**What is Ansible?**

* **Ansible** is an **open-source automation tool** developed by Red Hat.
* It is used for **IT automation, configuration management, provisioning, orchestration, and application deployment**.
* Ansible is **agentless** (no software/agent needs to be installed on client machines). It uses **SSH or WinRM** to communicate with systems.
* Configuration is defined in **YAML playbooks**, which makes it easy to read and write.

👉 Example Use Cases:

* Installing packages (Apache, Nginx, MySQL, etc.)
* Managing users, groups, and permissions
* Deploying applications
* Automating cloud provisioning (AWS, Azure, GCP)
* Orchestrating multi-tier environments

**What is Configuration Management?**

* **Configuration Management (CM)** is the practice of **automating the setup, maintenance, and consistency of systems**.
* It ensures that servers and applications are configured in a **desired, consistent state** across environments (Dev, QA, Prod).
* Instead of manually installing software and editing configs, CM tools (like **Ansible, Puppet, Chef, SaltStack**) automate these tasks.

👉 Key Benefits:

* **Consistency** → Every server is configured the same way, reducing "works on my machine" issues.
* **Scalability** → Manage 100s/1000s of servers with one playbook/policy.
* **Version Control** → Configurations stored as code ("Infrastructure as Code").
* **Auditing & Compliance** → Easily verify if systems meet security/compliance standards.

**Ansible as a Configuration Management Tool**

* Ansible is widely used for configuration management because:
  + It ensures systems are in the correct state (e.g., a package is installed, a service is running).
  + It’s **declarative** → you describe the state, Ansible ensures the system matches it.
  + Example: If you say "Nginx should be installed and running," Ansible checks and applies changes only if needed (**idempotent**).

✅ **In short**:

* **Ansible** = Automation tool (agentless, YAML-based).
* **Configuration Management** = Keeping IT systems in a consistent, desired state.
* **Ansible is one of the most popular tools used for configuration management.**

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

👉 IaC is about **provisioning and managing infrastructure** (servers, networks, storage, databases, cloud resources) using code instead of manual processes.

* **Focus**: Infrastructure setup (creating servers, VMs, load balancers, VPCs, etc.).
* **Goal**: Ensure infra is reproducible, consistent, and version-controlled.
* **Tools**:
  + Terraform
  + AWS CloudFormation
  + Pulumi
  + Azure Resource Manager (ARM)

✅ Example (Terraform):

resource "aws\_instance" "web" {

ami = "ami-0abcd1234"

instance\_type = "t2.micro"

}

👉 This code provisions an EC2 instance.

**🔹 2. Configuration Management (CM)**

👉 CM is about **configuring and managing the software and system state** of already provisioned infrastructure.

* **Focus**: Application setup (installing packages, updating configs, managing users, ensuring state).
* **Goal**: Keep environments consistent (Dev, QA, Prod).
* **Tools**:
  + Ansible
  + Puppet
  + Chef
  + SaltStack

✅ Example (Ansible Playbook):

- hosts: webservers

tasks:

- name: Install Apache

apt:

name: apache2

state: present

👉 This installs Apache on already provisioned servers.

**🔹 3. Key Differences**

| **Feature** | **Infrastructure as Code (IaC)** | **Configuration Management (CM)** |
| --- | --- | --- |
| **Scope** | Creates & manages infra (VMs, networks, storage, cloud services) | Manages software, OS config, services on infra |
| **Stage** | “Day 0 / Day 1” → provisioning stage | “Day 2+” → maintaining system state |
| **Idempotency** | Recreates infra from scratch if drift occurs | Enforces desired state without full rebuild |
| **Tools** | Terraform, CloudFormation, Pulumi | Ansible, Puppet, Chef, SaltStack |
| **Output** | Infrastructure resources | Configured servers/apps |

**🔹 4. How They Work Together**

In modern DevOps, **IaC + CM are often combined**:

1. **IaC** provisions servers, databases, and networks.
2. **CM** installs applications, configures services, and applies security hardening.

👉 Example Workflow:

* Terraform provisions **3 EC2 instances + load balancer**.
* Ansible configures **Nginx, deploys app code, sets firewall rules**.

✅ In short:

* **IaC = Build the house (infrastructure).**
* **CM = Furnish and maintain the house (software, configs).**

Ansible has broad use cases across **IT Operations, DevOps, Patching, and Compliance**. Let’s break it down:

**🔹 1. IT Operations Use Cases**

* **Server Provisioning** → Spin up and configure Linux/Windows servers in AWS, Azure, GCP, VMware, etc.
* **Application Deployment** → Deploy Java, Python, Node.js, or .NET apps consistently across environments.
* **Service Management** → Ensure services like Nginx, Apache, MySQL are always running with the right configs.
* **User & Access Management** → Create users, groups, and manage SSH keys/passwords.
* **Networking Automation** → Configure routers, switches, and firewalls (Cisco, Juniper, F5).

**🔹 2. DevOps Use Cases**

* **CI/CD Pipelines** → Integrate with Jenkins, GitHub Actions, or GitLab CI to deploy code automatically.
* **Infrastructure as Code (IaC)** → Manage infrastructure state using YAML playbooks.
* **Container Orchestration** → Automate Kubernetes (K8s) deployments, Helm charts, Docker container setups.
* **Multi-tier Orchestration** → Bring up a full stack (DB → Backend → Frontend → Load Balancer) in one playbook.
* **Secrets Management** → Integrate with Vault, AWS Secrets Manager, or Ansible Vault to manage credentials securely.

**🔹 3. Patching Use Cases**

* **OS Patch Management** → Apply latest security and kernel updates on Linux/Windows servers.
* **Automated Patch Scheduling** → Run patch playbooks during maintenance windows.
* **Selective Patching** → Patch only critical CVEs or specific services (like OpenSSL).
* **Rolling Updates** → Update servers in batches (avoid downtime in production).
* **Patch Reporting** → Generate reports of patched vs unpatched servers.

**🔹 4. Compliance & Security Use Cases**

* **Baseline Configuration Enforcement** → Ensure all servers follow CIS, NIST, ISO, HIPAA standards.
* **Policy as Code** → Define compliance rules in playbooks (e.g., password length, disabled root login).
* **Audit & Drift Detection** → Check for deviations from approved configurations.
* **Firewall & Security Hardening** → Apply standard iptables/firewall rules, disable insecure protocols (Telnet, FTP).
* **Log & Monitoring Setup** → Deploy and configure monitoring/alerting tools (ELK, Prometheus, Splunk).

✅ **Summary**:

* **IT** → Provisioning, deployments, user & service management.
* **DevOps** → CI/CD, IaC, container automation, orchestration.
* **Patching** → OS/software updates, patch scheduling, rolling updates.
* **Compliance** → Security baselines, auditing, hardening, reporting.

**Ansible Architecture**

Ansible follows a **simple, agentless architecture**. It doesn’t need any software/agent installed on client machines — it uses **SSH (Linux/Unix)** or **WinRM (Windows)** for communication.

**🔹 Main Components of Ansible**

**1. Control Node (Ansible Master)**

* The machine where **Ansible is installed**.
* Executes automation tasks by running **Playbooks**.
* Can be your laptop, a jump server, or a CI/CD server (like Jenkins).

**2. Managed Nodes (Clients/Targets)**

* The systems you want to configure/manage (Linux, Windows, network devices, cloud services).
* They don’t need Ansible installed — only **SSH/WinRM access**.

**3. Inventory**

* A file (INI, YAML, or dynamic) containing the list of **hosts/groups** Ansible will manage.
* Example (INI format):
* [webservers]
* web1 ansible\_host=192.168.1.10
* web2 ansible\_host=192.168.1.11
* [dbservers]
* db1 ansible\_host=192.168.1.20

**4. Playbooks**

* YAML files that define **desired configurations or tasks**.
* Example: Install and start Nginx:
* - name: Install and start Nginx
* hosts: webservers
* tasks:
* - name: Install Nginx
* apt:
* name: nginx
* state: present
* - name: Start Nginx
* service:
* name: nginx
* state: started

**5. Modules**

* Reusable units of code that perform specific tasks.
* Example: apt, yum, service, user, copy, git, docker\_container.
* Thousands of modules exist for OS, cloud, containers, and networking.

**6. Plugins**

* Extend Ansible’s functionality (logging, caching, callbacks).

**7. Ansible Galaxy**

* A community repository of **roles** (predefined automation).
* Example: ansible-galaxy install geerlingguy.nginx → installs an Nginx role.

**8. Roles**

* A structured way to organize playbooks (tasks, vars, handlers, templates).
* Useful for reusability and large-scale projects.

**9. Handlers**

* Special tasks triggered by notifications (e.g., restart service only if config changes).

**🔹 Communication Process (How Ansible Works)**

1. **User runs a Playbook** on the **Control Node** using ansible-playbook.
2. Ansible reads the **Inventory file** to determine which hosts to target.
3. The Control Node connects to **Managed Nodes** using **SSH (Linux)** or **WinRM (Windows)**.
4. Ansible sends **small modules (Python/PowerShell scripts)** to the Managed Node.
5. The module executes the task locally (e.g., install a package, start a service).
6. Results are sent back to the Control Node.
7. Modules are removed after execution (leaving no footprint).
8. Ansible ensures **idempotency** → only makes changes if required.

**🔹 High-Level Architecture Diagram (Textual)**

┌───────────────────────┐

│ Control Node │

│ (Ansible Installed) │

└─────────┬─────────────┘

│

┌─────────────┼───────────────────┐

│ │ │

┌───────────┐ ┌───────────┐ ┌───────────┐

│ Webserver │ │ DB Server │ │ Windows │

│ (Managed) │ │ (Managed) │ │ Server │

└───────────┘ └───────────┘ └───────────┘

^ ^ ^

│ │ │

(SSH) (SSH) (WinRM)

✅ **Summary:**

* **Control Node** runs playbooks.
* **Inventory** defines hosts.
* **Playbooks (YAML)** define what to do.
* **Modules** execute tasks.
* Communication happens via **SSH/WinRM**, making Ansible **agentless**.
* Ensures **idempotency** (same state, no duplicate changes).

**Ansible Advantages over Other CM Tools**

**1. Agentless Architecture**

* **Ansible**: No agents required. Uses **SSH (Linux)** or **WinRM (Windows)**.
* **Others**:
  + **Puppet & Chef** → Require agents running on each node + a master server.
  + **SaltStack** → Requires agents (minions).  
    ✅ Advantage: Easier setup, less overhead, fewer security concerns.

**2. Ease of Learning (YAML vs DSL)**

* **Ansible**: Uses **YAML (Playbooks)** → human-readable, easy for beginners.
* **Others**:
  + **Puppet** → Uses its own DSL (Domain Specific Language).
  + **Chef** → Uses Ruby DSL (developers love it, ops team struggles).  
    ✅ Advantage: Ops + Dev teams both can learn Ansible quickly.

**3. Idempotency Out-of-the-Box**

* **Ansible**: Ensures state automatically (won’t reinstall a package if it’s already installed).
* **Others**: Similar but often require more complex scripting or manifests.

**4. Simple Setup & Lightweight**

* **Ansible**: No special infrastructure needed → just install Ansible on a control node and start managing.
* **Others**: Puppet/Chef need Master–Agent setup, certificates, etc.  
  ✅ Advantage: Faster adoption, fewer resources needed.

**5. Better for Orchestration**

* **Ansible**: Not just configuration, but also **orchestration** (multi-tier deployments, app lifecycle, CI/CD).
* **Others**: Puppet and Chef focus more on config management; orchestration is weaker.

**6. Cloud & DevOps Integration**

* **Ansible**: Excellent cloud support (AWS, Azure, GCP) and works well in CI/CD pipelines (Jenkins, GitLab, etc.).
* **Others**: Cloud support exists but less straightforward.

**7. Security**

* **Ansible**: Uses **OpenSSH**, no agents → smaller attack surface.
* **Others**: Agents open more ports/services (Puppet Master, Chef Server).

**8. Community & Roles (Ansible Galaxy)**

* **Ansible Galaxy** → pre-built roles from community (like Docker, Kubernetes, Apache).
* Puppet has Forge, Chef has Supermarket, but **Ansible Galaxy is simpler & more widely adopted**.

**9. Push-Based Model (vs Pull-Based)**

* **Ansible**: **Push model** → Control Node pushes configs instantly.
* **Others**: Puppet & Chef → **Pull model** (agents periodically pull configs from master).  
  ✅ Advantage: Real-time execution and faster changes.

**10. Cost & Licensing**

* **Ansible**: Open-source, free. Enterprise version (Ansible Tower / AWX) for extra features.
* **Others**: Puppet & Chef Enterprise versions can be expensive.

**🔹 Summary Table**

| **Feature** | **Ansible (Best)** | **Puppet** | **Chef** | **SaltStack** |
| --- | --- | --- | --- | --- |
| **Architecture** | Agentless (SSH/WinRM) | Master–Agent | Master–Agent | Master–Minion |
| **Language** | YAML (easy) | DSL (Puppet DSL) | Ruby DSL | YAML + Python |
| **Setup Complexity** | Very simple | Medium–High | High | Medium |
| **Idempotency** | Yes (default) | Yes | Yes | Yes |
| **Orchestration** | Strong | Weak | Medium | Medium |
| **Push/Pull** | Push (real-time) | Pull (periodic) | Pull | Both (flexible) |
| **Community Support** | Strong (Galaxy) | Medium (Forge) | Medium (Supermarket) | Good |
| **Best For** | Cloud, DevOps, CI/CD | Large enterprise IT | Dev-heavy teams | Infra + networking |

✅ **In short**:

* **Ansible wins** in **simplicity, agentless design, orchestration, and DevOps/cloud integration.**
* **Puppet & Chef** are strong in **traditional enterprise environments** with large-scale, complex policies.
* **SaltStack** is good for **real-time infra and networking automation**.

**Ansible Prerequisites**

**1. Control Node (Where Ansible Runs)**

* **OS Support**: Linux/Unix (RHEL, CentOS, Ubuntu, Fedora, macOS).

Windows **cannot** be a control node.

* **Python**: Python **3.9 or higher** installed (Ansible is Python-based).
* **Ansible Installation**: Install via pip, apt, yum, or from source.
* **Connectivity**: Control Node must have **network access (SSH/WinRM)** to Managed Nodes.

**2. Managed Nodes (Target Systems)**

* **For Linux/Unix Nodes**:
  + SSH access (default port 22).
  + Python **2.7+ or 3.x** installed (most Linux distros already have it).
  + Sudo privileges if tasks need root access.
* **For Windows Nodes**:
  + Use **WinRM** (Windows Remote Management) instead of SSH.
  + PowerShell 3.0 or higher.
  + Proper authentication (username/password, Kerberos, or certificates).

**3. User Requirements**

* The Ansible Control Node user should have:
  + **SSH key-based authentication** (preferred) or password-based access to target nodes.
  + **Sudo privileges** on managed nodes if system-level changes are required.

**4. Networking Requirements**

* Control Node → Managed Node must have:
  + **SSH open** (Linux).
  + **WinRM open** (Windows, default ports 5985 HTTP / 5986 HTTPS).
* Proper **DNS or host resolution** for inventory hostnames.

**5. Inventory File**

* Define managed hosts in **INI or YAML format**.
* Example (/etc/ansible/hosts):
* [web]
* 192.168.1.10
* 192.168.1.11
* [db]
* 192.168.1.20 ansible\_user=admin ansible\_password=MyPass

**6. Optional (But Recommended)**

* **Version Control (Git)** → store playbooks and roles as code.
* **Ansible Galaxy** → use community roles.
* **Virtualenv (Python venv)** → isolate Ansible from system Python.
* **Ansible Tower/AWX** → GUI, RBAC, scheduling, reporting (for enterprises).

**✅ Quick Checklist**

* Control Node = Linux with Python + Ansible installed.
* Managed Node = Linux (SSH + Python) / Windows (WinRM + PowerShell).
* SSH/WinRM connectivity.
* Sudo/root privileges for automation.
* Inventory file ready.

**What is Ansible Inventory?**

* Inventory = A **list of target systems (managed nodes)** that Ansible will control.
* It can be written in **INI, YAML, or dynamically generated** from a script/Cloud API.
* By default, Ansible looks at /etc/ansible/hosts.
* You can group hosts (e.g., [webservers], [dbservers]) for easier management.

**🔹 Types of Ansible Inventory**

**1. Static Inventory**

* Manually defined in a file (hosts.ini or hosts.yml).
* Best for **small or stable environments**.

**Static Inventory Example (INI format)**

[webservers]

192.168.1.10 ansible\_user=ubuntu ansible\_ssh\_private\_key\_file=~/.ssh/id\_rsa

192.168.1.11 ansible\_user=ubuntu

[dbservers]

192.168.1.20 ansible\_user=admin ansible\_password=Passw0rd

**Static Inventory Example (YAML format)**

all:

children:

webservers:

hosts:

web1:

ansible\_host: 192.168.1.10

ansible\_user: ubuntu

web2:

ansible\_host: 192.168.1.11

ansible\_user: ubuntu

dbservers:

hosts:

db1:

ansible\_host: 192.168.1.20

ansible\_user: admin

👉 **Use case**: On-prem servers or lab environments where host IPs rarely change.

**2. Dynamic Inventory**

* Generated **on-the-fly** from cloud providers or external sources.
* Useful for **cloud environments** (AWS, Azure, GCP, OpenStack, VMware, Kubernetes) where servers scale dynamically.
* Instead of writing host IPs manually, Ansible queries the API to fetch live host details.

**How It Works**

* A **plugin or script** connects to the cloud provider.
* It retrieves the **list of running instances** and formats them as inventory for Ansible.

**Example: AWS Dynamic Inventory**

plugin: aws\_ec2

regions:

- ap-south-1

keyed\_groups:

- key: tags.Role

prefix: tag

* Here, Ansible will pull all running EC2 instances in ap-south-1 and group them by their **tags**.

**Example: Azure Dynamic Inventory**

plugin: azure\_rm

include\_vm\_resource\_groups:

- myResourceGroup

auth\_source: auto

👉 **Use case**:

* Auto-scaling environments in **AWS, Azure, GCP**.
* Kubernetes clusters where pods/nodes change dynamically.

**🔹 Key Differences: Static vs Dynamic**

| **Feature** | **Static Inventory** | **Dynamic Inventory** |
| --- | --- | --- |
| **Definition** | Hosts defined manually in a file | Hosts fetched dynamically from APIs/tools |
| **Best For** | Small, fixed environments (on-prem, labs) | Cloud/large-scale auto-scaling infra |
| **Scalability** | Low – must edit file for new hosts | High – auto-updates from provider |
| **Example** | hosts.ini file with IPs | AWS EC2, Azure RM, GCP, Kubernetes plugin |
| **Management Effort** | Manual | Automated |

✅ **Summary**:

* **Static Inventory** → manual, good for fixed infra.
* **Dynamic Inventory** → auto-discovered from cloud APIs, great for DevOps/cloud environments.

**Ad-hoc commands** are one of the first things you learn in Ansible. They’re used to run **one-time tasks quickly** without writing a Playbook.

**🔹 What are Ad-hoc Commands?**

* Syntax:
* ansible <host-pattern> -m <module> -a "<arguments>"
* **host-pattern** → group/host from inventory (like all, webservers, db1).
* **module** → the Ansible module to use (like ping, command, copy, yum).
* **arguments** → options for that module.

**🔹 Common Ad-hoc Command Examples**

**1. Check Connectivity (Ping)**

ansible all -m ping

ansible webservers -m ping

ansible dbservers -m ping

👉 Checks if all hosts in inventory are reachable via SSH.

**2. Run a Command**

ansible webservers -m command -a "uptime"

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

👉 Runs Linux commands (uptime, disk usage).

**3. Install a Package (Linux)**

ansible webservers -m yum -a "name=httpd state=present" --become

ansible dbservers -m apt -a "name=nginx state=latest" --become

👉 Installs Apache/NGINX on target hosts.

**4. Start/Stop/Restart a Service**

ansible webservers -m service -a "name=httpd state=started" --become

ansible webservers -m service -a "name=httpd state=restarted" --become

👉 Ensures webserver is running.

**5. Copy a File to Remote Hosts**

ansible all -m copy -a "src=/tmp/index.html dest=/var/www/html/index.html" --become

👉 Copies a local file to all managed nodes.

**6. Create a User**

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

👉 Creates a new user devops and adds it to the wheel group.

**7. Change File Permissions**

ansible all -m file -a "path=/tmp/test.sh mode=0755"

👉 Changes permissions on a file.

**8. Gather System Facts**

ansible all -m setup

👉 Collects detailed info (OS, IP, memory, CPU) about managed nodes.

**9. Reboot Servers**

ansible all -m reboot --become

👉 Reboots all servers gracefully.

**10. Check Installed Packages**

ansible webservers -m shell -a "rpm -qa | grep httpd"

👉 Runs shell command to check installed packages.

**🔹 Real-World Scenario**

👉 Imagine you’re a DevOps engineer managing **50 web servers**.

* Your manager asks:  
  *“Restart Nginx on all web servers now.”*

Without Ansible, you’d SSH into each server one by one 😓.  
With Ansible ad-hoc:

ansible webservers -m service -a "name=nginx state=restarted" --become

✨ Done in **seconds** across all servers.

✅ **Summary**:

* Ad-hoc commands = quick one-time tasks.
* Useful for **ping tests, package installs, service management, user creation, copying files, reboots**.
* Saves time vs manual SSH.

**More Useful Ansible Ad-hoc Commands**

**🔧 Package Management**

**Install multiple packages**

ansible all -m yum -a "name=httpd,mariadb-server state=present" --become

**Remove a package**

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

**🔧 Service Management**

**Enable a service on boot**

ansible webservers -m service -a "name=httpd enabled=yes" --become

**Stop a service**

ansible dbservers -m service -a "name=mysql state=stopped" --become

**🔧 File & Directory Management**

**Create a directory**

ansible all -m file -a "path=/opt/devops state=directory mode=0755" --become

**Remove a file/directory**

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

**Change file owner & group**

ansible all -m file -a "path=/var/www/html/index.html owner=apache group=apache" --become

**🔧 User & Group Management**

**Delete a user**

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

**Create a group**

ansible all -m group -a "name=developers state=present" --become

**🔧 Networking**

**Check open ports**

ansible all -m shell -a "netstat -tulnp | grep LISTEN"

**Test DNS resolution**

ansible all -m shell -a "nslookup google.com"

**🔧 Disk & Storage**

**Check disk usage**

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

**Mount a filesystem**

ansible all -m mount -a "path=/mnt/data src=/dev/sdb1 fstype=ext4 state=mounted" --become

**🔧 Archive & Compression**

**Extract a tar file**

ansible all -m unarchive -a "src=/tmp/app.tar.gz dest=/opt/app remote\_src=yes" --become

**Create a tar archive**

ansible all -m archive -a "path=/var/log dest=/tmp/logs.tar.gz" --become

**🔧 System Management**

**Check uptime**

ansible all -m command -a "uptime"

**Check memory usage**

ansible all -m shell -a "free -m"

**Check OS version**

ansible all -m shell -a "cat /etc/os-release"

**Reboot after updates**

ansible all -m reboot --become

**🔧 Copying & Fetching Files**

**Fetch log file from remote to control node**

ansible webservers -m fetch -a "src=/var/log/httpd/access.log dest=/tmp/logs/ flat=yes"

**Template with Jinja2**

ansible webservers -m template -a "src=nginx.conf.j2 dest=/etc/nginx/nginx.conf" --become

**🔧 Parallel Execution**

**Run with 10 hosts at a time**

ansible all -m ping -f 10

**✅ Summary**

* **System Checks** → uptime, memory, OS info
* **User/Group Management** → create/delete users & groups
* **Package/Service Mgmt** → install, update, restart
* **Files/Dirs** → copy, fetch, manage permissions
* **Networking** → DNS, ports, IP configs
* **Disk/Storage** → mount, tar, unarchive
* **Reboot & Maintenance**

**What is an Ansible Playbook?**

* A playbook is a YAML file (.yml or .yaml) that defines tasks to be executed on managed nodes.
* It’s more powerful than ad-hoc commands because it allows orchestration, multiple steps, conditionals, variables, loops, etc.

**Sample Ansible Playbook**

**Example 1: Install Apache Web Server**

---

- name: Install and Start Apache Web Server

hosts: webservers

become: yes # Run with sudo/root privileges

tasks:

- name: Install Apache

apt:

name: apache2

state: present

when: ansible\_os\_family == "Debian"

- name: Install Apache on RedHat

yum:

name: httpd

state: present

when: ansible\_os\_family == "RedHat"

- name: Ensure Apache is running

service:

name: "{{ 'apache2' if ansible\_os\_family == 'Debian' else 'httpd' }}"

state: started

enabled: yes

**Example 2: Create a User**

---

- name: Create DevOps User

hosts: all

become: yes

tasks:

- name: Add a new user

user:

name: devops

shell: /bin/bash

groups: sudo

state: present

**Example 3: Deploy a File**

---

- name: Deploy Configuration File

hosts: webservers

become: yes

tasks:

- name: Copy index.html

copy:

src: files/index.html

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

mode: '0644'

**How to Execute a Playbook**

1. **Run a playbook**
2. ansible-playbook install\_apache.yml
3. **Check playbook syntax**
4. ansible-playbook install\_apache.yml --syntax-check
5. **Dry run (check what changes would be made)**
6. ansible-playbook install\_apache.yml --check
7. **Limit execution to specific hosts**
8. ansible-playbook install\_apache.yml --limit web1
9. **Run with extra variables**
10. ansible-playbook deploy\_file.yml -e "filename=index.html"