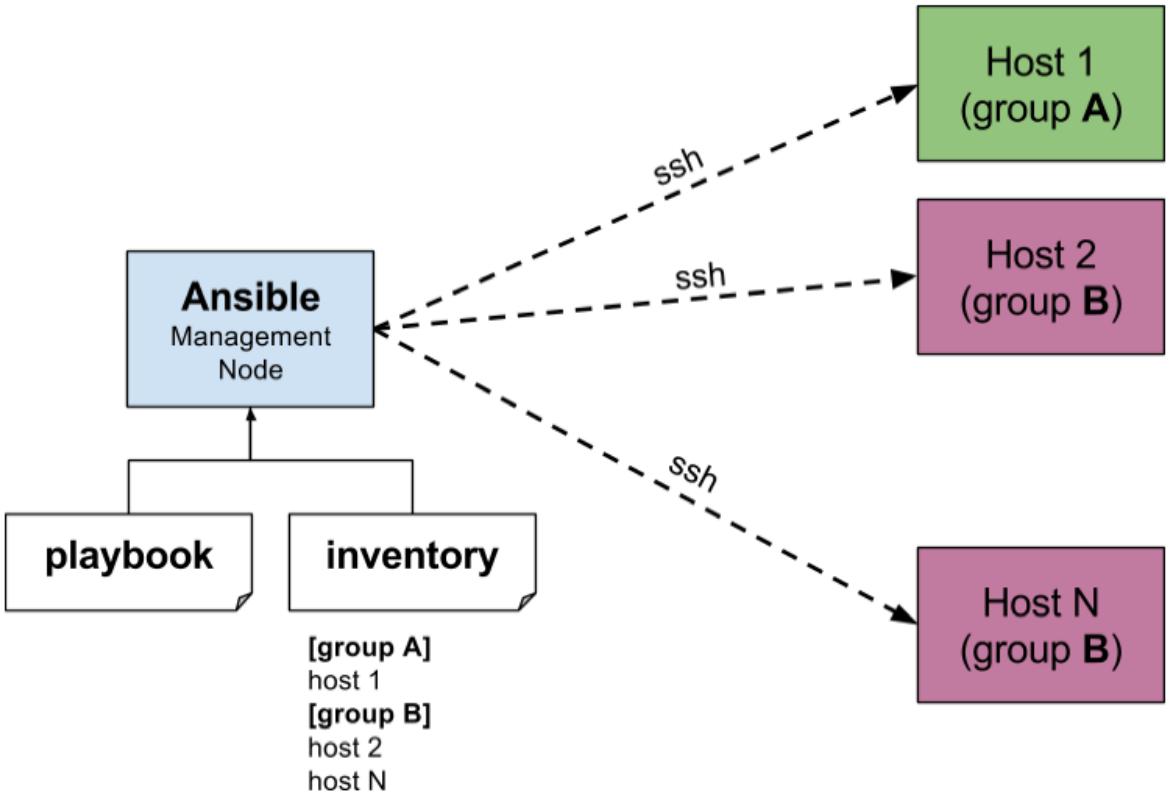
ANSIBLE DEVOPS

Ansible is an open-source automation tool used for configuration management, application deployment, task automation, and orchestration in IT environments. Developed by Red Hat, Ansible simplifies complex IT tasks by providing a platform-agnostic, agentless solution that uses a simple and human-readable language.

Key features of Ansible include:

- **Automation:** Ansible allows you to automate repetitive tasks and complex workflows, reducing the need for manual intervention. This includes tasks such as software installation, configuration management, and system updates.
- **Agentless Architecture:** Ansible operates in an agentless manner, meaning it doesn't require any software or agents to be installed on managed nodes.
- **Declarative Language:** Ansible uses a declarative language (YAML) to describe the desired state of the system. Instead of specifying step-by-step procedures, you define the intended configuration, and Ansible ensures the system matches that configuration.
- **Configuration Management:** Ansible excels in configuration management, ensuring consistency across multiple servers and environments.
- **Playbooks:** Ansible playbooks are scripts written in YAML that describe a set of tasks to be executed on remote nodes.
- **Community and Modules:** Ansible has a vibrant community that contributes to the development of modules, which are reusable units of work. These modules cover a wide range of tasks, from system administration to cloud provisioning.
- **Extensibility:** Ansible is extensible and can be integrated with other tools and systems, including version control systems (e.g., Git), continuous integration platforms, and monitoring solutions.
- **Idempotence:** Ansible promotes idempotent operations, meaning that applying the same operation multiple times has the same result as applying it once. This ensures predictability and reliability in configuration changes.

How Ansible Works?



Terminology

- Control Node: The machine from which Ansible is run. It's the system where you write and execute Ansible playbooks and commands.

- Managed Node: The machines that Ansible manages. Ansible communicates with these nodes to perform tasks and configurations. Managed nodes can be servers, network devices, or any machine with
- Inventory: An inventory is a list of managed nodes that Ansible can connect to and manage. It can be a simple text file, an INI file, or a dynamic script that dynamically generates the list of nodes.
- Playbook: A YAML file that defines a set of tasks to be executed on managed nodes. Playbooks are written in a human-readable format and describe the desired state of the system.
- Task: A single unit of work in Ansible. Tasks are defined in playbooks and represent actions to be taken on managed nodes, such as installing packages, copying files, or restarting services.
- Module: A module is a small piece of code that Ansible uses to perform a specific task on a managed node. Modules are executed on the remote machine and cover a wide range of functionalities, such as managing files, users, services, and more.
- Role: A reusable and self-contained collection of playbooks, variables, and tasks. Roles help organize and structure your Ansible projects, making it easier to manage and share common configurations.
- Facts: Ansible gathers information about managed nodes before executing tasks. These pieces of information are called facts and are accessible within playbooks. Facts include details about the system, network interfaces, and more.
- Play: A play is a set of tasks and configurations executed on a specific set of hosts defined in the playbook. A playbook can contain one or more plays.
- Handler: A handler is a special kind of task that only runs when notified by other tasks.
- Module Arguments: Parameters passed to Ansible modules to customize their behavior. Module arguments are specified in playbooks to configure the tasks.

LAB SETUP:

Pre-requisite
- 2 ubuntu VM (one for ansible-master and another for node): t2.micro
default VPC and create security group with all traffic open(Not recommended but use for practical purpose)

- Login to Ansible master VM

```
## Install Ansible

sudo su -

sudo apt get update

sudo apt upgrade -y

sudo apt-add-repository ppa:ansible/ansible

sudo apt install ansible

ansible --version

create a key pair : ssh-keygen

copy the public key manually on node

copy content of id_rsa.pub to node - /.ssh/authorized_keys file

on target machine: chmod -R go= ~/.ssh
```

This recursively removes all "group" and "other" permissions for the ~/.ssh/ directory.

```
cd /etc/ansible
```

```
vim hosts
```

```
add remote IP # IP of node machine- Private IP
```

```
#To add Amazon Linux as a Node
```

```
1] Get one amazon linux VM with instance type:t2.micro
```

```
2] On Controller : copy private_key /etc/ansible/private_key    # all private key content
```

```
3] provide permission : chmod 600 private_key
```

```
4] cd /etc/ansible  - vim hosts
```

```
5] Add amanzon linux IP details
```

```
    xx.xxx.xx.XX ansible_user=ec2-user ansible_ssh_private_key_file=/path/to/private_key
```

```
6] copy content of id_rsa.pub to node - /.ssh/authorized_keys file (in ec2_user)
```

```
7] on target machine: chmod -R go= ~/.ssh
```

```
8] on Amzon Linux : sudo vim /etc/ssh/sshd_config
```

```
remove comment : PubkeyAuthentication yes
```

```
9] sudo systemctl restart sshd
```

- Copy the public key to your node machines
- There are multiple ways to copy it
  1] copy the public key using ssh-copy-id
  2] copy the public key manually : Using this method
  We will manually append the content of your id_rsa.pub file to the ~/.ssh/authorized_keys file on your remote machine.
  cat ~/.ssh/id_rsa.pub
  on target machine: chmod -R go= ~/.ssh This recursively removes all “group” and “other” permissions for the ~/.ssh/ directory.
- check ssh connectivity between master to node: ssh root@<node_IP>

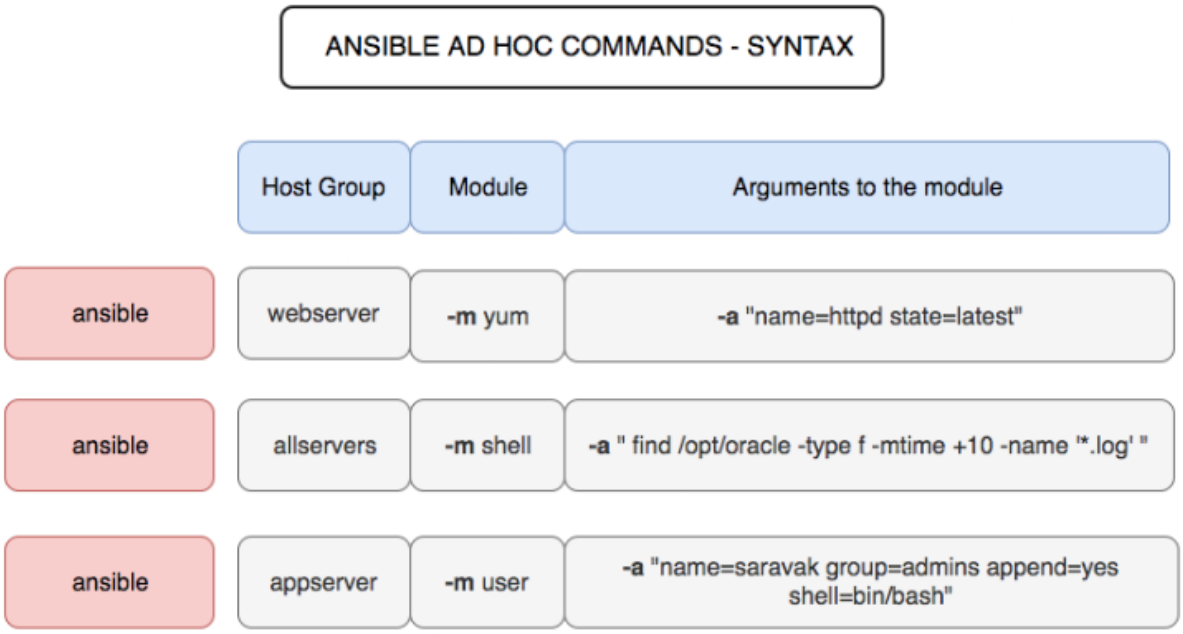
```
ansible all --list-hosts
```

Ad-Hoc Commands & Modules

An 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

A **module** is a reusable piece of code that performs a specific task on a managed node. Modules are the building blocks of Ansible playbooks and ad-hoc commands.

Press enter or click to view image in full size




```
##ad-hoc syntax:
```

```
ansible <inventory> -m <module> -a "<arguments>"
```

```
ansible all -m ping
```

```
ansible all -m setup
```

```
ansible all -m apt -a "name=nginx state=present" -b
```

```
ansible all -m apt -a "name=nginx state=absent" -b
```

```
sudo service nginx status  # check nginx service status
```

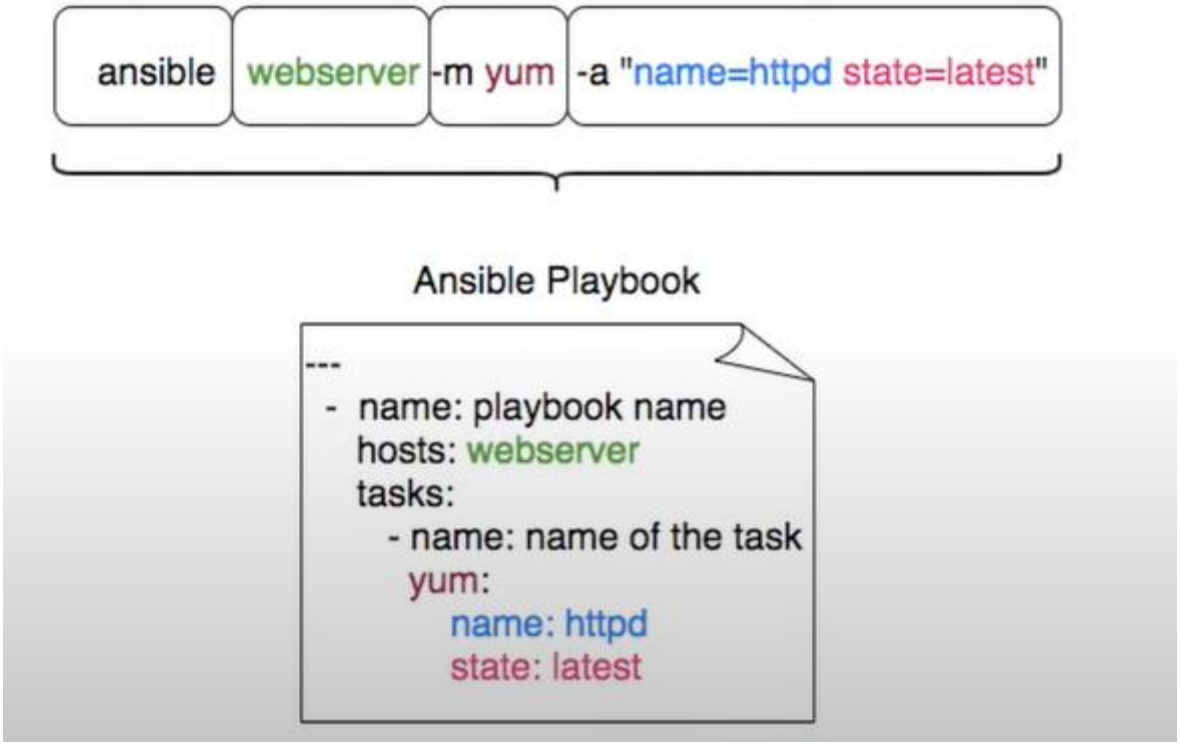
```
##b flag indicates that the command should be run with elevated privileges (sudo).
```

```
ansible all -m service -a "name=nginx state=restarted" -b
```

```
ansible all -m copy -a "src=/path/to/local/file.txt dest=/path/on/remote/file.txt" -b
```

```
ansible all -m apt -a "name=nginx state=present" -b
```

there are two ways of doing any automation using Ansible
Ad-hoc commands ()
ansible playbook



Modules

COPY

The `copy' module copies a file from the local or remote machine to a location on the remote machine.

```
ansible all -m copy -a 'content="I am awesome" dest=/tmp/class1.txt'
```

```
ansible all -m copy -a 'src=classs.txt dest=/opt'
```

```
if its failed do to lack of permission then
```

```
ansible all -m copy -a 'src=classs.txt dest=/opt' -u root -k
```

```
###copy file from node one location to another location.
```

```
ansible all -m copy -a 'src=/tmp/class1.txt dest=/opt remote_src=yes'
```

Command

The `command' module takes the command name followed by a list of space-delimited arguments. The given command will be executed on all selected nodes.

```
ansible all -m command -a 'ls -lh /tmp/classs.txt'
```

```
ansible all -m command -a 'cat /tmp/classs.txt'
```

```
ansible all -m command -a 'id'
```

shell

The `shell` module in Ansible is used to execute commands on the target hosts as if they were run directly on the command line of the remote server. This module allows you to use shell features, such as variable substitution, wildcard expansion, and shell pipes.

```
create a test.sh file
```

```
#!/bin/bash
```

```
echo "Hello welcome"
```

```
ansible all -m copy -a 'src=test.sh dest=/tmp mode=755'
```

```
ansible all -m shell -a '/tmp/test.sh'
```

raw

The `raw` module in Ansible is a powerful but less common module that allows you to execute raw commands on the target hosts without Ansible's typical module abstraction. It runs the specified command exactly as you write it, with no additional processing or interpretation by Ansible.

```
ansible all -m raw -a 'ls -lrt /tmp ; pwd'
```

file

The `file` module in Ansible is used for managing files and directories on remote hosts. It allows you to create, modify, and delete files and directories.

```
ansible all -m file -a 'path=/tmp/data state=directory'
```

```
to check directory is created or not : ansible all -m command -a 'ls -lrt /tmp'
```

delete directory

```
ansible all -m file -a 'path=/tmp/data state=absent'
```

```
ansible all -m file -a 'path=/tmp/data state=directory mode=0777 owner=root group=root'
```

fetch

The `fetch` module in Ansible is used to retrieve files from remote hosts and copy them to the local machine.

```
ansible all -m copy -a 'src=test.sh dest=/tmp mode=755'
```

```
ansible all -m fetch -a 'src=/tmp/test.sh dest=backup'
```

get_url

The `get_url` module in Ansible is used to download files from the internet and save them on the target machine.

```
ansible all -m get_url -a 'url=https://linux-training.be/linuxfun.pdf dest=/tmp/linuxfun.pdf'
```

lineinfile

The `lineinfile` module in Ansible is used for managing lines in a text file. It allows you to add, modify, or delete lines in a file

```
create a one file in corrent location
```

```
ansible all -m copy -a 'src=sagar.txt dest=/tmp'
```

```
ansible all -m lineinfile -a 'dest=/tmp/sagar.txt line="this is first project"'
```

```
ansible all -m command -a 'cat /tmp/sagar.txt'
```

```
add in first line of file
```

```
ansible all -m lineinfile -a 'dest=/tmp/sagar.txt line="Hi everyone" insertafter=BOF'
```

user

The `user` module in Ansible is used for managing user accounts on remote hosts. It allows you to create, modify, and delete user accounts, set user attributes, and manage user groups.

```
ansible all -m user -a 'name=amit state=present uid=1010 groups=root'
```

```
ansible all -m command -a 'id -a amit'
```

```
ansible all -m user -a 'name=amit state=absent uid=1010 groups=root'   ## remove user
```

group

The `group` module in Ansible is used for managing groups on remote hosts. It allows you to create, modify, and delete groups, as well as manage group membership.

```
ansible all -m group -a 'name=staff123 state=present gid=1030'
```

```
ansible all -m command -a 'tail -2 /etc/group'
```

```
ansible all -m group -a 'name=staff123 state=absent'
```

yum

The `yum` module in Ansible is used for managing packages on systems that use the YUM package manager, such as many Linux distributions based on Red Hat, CentOS, and Fedora. It allows you to install, update, remove, and manage packages, repositories, and groups.

apt

The `apt` module in Ansible is used for managing packages on systems that use the APT (Advanced Package Tool) package manager, commonly found on Debian-based Linux distributions like Ubuntu. The `apt` module allows you to install, update, remove, and manage packages and repositories.

```
ansible all -m apt -a 'name=zsh state=present'
```

```
ansible all -m command -a 'apt list zsh'
```

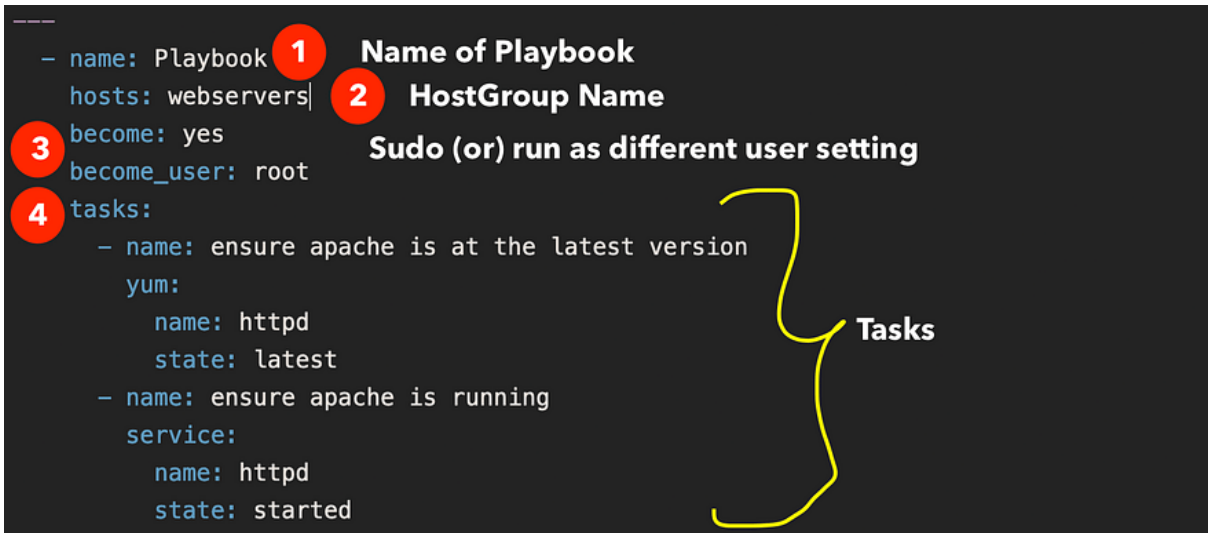
package

The `package` module in Ansible is a generic module for managing packages across different package managers. It provides a common interface for working with package management systems, abstracting the differences between systems like APT, YUM, DNF, Zypper, and others.

```
ansible all -m package -a 'name=zsh state=absent use=apt'
```

Ansible Playbook

An Ansible playbook is a script that defines a set of tasks to be executed on remote hosts. Playbooks are written in YAML format and provide a way to automate infrastructure configuration, application deployment, and other IT tasks. A playbook consists of one or more plays, where each play defines a set of tasks to be executed on a specific set of hosts.



Below is ansible playbook example. create first.yaml and use below code.

How to execute playbook :ansible-playbook first.yaml

```
- hosts: all

 tasks:




   - copy:

       src: /etc/ansible/test.sh

       dest: /tmp
```

```
        group: root

        mode: 777




    - file:

        path: /tmp/ansible

        state: directory
```

Ansible playbook to install apache2

```
---

- name: Install Apache2

 hosts: your_target_servers  # change host as per your requirement

 become: yes  # Run tasks with sudo




 tasks:
```

```
- name: Update apt cache

  apt:

    update_cache: yes



- name: Install Apache2

  apt:

    name: apache2

    state: present



- name: Start Apache2 service

  service:

    name: apache2

    state: started
```

run playbook based on os

```
#need two managed nodes - ubuntu and Amazon linux




- hosts: all

 gather_facts: true

 tasks:

 - name: run apt-get

   command: apt-get update

   when: ansible_distribution == "Ubuntu"

 - name: run yum update

   command: sudo yum install tree -y

   when: ansible_distribution == "Amazon"
```

Variables:

```
- hosts: all

 vars:

   mydir: /tmp/testing

   myfile: /etc/ansible/test.sh

   mypkg:

      - apache2

      - tree

 tasks:




   - name: "This tasks will create a new directory"

     file:

       path: "{{ mydir }}"

       state: directory
```

```
- name: "copy file"

  copy:

    src: "{{ myfile }}"

    dest: "{{ mydir}}"

    group: root

    mode: 777

- name: "create a new dir"

  file:

    path: /tmp/sagar

    state: directory




- name: "install package"
```

```
    apt:

      name: "{{ mypkg }}"

      state: latest
```

Now we will create separate file for all variables and provide file reference in main yaml file

```
# create a var.yaml file

mydir: /tmp/testing

myfile: /etc/ansible/test.sh

mypkg:

  - apache2

  - tree
```

```
# In main.yaml file update var file location

vars_files:
```

```
    - /etc/ansible/var.yaml
```

```
# You can pass variable from terminal also
```

```
ansible-playbook variable.yaml -e mydir=xxxxx myfile=xxxx mkpkg=xxx,xxxx
```

```
# If a variable is defined both in the YAML file and passed through the terminal, the terminal value takes precedence
```

Variable defines in Inventory file

```
# add variables in inventory file :
```

```
IP_address mydir=/tmp/testing myfile=/etc/ansible/test.sh mypkg=apache2,tree
```

```
Remove variable reference from variable.yaml file and run playbook
```

group vars

```
#In inventory file




[web]

XX.XX.XX.XX

[app]

XX.XX.XXX.XX




[web:vars]

mydir=/tmp/testing

myfile=/etc/ansible/test.sh

mypkg=apache2,tree
```

```
[app:vars]

mydir=/tmp/testing

myfile=/etc/ansible/test.sh

mypkg=apache2,tree




# run above variables.yaml playbook after removing variable reference
```

loop

```
- hosts: all

 tasks:



   - copy:

       src: "{{ item }}"

       dest: /tmp
```

```
    group: root

    mode: 777

  with_items:

      - /etc/ansible/test.sh

      - /etc/ansible/sagar.txt




- file:

    path: "{{ item }}"

    state: directory

  with_items:

      - /tmp/cloud1

      - /tmp/cloud2
```

loop with variable

```
- hosts: all

 vars:

   myfile:

     - /etc/ansible/test.sh

     - /etc/ansible/sagar.txt

   mydir:

     - /tmp/cloud1

     - /tmp/cloud2

 tasks:




   - copy:

       src: "{{ item }}"

       dest: /tmp
```

```
      group: root

      mode: 777

    with_items:

      - "{{ myfile }}"

  - file:

      path: "{{ item }}"

      state: directory

    with_items:

      - "{{ mydir }}"
```

nested loop

```
- hosts: all

 tasks:
```

```
  - copy:

      src: "{{ item[0] }}"

      dest: "{{ item[1] }}"

      group: root

      mode: 777

    with_items:

        - /etc/ansible/test.sh

        - /tmp/cloud1
```

tags:

Ansible tags are used to selectively run or skip specific tasks, roles, or entire plays within a playbook. They provide a way to organize and control the execution of tasks based on user-defined criteria.

```
Example

---
```

```
- name: installing apache2

 hosts: all

 become: yes




 tasks:

   - name: Install Apache2

     apt:

       name: apache2

       state: present

     tags:

       - apache




# how to run playbook with tag :
```

```
ansible-playbook <playbookname.yaml> --tags "<tag_name>"


# How to skip tags:  --skip-tags "database"
```

handlers

In Ansible, handlers are tasks that are triggered by other tasks. They are typically used to perform actions such as restarting a service or reloading configuration files in response to changes made during the playbook run. Handlers are only executed when notified by other tasks.

To notify a handler from a task, you can use the `notify` directive. Here's a basic example:

```
Example #1


- hosts: all


 tasks:


  - lineinfile:


          path: /etc/ssh/sshd_config


          line: DenyUsers harry


    notify: cloud
```

```
 handlers:

   - name: cloud

     service:

         name: sshd

         state: restarted
```

Example#2

```
---

- name: Install and configure Apache2

 hosts: your_target_hosts

 become: yes  # This allows the tasks to run with sudo privileges

 tasks:
```

```
  - name: Install Apache2

    apt:

      name: apache2

      state: present

    notify: restart apache




handlers:

  - name: restart apache

    service:

      name: apache2

      state: restarted
```

simple **Ansible playbook** to deploy a **static HTML page using Nginx**

```
static_site_deploy/
```

```
├── playbook.yml
├── files/
│   └── index.html
```

index.html (in files/index.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Welcome</title>
</head>
<body>
   <h1>Welcome to My Static Site!</h1>
</body>
```

```
</html>
```

```
#playbook.yaml
```

```
---
```

```
- name: Deploy static HTML site using Nginx
 hosts: all
 become: yes

 tasks:
   - name: Install Nginx
     apt:
       name: nginx
       state: present
       update_cache: yes
```

```
- name: Remove default Nginx index.html

  file:

    path: /var/www/html/index.nginx-debian.html

    state: absent



- name: Deploy custom index.html

  copy:

    src: files/index.html

    dest: /var/www/html/index.html

    owner: www-data

    group: www-data

    mode: '0644'



- name: Ensure Nginx is running

  service:
```

```
        name: nginx

        state: started

        enabled: yes
```

```
ansible-playbook -i your_inventory_file playbook.yml
```

```
Access your server IP in a browser:
```

```
http://<your-server-ip>
```

```
You should see: "Welcome to My Static Site!"
```

Roles

Ansible roles are a way to organize your Ansible tasks, handlers, variables, and other configuration elements in a reusable and modular fashion. They provide a structured way to break down complex configurations into smaller, more manageable components.

add image of tree command

```
cd /etc/ansible
```

```
mkdir roles && cd roles
```

```
ansible-galaxy role init <role_name>

# using tree command you can see roles directory structure

#Practical



In tasks - main.yaml



# tasks file for cloudknowledge

- name: Going to install apache package

 apt:

   name: "{{ apache_package }}"

   state: present



- name: Going to copy source code

 copy:

   src: index.html

   dest: "{{ apache_documentroot }}"
```

```
#in vars/main.yaml

# vars file for cloudknowledge


apache_package: apache2

apache_documentroot: /var/www/html/index.html


# In files/index.html and add some content

#in test/inventory - update your machine IP


# how to execute ansible role

in test folder : ansible-playbook test.yaml
```

Ansible vault

Ansible Vault is a feature of Ansible that allows you to **encrypt sensitive data** such as passwords, API tokens, SSH keys, and other secrets — **directly within your playbooks or variable files**.

This allows secure handling of secrets in **version-controlled repositories** like Git without exposing them in plaintext.

Basic commands

```
Create a vault file :     ansible-vault create secrets.yml
```

```
Edit an existing vault file :     ansible-vault edit secrets.yml
```

```
Encrypt an existing file :     ansible-vault encrypt file.yml
```

```
Decrypt an encrypted file     ansible-vault decrypt file.yml
```

```
View a vault file : ansible-vault view secrets.yml
```

```
Change vault password : ansible-vault rekey secrets.yml
```

```
Run playbook with vault : ansible-playbook playbook.yml --ask-vault-pass
```

```
Use password file : ansible-playbook playbook.yml --vault-password-file vault-pass.txt
```

Practice

```
Create an encrypted file:
```

```
ansible-vault create secret_vars.yaml   # It will ask for a password (choose and remember it).
Inside the file, add:


mysecret: "ThisIsTopSecret123"




To check if the file is encrypted: cat secret_vars.yaml




# create a playbook


# vault-playbook.yml


---


- name: Vault Test Playbook


 hosts: all


 become: true


 vars_files:


   - secret_vars.yml
```

```
 tasks:

   - name: Print the secret

     debug:

       msg: "The secret is {{ mysecret }}"
```

```
# Run the Playbook Using Vault Password

ansible-playbook vault-playbook.yml --ask-vault-pass
```

```
# Encrypt an Existing File

ansible-vault encrypt secret_vars.yaml

# decrypt :

ansible-vault decrypt secret_vars.yml
```

```
# Edit Vault File
```

```
ansible-vault edit secret_vars.yml
```

```
# Re-key with New Password
```

```
ansible-vault rekey secret_vars.yml
```

Dynamic Inventory

By default, **Ansible uses a static inventory** (a file listing hostnames or IPs), but in dynamic environments like **AWS, Azure, GCP, Kubernetes**, etc., where infrastructure changes frequently, maintaining static inventory isn't scalable.

This is where **dynamic inventory** comes in.

A **dynamic inventory** fetches the list of target hosts from an **external source (like cloud APIs)** each time you run an Ansible command or playbook. It dynamically builds a list of hosts based on filters such as tags, regions, etc.

```
# This is on master machine
```

```
sudo apt install python3-pip -y
```

```
sudo apt install python3-venv python3-full -y

python3 -m venv ~/myenv

source ~/myenv/bin/activate

pip install boto3

ansible-galaxy collection install amazon.aws




# Install aws cli on master

sudo apt install -y unzip curl

curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install

aws --version
```

```
#aws configure for connectivty with AWS resources

aws configure  <provide access key, secret key , region etc>




vi /etc/ansible/ansible.cfg

[inventory]

enable_plugins = aws_ec2

inventory = /opt/aws_ec2.yaml

private_key_file = /opt/key1.pem

remote_user = ubuntu

host_key_checking = False




#create a inventory file

cd /opt
```

```
vi aws_ec2.yaml
```

```
plugin: amazon.aws.aws_ec2

regions:

 - ap-south-1

filters:

 tag:Name: "Node"   # Quotes are recommended to avoid YAML parsing issues

keyed_groups:

 - key: tags.Name

   prefix: tag
```

```
#

ansible-inventory -i /opt/aws_ec2.yaml --list
```

```
# Create a copy playbook




---


- name: Copy file to EC2 instances tagged Name=Node


 hosts: tag_Node


 tasks:


   - name: Create a file with some content


     copy:


       dest: /tmp/hello_from_ansible.txt


       content: "Hello from Ansible on {{ inventory_hostname }}\n"
```

```
# ansible-playbook -i /opt/aws_ec2.yaml copy.yaml   -u ubuntu   --private-key /opt/key1.pem
```