Lesson: Configuration Management with Ansible

What you'll learn

- What **configuration management** is and how Ansible does it.
- Why Ansible is **agentless** and works over plain SSH.
- The core pieces: **inventory**, **playbooks**, **tasks**, and **modules**.
- Why Ansible runs are **idempotent**, and what that looks like in practice.
- Where Ansible fits next to Terraform and cloud-init.

Skill gained: you can read a simple Ansible playbook, explain what it would do to a server, and describe how Ansible connects to and configures machines.

The lesson

> Note: Ansible is **not deployed in the lab yet**. This lesson teaches it conceptually with small standalone examples. The servers and IPs shown are illustrative.

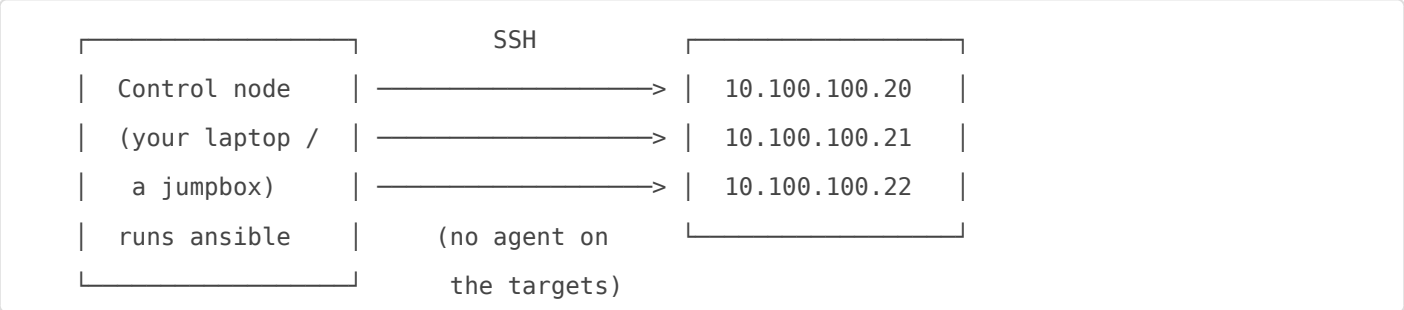
1. What configuration management means

Provisioning (Terraform, Chapter 2) creates the empty machine. **Configuration management** is everything that happens *inside* it afterward: install packages, write config files, create users, start services, keep them that way over time. Ansible is the most popular configuration-management tool.

Ansible is largely **declarative**: you describe the state you want ("nginx installed and running, config file present"), and Ansible checks the live machine and makes only the changes needed.

2. Agentless over SSH — the big idea

Many older config tools need an **agent** — a piece of software permanently running on every managed machine. Ansible doesn't. It is **agentless**: it connects to each machine over **SSH** (the same protocol you already use to log in), pushes over small instructions, runs them with Python, and disconnects.



Why this matters: nothing to install or maintain on the targets, and if a machine can be reached by SSH, Ansible can manage it. You run everything from one **control node** (your machine, or a bastion like the lab's **Jumpbox**).

3. Inventory: the list of machines

The **inventory** tells Ansible *which* machines to manage and how to group them. The simplest form is an INI-style file:

```
[webservers]
web1 ansible_host=10.100.100.20
web2 ansible_host=10.100.100.21

[dbservers]
db1 ansible_host=10.100.100.13

[all:vars]
ansible_user=omniops
```

Groups (`webservers`, `dbservers`) let you target many machines at once. You could also write inventory as YAML. In real setups, inventory can be **dynamic** — generated automatically from a cloud's API — but a static file is perfect to start.

4. Playbooks, plays, tasks, modules

The vocabulary, from biggest to smallest:

- **Playbook** — a YAML file describing what to do. The top-level thing you run.
- **Play** — a section of a playbook that maps a group of hosts to a list of tasks.
- **Task** — one unit of work ("install nginx").

- **Module** — the reusable code that actually performs a task (e.g. `apt`, `copy`, `service`). Ansible ships with thousands.

Here's a complete, readable playbook:

```
- name: Configure web servers
  hosts: webservers          # which group from inventory
  become: true               # use sudo
  tasks:

    - name: Install nginx
      ansible.builtin.apt:
        name: nginx
        state: present       # "present" = make sure it's installed

    - name: Deploy the site config
      ansible.builtin.copy:
        src: ./nginx.conf
        dest: /etc/nginx/nginx.conf
      notify: Restart nginx   # trigger a handler if this changed

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

  handlers:
    - name: Restart nginx
      ansible.builtin.service:
        name: nginx
        state: restarted
```

Read it top to bottom: it targets the `webservers` group, becomes root, installs nginx, copies a config, and ensures the service runs. A **handler** is a special task that runs only when notified (here: restart nginx *only if* the config actually changed).

5. Idempotency in action

Each module checks the machine's real state before acting. `state: present` means "make sure nginx is installed" — if it's already there, the module does nothing. This is **idempotency** (Chapter

1): run the playbook ten times, the server ends up identical every time.

When you run a playbook, Ansible reports a colored summary per host:

```
PLAY RECAP *********************************************************
web1 : ok=4    changed=1    unreachable=0    failed=0
web2 : ok=4    changed=0    unreachable=0    failed=0
```

- **ok** = task checked, state already correct or just made correct.
- **changed** = Ansible actually altered something.
- A second run on an already-correct server should show **changed=0** everywhere. That's the proof your playbook is idempotent.

6. Running it

Two commands cover the basics:

```
# Run a quick one-off command (the "ping" module just checks connectivity)
ansible all -i inventory.ini -m ping

# Run a full playbook
ansible-playbook -i inventory.ini site.yml
```

There's also `--check` (a dry run that reports what *would* change without changing it) — Ansible's equivalent of `terraform plan`. Use it when you're nervous about a change.

7. Where Ansible fits

Great at:

- Installing and configuring software inside existing machines.
- Pushing changes to many servers at once, repeatably.
- Ongoing management — re-run anytime to correct drift.
- Orchestrating multi-step rollouts (update these, then those).

Not its strong suit:

- Creating the underlying infrastructure (VMs, networks). It *can*, via cloud modules, but **Terraform** is usually cleaner for provisioning.
- Things that should only ever happen once at boot, where **cloud-init** is simpler.

The common combination: **Terraform** makes the VMs, **cloud-init** does minimal first-boot setup, and **Ansible** does the detailed, ongoing configuration. In the lab, you reach internal VMs through

the **Jumpbox** bastion — exactly the kind of place an Ansible control node would live. Chapter 5 covers choosing and combining tools.

Dig deeper

- Ansible getting started
- Ansible playbooks: intro
- How to build your inventory
- Ansible builtin modules index
- Ansible: check mode (dry run)

Search terms

- `ansible playbook tutorial for beginners`
- `ansible agentless ssh how it works`
- `ansible inventory file example`
- `ansible idempotent changed ok recap`
- `ansible modules apt copy service`

Check yourself

1. What does "configuration management" cover that provisioning does not?
2. What does "agentless" mean, and what protocol does Ansible use to reach machines?
3. Define inventory, playbook, task, and module in one sentence each.
4. On a second run against an already-configured server, what should the `changed` count be, and why?
5. Why might a team use Terraform *and* Ansible together rather than just one?

Revision #3
Created 2026-05-28 16:36:00 UTC by Eslam Shapsough
Updated 2026-05-28 16:40:00 UTC by Eslam Shapsough