# Ansible Fundamentals

---

1. What is Ansible?
2. Ansible Components
3. Ansible CLI
4. First Hands-On

**Ansible** is an open-source automation tool. It is a robot that does the repetitive work for you.

Instead of logging into 100 servers one by one to install software or change a setting, you write the instructions once, and Ansible pushes those changes to all 100 servers at the same time.

Ansible vs Jenkins

### What Ansible is for

- **Configuration management**
    - Install packages, configure services, edit files, manage users, deploy apps.
- **Idempotent changes**
    - “Make the server look like this,” safely repeatable.
    - An **Idempotent** change means that **performing an operation multiple times produces the same result as performing it once**
- **Runs tasks on machines**

    - Over SSH/WinRM (typically), against fleets of hosts.

## What Jenkins is for

- **CI/CD automation server**
    - Build, test, package, scan, and deploy based on triggers (git push, PR, schedule).
- **Pipeline and workflow**
    - Stages, approvals, artifacts, build logs, test reports, notifications.
- **Integrations**
    - GitHub/GitLab, Docker, Kubernetes, SonarQube, artifact repos, secrets tools, etc.

## Where they overlap

- Jenkins can “run scripts” and can call Ansible.
- Ansible can “deploy” code and can run from cron or manually.
- But Ansible alone does not naturally provide a full CI system: PR triggers, build queues, artifact storage, rich pipeline UI, test reporting, etc.

## Typical real-world setup

- **Jenkins does the pipeline**
    - Trigger on PR/merge -> build/test -> create artifact (Docker image) -> approval -> deploy.
- **Ansible does the deployment and server config**
    - Pull the built artifact/version and configure/roll it out across servers.

## Simple rule of thumb

- If you need **build/test/approval/pipeline visibility and triggers**: Jenkins (or Github actions ).
- If you need **repeatable infrastructure/app configuration on hosts**: Ansible.

Ansible vs Argocd

**Ansible**

- Push-based automation: a controller runs tasks and applies changes to targets.
- Great for VMs, bare metal, OS config, middleware setup, and also can apply K8s YAML.

**Argo CD**

- Kubernetes-only (practically): pull-based reconciliation.
- Continuously makes the cluster match what's in Git (desired state).

**Ansible vs puppet**

**Ansible**: push-based. Runs playbooks over SSH, applies desired state, can verify during the run, then stops. No continuous monitoring or enforcement unless you run it again.

**Puppet**: pull-based. Agent runs on servers periodically, continuously enforces desired state, detects drift and fixes it automatically.

**After deployment**

- With Ansible: if Nginx/container is later stopped or config is changed manually, Ansible will not know unless scheduled to run again.

- With Puppet: agent run will notice and revert to the defined state.

**Tower/AWX**

- Does not keep SSH connections alive.
- It schedules and runs Ansible repeatedly (like a controller), so you get periodic enforcement, not continuous.

**For “is it running” between runs**

- Use monitoring and service auto-restart (systemd, Docker restart policy, ASG health checks, etc.).

Ansible vs Terraform

**Ansible does not replace Terraform**

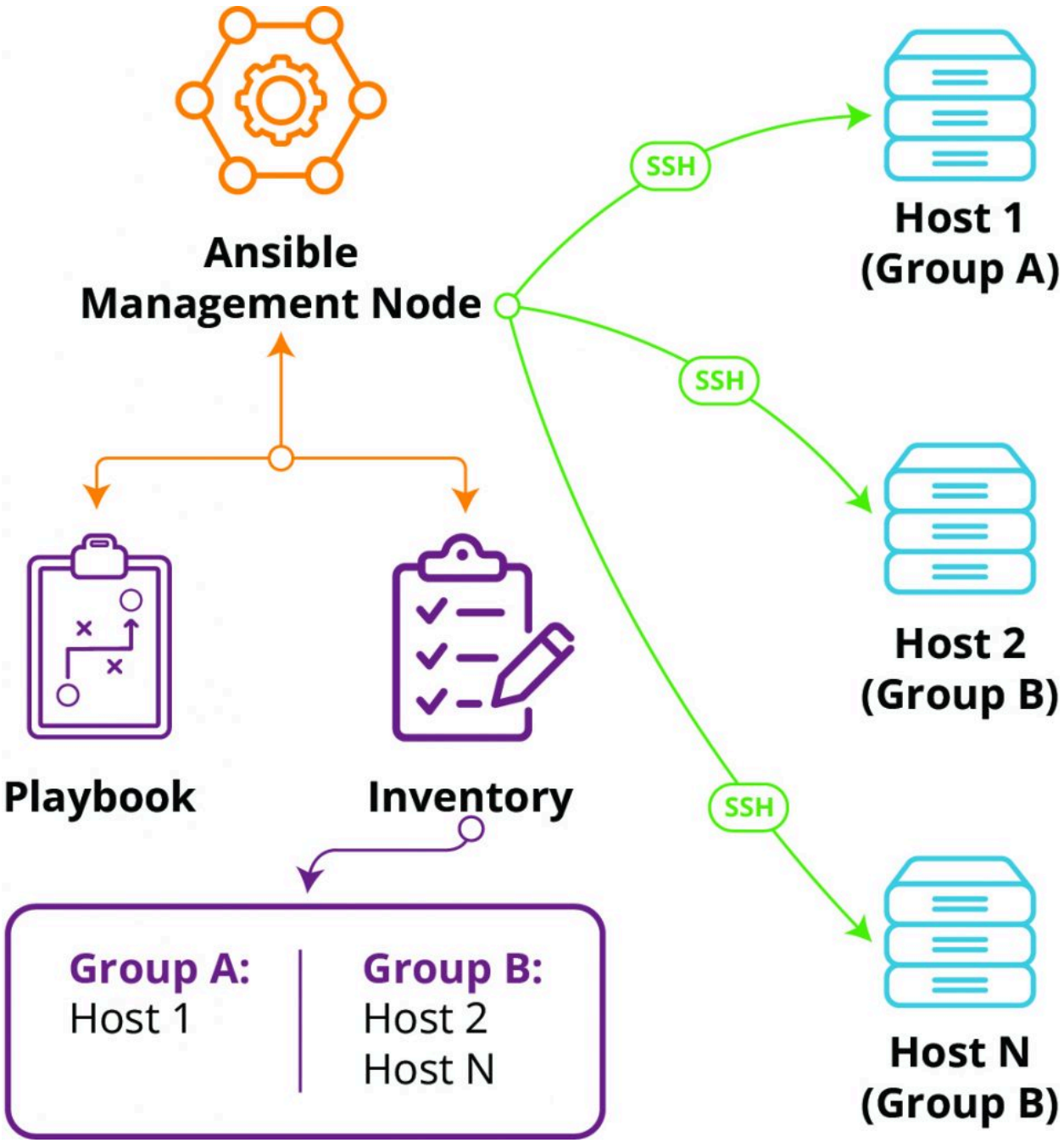
- Terraform: provisions cloud infrastructure (VPC, IAM, EC2, ALB, RDS) with state and plan.
- Ansible: configures what runs on those machines.

## Ansible components:

1. Modules -> simple unit that executes a task, like run nginx, configure nginx etc
2. Playbooks -> (collection of play)
3. Tasks -> action to be performed
4. Host keyword
5. Variables

6. Hosts details are stored in inventory file
7. Ansible tower: central UI dashboard

Collection of tasks (uses modules) -> play, collection of play -> playbooks



Hands-on:

## The 5 Core Commands

Teach these five binaries. They are the primary tools you interact with.

1. **`ansible`**: Runs a single, simple task (Ad-Hoc).
2. **`ansible-playbook`**: Runs a complex script (Playbook).
3. **`ansible-inventory`**: Views and debugs the host list.
4. **`ansible-doc`**: The built-in instruction manual for modules.
5. **`ansible-config`**: Manages configuration settings.

**Inventory:**

```
cat hosts.ini
```

Ask Ansible to parse it:

```
ansible-inventory -i hosts.ini --graph
```

**Modules:**

```
ansible web -i hosts.ini -m ping
```

-m ping

**Task**

A single instruction to run a module with specific arguments.

Run a task to install `git`:

```
ansible web -i hosts.ini -m apt -a "name=git state=present" -b
```

**Playbooks**

A file containing a list of tasks.

```
nano class_lab.yml
```

```
---
- name: Class Lab Setup
  hosts: web
  become: yes
  tasks:
    - name: Install Git
      apt:
        name: git
        state: present
```

**Host Keyword**

The line that maps the Playbook to the Inventory group.

In `class_lab.yml`, the line `hosts: web` tells Ansible to look at the `[web]` header in `hosts.ini`.

**Variables**

Values defined once and referenced multiple times.

```
---
- name: Class Lab Setup
  hosts: web
  become: yes
  vars:
```

```
    software_name: unzip    # <--- Variable Defined

  tasks:
    - name: Install Software
      apt:
        name: "{{ software_name }}"  # <--- Variable Used
        state: present
```

**Execution (Running the Playbook)**

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

1. The Connectivity Check

```
ansible web -i hosts.ini -m ping
```

2. File Transfer (Copy)
   Copy a file from your laptop (control node) to 100 remote servers:

```
ansible web -m copy -a "src=/etc/hosts dest=/tmp/hosts"
```

3. Installing a Package (Admin Task)
   Ensure Git is installed on all servers. `become`: This tells Ansible to run with sudo (root privileges).

```
ansible all -m apt -a "name=git state=present" --become
```

4. Checking System Status
   Check free memory on all web servers:

```
ansible web -m shell -a "free -m"
```

Can have pipes in shell, its more powerful than command module

```
ansible all -i inventory.ini -m command -a "ls /tmp"
```

No pipes, no redirects.

Q. If Ansible needs the Key to connect, how do you get the Key there in the first place?

You definitely do **not** log into 1000 servers manually. Here are the three industry-standard ways to solve this.

Method 1: The "Birth Certificate" (Cloud-Init / User Data)

Method 2: The "Golden Image" (AMI)

Method 3: The "Bootstrap" custom script