https://mindmajix.com/ansible-interview-questions

Configuration Management
----

ansible-senario based playbooks:-
https://chatgpt.com/share/6ce5a4b1-c3fe-48ba-a8c6-ef1f66da1483

This is process of configuring remote servers from one point of control.

Advantages

Upgrading Ansible by using the command: sudo pip install ansible==<version-number>

1) Provisioning of servers
The applications that should be installed on server can be done very quickly from a single centralized location.

2) Idempotent
Configuration management tools are used to bring the server to a particular state, called as desired state. If a server already in the desired state, configuration management tools will not reconfigure that server.

Note: Cofiguration management tools cannot be used for installing OS from the scratch.
They can be used only for managing the applications on top of the OS.

Configutaion management tools - Ansible, chef, puppet, salt etc

+++
Ansible -- It is a open source configuration management tool, created using Python.
Main machine in which anisble is installed, is called as controller.
Remote severs that Ansible configures, are called as managed nodes.

Ansible uses agent less policy for configures remote servers ie Ansible is installed only on 1 machine, and we do not require any client side software to be installed on the remote serers.

Ansible performs configuration management through password less ssh.

what are ansible tags?

If you have a large playbook, it may be useful to run only specific parts of it instead of running the entire playbook. You can do this with Ansible tags. Using tags to execute or skip selected tasks is a two-step process:

===>Add tags to your tasks, either individually or with tag inheritance from a block, play, role, or import.

===>Select or skip tags when you run your playbook.

+++++++++++++++++++++++++++++++++++
Create 4 Servers (Ubuntu 18)
1 is controller
3 are managed nodes

Name the instances as
Controller
Server1
Server2

Sailent features of Ansible

> Agent less Architecture
> Idempotency
> Large Collection of Built in methods
> Inventory Management
> Role Based Structure
> Integration with other Tools
> Secure
> Ansible Galaxy
> Ansible Tower

Server3

Ubuntu machines default come with Python3
Ansible supports Python2
We need to downgrade the machines from python3 to Python2

Connect server1
Check the version

```
$ python3 --version
```

To Install Python2

```
$ sudo apt-get update
$ sudo apt-get dist-upgrade ( It will point to  older apt repository  where python2 is available)

$  sudo apt-get install -y python2.7 python-pip

$ sudo apt-get install python3-pip
```

Now check the version of python

```
$ python --version
```

+++++++++++++++++++++++++

Establish password less ssh connection

```
$ sudo passwd ubuntu
```

(lets give the password as ubuntu only)

```
$ sudo vim /etc/ssh/sshd_config
```

change
PasswordAuthentication yes
Save and QUIT

```
$ sudo service ssh restart
$ exit
```

++++++++++++++++++

Repeat the same steps in server2 and server3

++++++++++++++++

Now, Connect to controller
Even in controller also python2 version should be available

(So, run the same commands)

```
$ sudo apt-get update
$ sudo apt-get dist-upgrade

$  sudo apt-get install -y python2.7 python-pip
```

Now check the version of python

Ansible Galaxy:

is a community driven Repository for Ansible-roles, Allowing users to share and downlode roles developed by others

Dynamic inventory

"Dynamic inventory in Ansible helps you automatically get information about your servers from sources like cloud providers.This is really helpful in cloud environments where servers can change frequently.

With dynamic inventory, you get the latest information about your servers every time you run an Ansible command, so you always have current data. Ansible has tools for popular cloud services like AWS and Azure, which makes it easy to pull in information about your cloud servers.

For example, if we want to manage AWS EC2 instances, we can create a file that tells Ansible to use the AWS plugin to get a list of running instances. When we run our playbook, Ansible will automatically fetch the current instances.

The benefits of dynamic inventory include better management of many servers, saving time, and reducing the chance of mistakes since we don't need to update a list manually. Overall, dynamic inventory helps us automate and adapt to changes in our infrastructure more easily."

Prerequisites:

AWS Credentials:

Make sure your AWS credentials are configured. You can set these up using the AWS CLI with aws configure or by exporting the AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY environment variables.

Ansible Version:

Ensure you have Ansible installed (version 2.9 or higher is recommended) along with the boto3 and botocore libraries, which are required for interacting with AWS.

Install Required Collections:

You need to have the amazon.aws collection installed. You can install it using: ansible-galaxy collection install amazon.aws

Example Inventory File

Create a file called ec2.yml for your dynamic inventory configuration:

```
# ec2.yml
plugin: aws_ec2
regions:
  - us-west-1  # Change this to your preferred AWS region
filters:
  instance-state-name: running  # Fetch only running instances
```

```
$ python --version
```

++++++++++++++++++++
Now , We need to generate ssh connections

```
$ ssh-keygen
```

Now copy the key to managed nodes

```
$ ssh-copy-id ubuntu@172.31.0.98   ( private Ip of server1 )

$ ssh-copy-id ubuntu@172.31.1.183  ( private Ip of server2 )

$ ssh-copy-id ubuntu@172.31.14.179  ( private Ip of server3 )
```

+++++++++++++++++++++++
Installing ansible now

Connect to controller.

```
$ sudo apt-get install software-properties-common
```

(software-properties-common , is a base package which is required to install ansible)

```
$ sudo apt-add-repository ppa:ansible/ansible

$ sudo apt-get update

$ sudo apt-get install -y ansible
```

++++++++++++++++++
To check ther version of ansible

```
$ ansible --version
```

+++++++++++++
Write the ip address of nodes in the inventory file

```
$ cd /etc/ansible
$ ls

$ sudo vim hosts
```

insert the private ip addresss of 3 servers
save and quit

```
$ ls -la     ( to see the list in the current machine )
$ ansible all  -a  'ls  -la'    ( you will get the list of the files in all managed nodes )
```

Example Playbook

Now, create an Ansible playbook named manage_ec2_instances.yml. This example playbook will gather information about the EC2 instances and run a command (like checking the uptime) on each of them:

```
# manage_ec2_instances.yml
---
- name: Manage AWS EC2 Instances
  hosts: tag_Environment_production  # Change this to
match your tag or group
  gather_facts: false
  tasks:
    - name: Check the uptime of EC2 instances
      ansible.builtin.command: uptime
      register: uptime_output

    - name: Display uptime
      ansible.builtin.debug:
        var: uptime_output.stdout_lines
```

Running the Playbook: ansible-playbook -i ec2.yml manage_ec2_instances.yml

Difference between Docker and Ansible??

Docker container is implemented with host OS software including process, chroot, cgroup, network and so on to util ize independent environment directly on host OS. On the other hand, Ansible is a configuration management tool. ... This tool just manages to automate installation and configuration to all the servers

===
what is Module???

Modules (also referred to as “task plugins” or “library plugins”)   can be used from the command line or in a playbook task. Ansible executes each module, usually on the remote managed node, and collects return values.

----
in ansible we can configure the servers in 2-ways i.e by using
1) adhoc commands
2) playbooks

adhoc commands

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

what are the Important modules in ansible ?
----
1) command - This module is used for executing basic linux commands on managed nodes.
2) shell - This module is used to execute commands which involved redirection and piping and to execute shell scripts on managed nodes.
3) ping -- This module is used to check if the remote server is pingable or not.
4) user -- This module is used for user management like create user, setting password, assign home directory etc
5) copy -- This module is used to copy the files and folders from controller to managed nodes
6) fetch -- This module is used to copy files and folder from managed nodes to controller
7) file -- This module is used for creating or deleting files and folders on managed nodes.
8) stat -- Used to capture detailed information about files and folders present in managed nodes.
9) debug -- Used to display output of any module
10) apt   -- Used for performing package management on managed nodes ie installing softwares / upgrading repositories etc . It works on ubuntu, debain flavours of linux.
11) yum -- similar to apt module. It works on Red hat linux, centos etc
12) git -- used to perform git version controlling on managed nodes
13) replace -- This is used to replace specific text in configuration file with some other text.
14) service -- used for starting / stoping / restarting services on managed nodes.
15) include -- Used for calling child play books from parent play book
16) uri   -- useful in checking if remote url is reachable or not.
17) docker_container -- used to execute docker commands related to container management on managed nodes
18) docker_image -- used to execute commands related to docker images on managed nodes.
19) docker_login -- used to login to docker hub from managed nodes.
20) setup   -- used to capturing system information related to the managed nodes.

++

```
$ ansible all -i /etc/ansible/hosts -m command -a 'free'
```

```
$ ansible all -i /etc/ansible/hosts -m command -a 'touch file1'
```

we want to back up both web application files and a MySQL database on a web server. The backup should be saved to a remote storage location (e.g., an NFS share or an S3 bucket).

1. Backing Up Files with Ansible

To back up application files located in /var/www/html, we’ll:

To check the file which is created

```
$ ssh 172.31.2.173    ( this command will go that machine  )
$  ls
$  exit    ( to come back to controller )
```

++++++++++++++++

To install docker in all managed nodes

```
$ ansible all -i /etc/ansible/hosts -m shell -a 'curl  -fsSL  https://get.docker.com  -o get-docker.sh'

$ ansible all -i /etc/ansible/hosts -m shell -a 'sh get-docker.sh'
```

+++++++++++++

To check docker is installed or not

```
$ ssh 172.31.2.173
$ docker  --version
$  exit  (  to come back to controller )
```

+++++++++++++++++

Notes:

Ansible performs remote configurations in 2 ways
1) using adhoc commands
2) using play books

Syntx of adhoc commands

```
$ ansible  all/group_name/ipaddress -i  path_of_inventory_file -m modulename  -a  'arguments'
```

+++++++++++++++++

Ansible command module to check the memory info on all managed nodes

```
$ ansible all -i /etc/ansible/hosts -m command -a 'free'
```

++++++++++++++++++

To open the default inventory file

```
$ sudo vim  /etc/ansible/hosts
```

(Observation: 3 ip address are available)

+++++++++++++++++

Now, I copy the first two IP address (in a new notepad file)
quit the inventory file

+++++++++++++

1) Create a backup directory with a timestamp to ensure each backup is unique.
2) Use Ansible's synchronize or copy module to transfer files to the backup directory.
3) Optionally, compress the backup to save space.

Example Ansible Playbook for File Backup

```
---
- name: Backup web application files
  hosts: webservers
  vars:
    backup_base_dir: "/backups"
    app_files_dir: "/var/www/html"
    timestamp: "{{ ansible_date_time.iso8601 }}"

  tasks:
    - name: Create timestamped backup directory
      file:
        path: "{{ backup_base_dir }}/files/{{ timestamp }}"
        state: directory
        mode: '0755'
    - name: Copy application files to backup directory
      synchronize:
        src: "{{ app_files_dir }}"
        dest: "{{ backup_base_dir }}/files/{{ timestamp }}/"
        rsync_opts:
          - "--compress"

    - name: Archive and compress backup files (optional)
      command: tar -czf "{{ backup_base_dir }}/files/{{ timestamp }}.tar.gz" -C "{{
                                             backup_base_dir }}/files/{{ timestamp }}" .
      when: backup_compression | default(true)
```

Explanation:

The timestamp variable creates unique backup directories to avoid overwriting.
The file module creates the backup directory.
The synchronize module copies the files from app_files_dir to the backup location.
We optionally compress the backup using tar to save space

2. Backing Up MySQL Database with Ansible

For database backups, we'll:

1) Use the mysql_db Ansible module to dump the database.
2) Store the backup files in the same backup directory as the application files.

Example Ansible Playbook for Database Backup

```
---
- name: Backup MySQL database
  hosts: dbservers
  vars:
    backup_base_dir: "/backups"
    db_name: "my_database"
    db_user: "backup_user"
    db_password: "secure_password"
    timestamp: "{{ ansible_date_time.iso8601 }}"
  tasks:
    - name: Create timestamped backup directory for database
      file:
        path: "{{ backup_base_dir }}/db/{{ timestamp }}"
        state: directory
        mode: '0755'

    - name: Dump MySQL database to backup directory
      command: >
        mysqldump -u {{ db_user }} -p{{ db_password }} {{ db_name }}
        > "{{ backup_base_dir }}/db/{{ timestamp }}/{{ db_name }}.sql"
      no_log: true
```

Create my own inventory file

```
$ vim  myinventory
```

go to insert mode

paste two ip address
save and quit

+++++++++++
To check the inventory file

```
$ cat myinventory
```

+++++++++++

```
$ ansible all -i myinventory -m command -a 'free'
```

Observation: free command works on only two machines

+++++++++++++
If you do not mention the inventory file, it takes default inventory file.
ex:

```
$ ansible all  -m command -a 'free'
```

++++++++++++++++++++++++++++++++

command module is the default module in ansible

```
$ ansible all  -a 'free'
```

++++++++++++++++++++++

Note:
The defualt inventory file is /etc/ansible/hosts and when using this inventory file, we need not use -i option.
ex:

```
$ ansible all  -m command -a 'free'
```

The default module is command. When using command module we need not use -m option
ex:

```
$ ansible all  -a 'free'
```

Shell Module

Explanation:
----

1) The file module creates a timestamped directory for the database backup.
2) The command module runs mysqldump, dumping the database to an SQL file.
3) We use no_log: true to avoid logging sensitive information.

3. Transferring Backups to Remote Storage

In a production environment, you may want to send backups to a remote storage solution such as Amazon S3, an NFS share, or a backup server. Here's an example of uploading the backup to an NFS share.

Example Ansible Playbook for Transferring Backups

```
---
- name: Transfer backups to remote storage
  hosts: all
  vars:
    backup_base_dir: "/backups"
    nfs_mount: "/mnt/backup_nfs"

  tasks:
    - name: Ensure NFS share is mounted
      mount:
        path: "{{ nfs_mount }}"
        src: "nfs-server:/export/backups"
        fstype: nfs
        state: mounted

    - name: Copy compressed backup files to NFS share
      copy:
        src: "{{ backup_base_dir }}/files/{{ timestamp }}.tar.gz"
        dest: "{{ nfs_mount }}/files/{{ inventory_hostname }}-{{ timestamp }}.tar.gz"
      ignore_errors: true

    - name: Copy MySQL backup to NFS share
      copy:
        src: "{{ backup_base_dir }}/db/{{ timestamp }}/{{ db_name }}.sql"
        dest: "{{ nfs_mount }}/db/{{ inventory_hostname }}-{{ timestamp }}.sql"
      ignore_errors: true
```

Explanation:
----

1) The mount module ensures that the NFS share is accessible.
2) The copy module transfers the compressed backup files and database dump to the NFS mount.

4. Scheduling Backups in Real-Time

To run backups periodically, you can schedule this playbook to run at a specific time, like nightly, using Ansible Tower/AWX or a cron job. Here's how you might set up a cron job using Ansible:

Scheduling Backups with Ansible Cron Module

```
---
- name: Schedule nightly backup
  hosts: localhost
  tasks:
    - name: Add cron job for nightly backup
      cron:
        name: "Nightly Backup"
        minute: "0"
        hour: "2"
        user: "ansible"
        job: "ansible-playbook /path/to/backup_playbook.yml"
```

Note: This cron job will trigger the backup playbook at 2:00 AM every night.

Setting up alerts in Ansible to notify you of the success or failure of backups, or if the backup directory exceeds a certain size, is essential for proactive monitoring. You can use tools like Slack, email, or log monitoring systems to receive these notifications automatically.

ansible command to execute ls -la and store the output into file1 on all the managed nodes.

```
$ ansible  all  -m  shell -a 'ls -la > file2'
```

To check the file which is created

```
$ ssh 172.31.12.239
$ ls
$ exit   ( to come back to controller )
```

+++++++++++++++
command to install docker on all managed nodes

```
$ ansible all  -m shell -a 'curl  -fsSL  https://get.docker.com  -o get-docker.sh'

$ ansible all -m shell -a 'sh get-docker.sh'
```

++++++++++++++++++++++++
User Module:
(From controller)
To create new user

```
$ sudo  useradd sai
$ vim /etc/passwd     (  User will be created in this file )
```

To set the password

```
$ sudo passwd  sai      ( sai is the username)
```

+++++++++++
Now, i want to create user in all managed nodes

```
$ ansible  all -m user  -a  'name=anu  password=sunil'
```

(we ger error : permission denied)

```
$ ansible  all -m user  -a  'name=anu  password=sunil'  -b  ( become , for higher privileges on managed nodes )
```

++++++++++++++
To check if user is create or not

```
$  ssh 172.31.12.239
$ vim  /etc/passwd
$  exit
```

+++++++++++++++++++++
Command to create user and set home directory, user id, default working shell etc
Another example

```
$ ansible all -m user  -a 'name=Ravi password=freefree uid=1234 comment="A regular user"  home=/home/ubuntu/
```

1. Alerting on Backup Status (Success/Failure) via Slack

To send alerts to Slack, you'll need:

> A Slack webhook URL to post messages to a specific channel.
> An Ansible playbook that checks for the status of each backup step and posts alerts if a step fails or succeeds.

Slack Notification Setup
Create a Slack Incoming Webhook:
> Go to the Slack API and create an Incoming Webhook.
> Copy the generated webhook URL.
Include Slack Notification in the Backup Playbook

Here's an example of a modified backup playbook that notifies Slack on success or failure:

```
---
- name: Backup web files and notify Slack on status
  hosts: webservers
  vars:
    backup_base_dir: "/backups"
    app_files_dir: "/var/www/html"
    timestamp: "{{ ansible_date_time.iso8601 }}"
    slack_webhook_url: "https://hooks.slack.com/services/XXXX/YYYY/ZZZZ"

  tasks:
    - name: Create timestamped backup directory
      file:
        path: "{{ backup_base_dir }}/files/{{ timestamp }}"
        state: directory
        mode: '0755'
      register: create_dir
      ignore_errors: yes

    - name: Copy application files to backup directory
      synchronize:
        src: "{{ app_files_dir }}"
        dest: "{{ backup_base_dir }}/files/{{ timestamp }}/"
      register: sync_files
      ignore_errors: yes

    - name: Send Slack notification if backup succeeded
      uri:
        url: "{{ slack_webhook_url }}"
        method: POST
        body: '{"text": ":white_check_mark: Backup completed successfully for {{
                   inventory_hostname }}."}'
        headers:
          Content-Type: "application/json"
      when: create_dir is succeeded and sync_files is succeeded
      delegate_to: localhost

    - name: Send Slack notification if backup failed
      uri:
        url: "{{ slack_webhook_url }}"
        method: POST
        body: '{"text": ":x: Backup failed for {{ inventory_hostname }}. Check logs for
                   details."}'
        headers:
          Content-Type: "application/json"
      when: create_dir is failed or sync_files is failed
      delegate_to: localhost
```

Ravi shell=/bin/bash' -b

To check for the new user

```
$  ssh 172.31.44.218
$ vim  /etc/passwd
```

++++++++++

Install git in all managed nodes

```
$ ansible all -m apt  -a 'name=git  state=present' -b
```

Observation:
We get "changed": false
(That means git is already installed on it. The command has no effect in the nodes)

Now , run the below command

```
$ ansible all -m apt  -a 'name=git  state=absent' -b
```

(absent means - uninstall)

output, we get in yellow color
(scroll up) we get "changed":true
(The command is effected the instance)

Now if we run the below command (with present option)

```
$ ansible all -m apt  -a 'name=git  state=present' -b
```

we get "changed":true

Notes:
apt module -- This is used for package management.
1) ansible all -m apt -a 'name=git state=present' -b

state=present is for installation
state=latest   for upgradation
state=absent   for uninstallation

++

I wan to update apt-repositoty and install tomcat8

```
ansible all -m  apt -a 'name=tomcat8 state=present  update_cache=yes' -b
```

Explanation:
----

1)The uri module posts messages to Slack using the webhook URL
.
2)Separate tasks notify Slack if the backup succeeds (using the when condition is succeeded) or fails (using the when condition is failed).

3)delegate_to: localhost ensures that the Slack message is sent from the control node, not the target server.

2. Alerting via Email (SMTP) on Backup Status

To send email alerts, you'll need:

- >> Access to an SMTP server or a mail relay.
- >> The mail module in Ansible to send emails.

Email Notification Example

This example sends an email based on backup success or failure:

```
---
- name: Backup web files and notify via email
  hosts: webservers
  vars:
    backup_base_dir: "/backups"
    app_files_dir: "/var/www/html"
    timestamp: "{{ ansible_date_time.iso8601 }}"
    email_recipient: "admin@example.com"
    email_sender: "ansible@example.com"
  tasks:
    - name: Create timestamped backup directory
      file:
        path: "{{ backup_base_dir }}/files/{{ timestamp }}"
        state: directory
        mode: '0755'
      register: create_dir
      ignore_errors: yes

    - name: Copy application files to backup directory
      synchronize:
        src: "{{ app_files_dir }}"
        dest: "{{ backup_base_dir }}/files/{{ timestamp }}/"
      register: sync_files
      ignore_errors: yes

    - name: Send success email notification
      mail:
        host: smtp.example.com
        port: 587
        to: "{{ email_recipient }}"
        from: "{{ email_sender }}"
        subject: "Backup Successful on {{ inventory_hostname }}"
        body: "The backup completed successfully for {{ inventory_hostname }} at
              {{ timestamp }}."
      when: create_dir is succeeded and sync_files is succeeded
      delegate_to: localhost

    - name: Send failure email notification
      mail:
        host: smtp.example.com
        port: 587
        to: "{{ email_recipient }}"
        from: "{{ email_sender }}"
        subject: "Backup Failed on {{ inventory_hostname }}"
        body: "The backup failed for {{ inventory_hostname }}. Please check the
               logs for more details."
      when: create_dir is failed or sync_files is failed
      delegate_to: localhost
```

The above command will update apt repository and install tomcat8
To update apt-repository on managed nodes update_cache=yes is used

++++++++++++++++++++++++++

File module

This is used to create files and folder on managed nodes

```
ansible all  -m  file -a  'name=/tmp/file5  state=touch'
```

To check the file which is create

```
$ ssh 172.31.12.239
$ cd  /tmp
$ ls
$ exit
```

TO create a directory

```
ansible all  -m  file -a  'name=/tmp/dir1  state=directory'
```

To check the directory

```
$  ssh 172.31.39.33
$ cd  /tmp
$ ls
$ exit
```

To delete the file

```
ansible all  -m  file -a  'name=/tmp/file5  state=absent'
```

+++++++++++++++++++++++++
Notes:
Command to create a file on all managed nodes

```
ansible all  -m  file -a  'name=/tmp/file1  state=touch'
```

state=touch   is to create files
state=directory is to create directory
state=absent is for deleting file/directory

+++++++++++++++++++++
Now,
To know the current user

```
$ whoami
$  ansible all  -m  file -a  'name=file1  state=touch'
```

Now go to managed nodes and check the permission of the file

```
$ ssh 172.31.12.239
$ ls -l file1
```

Observe the permissions are   rw-rw-r--

Explanation:

>> The mail module sends an email with different subjects and bodies depending on whether the backup succeeded or failed.
>> Replace smtp.example.com with your SMTP server.

3. Monitoring Backup Directory Size and Setting an Alert

To monitor the backup directory size and trigger an alert if it exceeds a threshold, use the stat module to check the size and send an alert if it's over a defined limit.

Example: Alert on Backup Directory Size

Here's a playbook snippet that checks the size of the backup directory and sends an alert if it exceeds 10 GB:

```
---
- name: Check backup directory size and alert if exceeds limit
  hosts: webservers
  vars:
    backup_dir: "/backups/files"
    size_limit: 10737418240  # 10 GB in bytes
    slack_webhook_url: "https://hooks.slack.com/services/XXXX/YYYY/ZZZZ"

  tasks:
    - name: Get backup directory size
      stat:
        path: "{{ backup_dir }}"
      register: backup_dir_info

    - name: Send Slack alert if backup directory exceeds size limit
      uri:
        url: "{{ slack_webhook_url }}"
        method: POST
        body: '{"text": ":warning: Backup directory on {{ inventory_hostname }}
               exceeds size limit of 10 GB."}'
        headers:
          Content-Type: "application/json"
      when: backup_dir_info.stat.size > size_limit
      delegate_to: localhost
```

Explanation:

>> The stat module retrieves the size of the backup directory.
>> If the size exceeds the threshold (size_limit), a Slack notification is sent.
>> Adjust size_limit based on your storage policy.

4. Centralized Logging for Backup Status

Using centralized logging tools (like ELK Stack or Splunk) is another way to track backup statuses. Ansible logs can be shipped to a central server for real-time monitoring and alerting if a failure occurs.

Steps:
>> Enable Ansible logging: Configure Ansible to log to a central log file.

>> Ship logs: Use a log shipper like Filebeat to send logs to a centralized system where alert rules can trigger notifications on failures or errors.

Permission Types: For files and directories, there are 4 types of permissions.
1) r -->Read
2) w --> Write
3) x -->Execute
4) - -->No Permission

Numeric Permissions:

We can specify permissions by using octal number.
Octal means base-8 and allowed digits are 0 to 7
0 -- >000 --> No Permission
1 --> 001 --> Execute Permission
2 -->010 --> Write Permission
3 --> 011 --> Write and execute Permissions
4 -->100 --> Read Permission
5 --> 101 -->Read and execute Permissions
6 --> 110 --> Read and write Permission
7 -->111 --> Read, Write and execute Permissions

Note:
----
4 --> Read Permission
2 --> Write Permission
1 --> Execute Permission

It is more easy to remember
----
5 --> 4+1 --> r-x
3 --> 2+1 --> -wx
6 --> 4+2 --> rw-
7 -->4+2+1 -->rwx

Now, I want to change the permissions from controller

```
$ exit   ( will come back to controller )
$  ansible all  -m  file -a  'name=file1  state=touch owner=Anu group=Ravi  mode=700'  -b
```

The above command will execute only if Anu user and Ravi group is available in all nodes.

Notes:
File module can be used to change the ownership, group ownership and permissions on the file.

Copy Module

This is used for copying the files from controller into managed nodes.

We know in the file /etc/passwd we have all the information about users

Now I want to copy the file into all nodes

```
$ ansible all -m  copy  -a 'src=/etc/passwd dest=/tmp'
```

1. Automated Server Reboots

Automating server reboots is common for maintenance, applying kernel updates, or resolving issues. With Ansible, you can control the timing, add delays, and ensure services are back online after rebooting.

Example Playbook for Controlled Server Reboots

Here's a playbook that:

1. Notifies users (e.g., via Slack) before rebooting.
2. Reboots servers in batches to avoid downtime.
3. Verifies that essential services are running after the reboot.

```yaml
---
- name: Controlled server reboots with notifications
  hosts: webservers
  serial: 2  # Reboot two servers at a time to maintain availability
  vars:
    slack_webhook_url: "https://hooks.slack.com/services/XXXX/YYYY/ZZZZ"
    critical_services:
      - nginx
      - mysql

  tasks:
    - name: Notify users of upcoming reboot on Slack
      uri:
        url: "{{ slack_webhook_url }}"
        method: POST
        body: '{"text": ":warning: Rebooting server {{ inventory_hostname }} for maintenance."}'
        headers:
          Content-Type: "application/json"
      delegate_to: localhost

    - name: Reboot the server
      reboot:
        msg: "Scheduled maintenance reboot"
        pre_reboot_delay: 30
        post_reboot_delay: 60
      register: reboot_status

    - name: Check if critical services are running after reboot
      service:
        name: "{{ item }}"
        state: started
      loop: "{{ critical_services }}"
      ignore_errors: yes

- name: Notify users of reboot completion on Slack
      uri:
        url: "{{ slack_webhook_url }}"
        method: POST
        body: '{"text": ":white_check_mark: Server {{ inventory_hostname }} rebooted and services are running."}'
        headers:
          Content-Type: "application/json"
      when: reboot_status is succeeded
      delegate_to: localhost
```

Explanation:

>> serial: 2: Reboots two servers at a time to ensure that the load balancer or other servers can handle incoming requests.

>> uri (Slack notification): Alerts users before and after the reboot to inform them of maintenance actions.

>> reboot module: Handles the reboot, with delays before and after for graceful shutdown and restart.

>> service module: Verifies that critical services (like nginx and mysql) are up after rebooting. This step ensures applications are operational.

2. Application Maintenance (Restarting Services, Clearing Cache, Running Migrations)

To check the file which is copies

```
$ ssh 172.31.12.239
$ cd /tmp
$ ls
$ exit
```

++++++++++++++

Scenario:

I want to create tomcat users file in controller and copy the file in all the nodes

```
$ sudo vim tomcat-users.xml
```

Go to Insert mode

```
<tomcat-users>
  <user  username="training"  password="freefree"  roles="manager-script"/>
<tomcat-users>
```

```
:wq
```

```
$  ansible all -m  copy  -a  'src=tomcat-users.xml  dest=/etc/tomcat8'  -b
```

To check the file

```
$ ssh 172.31.12.239
$ cd /etc/tomcat8
$ ls
```

Open that file to check the contents

```
$  sudo  cat  tomcat-users.xml
```

+++++++++++++++++++++++++++++++++

Ansible command to copy /etc/passwd file to all the managed nodes

```
$ ansible all -m  copy  -a 'src=/etc/passwd dest=/tmp'
```

++++++++++++++++++++++

Create a tomcat-users.xml file on controller and copy it into all managed nodes into default location of tomcat ie /etc/tomcat8

```
$ sudo vim tomcat-users.xml
```

Go to Insert mode

```
<tomcat-users>
  <user  username="training"  password="freefree"  roles="manager-script"/>
<tomcat-users>
:wq
```

```
$  ansible all -m  copy  -a  'src=tomcat-users.xml  dest=/etc/tomcat8'  -b
```

Application maintenance often involves restarting services, clearing caches, and applying database migrations. Here's how you can use Ansible for typical maintenance tasks.

Example Playbook for Application Maintenance

This example handles:

>> Clearing application caches.

>> Restarting necessary services.

>> Running database migrations.

```
---
- name: Application maintenance tasks
  hosts: appservers
  vars:
    app_cache_dir: "/var/www/html/app/cache"
    critical_services:
      - nginx
      - php-fpm
    slack_webhook_url: "https://hooks.slack.com/services/XXXX/YYYY/ZZZZ"

  tasks:
    - name: Notify users of maintenance start on Slack
      uri:
        url: "{{ slack_webhook_url }}"
        method: POST
        body: '{"text": ":construction: Starting maintenance on {{ inventory_hostname }}."}'
        headers:
          Content-Type: "application/json"
      delegate_to: localhost

    - name: Clear application cache
      file:
        path: "{{ app_cache_dir }}"
        state: absent
      ignore_errors: yes

    - name: Recreate application cache directory
      file:
        path: "{{ app_cache_dir }}"
        state: directory
        mode: '0755'

    - name: Restart critical services
      service:
        name: "{{ item }}"
        state: restarted
      loop: "{{ critical_services }}"

    - name: Run database migrations
      command: "php /var/www/html/app/artisan migrate --force"
      register: migration_result
      failed_when: migration_result.rc != 0

    - name: Notify users of maintenance completion on Slack
      uri:
        url: "{{ slack_webhook_url }}"
        method: POST
        body: '{"text": ":white_check_mark: Maintenance completed on {{ inventory_hostname }}."}'
        headers:
          Content-Type: "application/json"
      delegate_to: localhost
```

Explanation:

>> Clear application cache: The file module removes the cache directory to clear cached data.

>> Recreate cache directory: Ensures the cache directory is ready for use after clearing.

>> Restart critical services: Restarts services like nginx and php-fpm, ensuring that the app can serve requests smoothly.

>> Run database migrations: Runs database migrations using a command. The failed_when condition will flag this step as failed if the migration command fails.

>> Slack notifications: Notifies users before and after maintenance.

3. Rolling Maintenance Across Multiple Servers

To avoid downtime, you can perform maintenance in a rolling fashion, where you apply maintenance tasks to a few servers at a time. This approach helps maintain service availability.

Example Playbook for Rolling Maintenance

++

Create a file on the controller machine

```
$ cat > newfile1
aaaa
bbbbb
ccccc
ddddd
Ctrl+d
```

```
$ ls -l newfile1
```

we get the permissions
rw-rw-r--
When we copy the file we have the same permissions

```
$ ansible all -m copy -a 'src=newfile1 dest=/home/ubuntu'
```

To got managed node and check the permissions on the file. It remains the same

```
$ ssh 172.31.39.33
$ ls -l newfile1
$ exit
```

Command to copy with changes permissions

```
$ ansible all -m copy -a 'src=newfile1 dest=/home/ubuntu owner=root group=root mode=760' -b
```

Now, go to node and check the permissions

```
$ ssh 172.31.35.79
$ ls -l newfile1
$ exit
```

Notes:
Copy module is used to change the ownership, group ownership and permissions of the files that are copied to mana ged nodes.

```
$ ansible all -m copy -a 'src=newfile1 dest=/home/ubuntu owner=root group=root mode=760' -b
```

++

To copy the file , by replacing the old content with new content

```
$ ansible all -m copy -a 'content="sunil\n" dest=newfile1' -b
```

TO to managed node and check the content

```
$ ssh 172.31.11.96
$ sudo cat newfile1
$ exit
```

Notes: Copy module can also send content into the file

```
$ ansible all -m copy -a 'content="sunil\n" dest=newfile1' -b
```

```
---
- name: Rolling application maintenance
  hosts: appservers
  serial: 2  # Maintain application availability by updating two servers at a time
  vars:
    slack_webhook_url: "https://hooks.slack.com/services/XXXX/YYYY/ZZZZ"

  tasks:
    - name: Notify users of rolling maintenance start on Slack
      uri:
        url: "{{ slack_webhook_url }}"
        method: POST
        body: '{"text": ":construction: Rolling maintenance starting on {{ inventory_hostname
              }}."}'
        headers:
          Content-Type: "application/json"
      delegate_to: localhost

    - name: Update application files
      git:
        repo: "https://github.com/example/repo.git"
        dest: "/var/www/html/app"
        version: "main"

    - name: Restart application services
      service:
        name: nginx
        state: restarted

    - name: Notify users of maintenance completion for server
      uri:
        url: "{{ slack_webhook_url }}"
        method: POST
        body: '{"text": ":white_check_mark: Rolling maintenance completed on {{ inventory_hostname }}."}'
        headers:
          Content-Type: "application/json"
      delegate_to: localhost
```

Explanation:

>> serial: 2: Applies the playbook to two servers at a time to ensure the app remains available.
>> git module: Pulls the latest version of the application from a Git repository, updating it without needing a full redeployment.
>> service module: Restarts nginx after updating files.
>> Slack notifications: Notifies users when maintenance starts and completes on each server.

Best Practices for Server Reboots and Maintenance with Ansible

==

>> Rolling Reboots and Maintenance: Use the serial option to reboot or update servers in batches. This approach helps avoid service interruptions and maintains availability.

>> Notify Users: Notify teams or end-users before and after maintenance tasks. Integrate with tools like Slack, email, or ticketing systems for automated communication.

>> Validation: After reboots or maintenance, validate that services and applications are running as expected. Ansible tasks like service status checks, endpoint health checks, or load balancer validations help ensure uptime.

>> Testing: Run playbooks in a staging or testing environment first, especially when applying updates or running database migrations. This prevents unexpected issues in production.

>> Scheduling with Ansible Tower/AWX: If reboots or maintenance need to occur during specific windows (e.g., after hours), use Ansible Tower or AWX to schedule playbook runs. This ensures tasks happen automatically at the right time.

++++++++++++++++++++++

Fetch Module (opposite of copy module)

Go to managed node

```
$ ssh 172-31-35-79
$ cd /etc/tomcat8
$ ls
```

There is server.xml file
I want to get the file (server.xml) from node to controller

```
$ exit ( come back to controller )
$ ansible all -m fetch -a 'src=/etc/tomcat8/server.xml dest=/tmp' -b
```

Now to got tmp folder

```
$ cd /tmp
$ ls
```

You will find three folders. The names of the folers are IP address of managed nodes

```
$ cd 172.31.35.102
$ ls
$ cd etc
$ ls
$ cd tomcat8
$ ls
```

Notes:
Fetch module is used to copy files from managed nodes to controller.
Command to copy tomcat-server.xml file from all managed nodes into /tmp folder on the controller.

```
$ ansible all -m fetch -a 'src=/etc/tomcat8/server.xml dest=/tmp' -b
```

Git Modules

This is used to perform git version controlling on the managed nodes.

```
ansible all -m git -a 'repo=https://github.com/sunildevops77/repo1.git dest=/tmp/mygit' -b
```

The above command will download the files in all managed nodes.

Go to managed node and check

```
$ ssh 172.31.35.79
$ cd /tmp
$ ls
$ cd mygit
$ ls
$ exit
```

Notes:

Ansible command to clone remote git repository into all managed nodes

```
ansible all -m  git  -a  'repo=https://github.com/sunildevops77/rep1.git  dest=/tmp/mygit' -b
```

++++++++++++++++++

Service Module

This is used for starting/ stoping / restarting the services.

Ansible command to restart tomcat8 on all managed nodes

```
$  ansible all   -m service    -a 'name=tomcat8  state=restarted'  -b
```

state=restarted is for restarting a service
state=stopped is for stopping a running service
state=started is for starting a stopped service

Replace module

Go to managed node

```
$ ssh 172.31.36.52
$ cd /etc/tomcat8/
$ ls
$ sudo  vim  server.xml
```

Look for connector port , to see the port number in which it is running. (line 74)

Now, we want to change the port number on all managed nodes, in this scenario
we use replace module.

Quit the server.xml file

```
$ exit   ( to come back to controller )
$  ansible all  -m replace  -a 'regexp=8080 replace=9090 path=/etc/tomcat8/server.xml'  -b
```

Lets check tomcat is respoding on 9090 port in managed node

Get public DNS from aws
ec2-13-251-114-207.ap-southeast-1.compute.amazonaws.com
ec2-13-234-48-168.ap-south-1.compute.amazonaws.com

Open Browser
URL --- ec2-13-251-114-207.ap-southeast-1.compute.amazonaws.com:9090

We will not get the page, becuase we need to restart the service

```
$  ansible all   -m service    -a 'name=tomcat8  state=restarted'  -b
```

Now, try the above URL --- it Works!!

replace module

This is used for replacing a specific string with other string.
Ex:
Ansible command to change the port number of tomcat from 8080 to 9090

```
$ ansible all  -m replace  -a 'regexp=8080 replace=9090 path=/etc/tomcat8/server.xml'  -b
```

uri module

I want to check facebook is reachable for not in all managed nodes.

```
$ ansible all -m   uri  -a 'url=http://facebook.com'
```

In the output (green color) status - 200

Give a invalid url , we get status as -1
Ex:

```
$ ansible all -m   uri  -a 'url=http://hgyi9cb.com'
```

Now, I want to stop tomcat in all managed nodes (Just repeat)

```
$  ansible all   -m service    -a 'name=tomcat8  state=stopped'  -b
```

Notes:
urI module is used to check if the url is reachable or not.
Command to check if facebook.com is reachable on all managed nodes.

```
$ ansible all -m   uri  -a 'url=http://facebook.com  status=200'
```

++++++++++++++++++++++++++++++++++++
Lets have an example of all modules
Requirement: I want to install tomcat all manages nodes , then i want to copy users.xml in all managed nodes,
I want to change port number of tomcat , then i want to restart the service, finally i want to check
url is reachable or not.

1st we need to unintall tomcat in all managed nodes.

```
$ ansible  all  -m  apt -a 'name=tomcat8 state=absent  purge=yes'  -b
```

```
$  ansible all -m  apt -a 'name=tomcat8 state=present' -b

$  ansible all -m  copy  -a  'src=tomcat-users.xml  dest=/etc/tomcat8'  -b
$ ansible all  -m replace  -a 'regexp=8080 replace=9090 path=/etc/tomcat8/server.xml'  -b
$  ansible all   -m service    -a 'name=tomcat8  state=restarted'  -b
```

To check tomcat is running individually on all servers,
take the private ip of all nodes
172.31.11.96
172.31.6.207
172.31.12.138

```
$ ansible all -m   uri  -a 'url=http://172.31.11.96:9090'
```

It returns status as 200

Similarly check the other two nodes

```
$ ansible all -m   uri  -a 'url=http://172.31.6.207:9090'
$ ansible all -m   uri  -a 'url=http://172.31.12.138:9090'
```

++++++++++++++++++++++
Notes:
Requirement.

I want to install tomcat all modules.Copy tomcat-users.xml in all managed nodes.
Change port number of tomcat from 8080 to 9090. Restart the tomcat8 service.
Finally i want to check url is reachable or not.

```
$  ansible all -m  apt -a 'name=tomcat8 state=present' -b

$  ansible all -m  copy  -a  'src=tomcat-users.xml  dest=/etc/tomcat8'  -b
$ ansible all  -m replace  -a 'regexp=8080 replace=9090 path=/etc/tomcat8/server.xml'  -b
$  ansible all   -m service    -a 'name=tomcat8  state=restarted'  -b
```

To check tomcat is running individually on all servers,
take the private ip of all nodes
172.31.11.96
172.31.6.207
172.31.12.138

```
$ ansible all -m   uri  -a 'url=http://172.31.11.96:9090  status=200'
```
It returns status as 200

Similarly check the other two nodes
```
$ ansible all -m   uri  -a 'url=http://172.31.6.207:9090  status=200'
$ ansible all -m   uri  -a 'url=http://172.31.12.138:9090  status=200'
```

what is Play books

Notes:
Adhoc commands are capable of working only on one module and one set of arguments.
When we want to perform complex configuration management activities,
adhoc commands will be difficult to manage.

In such scenarios, we use play books.
Play book is combination of plays.
Each play is designed to do some activity on the managed nodes.
These plays are created to work on single host or a group of hosts or all the hosts.

The main advantage of play books is reusability.
Play books are created using yaml files.

```
$ mkdir  playbooks
$ cd playbooks
$ vim playbook1.yml
INSERT   mode

---
```

```
- name: Install git and clone a remote repository
  hosts: all
  tasks:
    - name: Install git
      apt:
       name: git
       state: present
       update_cache: yes
    - name: clone remote git repository
      git:
        repo: https://github.com/sunilkumark11/git-9am-batch.git
        dest: /home/ubuntu/newgit
...
```

To check the syntax:

```
$ ansible-playbook  playbook1.yml  --syntax-check
```

(Do not use tab when creating yml file)

To run the playbook

```
$ ansible-playbook  playbook1.yml  -b
```

++++++++++++++++++++++++++++++++

Play books

Notes:
Adhoc commands are capable of working only on one module and one set of arguments.
When we want to perform complex configuration management activities,
adhoc commands will be difficult to manage.

In such scenarios, we use play books.
Play book is combination of plays.
Each play is designed to do some activity on the managed nodes.
These plays are created to work on single host or a group of hosts or all the hosts.

The main advantage of play books is reusability.
Play books are created using yaml files.

```
$ mkdir  playbooks
$ cd playbooks
$ vim playbook1.yml
INSERT   mode
```

```
---
- name: Install git and clone a remote repository
  hosts: all
  tasks:
    - name: Install git
```

```
      apt:
       name: git
       state: present
       update_cache: yes
    - name: clone remote git repository
      git:
        repo: https://github.com/sunilkumark11/git-9am-batch.git
        dest: /home/ubuntu/newgit
...
```

To check the syntax:

```
$ ansible-playbook  playbook1.yml  --syntax-check
```

(Do not use tab when creating yml file)

To run the playbook

```
$ ansible-playbook  playbook1.yml  -b
```

++++++++++++++++++++++++++++

2nd example on playbook

Create user on all managed nodes and I want to copy passwd file.

```
$ vim playbook2.yml
```

```
---
- name: Create user and copy passwd file
  hosts: all
  tasks:
          - name: User creation
            user:
             name: kiran
             password: sunilsunil
             uid: 6779
             home: /home/kiran
          - name: Copy password into users home dir
            copy:
             src: /etc/passwd
             dest: /home/kiran
...
```

*****importent point******
----
in the above playbook when we are creating the user it is not a good pratice like to give the password directly.
we have to encrypt the password is the good pratice by making use of command called (openssl passwd)
after that it will ask the password to encrypt
----->openssl passwd
then it will ask the password like------->password:
after entering the password it will ask once again for verification like---------->verifying - password:
after validating the password it will encrypt the password,this encrypted password we can use inside the playbook f

or the security purpose.

Save and quit
```
$
```

Check the syntax:
```
$ ansible-playbook  playbook2.yml  --syntax-check
```

To run
```
$ ansible-playbook  playbook2.yml  -b
```

TO check user is created in managed nodes:
```
$ ssh  172.31.2.173
$ vim /etc/passwd
```

To check if passwd file is copied to /home/kiran
```
$  cd /home/kiran
$ ls
$ exit
```

Ex 3: Playbook to configure tomcat8   (earlier example)

1st uninstall tomcat
```
$ ansible  all  -m  apt -a 'name=tomcat8 state=absent  purge=yes'  -b
```

```
$ vim playbook3.yml

---
- name: Configure  tomcat8
  hosts: all
  tasks:
   - name: Install tomcat8
     apt:
      name: tomcat8
      state: present
   - name: copy tomcat-users.xml file
     copy:
      src:  /home/ubuntu/tomcat-users.xml
      dest: /etc/tomcat8
   - name: change port of tomcat from 8080 to 9090
     replace:
      regexp: 8080
      replace: 9090
      path: /etc/tomcat8/server.xml
   - name: restart tomcat8
     service:
      name: tomcat8
      state: restarted
```

Have you done anything with AD ?
==============================

To manage Active Directory in real-time using Ansible, you'll need to ensure your Ansible control machine can interact with your AD infrastructure immediately and perform tasks as soon as they are needed. Here's how you can set up and execute such tasks in real-time:

Prerequisites
----

1. Ansible Control Node: Ensure Ansible is installed and configured on your control node.
2. Windows Remote Management (WinRM): Configure WinRM on your Windows servers to allow Ansible to communicate with them.
3. AD Modules: Install required Ansible modules for AD management (win_domain_user, win_domain_group, win_domain_membership).
4. Python Modules: Install pywinrm on your Ansible control machine to allow Ansible to communicate with Windows servers.

Step-by-Step Guide

1.Inventory Setup: Configure your inventory to include your Windows servers and domain controller.

2. Real-Time Triggering: Use Ansible Tower/AWX or cron jobs/scheduled tasks on the control node to trigger playbooks in response to specific events.

Note:-   Ansible uses inventory files to manage the list of servers it will manage. These files can be written in INI or YAML format.

Example Inventory File (`hosts.ini`)

Let's create an ` hosts.ini ` file:

```
[windows_servers]
server1.example.com
server2.example.com

[domain_controllers]
dc1.example.com
```

Note:

Here, [windows_servers] and [domain_controllers] are groups containing your Windows servers and domain controllers, respectively.

Example Playbooks
----

1. Create an AD User:- Create a playbook to create an AD user (create_ad_user.yml):

```
---
- name: Create AD user
  hosts: domain_controllers
  gather_facts: no
  tasks:
    - name: Create a new AD user
      win_domain_user:
        name: johndoe
        password: Pa$$w0rd123
        given_name: John
        surname: Doe
        user_ou: "OU=Users,DC=example,DC=com"
        state: present
        enabled: yes
      register: user_creation

    - name: Debug user creation result
      debug:
        var: user_creation
```

```
   - name: check url response of server 1
     uri:
      url: http://172.31.7.134:9090
   - name:   check url response of server 2
     uri:
      url: http://172.31.3.46:9090
...
```

```
$ ansible-playbook  playbook3.yml  --syntax-check
$ ansible-playbook  playbook3.yml  -b
```

++++++++++++++++++++++++

Requirment:
Install apache2 in all managed nodes, Place our own content in default homepage

```
$ cd playbooks
$ vim playbook4.yml
```

```
---
- name: configuring apache2
  hosts: all
  tasks:
   - name: Install apache2
     apt:
      name: apache2
      state: present
```

Save and quit

```
$ ansible-playbook  playbook4.yml  -b
```

To check apache2 is installed

```
$ ssh 172.31.12.239
```

(Homepage of apache2 is present in /var/www/html)

```
$ cd  /var/www/html
$ ls
```

we get index.html (this html file is default homepage of apache)
Editing the index.html page
This is possible using copy module.

```
$ exit
$ vim playbook4.yml
```

```
- name: configuring apache2
  hosts: all
  tasks:
   - name: Install apache2
```

2. Add a User to an AD Group:

Create a playbook to add a user to an AD group (` add_user_to_group.yml`):

```
---
- name: Add AD user to group
  hosts: domain_controllers
  gather_facts: no
  tasks:
    - name: Add user to AD group
      win_domain_group_membership:
        name: 'Domain Admins'
        members:
          - johndoe
        state: present
      register: group_membership

    - name: Debug group membership result
      debug:
        var: group_membership
```

3. Join a Machine to an AD Domain:

Create a playbook to join a Windows machine to an AD domain (`join_domain.yml`):

```
---
- name: Join machine to AD domain
  hosts: windows_servers
  gather_facts: no
  tasks:
    - name: Ensure the machine is joined to the domain
      win_domain_membership:
        dns_domain_name: example.com
        domain_admin_user: adminuser
        domain_admin_password: P@ssw0rd
        state: domain
      register: domain_join

    - name: Reboot if domain join required it
      win_reboot:
      when: domain_join.reboot_required
```

Real-Time Triggering

To trigger playbooks in real-time, you have two primary options:

1. Using Ansible Tower/AWX
2. Using Cron Jobs/Scheduled Tasks

Option 1: Using Ansible Tower/AWX

Ansible Tower (commercial) and AWX (open-source) provide a web-based UI and REST API for managing Ansible playbooks and inventories.

Step-by-Step Guide for Ansible Tower/AWX

Note:-->> Install Ansible Tower/AWX: Follow the official installation guide for Ansible Tower or AWX installation guide for AWX.

>> Create an Inventory:
1) Go to the Inventories section.
2) Click on the Add button to create a new inventory.
3) Add your Windows servers and domain controllers to this inventory.

>> Create Credentials:
1) Navigate to the Credentials section.
2) Add the necessary credentials for connecting to your Windows
3) servers (e.g., SSH keys, WinRM credentials).

```
    apt:
     name: apache2
     state: present
   - name: Edit index.html file
    copy:
     content: "Welcome to Playbooks\n"
     dest: /var/www/html/index.html
```

save and quit

```
$ ansible-playbook  playbook4.yml  -b
```

++++++++++++++++++++++

How to open url in terminal?
by using elinks
Ex:

```
$  elinks http://google.com
```

We get error (elinks not found)

Let's install elinks

```
$ sudo apt-get install -y elinks
```

Now run the command

```
$  elinks http://google.com
```

Now we want to look at index.html file in managed nodes

```
$ elinks http://15.207.99.5
```

After editing the index.html file, i need to restart the service and check the url response

```
$ vim playbook4.yml

---
- name: configuring apache2
  hosts: all
  tasks:
   - name: Install apache2
    apt:
     name: apache2
     state: present
   - name: Edit index.html file
    copy:
     content: "Welcome to playbooks\n"
     dest: /var/www/html/index.html
   - name: Restart apache2
    service:
     name: apache2
     state: restarted
   - name: check url response of server1
    uri:
     url: http://172.31.7.134
     status: 200
```

>> Create a Job Template:

1) Go to the Templates section.
2) Click on the Add button and select Job Template.
3) Configure the job template with the following settings:

   Name: Name of your job template.
   Inventory: Select the inventory created in step 2.
   Project: Select or create a project that contains your playbooks.
   Playbook: Select the playbook you want to execute.
   Credentials: Select the credentials created in step 3.

>> Create Schedules or Webhooks:

1) Schedules: You can schedule the job template to run at specific intervals (e.g., daily, weekly).

   i) Go to the Schedules tab of the job template.
   ii) Click on the Add button to create a new schedule.
   iii) Define the frequency and time for the job to run

Example Job Template Configuration

Option 2: Using Cron Jobs/Scheduled Tasks

For a simpler setup without a web UI, you can use cron jobs (Linux) or Task Scheduler (Windows) on the Ansible control node to trigger playbooks.

Step-by-Step Guide for Cron Jobs

Open Crontab File:

>> Open the crontab file for editing:

```
crontab -e
```

Add a Cron Job Entry:

>> Add an entry to the crontab file to run the playbook at the desired interval (e.g., every 5 minutes):

```
*/5 * * * * ansible-playbook -i /path/to/hosts.ini /path/to/playbook.yml
```

```
   - name: check url response of server2
     uri:
      url: http://172.31.3.46
      status: 200
   - name: check url response of server3
     uri:
      url: http://172.31.2.140
      status: 200
...
```

```
 ansible-playbook  playbook4.yml  -b
```

Notes:
Ex: Ansible playbook for configure apache2

++
Creating reusable playbooks using variables
3 Types of variables
1) Global scope variables   (highest priority) - we pass values from command prompt
2) Host scope variables
3) play scope variables   (least priority)

Ex of Global scope variables

```
$ vim playbook5.yml

---
- name: Install software packages
  hosts: all
  tasks:
   - name: Install/uninstall/update etc
     apt:
      name: tree
      state: present
      update_cache: yes
...
```

If we run the above play book 10 times, what happens? tree package will install 10 times.
The above play book is not reusable.

we make small changes to the above code

```
$ vim playbook5.yml

---
- name: Install software packages
  hosts: all
  tasks:
   - name: Install/uninstall/update etc
     apt:
```

```
     name: "{{a}}"
     state: "{{b}}"
     update_cache: "{{c}}"
...
```

To run the playbook by passing values to the variables

```
$ ansible-playbook  playbook5.yml  --extra-vars "a=git b=absent c=no"  -b
```

(The above command will uninstall git from all nodes)

Run the same playbook with diffrent values

```
$ ansible-playbook  playbook5.yml  --extra-vars "a=tree b=present c=no"  -b
```

+++++++++++++++++++

Before going to host scope variables,
lets discuss play scope variables

Playscope variables are defined within the playbook and they can effect only in one single play.

Ex:

```
$ vim playbook7.yml

---
- name: Using play scope variable
  hosts: all
  vars:
   - a: tomcat8
   - b: present
   - c: no
  tasks:
   - name: Install tomcat8
     apt:
      name: "{{a}}"
      state: "{{b}}"
      update_cache: "{{c}}"
...

$ ansible-playbook  playbook7.yml  -b
```

(It will install tomcat8)

We can run by using extra vars from command line

```
$ ansible-playbook  playbook7.yml    --extra-vars "a=tree b=present c=no"  -b
```

How would you automation using ansible if every server get 500 GB ?

If you need to ensure that every server in your infrastructure gets a 500 GB disk, you can create an Ansible playbook that performs the necessary steps.

Example:-

In this playbook: - ============>>>>
----

1. Check if the disk is present: The lsblk command checks if an additional disk (sdb) is already present.

2. Create a 500 GB disk: If the disk is not present, it uses the dd command to create a disk file of 500 GB.

3. Create a filesystem: It formats the disk with ext4.

4. Mount the disk: It mounts the disk to /mnt/data.

5. Disk Naming: Adjust the disk naming and paths (/path/to/diskfile, sdb, /mnt/data) as per your environment.

6. Disk Naming: Adjust the disk naming and paths (/path/to/diskfile, sdb, /mnt/data) as per your environment.

```
---
- name: Ensure all servers have a 500 GB disk
  hosts: all_servers
  become: yes

  tasks:
    - name: Check if additional disk is present
      command: lsblk | grep sdb
      register: disk_check
      ignore_errors: true

    - name: Add new 500 GB disk if not present
      block:
        - name: Create a 500 GB disk
          shell: dd if=/dev/zero of=/path/to/diskfile bs=1G count=500
          when: disk_check.rc != 0

        - name: Create a filesystem on the new disk
          filesystem:
            fstype: ext4
            dev: /path/to/diskfile

        - name: Mount the new disk
          mount:
            path: /mnt/data
            src: /path/to/diskfile
            fstype: ext4
            opts: defaults
            state: mounted

      when: disk_check.rc != 0
```

7. Idempotence: Ensure the playbook can be run multiple times without causing issues, which is handled by the when condition checking the result of disk_check.

The above command will install tree because global scope variables have higher priority

Notes:
Playscope variables
These variables are definied at level of individual plays and they can effect only one play.

Ex:

```
---
- name: Using play scope variable
  hosts: all
  vars:
   - a: tomcat8
   - b: present
   - c: no
  tasks:
   - name: Install tomcat8
     apt:
      name: "{{a}}"
      state: "{{b}}"
      update_cache: "{{c}}"
...
```

Note: The above playbook works like a template, who's default behaviour is to install tomcat8
But, we can by pass that behaviour and make it work in some other software by passing the variables as extra vars

```
$ ansible-playbook  playbook7.yml  -b  --extra-vars "a=tree b=present c=no"  -b
```

The above command will install tree because global scope variables have higher priority

Notes:
Playscope variables
These variables are definied at level of individual plays and they can effect only one play.

Ex:

```
---
- name: Using play scope variable
  hosts: all
  vars:
   - a: tomcat8
   - b: present
   - c: no
  tasks:
   - name: Install tomcat8
     apt:
```

```
    name: "{{a}}"
    state: "{{b}}"
    update_cache: "{{c}}"
...
```

Note: The above playbook works like a template, who's default behaviour is to install tomcat8
But, we can by pass that behaviour and make it work in some other software by passing the variables as extra vars

++++++++++++++++++++

++
Today we will discuss about host scope variables
Lets create one more managed node.
So, we will have 1 controller 4 nodes.
In step 6 -- Add rule -- All Traffic -- Anywhere

Check the version in the new node

```
$ python3 --version
```

We need to downgrade the machines from python3 to Python2
To downgrade

```
$ sudo apt-get update
$ sudo apt-get dist-upgrade ( It will point to  older apt repository  where python2 is available)

$  sudo apt-get install -y python2.7 python-pip
```

Now check the version of python

```
$ python --version
```

Establish password less ssh connection

```
$ sudo passwd ubuntu
```

(lets give the password as ubuntu only)

```
$ sudo vim /etc/ssh/sshd_config
```

change
PasswordAuthentication yes
Save and QUIT

```
$ sudo service ssh restart
$ exit
```

++++++++++++++++
Now, Connect to controller
Now , We need to generate ssh connections

```
$ ssh-keygen
```

Now copy the key to managed nodes

```
$ ssh-copy-id ubuntu@172.31.6.241  ( private Ip of server4 )
```

+++++++++++

Now, we need to add the information of managed nodes in the inventory file.
Location of inventory file   /etc/ansible

```
$ cd /etc/ansible
$ ls
$ sudo vim hosts
```
insert the private ip addresss of 4th server
save and quit

```
$ ansible all  -a  'ls  -la'    ( you will get the list of the files in all managed nodes )
```

++++++++++++++++++
We can do grouping using [groupname]

Ex:

To do grouping

```
$ sudo vim hosts

[webserver]
172.31.11.96
172.31.6.207
[appserver]
172.31.12.138
[dbserver]
172.31.31.161
```

+++++++++++++++++++

```
$ ansible appserver  -a 'free'    ( It runs on one machine 172.31.12.138)

$ ansible webserver  -a 'free'   ( It runs on two machines )

$ ansible all  -a 'free'
```

+++++++++++++++++++++++
We can perform grouping on groups

```
$ sudo vim hosts

[webserver]
172.31.11.96
172.31.6.207
[appserver]
172.31.12.138
[dbserver]
172.31.31.161
[india:children]
webserver
dbserver
```

How can you optimize the execution of Ansible playbooks for a large number of servers?

Ansible can execute tasks in parallel, and I would leverage this feature to optimize playbook execution. I can configure the forks setting in the Ansible configuration file or use the -f option during playbook execution to specify the number of parallel processes. This allows tasks to run concurrently across multiple servers, improving overall efficiency.

```
ansible.cfg
        [defaults]
        forks = 20
```

In Ansible, `changed_when` and `failed_when` are used to control task outcomes based on custom conditions.

changed_when: false

You want to run a task to check if a file exists, but you don't want Ansible to consider the system "changed" just because the task ran. For example, you are checking for a log file.

EX:-

```
- name: Check if the log file exists
  stat:
    path: /var/log/myapp.log
  register: log_file

- name: Notify if the log file exists
  debug:
    msg: "Log file exists."
  when: log_file.stat.exists
  changed_when: false
```

```
$ ansible india  -a 'free'
```

Grouping in inventory file

```
$ sudo  vim  /etc/ansible/hosts

[webserver]
172.31.11.96
172.31.6.207
[appserver]
172.31.12.138
[dbserver]
172.31.31.161
[india:children]
webserver
dbserver
```

Host scope variables

These variables are classified into 2 types
1) Variables to work on group of hosts
2) Variables to work on single hosts

Variables to work on group of hosts

These variables are designed to work on group of hosts.
They are definined in a folder called group_vars
This group_vars folder should be presnent in the same folder where all the playbooks are present.
In this group_vars folder, we should create a file who's name is same as group_name in Inventory file.
In this file we create variables.

Varible which works on group of hosts

```
$ cd  ( enter)
$ cd playbooks
$ ls
```

Varibles which work in group of hosts are divided into two types
1) Variables which work in group of machines
2) Variables which work on one machine

Variables which work in group of machines

```
playbooks$ mkdir group_vars
```

Note: group_vars folder should be present in the same location of playbook files.

```
$ cd group_vars
$  vim webserver
```

failed_when

You want to run a command and only mark the task as failed if a specific error appears in the command's output. For example, running a script and checking for a critical error.

EX:-

```
- name: Run the script
  shell: /usr/local/bin/myscript.sh
  register: script_output
  ignore_errors: yes

- name: Check script output for critical error
  debug:
    msg: "The script ran successfully."
  failed_when: "'CRITICAL ERROR' in script_output.stderr"
```

The shell module runs /usr/local/bin/myscript.sh and saves the result in script_output.

ignore_errors: yes allows the playbook to continue even if the script fails.

The debug task prints a success message.

failed_when: "'CRITICAL ERROR' in script_output.stderr" checks if " CRITICAL ERROR" is in the stderr output of the script. If it is, Ansible will mark this task as failed.

To copy a file from Server-A to Server-B using an Ansible playbook, you can follow these steps:

Configure Inventory: First, define both Server-A and Server-B in your Ansible inventory.
Write Playbook: The playbook will first fetch the file from Server-A using the fetch module, and then copy it to Server-B using the copy or template module.

Inventory (hosts.ini):

```
[servers]
server-a ansible_host=server-a-ip
server-b ansible_host=server-b-ip
```

Ansible Playbook (copy_file.yml):

```
- name: Copy file from Server-A to Server-B
  hosts: server-a
  tasks:
    - name: Fetch file from Server-A
      fetch:
        src: /path/to/source/file/on/server-a   # Path of file on Server-A
        dest: /tmp/local_copy                   # Local directory on Ansible control machine
        flat: yes                               # Ensures it copies without creating subdirectories

- name: Copy the file to Server-B
  hosts: server-b
  tasks:
    - name: Copy file from Ansible control machine to Server-B
      copy:
        src: /tmp/local_copy/file               # File that was fetched from Server-A
        dest: /path/to/destination/on/server-b  # Destination path on Server-B
```

Explanation:
Inventory File: server-a and server-b are defined in the hosts.ini inventory file, where you replace server-a-ip and server-b-ip with the actual IP addresses of the servers.

Playbook (copy_file.yml):First Play: This targets server-a and uses the fetch module to pull the file from server-a to the Ansible control machine (/tmp/local_copy/).
Second Play: This targets server-b and uses the copy module to send the fetched file to server-b.

```
ansible-playbook -i hosts.ini copy_file.yml
```

```
a: Prakash
b: logiclabs
c: /home/Prakash
d: 67809
e: /bin/bash
```

Save and Quit

```
$ cd ..
playbooks$ vim playbook8.yml
```

```
---
- name: Using host scope variables
  hosts: webserver
  tasks:
   - name: User creation
     user:
      name: "{{a}}"
      password: "{{b}}"
      home: "{{c}}"
      uid: "{{d}}"
      shell: "{{e}}"
...
```

save and quit

TO run the playbook

```
$ ansible-playbook playbook8.yml  -b  ( It runs on two machines)
```

++++++++++++++++++++++++++++

Lets add few more variables

```
$ cd group_vars
$ vim webserver
```

```
a: Prakash
b: durgasoft
c: /home/Prakash
d: 67809
e: /bin/bash
f: tree
g: present
h: no
```

save and quit

```
$ cd ..
$ vim playbook9.yml
```

```
---
- name: Using host scope variables
```

TAGS:-

Tags in Ansible, allows you to run specific parts of your playbooks. They help in managing and organizing playbooks, especially when dealing with large and complex deployments.

Scenario: Deploying a Web Application

Imagine you have a playbook that sets up a web server, deploys a web application, and configures the database. You might want to run only the web server setup during development and skip the database configuration, or vice versa.

```
---
- name: Setup web server
  hosts: webservers
  become: yes
  tasks:
    - name: Install Nginx
      apt:
        name: nginx
        state: present
      tags: webserver

    - name: Start Nginx
      service:
        name: nginx
        state: started
      tags: webserver

- name: Deploy web application
  hosts: webservers
  become: yes
  tasks:
    - name: Copy application files
      copy:
        src: /path/to/app/
        dest: /var/www/html/
      tags: app

- name: Configure database
  hosts: dbservers
  become: yes
  tasks:
    - name: Install MySQL
      apt:
        name: mysql-server
        state: present
      tags: database

    - name: Create application database
      mysql_db:
        name: app_db
        state: present
      tags: database
```

Run Only Web Server Tasks:

```
ansible-playbook playbook.yml --tags "webserver"
```

Run Only Application Deployment Tasks:

```
ansible-playbook playbook.yml --tags "app"
```

Skip Database Configuration:

```
ansible-playbook playbook.yml --skip-tags "database"
```

```
  hosts: webserver
  tasks:
   - name: Install software
     apt:
      name: "{{f}}"
      state: "{{g}}"
      update_cache: "{{h}}"
...
```

```
$ ansible-playbook  playbook9.yml  -b
```

+++++++++++++++++++++++++

Variables to work on single hosts

Variables to work on single hosts
These variables are designed on single machine.
Thet are created in folder called host_wars
This host_wars folder should be created in the same location of where the playbooks are present.

```
playbooks$ mkdir host_vars
$ cd host_vars
$ vim 172.31.6.241        ( 172.31.6.241  private Ip of server4 )

a: firewalld
b: present
c: yes

save and quit
$ cd ..
$ vim playbook10.yml

---
- name: Use host scope variables
  hosts: 172.31.6.241
  tasks:
   - name: Install firewall
     apt:
      name: "{{a}}"
      state: "{{b}}"
      update_cache: "{{c}}"
...

save and quit

$ ansible-playbook  playbook10.yml  -b
```

+++

Implementing loops

Run Web Server and Application Tasks:

```
ansible-playbook playbook.yml --tags "webserver,app"
```

Advantages of Using Tags
====================

Selective Execution: Run only the parts of the playbook that you need, which saves time and resources.

Flexibility: Easily test specific parts of your setup without running the entire playbook.

Maintainability: Simplifies the management of large playbooks by logically grouping related tasks.

We need to check a file.. If that file is not present.. A file should be created with specific permissions

```
- name: Ensure a specific file exists with the correct permissions
  hosts: all
  become: yes
  vars:
    file_path: /path/to/your/file
    file_owner: root
    file_group: root
    file_mode: '0644'
  tasks:
    - name: Check if the file exists
      stat:
        path: "{{ file_path }}"
      register: file_stat

    - name: Create the file with specific permissions if it does not exist
      file:
        path: "{{ file_path }}"
        state: touch
        owner: "{{ file_owner }}"
        group: "{{ file_group }}"
        mode: "{{ file_mode }}"
      when: not file_stat.stat.exists
```

ensure that the HTTPD service (or its equivalent on other platforms) is running on Windows, Ubuntu, and RHEL, you can use Ansible's ansible_facts and platform-specific tasks. Below is an example playbook that checks and ensures the service is running, adapting to different operating systems:

```
---
- name: Ensure HTTPD service is running on different platforms
  hosts: all
  become: yes
  gather_facts: yes

  vars:
    services:
      - { name: "httpd", os_family: "RedHat" }
      - { name: "apache2", os_family: "Debian" }
      - { name: "W3SVC", os_family: "Windows" }

  tasks:
    - name: Ensure the HTTPD service is running
      block:
        - name: Set service name based on OS
          set_fact:
            service_name: "{{ item.name }}"
          when: ansible_facts['os_family'] == item.os_family
          loop: "{{ services }}"

        - name: Start and enable the HTTPD service on Linux
          service:
            name: "{{ service_name }}"
            state: started
            enabled: yes
          when: ansible_facts['os_family'] in ['RedHat', 'Debian']

        - name: Start and enable the HTTPD service on Windows
          win_service:
            name: "{{ service_name }}"
            start_mode: auto
            state: started
          when: ansible_facts['os_family'] == 'Windows'

      when: service_name is defined
```

Notes: Modules in ansible can be executed multiple times using loops.

```
$ vim playbook11.yml

- name: Install software packages
  hosts: webserver
  tasks:
   - name: Install software
     apt:
      name: "{{item}}"
      state: present
      update_cache: no
     with_items:
      - tree
      - git
      - default-jdk
      - apache2
...

$ ansible-playbook  playbook11.yml  -b
```

Ex: Playbook to install diffrent s/w packages

```
$ vim playbook11.yml

- name: Install software packages
  hosts: webserver
  tasks:
   - name: Install software
     apt:
      name: "{{item}}"
      state: present
      update_cache: no
     with_items:
      - tree
      - git
      - default-jdk
      - apache2
...
```

++++++++++++++++++++++++

Requirement:
Tree needs to be installed
Git needs to be unintalled
jdk needs to be updated
apache needs to be installed and update cache

```
$ cd playbooks
$ vim playbook12.yml
```

OR

```
- name: Ensure web server services are running
  hosts: all
  become: yes
  vars:
    services:
      Debian: apache2
      RedHat: httpd
      Windows: Tomcat8
  tasks:
    - name: Gather service facts
      service_facts:

    - name: Ensure web server service is running on different platforms
      block:
        - name: Identify OS
          ansible.builtin.setup:
            filter: ansible_os_family

        - name: Ensure web server service is running on Debian/Ubuntu
          block:
            - name: Check if the service is running
              shell: systemctl is-active --quiet {{ services.Debian }}
              register: service_status
              ignore_errors: yes

            - name: Restart the service if it is not running
              service:
                name: "{{ services.Debian }}"
                state: restarted
              when: service_status.rc != 0
              notify: Restart {{ services.Debian }}
          when: ansible_os_family == "Debian"

        - name: Ensure web server service is running on RHEL/CentOS
          block:
            - name: Check if the service is running
              shell: systemctl is-active --quiet {{ services.RedHat }}
              register: service_status
              ignore_errors: yes

            - name: Restart the service if it is not running
              service:
                name: "{{ services.RedHat }}"
                state: restarted
              when: service_status.rc != 0
              notify: Restart {{ services.RedHat }}
          when: ansible_os_family == "RedHat"

        - name: Ensure Apache Tomcat 8 is running on Windows
          block:
            - name: Check if the Tomcat8 service is running
              win_shell: Get-Service -Name {{ services.Windows }} | Select-Object -ExpandProperty
Status
              register: service_status

            - name: Start the Tomcat8 service if it is not running
              win_service:
                name: "{{ services.Windows }}"
                state: started
              when: service_status.stdout != 'Running'
              notify: Restart {{ services.Windows }}
          when: ansible_os_family == "Windows"

  handlers:
    - name: Restart {{ services.Debian }}
      service:
        name: "{{ services.Debian }}"
        state: restarted

    - name: Restart {{ services.RedHat }}
      service:
        name: "{{ services.RedHat }}"
        state: restarted

    - name: Restart {{ services.Windows }}
      win_service:
        name: "{{ services.Windows }}"
        state: restarted
```

```
---
- name: Install software packages
  hosts: webserver
  tasks:
   - name: Install software
     apt:
      name: "{{item.a}}"
      state: "{{item.b}}"
      update_cache: "{{item.c}}"
     with_items:
      - {a: tree,b: present,c: no}
      - {a: git,b: absent,c: no}
      - {a: default-jdk,b: absent,c: no}
      - {a: apache2,b: present,c: yes}
...
```

save and quit

Explanation:
==========
Variables (vars):

Define a dictionary services with keys for each OS family (Debian, RedHat, Windows) and their respective service names.

Gather Service Facts:

Use service_facts to gather information about the services on the hosts.

Blocks:

Use blocks to group tasks for different OS families.
Each block contains tasks to check the service status and restart it if not running.

Handlers:

Define handlers to restart the services. Handlers are triggered using the notify directive when a task requires it (e.g., when a service needs restarting).
Handlers are named dynamically using the service names from the vars.

Service Management:

For Debian/Ubuntu and RHEL/CentOS, use systemctl to check and manage the services.
For Windows, use PowerShell to check the service status and win_service to manage the service.

```
$ ansible-playbook  playbook12.yml  -b
```

++++++++++++++++++++++
Ex: For working on multiple modules with multiple with_items.

Requirement: To create multiple users and files/directories in user's home directories.

```
$ vim playbook13.yml
---

---
- name: Create users and create files/dir in users home dir
  hosts: all
  tasks:
   - name: Create multiple users
     user:
      name: "{{item.a}}"
      password: "{{item.b}}"
      home: "{{item.c}}"
     with_items:
      - {a: Farhan,b: durgasoft,c: /home/Farhan}
      - {a: Ravi,b: durgasoft,c: /home/ubuntu/Ravi}
   - name: creating files and directories in users home dir
     file:
      name: "{{item.a}}"
      state: "{{item.b}}"
     with_items:
      - {a: /home/Farhan/file1,b: touch}
      - {a: /home/ubuntu/Ravi/dir1,b: directory}
...
```

```
# vars.yml
users:
  - name: Farhan
    password: durgasoft
    home: /home/Farhan
  - name: Ravi
    password: durgasoft
    home: /home/ubuntu/Ravi

files_and_dirs:
  - path: /home/Farhan/file1
    state: touch
    owner: Farhan
    group: Farhan
    mode: '0644'
  - path: /home/ubuntu/Ravi/dir1
    state: directory
    owner: Ravi
    group: Ravi
    mode: '0755'
```

save and quit

```
$ ansible-playbook  playbook13.yml  -b
```

To check , user is created or not?

```
$ ssh 172.31.11.96
$ vim /etc/passwd
```

TO check files and dir are created or not

```
$ cd /home/Farhan
$ ls     ( we can see the file)
$ cd
$ pwd
$ cd Ravi
$ ls  ( we can see the dir )
$ exit
```

```
---
- name: Create users and create files/dir in users home dir
  hosts: all
  become: yes

  vars_files:
    - vars.yml

  tasks:
    - name: Create multiple users
      user:
        name: "{{ item.name }}"
        password: "{{ item.password }}"
        home: "{{ item.home }}"
        state: present
      loop: "{{ users }}"

    - name: Create files and directories in users' home dirs
      file:
        path: "{{ item.path }}"
        state: "{{ item.state }}"
        owner: "{{ item.owner }}"
        group: "{{ item.group }}"
        mode: "{{ item.mode }}"
      loop: "{{ files_and_dirs }}"
```

+++++++++++++++++
Handlers

Handler is a piece of code which is executed, if some other module is executed successfully and it has made some changes.

Handlers are always executed only after all the tasks are executed.
Handlers are executed in the order that are mentioned in the handler section, and not in the order they are called in the tasks section.
Even if handler is called multiple times in the tasks section, it will be executed only once.

Requirement:

```
$ vim playbook14.yml

---
- name: Confugure apache2 using handlers
  hosts: all
  tasks:
   - name: Install apache2
     apt:
      name: apache2
      state: present
   - name: Edit index.html file
     copy:
      content: "Logiclabs\n"
      dest: /var/www/html/index.html
     notify: Restart apache2
  handlers:
   - name: Restart apache2
     service:
      name: apache2
      state: restarted
...
```

```
---
- name: Ensure httpd service is running
  hosts: all
  become: yes

  tasks:
    - name: Gather service facts
      service_facts:

    - name: Check if the httpd service is running
      shell: systemctl is-active --quiet httpd
      register: service_status
      ignore_errors: yes

    - name: Notify handler to restart httpd if it is not running
      meta: flush_handlers
      notify: Restart httpd service
      when: service_status.rc != 0

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

Notify Handler to Restart httpd if Not Running:

The 'meta: flush_handlers' task ensures that handlers are run as soon as they are notified. This can be useful in cases where you want the handler to be executed immediately rather than at the end of the playbook.
The notify: Restart httpd service directive triggers the Restart httpd service handler if the condition 'when: service_status.rc != 0' is met (i.e., if the httpd service is not running).

```
$ ansible-playbook  playbook14.yml  -b
```

Note:
As editing the index.html file is successfull, handler is executed.

If you re run the playbook, handler is not executed.

Using service command:

```
- name: Check if the httpd service is running
  shell: service httpd status | grep -q "running"
  register: service_status
  ignore_errors: yes
```

Using ps command:

```
- name: Check if the httpd service is running
  shell: ps aux | grep -v grep | grep -q httpd
  register: service_status
  ignore_errors: yes
```

++++++++++++++++++++++
Error Handling

If any module fails in ansible,the execution of the playbook terminates over there.
When we know that certain module might fail, and still we want to continue playbook execution, we can use error h andling.
The section of code which might generate an error should be given in block section.
If it generates an error, the control comes to rescue section.
Always section is executed every time, irespective of whether the block is successfull or failure.

```
$ vim playbook15.yml

---
- name: Error handling
  hosts: all
  tasks:
   - block:
      - name: Install apache1
        apt:
         name: apache1
         state: present
     rescue:
      - name: Install apache2
        apt:
         name: apache2
         state: present
     always:
      - name: Check url response
        uri:
         url: "{{item}}"
        with_items:
         - http://172.31.7.134
         - http://172.31.3.46
         - http://172.31.2.140
         - http://172.31.6.241
...

$ ansible-playbook  playbook15.yml  -b
```

By using ansible how you can check wether the service is running or not, if it not running need to restart in a real time?

```
---
- name: Ensure httpd service is running
  hosts: all
  become: yes
  tasks:
    - name: Gather service facts
      service_facts:

    - name: Check if the httpd service is running and restart if not
      block:
        - name: Check if the httpd service is running
          shell: systemctl is-active --quiet httpd
          register: service_status
          ignore_errors: yes

        - name: Restart the httpd service if it is not running
          service:
            name: httpd
            state: restarted
          when: service_status.rc != 0
      when: "'httpd' in ansible_facts.services"
```

Gather Service Facts:

The 'service_facts' module is used to gather information about the services on the target hosts. This populates 'ansible_facts.services' with data about available services.

A 'block' is used to group related tasks and handle errors properly.

The 'shell' module executes the 'systemctl is-active --quiet httpd' command to check if the httpd service is active. The result is stored in service_status. The ignore_errors: yes directive allows the playbook to continue even if this command fails (i.e., if the service is not running).

The 'service' module restarts the httpd service if it is not running. The when condition 'service_status.rc != 0' checks the return code of the previous command to determine if the service needs to be restarted.

The outer when condition ensures this block only runs if the httpd service is found in 'ansible_facts.services'.

Note:

Using ansible-playbook with --retry

Ansible doesn't have a built-in retry mechanism for failed tasks in a straight forward way, but you can create a retry file after a playbook run. This feature allows you to rerun the failed tasks.

```
ansible-playbook playbook.yml --limit '@playbook.retry'
```

==

Ansible Vault

Ansible Vault is for security , if we have any confidential information in that playbook we can restrict the people to accessing by using the command called Ansible Vault

if our-requirement is to ----->restrict people from executing/seeing the playbook by making use of command called Ansible Vault
Ansible Vault is encrypts the playbook with a password
and it will ask to setup a password
only who knows the password can execute it other's can not execute it
when we are executeing the playbook ,   it decrypts and executes on the fly ,so here we no need to decrypt explecitly

And also we can change the password according to the reqirement by making use of keyword called rekey

----

syntax for encrypting the playbook
================================
ansible-vault encrypt playbookname.yml
--->it will ask the password

after that if we open the playbook ---->the content in the playbook is in encryped form

we can execute the palybook by making use of command
syntax:
----
ansible-playbook playbookname.yml --ask-vault-pass
----
then it will ask the password
after that it will execute

here if our requirement is to decrypt the playbook
syntax
----
ansible-vault decrypt playbookname.yml

https://docs.ansible.com/ansible/latest/inventory_guide/intro_patterns.html

Passing host group name in ansible-playbook command line

use --limit to specify the group like so

```
ansible-playbook -i /etc/ansible/hosts playbook.yml --limit <group-name>
```

you want to target inventory groups dynamically; a simple approach is to pass the hosts as variable in the playbook.

```
---
hosts: “{{target}}”
tasks:
```

While running the playbook pass the target variable like

```
ansible-playbook playbook.yml -i  /etc/ansible/hosts  -e target=<group-name>
```

after that it will ask the password------>and if the password is currect ------>it will decrypt

changeing the password to the playbook
----

ansible-vault rekey playbookname.yml

example:
----
ansible-vault encrypt sample.yml   ----->to encrypt
ansible-vault decrypt sample.yml   ------>to decrypt
ansible-vault rekey sample.yml   ------->changing the password to the yaml file
ansible-playbook sample.yml --ask-vault-pass   -----> to execute

----

what is roles ??

Roles automatically load related vars, files, tasks, handlers, and other Ansible artifacts based on a known file structu re. After you group your content in roles, you can easily reuse them and share them with other users.

first we have to create the playbook ---->under playbook we can create the roles
ex: playbookroles.yml

```
-hosts: webservers
 roles:
  - role123
  -role456
```

what is the difference between roles and playbooks?

Playbooks: These are YAML files where you write a list of tasks to perform on your servers directly.

Roles: These is a way to organize and reuse a set of related tasks and other resources in a structured format. They can be included in playbooks to keep things modular and reusable.

in ------------>/etc/ansible   we can find the directory roles, in----->/etc/ansible/roles ------------>
we can create the directorys called role123 and role456 ----->inside these roles we can create the following directory s

1)tasks
2)vars
3)defaults
4)templetes
5)files
6)handlers

To create a role in Ansible, you can use the ansible-galaxy command. This command-line tool helps manage roles in Ansible. The basic syntax for creating a new role is

```
ansible-galaxy init <role_name>
```

Replace <role_name> with the name of your desired role. For example, if you want to create a role named my_role, you would run: ansible-galaxy init my_role

This command will create a directory structure for the role, which includes various subdirectories and files to organize tasks, handlers, defaults, variables, files, templates, and metadata.

under every directory we can create the file called main.yml

firstly under tasks main.yml will be executed based on the values/variable we are given in the main.yml
remaining directorys main.yml files will be executed

variable precidance
----
playbook have the high/first presidance
vars have the second precidance
defaults have the last precidance

here we can define the variables in playbook,vars and defaults
if we are given the variables in all palybook,vars and defaults it will take the playbook variables what we are mencti on
if we give the variable in vars and defults it will take the values inside the vars because vars have the high precidanc e

if we are given the variable values only in the dafaults it will execute

Here's what the generated structure looks like:

```
my_role/
 README.md
 defaults
   main.yml
 files
 handlers
   main.yml
 meta
   main.yml
 tasks
   main.yml
 templates
 tests
   inventory
   test.yml
 vars
    main.yml
```

Ansible-Advantages
----
1) Provisioning of servers
The applications that should be installed on server can be done very quickly from a single centralized location.

2) Idempotent
Configuration management tools are used to bring the server to a particular state, called as desired state.
 If a server already in the desired state, configuration management tools will not reconfigure that server.