**ANSIBLE**

In ansible we have control node and manage nodes. Assume control node is one virtual machine manage nodes also virtual machines if we get task to install java on manage nodes you have to write a playbook on control nodes no need to go to each server and write it.As a prerequist manage ode should have python.

Ansible is a popular open-source automation tool designed for simplifying IT tasks, including configuration management, application deployment, and orchestration. Here’s why Ansible is needed:

Orchestration in Ansible involves managing the interactions and dependencies between different systems, services, or tasks to achieve a coordinated outcome. Unlike simple provisioning, orchestration focuses on automating complex workflows where multiple components need to work together seamlessly.

**1. Automation of Repetitive Tasks**

* **Problem:** Managing IT infrastructure often involves repetitive tasks like configuring servers, deploying applications, and ensuring consistency across environments.
* **Solution:** Ansible automates these tasks, reducing manual effort and the risk of human error.

**2. Agentless Architecture**

* **Problem:** Many automation tools require agents or software to be installed on the managed nodes, adding complexity and overhead.
* **Solution:** Ansible is agentless. It uses SSH (or WinRM for Windows) to communicate with systems, making it lightweight and easy to deploy.

**3. Scalability**

* **Problem:** As infrastructure grows, managing a large number of systems manually or with scripts becomes unmanageable.
* **Solution:** Ansible scales seamlessly from managing a few servers to thousands, enabling consistent and efficient operations.

**4. Infrastructure as Code (IaC)**

* **Problem:** Manual configuration is error-prone and doesn't support version control or auditing.
* **Solution:** Ansible uses simple, human-readable YAML files to define configurations, making it easy to version, audit, and share infrastructure definitions.

**5. Cross-Platform Support**

* **Problem:** Organizations often use a mix of operating systems and environments, such as Linux, Windows, cloud platforms, and containers.
* **Solution:** Ansible supports diverse platforms and environments, allowing centralized management from a single tool.

**6. Rapid Deployment**

* **Problem:** Setting up environments and deploying applications manually can take hours or days.
* **Solution:** Ansible automates deployments, enabling rapid and consistent rollouts across environments.

**7. Security and Compliance**

* **Problem:** Maintaining consistent security policies and compliance across multiple systems is challenging.
* **Solution:** Ansible can enforce security baselines, perform regular audits, and ensure compliance with industry standards.

**8. Simplified Learning Curve**

* **Problem:** Complex tools require significant time to learn and use effectively.
* **Solution:** Ansible’s declarative, YAML-based syntax is simple and intuitive, requiring minimal training compared to other automation tools.

**Use Cases for Ansible**

* Configuration management (e.g., setting up servers with required software).
* Application deployment (e.g., deploying web apps or databases).
* Infrastructure provisioning (e.g., provisioning VMs or cloud resources).
* Orchestration (e.g., coordinating multi-tier deployments).
* Continuous delivery/continuous integration (CI/CD) pipelines.

In short, Ansible helps streamline IT operations, improves efficiency, and enables teams to focus on strategic initiatives instead of repetitive tasks.

Shell/python/ansible

Shell only works for linux operation sytems.

Python maintainance after sometime would be tough if we need to install modules also woul be tough.

If you want to communicate with api good to go with python ansible don’t have modules for it.

**Ansible passwordless authentication**

Passwordless authentication in Ansible typically refers to the ability to run tasks or playbooks on managed nodes (remote machines) without having to manually enter passwords. This is achieved by using SSH key-based authentication.

**Inventory**

Ansible **inventory** is a file or collection of sources that tells Ansible which machines (hosts) to manage. It defines groups, individual hosts, and related variables. Here's a simple breakdown of **types of inventories** and their explanation:

### ****1. Static Inventory****

A **static inventory** is a file (e.g., hosts or inventory.ini) where you manually list hosts and groups.

* **Format:** INI or YAML
* **Use Case:** Small, stable environments.

#### Example (INI format):

ini

Copy code

[web\_servers]

server1.example.com

server2.example.com

[db\_servers]

db1.example.com ansible\_user=dbadmin ansible\_port=2222

* web\_servers and db\_servers are **groups**.
* Hosts can have **variables** like ansible\_user (SSH user) and ansible\_port.

#### Example (YAML format):

yaml

Copy code

all:

children:

web\_servers:

hosts:

server1.example.com:

server2.example.com:

db\_servers:

hosts:

db1.example.com:

ansible\_user: dbadmin

ansible\_port: 2222

### ****2. Dynamic Inventory****

A **dynamic inventory** is generated **on the fly** using a script, plugin, or cloud service API (e.g., AWS, Azure).

* **Use Case:** Large or dynamic environments like cloud or container systems.

#### Example: AWS EC2 Inventory Plugin

* Ansible queries AWS to get all EC2 instances dynamically.
* No manual listing of hosts is required.

To use dynamic inventory:

bash

Copy code

ansible-inventory -i aws\_ec2.yaml --graph

**Adhoc Commands:(one line command to complete task)**

If you want to do simple tasks like installing apache server, installing or unistalling softwares.

Ansible ad-hoc commands allow you to run tasks without writing a full playbook. These commands are quick, one-off operations for performing actions on managed nodes.

Here’s an overview and some examples of **Ansible ad-hoc commands**.

## ****General Syntax****

bash

Copy code

ansible <group\_or\_host> -m <module> -a "<arguments>" -i <inventory>"

* **<group\_or\_host>**: Group name or hostname from the inventory.
* **-m**: Specifies the module to use.
* **-a**: Arguments for the module.
* **-i**: Path to the inventory file (optional if default is used).

Share

VR

##### You said:

difference between playbook and adhoc commands

###### ChatGPT said:

ChatGPT

The difference between **playbooks** and **ad-hoc commands** in **Ansible** primarily comes down to their scope, purpose, and how they are executed. Here's a breakdown:

### 1. ****Ad-hoc Commands****

* **Purpose**: Used for quick, one-off tasks that don’t need to be saved or reused.
* **Execution**: Run directly from the command line using the ansible command.
* **Scope**: Typically used for simple tasks, such as checking system status, copying a file, or restarting a service.
* **State**: Do not maintain state, as they are not declarative.
* **Syntax Example**:

bash

Copy code

ansible all -m ping

ansible webservers -a "systemctl restart apache2"

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

* + -m specifies the module to use.
  + -a passes arguments to the module.
* **When to Use**:
  + Quick tasks, troubleshooting, or testing.
  + When you don’t need to save or repeat the task later.

### 2. ****Playbooks****

* **Purpose**: Used for automating more complex, multi-step processes or workflows.
* **Execution**: Run using the ansible-playbook command.
* **Scope**: A YAML file that defines multiple tasks, configurations, or workflows in a structured and reusable way.
* **State**: Declarative and idempotent (ensures the desired system state).
* **Syntax Example**:

yaml

Copy code

---

- name: Install Apache and start service

hosts: webservers

become: yes

tasks:

- name: Install Apache

apt:

name: apache2

state: present

- name: Start Apache service

service:

name: apache2

state: started

* + Playbooks are written in YAML format.
  + They define tasks, loops, variables, conditionals, and handlers.
* **When to Use**:
  + Automating repetitive tasks.
  + Managing configuration across multiple servers.
  + Creating a repeatable and reusable infrastructure setup.

## ****Examples of Ansible Ad-hoc Commands****

### 1. ****Ping Hosts****

Checks connectivity to the managed nodes.

bash

Copy code

ansible all -m ping

* **all**: Target all hosts in the inventory.
* **ping**: Ansible module for connectivity testing.

### 2. ****Gather Facts****

Retrieve information (facts) about a host.

bash

Copy code

ansible all -m setup

### 3. ****Run Shell Commands****

Execute a command on remote nodes.

bash

Copy code

ansible webservers -m shell -a "uptime"

* **webservers**: Target hosts in the "webservers" group.
* **shell**: Run commands through the shell.
* **uptime**: Command to run.

### 4. ****Copy Files****

Copy a file from control node to a remote host.

bash

Copy code

ansible all -m copy -a "src=/path/to/file dest=/tmp/file mode=644"

### 5. ****Create or Remove a Directory****

bash

Copy code

# Create directory

ansible all -m file -a "path=/tmp/testdir state=directory mode=0755"

# Remove directory

ansible all -m file -a "path=/tmp/testdir state=absent"

### 6. ****Manage Packages****

#### Install a Package

bash

Copy code

ansible all -m yum -a "name=httpd state=present" # For CentOS/RHEL

ansible all -m apt -a "name=nginx state=present" # For Ubuntu/Debian

#### Remove a Package

bash

Copy code

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

### 7. ****Restart Services****

bash

Copy code

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

### 8. ****Change Permissions (Chmod)****

bash

Copy code

ansible all -m file -a "path=/tmp/testfile mode=0644"

### 9. ****Add or Remove Users****

#### Add a User

bash

Copy code

ansible all -m user -a "name=devops state=present"

#### Remove a User

bash

Copy code

ansible all -m user -a "name=devops state=absent"

### 10. ****Check Free Disk Space****

bash

Copy code

ansible all -m command -a "df -h"

### 11. ****Manage Cron Jobs****

Add a cron job:

bash

Copy code

ansible all -m cron -a "name='cleanup\_tmp' minute='0' hour='3' job='rm -rf /tmp/\*'"

### 12. ****Reboot a Server****

bash

Copy code

ansible all -m reboot

### ****Specifying Inventory File****

By default, Ansible uses /etc/ansible/hosts for inventory. To use a custom inventory file:

bash

Copy code

ansible all -i /path/to/inventory -m ping

## ****Key Modules for Ad-hoc Commands****

* **ping**: Test connection.
* **command**: Run a command (no shell features).
* **shell**: Run a command through a shell.
* **copy**: Copy files.
* **file**: Manage file attributes (directories, files, symlinks).
* **user**: Manage users.
* **service**: Manage services.
* **apt/yum**: Package management.
* **reboot**: Reboot systems.
* **cron**: Manage cron jobs.
* **setup**: Gather facts about a host.

### ****Examples with Specific Targets****

* **Target specific hosts**:

bash

Copy code

ansible webservers -m shell -a "df -h"

* **Using pattern matching**:

bash

Copy code

ansible 'web\*' -m ping

* **Run as a specific user**:

bash

Copy code

ansible all -b -m yum -a "name=vim state=present"

* + **-b**: Run the command with privilege escalation (e.g., sudo).

### ****Useful Tips****

* Add **-o** to display output in a single line:

bash

Copy code

ansible all -m ping -o

* Use **--limit** to restrict execution:

bash

Copy code

ansible all --limit "host1,host2" -m ping

Ad-hoc commands are great for quick tasks, but for more complex automation, **playbooks** are preferred.

4o

##### You said:

gather facts ansible

###### ChatGPT said:

ChatGPT

In Ansible, **gathering facts** refers to collecting system information about managed nodes. This information, known as **facts**, includes details like the OS, IP address, CPU, memory, disk space, and other system-specific attributes.

## ****How to Gather Facts****

### 1. ****Default Fact Gathering in Playbooks****

By default, Ansible gathers facts at the start of any playbook execution using the setup module.

Example Playbook:

yaml

Copy code

- hosts: all

tasks:

- name: Display gathered facts

debug:

var: ansible\_facts

### 2. ****Ad-hoc Command for Gathering Facts****

You can use the setup module with an ad-hoc command to gather and display facts.

bash

Copy code

ansible all -m setup

* **all**: Targets all hosts in the inventory.
* **setup**: The module used to gather system facts.

### 3. ****Filtering Specific Facts****

If you only need specific facts, you can filter them using the filter parameter.

#### Example: Filter facts related to "memory"

bash

Copy code

ansible all -m setup -a 'filter=ansible\_memory\*'

#### Example: Get network-related facts

bash

Copy code

ansible all -m setup -a 'filter=ansible\_default\_ipv4'

### 4. ****Disabling Fact Gathering****

If you don’t need facts gathered automatically in a playbook, you can disable it to save time.

yaml

Copy code

- hosts: all

gather\_facts: no

tasks:

- name: Task without facts

debug:

msg: "Facts gathering is disabled."

### 5. ****Using Gathered Facts****

Once gathered, the facts are stored in the ansible\_facts dictionary and can be used in your playbooks.

#### Example: Display the IP Address

yaml

Copy code

- hosts: all

tasks:

- name: Display the default IPv4 address

debug:

msg: "The IP address is {{ ansible\_facts['default\_ipv4']['address'] }}"

#### Example: Use CPU cores for decision-making

yaml

Copy code

- hosts: all

tasks:

- name: Install software if 4 or more CPUs exist

debug:

msg: "This host has {{ ansible\_processor\_cores }} CPU cores."

when: ansible\_processor\_cores >= 4

## ****List of Useful Facts****

Here are some commonly used facts:

| **Fact Key** | **Description** |
| --- | --- |
| ansible\_os\_family | OS family (e.g., RedHat, Debian). |
| ansible\_distribution | OS distribution name (e.g., Ubuntu). |
| ansible\_memtotal\_mb | Total memory (RAM) in MB. |
| ansible\_processor\_cores | Number of CPU cores. |
| ansible\_default\_ipv4 | Default IPv4 network details. |
| ansible\_architecture | System architecture (e.g., x86\_64). |
| ansible\_kernel | Kernel version. |
| ansible\_hostname | System hostname. |
| ansible\_date\_time | Date and time details. |

(Devops engineer work with adhoc commands to managing files, deleting files, updating files)

### Key Differences Summary:

| **Aspect** | **Ad-hoc Commands** | **Playbooks** |
| --- | --- | --- |
| **Use Case** | One-off, quick tasks | Complex, repeatable, multi-step tasks |
| **Execution** | Directly from CLI (ansible command) | Using YAML files (ansible-playbook) |
| **Idempotence** | No guarantee (depends on the task) | Yes, always ensures desired state |
| **State Management** | Imperative | Declarative and consistent |
| **Complexity** | Simple | Structured and supports advanced logic |
| **Reusability** | No | Yes, reusable workflows |
| **Syntax** | Inline arguments or simple flags | YAML-based structured tasks |
| **YAML** |  |  |

Yaml is much more human readable.

# Understanding YAML

YAML (YAML Ain't Markup Language) is a human-readable data serialization format that is commonly used for configuration files and data exchange between languages with different data structures.

## YAML Syntax

### Strings, Numbers and Booleans:

string: Hello, World!

number: 42

boolean: true

### List

fruits:

- Apple

- Orange

- Banana

### Dictionary

person:

name: John Doe

age: 30

city: New York

### List of dictionaries

YAML allows nesting of lists and dictionaries to represent more complex data.

family:

parents:

- name: Jane

age: 50

- name: John

age: 52

children:

- name: Jimmy

age: 22

- name: Jenny

age: 20

Ansible **modules** are small programs used to perform specific tasks within an Ansible playbook or ad-hoc commands. Modules are the building blocks of Ansible automation, allowing you to configure systems, deploy applications, and orchestrate IT tasks.

### ****Types of Ansible Modules****

Ansible modules are categorized based on their functions. Here are some of the key categories:

### 1. ****Core Modules****

These are the primary modules shipped with Ansible by default.

**Examples:**

* ping - Checks if the target host is reachable.
* setup - Gathers facts about the target system.
* command - Executes a command on the target system.
* shell - Runs shell commands on remote hosts.
* copy - Copies files to a remote machine.

### 2. ****System Modules****

These modules manage system-level tasks.

**Examples:**

* user - Manages user accounts.
* group - Manages groups on the system.
* service - Controls system services (e.g., start, stop, restart).
* cron - Manages cron jobs.

### 3. ****Files Modules****

Modules used for file and directory operations.

**Examples:**

* file - Sets permissions, creates, or removes files/directories.
* copy - Copies files from local to remote systems.
* fetch - Fetches files from remote systems to local systems.
* template - Deploys Jinja2 templates.

### 4. ****Package Management Modules****

Manages software packages on various systems.

**Examples:**

* apt - Manages Debian-based package installation.
* yum - Manages packages on RedHat-based systems.
* dnf - For managing packages on newer RedHat systems.
* pip - Manages Python packages.
* gem - Installs Ruby gems.

### 5. ****Cloud Modules****

These modules are used to manage cloud infrastructure.

**Examples:**

* aws\_ec2 - Manages AWS EC2 instances.
* azure\_rm - Manages resources on Microsoft Azure.
* gcp\_compute - Manages Google Cloud Platform resources.
* openstack - Manages OpenStack cloud resources.

### 6. ****Networking Modules****

Manages network devices and configurations.

**Examples:**

* ios\_config - Configures Cisco IOS devices.
* nxos\_config - Configures Cisco NX-OS devices.
* net\_ping - Pings hosts from network devices.
* juniper\_junos\_config - Configures Juniper devices.

### 7. ****Database Modules****

Manages databases like MySQL, PostgreSQL, etc.

**Examples:**

* mysql\_db - Manages MySQL databases.
* postgresql\_db - Manages PostgreSQL databases.
* mongodb\_user - Manages MongoDB users.

### 8. ****Cloud Automation Modules****

For automating cloud-based deployments and services.

**Examples:**

* k8s - Manages Kubernetes resources.
* docker\_container - Manages Docker containers.
* terraform - Executes Terraform automation.

### 9. ****Utilities Modules****

General-purpose modules for diverse tasks.

**Examples:**

* wait\_for - Waits for conditions like ports or files to become available.
* uri - Interacts with HTTP/HTTPS endpoints.
* set\_fact - Defines custom variables dynamically.

### ****Custom Modules****

If Ansible's built-in modules don't suit your needs, you can create **custom modules** in Python, Bash, or any scripting language.

#### ****Example of a Simple Custom Module****

Custom module example in Python:

python

Copy code

#!/usr/bin/python

import json

print(json.dumps({

"changed": True,

"msg": "Hello from custom module!"

}))

Place it in the library/ directory of your playbook to make it usable.

### ****How to Use Ansible Modules****

#### Example Playbook:

yaml

Copy code

- name: Example playbook using modules

hosts: all

tasks:

- name: Ping the hosts

ansible.builtin.ping:

- name: Install Apache package

ansible.builtin.yum:

name: httpd

state: present

- name: Start Apache service

ansible.builtin.service:

name: httpd

state: started

enabled: true

### Key Notes:

1. **Module Arguments** - Modules accept specific parameters.
2. **Idempotency** - Ansible modules ensure that running the same playbook multiple times does not cause changes unless required.
3. **Return Values** - Modules return JSON data that can be processed in playbooks.