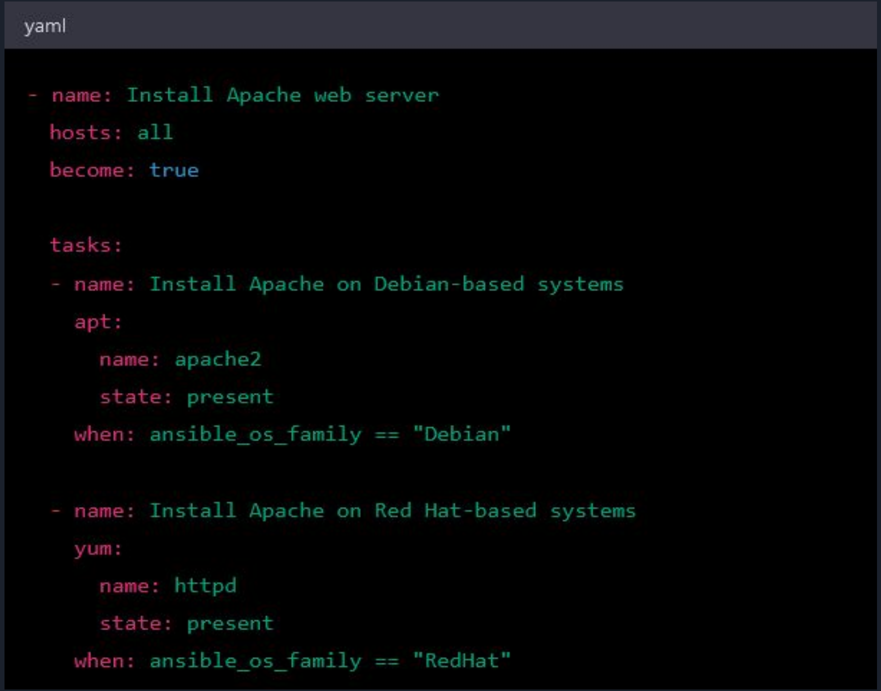
```
--- # target and task playbook
- hosts: demo
  user: ansible
  become: yes
  connection: ssh

  tasks:
        - name: add the list of  users
          user: name='{{item}}' state=present
          with_items:
                  - Bilal
                  - Mazhar
                  - Ali
```



```
bilal_mazhar:x:1003:1003::/home/bilal_mazhar:/bin/bash
bilal_admin:x:1004:1004::/home/bilal_admin:/bin/bash
bilal_ansible:x:1005:1005::/home/bilal_ansible:/bin/bash
Bilal:x:1006:1006::/home/Bilal:/bin/bash
Mazhar:x:1007:1007::/home/Mazhar:/bin/bash
Ali:x:1008:1008::/home/Ali:/bin/bash
[ansible@localhost ~]$
```

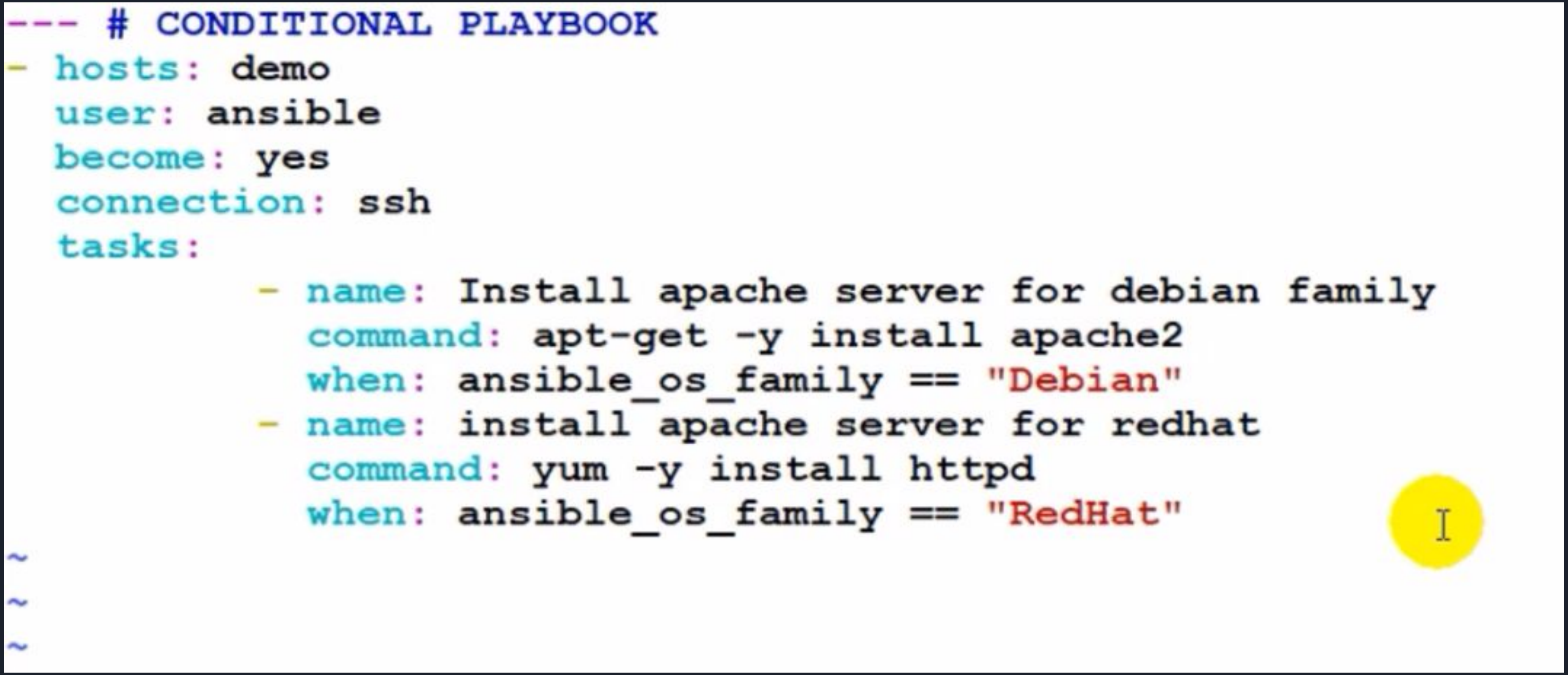
Conditions

→ Conditions, also known as "when" statements, allow you to control the execution of tasks in an Ansible playbook based on certain conditions. Here is an example of an Ansible playbook that uses a condition:



```yaml
- name: Install Apache web server
  hosts: all
  become: true

  tasks:
  - name: Install Apache on Debian-based systems
    apt:
      name: apache2
      state: present
    when: ansible_os_family == "Debian"

  - name: Install Apache on Red Hat-based systems
    yum:
      name: httpd
      state: present
    when: ansible_os_family == "RedHat"
```

Example :



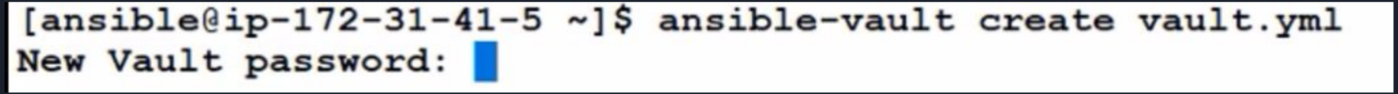
```
--- # CONDITIONAL PLAYBOOK
- hosts: demo
  user: ansible
  become: yes
  connection: ssh
  tasks:
          - name: Install apache server for debian family
            command: apt-get -y install apache2
            when: ansible_os_family == "Debian"
          - name: install apache server for redhat
            command: yum -y install httpd
            when: ansible_os_family == "RedHat"
~
~
~
```

Vault

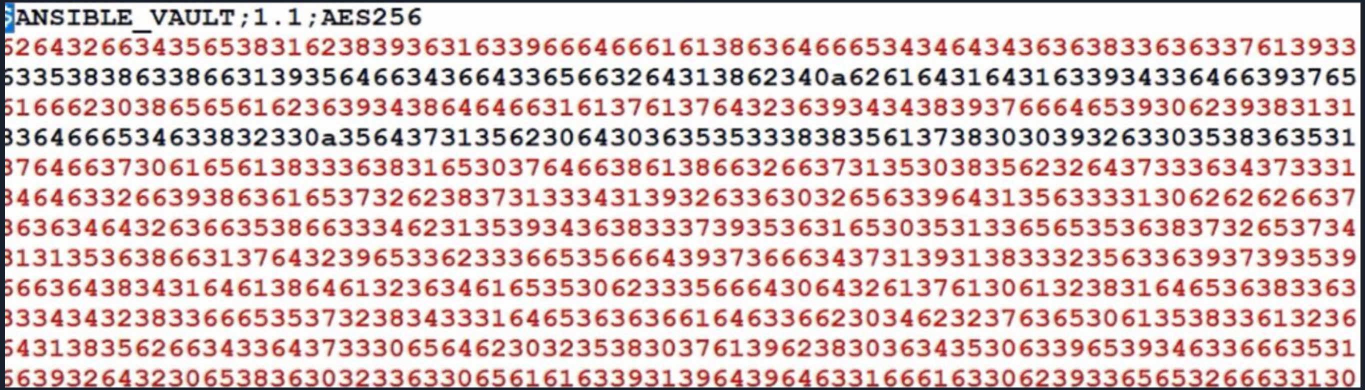
→ In Ansible, Vault is a feature that allows you to encrypt sensitive data such as passwords, API keys, or other secrets, in a playbook or variable file. This ensures that sensitive data is not stored in plaintext, and helps to protect against unauthorized access to sensitive information.

- Creating a new encrypted playbook = “ **ansible-vault create abc.yml** “ ,
- Edit the Encrypted Playbook = “ **ansible-vault edit abc.yml** “ ,
- To change password = “ **ansible-vault rekey abc.yml**”,
- To encrypt the existing Playbook = “**ansible-vault encrypt abc.yml**”,
- To decrypt the exiting playbook = “ **ansible-vault decrypt abc.yml** “

Example : 1

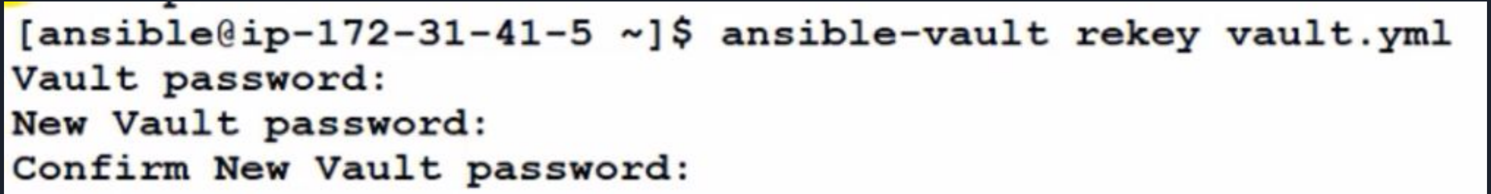


```
[ansible@ip-172-31-41-5 ~]$ ansible-vault create vault.yml
New Vault password: 
```



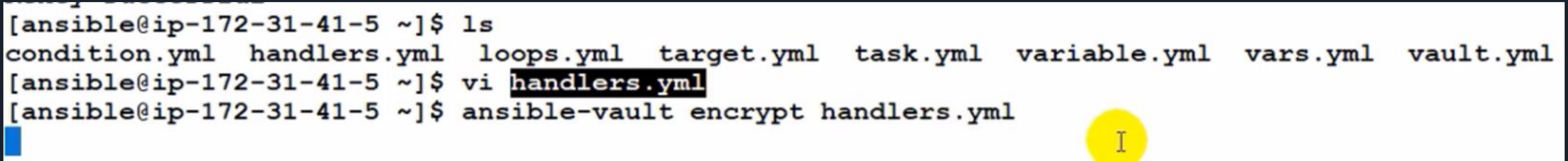
```
$ANSIBLE_VAULT;1.1;AES256
52643266343565383162383936316339666466616138636466653434643436363833636337613933
63353838633866313935646634366433656632643138623402a62616431643163393433646639376 5
51666230386565616236393438646466316137613764323639343438393766646539306239383131
33646665346338323330a3564373135623064303635353338383561373830303932633035383635 31
37646637306165613833363831653037646638613866326637313530383562326437333634373331
34646332663938636165373262383731333431393263363032656339643135633331306262626637
36363464326366353866333462313539343638333739353631653035313365653536383732653734
31313536386631376432396533623336653566643937366634373139313833323563363937393539
56636438343164613864613236346165353062333566643064326137613061323831646536383363
33343432383366653537323834333164653636366164633662303462323763653061353833613236
54313835626634336437333065646230323538303761396238303634353063396539346336663531
56393264323065383630323363306561616339313964396463316661633062393365653266633130
```

Example : 2



```
[ansible@ip-172-31-41-5 ~]$ ansible-vault rekey vault.yml
Vault password:
New Vault password:
Confirm New Vault password:
```

Example : 3

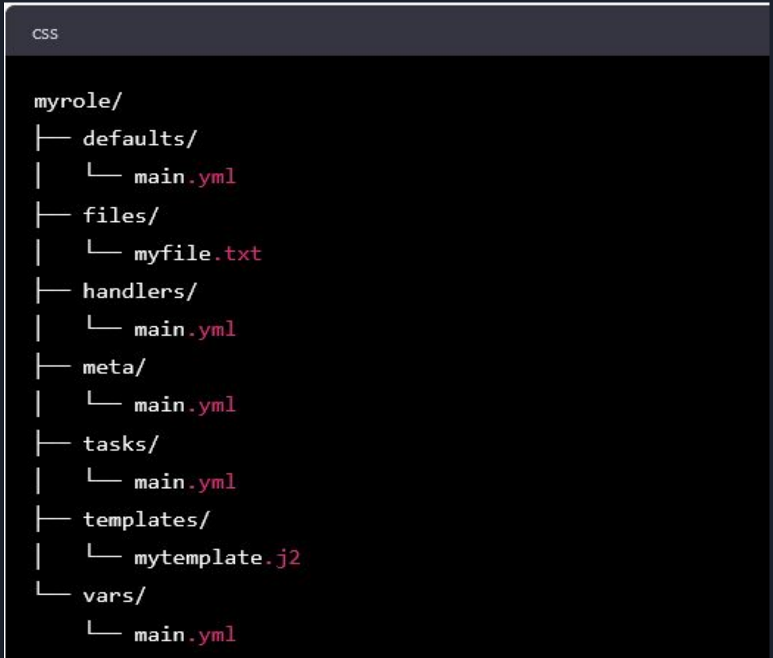


```
[ansible@ip-172-31-41-5 ~]$ ls
condition.yml  handlers.yml  loops.yml  target.yml  task.yml  variable.yml  vars.yml  vault.yml
[ansible@ip-172-31-41-5 ~]$ vi handlers.yml
[ansible@ip-172-31-41-5 ~]$ ansible-vault encrypt handlers.yml
```

Roles

→ In Ansible, a role is a predefined set of tasks, files, templates, and variables that can be reused across multiple playbooks. Roles help to simplify the management of complex playbooks by providing a structured way to organize and share reusable components.

→ Roles are typically organized into directories that follow a standard structure, such as:



```
myrole/
├── defaults/
│   └── main.yml
├── files/
│   └── myfile.txt
├── handlers/
│   └── main.yml
├── meta/
│   └── main.yml
├── tasks/
│   └── main.yml
├── templates/
│   └── mytemplate.j2
└── vars/
    └── main.yml
```

Role	Description
defaults/main.yml	Default variables for the role
files/	Static files that can be transferred to hosts
handlers/main.yml	Event-driven tasks that respond to notifications
meta/main.yml	Metadata and dependencies for the role
tasks/main.yml	The main set of tasks for the role
templates/	Jinja2 templates that can be rendered and transferred to hosts
vars/main.yml	**Variables for the role**