Unit-3

Part-B

Configuration Management with Ansible:

1. **Introduction to Ansible,**
2. **Ansible architecture and installation,**
3. **Inventory management,**
4. **Ad-hoc commands and playbooks,**
5. **Roles and modules in Ansible,**
6. **Writing and managing Ansible playbooks,**
7. **Integrating Ansible with other tools**.

1.Introduction to Ansible

Ansible:

Ansible is a simple, powerful, and agentless IT automation tool that is used for IT configuration management, application deployment, and perform repetitive task. It simplifies complex tasks by automating them using simple YAML-based scripts called Playbooks.

Why Ansible is important

• Automates repetitive tasks (e.g., software installation, configuration updates).

• Manages multiple servers with a single script.

• Works across platforms (Linux, Windows, Cloud, On-Premise).

• Integrates with DevOps tools like Jenkins, Docker, Kubernetes.

• Enhances security by enforcing configuration consistency.

Overview :

Ansible is an IT automation engine that can automate various IT needs. And it has features like application deployment that means you can deploy your application easily as per your requirements, cloud provisioning, configuration management is also the main feature where you can configure and describe your automation job, and intra-service orchestration. In this, (Yet Another Markup Language)YAML is used for configuring that helps for describing automation jobs as per requirement. It is Designed for multi-tier deployments, Ansible models the IT

infrastructure by describing how various systems interrelate, instead of managing one system at a time.

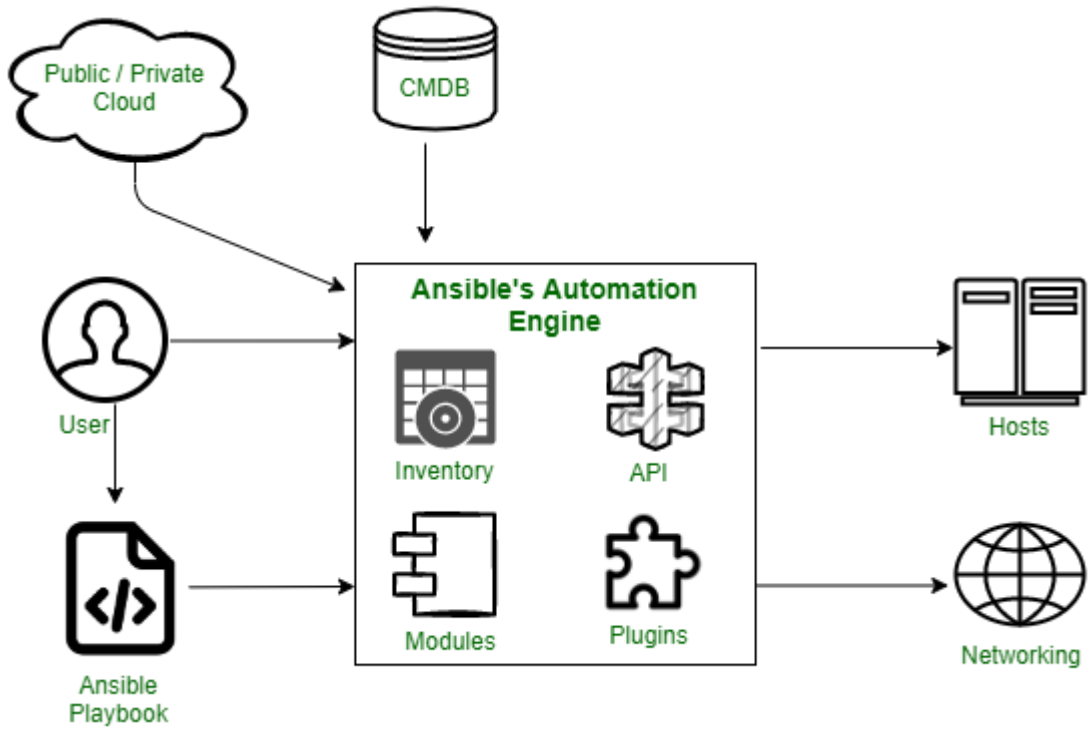
Features :

In this, It uses no extra functionality and cost like no agents and no extra custom security infrastructure, hence it is easy to deploy.

It uses a very simple language called YAML (Yet Another Markup Language) in the form of Ansible Playbooks and you can configure it as per your requirement, and it helps describe the automation jobs in a way that looks like basic English.

The Ansible Automation Engine has a direct interaction with the users who write playbooks and also interacts with cloud services and the Configuration Management Database (CMDB).

2. Ansible architecture and installation



a.Inventories -

Ansible inventories are lists of hosts with their IP addresses, servers, and databases which have to be managed via an SSH for UNIX, Linux, or Networking devices, and WinRM for Windows systems.

b.APIs -
Application Programming Interface or APIs are used as a mode of transport for public and private cloud services.

c.Modules -
Modules are executed directly on remote hosts through playbooks and can control resources like services, packages, files, or execute system commands. They act on system files, install packages and make API calls to the service network. There are over 450 Ansible that provide modules that automate various jobs in an environment. For example, Cloud Modules like Cloud Formation create or delete an AWS cloud formation stack.

d.Plugins -
Plugins are pieces of code that augment Ansible's core functionality and allow executing Ansible tasks as a job build step. Ansible ships with several handy plugins and one can also write it on their own. For example, Action plugins act as front-ends to modules and can execute tasks on the controller before calling the modules themselves.

e.Networking-
Ansible uses a simple, powerful, and agent-less automation framework to automate network tasks. It uses a separate data model and spans different network hardware.

f.Hosts -
Hosts refer to the nodes or systems (Linux, Windows, etc) which are automated by Ansible.

g.Playbooks -
Playbooks are simple files written in YAML format which describe the tasks to be executed by Ansible. Playbooks can declare configurations, orchestrate the steps of any manual ordered process and can also launch various tasks.

h.CMDB -
It stands for Configuration Management Database (CMDB). In this, it holds data to a collection of IT assets, and it is a repository or data warehouse where we will store this kind of data, and It also defines the relationships between such assets.

I.Cloud -
It is a network of remote servers hosted on the internet to store, manage and process data instead of storing it on a local server.

J.Control Node

The Control Node is the machine where Ansible is installed and executed. It is responsible for managing and orchestrating the execution of automation tasks.

K.Managed Nodes (Hosts)

These are the target machines (Linux, Windows, or network devices) where Ansible executes tasks. Managed nodes do not require Ansible to be installed, making it agentless.

Inventory

The inventory file lists all the managed nodes (hosts) that Ansible will control. Managed Nodes

(Hosts)

[Webserver]

125.11.58.85

Playbooks

Playbooks are YAML files that define automation tasks in a structured manner.

YAML Structure:

- name: Install Apache

hosts: webservers

tasks:

- name: Install Apache on Debian

apt:

name: apache2

state: present

systemd:

name: apache

state: started

enabled: yes

Installation

For Installing ansible in Linux (Ubuntu) System we have some steps.

 Sudo apt update

 sudo apt install -y software-properties-common

 Sudo apt install ansible

For Installing ansible in Windows System we have some steps.

 Open windows power shell as administrator mode

 Run command “wsl –install”

 Verify WSL installation using “wsl --list –online” command

 Install Ubuntu as the default WSL distribution using “wsl --install -d Ubuntu” After installation of ubuntu, open ubuntu from start menu and run the ubuntu and create and account

 Now run “sudo apt update command” for update the dependencies

 Install Ansible using “sudo apt install ansible” command

3.Inventory management

Inventory management in Ansible, refers to the process of defining, organizing, and managing the list of target hosts that Ansible will control. The inventory file contains details such as hostnames, IP addresses, groups, and connection parameters.

 A target host may be a Cloud instance, Network Device, or physical servers.

Mainly we have two types of inventory:

Static Inventory: Defined by manually using .ini or .yml file for configure and manage the host server.

A. [web_servers]

web1.example.com

web2.example.com

B. [db_servers]

db1.example.com ansible_host=192.168.1.10 ansible_user=root

Dynamic Inventory: Dynamic inventory allows Ansible to fetch host details from cloud providers, container platforms, or other external sources dynamically. This is useful for large, frequently changing infrastructures.

Ex: AWS, Azure, Cisco, MS-365, VPN Services, GoogleHome and Alexa, Cryptocurrency, Airtel, Jio Starlink etc.

To enable AWS dynamic inventory, create a configuration file (aws_ec2.yml)

plugin: aws_ec2

regions:

- us-east-1

keyed_groups:

- key: tags.Environment

prefix: env_

Show the list of dynamic inventory

ansible-inventory -i aws_ec2.yml –list

Managing Inventory in Ansible

1. First Check all inventory host list using given command

ansible-inventory -i inventory.yml –list

2. Check weather it is connected or not with host server with given command

ansible all -m ping -i inventory.yml

3. Run ad-hoc command on any one host for checking uptime downtime details using given

command

ansible web_servers -m command -a "uptime" -i inventory.yml

4.Ad-hoc commands and playbooks

Ad-Hoc Commands and playbook

Ad-hoc command in Ansible is a single line or linear command that is used for execute single task or one or more managed node without writing a long playbook. it is often used for Restart,

stop, ping and copy a file. Ad-hoc command is a light-weight and easy but they are not a reusable command.

Basics Syntax of Ad-hoc command

ansible <host-pattern> -m <module> -a "<module-arguments>"

Some Ad-Hoc Commands

1. Ping a single host or all host:

 ansible <host-name> -m ping

 ansible all -m ping

2. Restart single services or all services:

 ansible <host-name> -m service -a "name=httpd state=restarted“

 ansible all -m service -a "name=httpd state=restarted“

3. Copy a file to a remote server:

 ansible all -m copy -a "src=/home/user/file.txt dest=/tmp/file.txt“

4. Install a package using yum command:

 ansible all -m yum -a "name=httpd state=present"

Playbook:-

An Ansible Playbook is a YAML file that defines a series of automation tasks to be executed on

remote machines. Unlike ad-hoc commands (which are for one-off tasks), playbooks are declarative,

repeatable, and designed for complex configurations and orchestration.

YAML Syntax: Playbooks are written in YAML, which is human-readable and structured using

indentation.

Play: A play is a mapping between a group of hosts and tasks. A playbook can contain one or more

plays.

- **YAML File:** Playbooks are just text files ending with .yaml or .yml.

- **Hosts:** You tell the playbook *which* servers to run on (e.g., all servers, web servers, database servers).
- **Tasks:** These are the individual steps or actions Ansible will perform. Each task uses an Ansible **module** to do something specific.
- **Modules:** These are like specialized tools. For example:
    - ansible.builtin.ping: Checks if a server is reachable.
    - ansible.builtin.apt or ansible.builtin.yum: Installs software packages.
    - ansible.builtin.copy: Copies files.
    - ansible.builtin.service: Starts, stops, or restarts services.
- **become: true:** This means "run this task with elevated privileges," usually like sudo (acting as an administrator).
- **Order Matters:** Tasks in a playbook run from top to bottom, in the order you list them.

Small Example: Installing Nginx

Let's say you want to install the Nginx web server on a machine. Here's a super simple playbook:

YAML

```
---
- name: Install Nginx web server
  hosts: webserver_group # This refers to a group of servers in your inventory
  become: true           # We need root privileges to install software

  tasks:
    - name: Ensure Nginx package is installed
      ansible.builtin.apt:
        name: nginx
        state: present

    - name: Ensure Nginx service is running and enabled
      ansible.builtin.service:
        name: nginx
        state: started
        enabled: true
```

Simple YAML Structure and YAML Structure with Roles.

Simple YAML	YAML with roles
- name: Install the correct web server for RHEL import_tasks: redhat.yml when: ansible_facts['os_family']|lower == 'redhat' - name: Install the correct web server for Debian import_tasks: debian.yml when: ansible_facts['os_family']|lower == 'debian' - name: Install web server ansible.builtin.yum: name: "httpd" state: present - name: Install web server ansible.builtin.apt: name: "apache2" state: present	# roles/example/tasks/main.yml - name: Install the correct web server for RHEL import_tasks: redhat.yml when: ansible_facts['os_family']|lower == 'redhat' - name: Install the correct web server for Debian import_tasks: debian.yml when: ansible_facts['os_family']|lower == 'debian' # roles/example/tasks/redhat.yml - name: Install web server ansible.builtin.yum: name: "httpd" state: present # roles/example/tasks/debian.yml - name: Install web server ansible.builtin.apt: name: "apache2" state: present

5.Roles and modules in Ansible

Roles:

In Ansible, Roles is a structured way of organizing playbooks, making complex playbooks more manageable, reusable, and scalable using breakdown a long playbook into a different small files. It can make complex playbook Easier to manage and share.

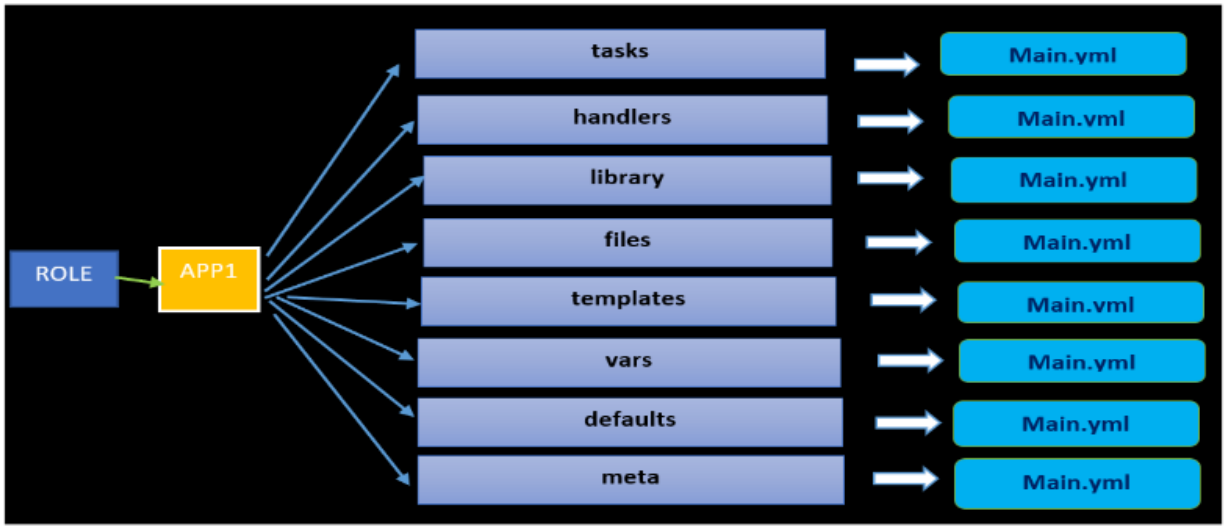


Fig: Architecture of Roles.

Purpose of Each Folder

 tasks/ – Main list of tasks to be executed.

 handlers/ – Tasks triggered by notify.

 files/ – Static files to copy to hosts.

 templates/ – contain templates for configuration files.

 vars/ – Variables with higher precedence.

 defaults/ – Default variables lowest precedence.

 meta/ – Contain Role metadata and dependencies.

Why roles are Important

 Reusability: you can create it once and reuse it in different projects and environments.

 Modularity: Roles allow breaking down playbooks into reusable components. Each role can manage a particular aspect of the system, such as installing a web server or configuring a database.

 Maintainability: By organizing tasks, variables, handlers, and other components into separate directories, roles make the codebase easier to navigate and maintain.

 Scalability: As infrastructure grows, roles help manage complexity by providing a clear structure, making it easier to scale configurations.

Use Cases of Ansible Role

Ansible Roles are used in a variety of contexts, including:

 Infrastructure as Code (IaC): Managing and provisioning infrastructure in a consistent and repeatable manner.

 Application Deployment: Deploying applications with all necessary dependencies and configurations.

 Configuration Management: Ensuring systems are configured correctly and consistently across environments.

 Continuous Integration/Continuous Deployment (CI/CD): Integrating with CI/CD pipelines to automate the deployment process.

When to Use Ansible Roles

Ansible Roles should be used when:

 Managing Large Codebases: For projects with large and complex playbooks, roles help in organizing and simplifying the code.

 Promoting Code Reusability: When there is a need to use the same configuration across multiple projects or environments.

 Improving Collaboration: In teams where multiple developers or operators are working on the same codebase, roles enhance collaboration by providing a clear structure.

 Automating Repetitive Tasks: For tasks that are repetitive and consistent across different environments, roles ensure standardization and efficiency.

How can we Install a Roles in Ansible:

In Ansible, we have a Ansible Packge Manager that is called “Ansible-galaxy” that is used for roles and connection. It connects to Ansible Galaxy a public repository (https://galaxy.ansible.com/) where developers and organizations share roles and collections. It can also work with private Galaxy servers or local directories.

Modules in Ansible:

In Ansible, Modules is a keyword or predefines small tasks or tool, that is used for performs specific task like.. Install a package or server, copy a file, create a user etc.

Common Modules Examples in Ansible:

 apt, yum, dnf: manage packages

 copy: copy files to remote machines

 file: set file properties

 service: start/stop services

 user: manage users

 command, shell: run shell commands

 git: clone repositories

Example:

- name: Install nginx

apt:

name: nginx

state: present

6.Writing and managing Ansible playbooks

Writing and managing Ansible playbooks is a crucial part of using Ansible effectively. (Write any YAML file) We know that, An Ansible playbook is a YAML file that defines one or more plays or tasks. Each play maps a group of hosts to tasks that define what you want Ansible to do. In Ansible, playbooks can be written and managed in two different ways

1. Simple Structure

2. Advance Structure with ansible roles

1. Simple Structure

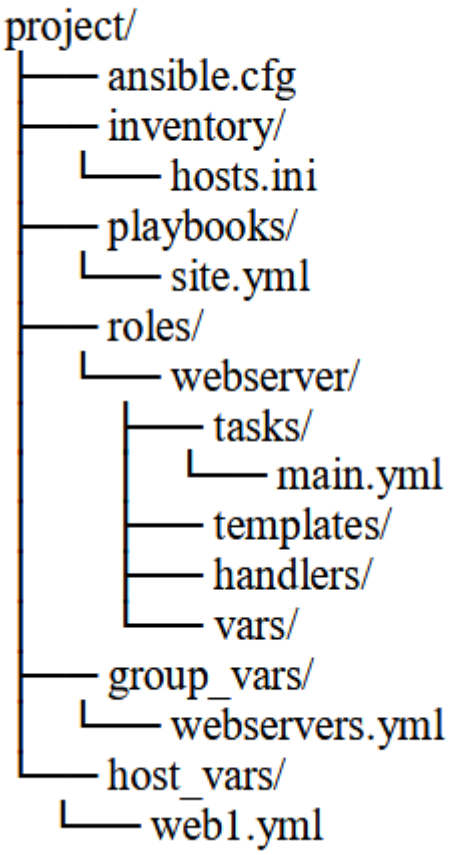
project/

├── inventory.ini

├── playbook.yml

└── ansible.cfg

2. Advance Structure with ansible roles.



Key Component Description

Component	Description
name	Human-readable name for the play or task
hosts	Target hosts from your inventory
vars	Define variables
tasks	List of actions to perform
roles	Include reusable role structures

7.Integrating Ansible with other tools

1. **Integrating with Git**

Purpose: Manage playbooks as code (IaC), mange version control and enable triggers via webhooks.

Integration:

- Store playbooks in Git.
- Trigger Ansible runs via GitHub Actions or GitLab CI/CD.
- Use ansible-pull for pull-based deployment models.

2. **Integrating with Jenkins**

Purpose: Automate infrastructure changes and deployments as part of CI/CD pipelines.

We have an option to write Ansible playbook inside Jenkins pipeline scripts or as a post -build step.

After that it can Trigger playbooks to deploy applications or configure environments.

Command:

ansible-playbook -i inventory site.yml

3. Integrating with Docker and Kubernates

Docker:

Use Ansible to build images, start/stop containers, and manage Docker Compose.

Ansible has built-in Docker modules (e.g., community.docker.docker_container).

Kubernetes:

Deploy and manage Kubernetes resources using Ansible modules or kubectl commands.

Combine with Helm charts via (community.kubernetes.helm) command

4. **Integrating with Cloud Services**

Purpose: Provision and manage cloud resources.

Integration:

 Ansible has cloud-specific modules and collections (e.g., amazon.aws, azure.azcollection).

 Use Ansible to automate EC2 instances, S3, VPC, load balancers, etc.

5. Integrating with Terraform

Purpose: Provision infrastructure (like servers, networks), then configure it with Ansible.

Integration:

 Terraform provisions infrastructure.

 Use null_resource and local-exec or remote-exec to call Ansible.

 Output dynamic inventory from Terraform to feed into Ansible.

6. integrating with monitoring tool like Grafana, Promethus.

Purpose: Automate alert responses or configure monitoring setups.We can Use Ansible with Monitering tool for automate the Alert System in our Application using ansible playbook and webhook.