Ansible Playbooks & Modules

1. What is a playbook?
2. Playbook Anatomy
3. Commonly Used Modules
4. Running Playbooks
5. Internal Process of Playbooks and Modules

Recap:

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

Method 2: The "Golden Image" (AMI)

Method 3: The "Bootstrap" custom script

Why Playbooks Over Ad-Hoc Commands?

- Order is manual
- No documentation
- Hard to reuse
- Unsafe for production

What is a Playbook?

A **Playbook** is the **Recipe**.

- It is a file where you write down every single step needed to configure a server.
- It is **repeatable**: You can run it today, tomorrow, or next year, and it will produce the exact same result.
- It is written in **YAML**, so it reads like English, not code.

Playbook Anatomy

A Playbook is like a sandwich. It has layers. Let's dissect one.

The Structure:

1. **Header (`---`)**: Marks the start of the YAML file.
2. **Target (`hosts`)**: Who are we cooking for? (e.g., `web` servers).
3. **Privilege (`become`)**: Do we need to be the boss? (`sudo` access).
4. **Tasks**: The actual steps (Install this, Copy that, Start this).

The Rules of YAML:

- **Indentation is Life:** You cannot use Tabs. You must use Spaces (usually 2 spaces per indent).
- **Lists:** Use a dash (`-`) to make a list.
- **Key-Value Pairs:** Use a colon (`key: value`).

Commonly Used Modules (The Toolbox)

In 90% of your work, you will use these 5 "Power Modules."

Module	What it does	Real-World Example
`apt`	Manages packages	"Install Nginx and Git."
`copy`	Moves files	"Push my `index.html` from my laptop to the server."
`file`	Manages file attributes	"Create a directory named `/data` and give it 755 permissions."
`service`	Manages states	"Start Nginx and make sure it starts automatically on reboot."
`user`	Manages people	"Create a user named `deploy`."

Hands-On: Writing "The Master Recipe"

Let's combine everything into one robust file. We will configure a complete Web Server with a custom page.

Step 1: Create the file on your Master Node.

```
nano master_setup.yml
```

Step 2: Type this in (Pay attention to the spacing!):

```
---
- name: Setup Production Web Server
  hosts: web
  become: yes  # We need root to install software

  tasks:
    # 1. Update the apt cache (like running 'apt update')
    - name: Update apt cache
      apt:
        update_cache: yes

    # 2. Install Nginx
    - name: Install Nginx
      apt:
        name: nginx
        state: present

    # 3. Create a custom folder
    - name: Create Website Directory
      file:
        path: /var/www/html/my_site
        state: directory
        mode: '0755'

    # 4. Copy our custom homepage
    # (We write content directly here to save time)
    - name: Deploy Home Page
      copy:
        content: "<h1>Hello from Ansible Class 2!</h1>"
        dest: /var/www/html/my_site/index.html

    # 5. Start the Service
```

```
    - name: Start Nginx
      service:
        name: nginx
        state: started
        enabled: yes
```

Running Playbooks

Now we execute the recipe.

Command:

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

While it runs:

1. **Gathering Facts:** Watch it pause at the start. It is "logging in" and checking the server details.
2. **Color Coding:**
    - **Yellow:** "I changed something" (e.g., installed Nginx).
    - **Green:** "It was already correct" (e.g., running it a second time).
    - **Red:** "Something failed."

Validation:

Open your browser or use curl to check if your custom site exists:

---

5. Internal Process (How it actually works)

This is the "Under the Hood" section that separates juniors from seniors.

When you hit Enter, Ansible performs a **4-Step Dance**:

1. Connect & Convert:
   Ansible reads your YAML playbook. It translates your apt task into a small Python script.
2. Transport (SFTP/SCP):
   It takes that small Python script and securely uploads it to the Target Server (usually to a hidden folder like ~/.ansible/tmp/).
3. Execute:
   It executes the Python script locally on the Target Server.
    - *Key Point:* The processing power is used on the Target, not the Master.

4. Cleanup:
   Once finished, it deletes the script from the Target and reports the JSON result back to the Master.

Q: "What actually happens on the target server when I run a playbook? Is there an agent running?"

> **Answer:** No agent is running. Ansible uploads a temporary Python script (module) to the target via SSH, executes it, and then deletes it immediately after the task is done.

Q: "If I change the content of my index.html in the playbook and run it again, what happens?"

> **Answer:** The copy module calculates the checksum (hash) of the file. It sees the local string is different from the file on the server. It will overwrite the file on the server and report "Changed" (Yellow).

Step-by-Step Internal Process

Step 1: Playbook Parsing (Control Node)

- Ansible reads the YAML playbook file.
- It validates:
    - Indentation
    - Syntax
    - Structure (plays, tasks, modules)

If parsing fails:

- Execution stops immediately
- No server is contacted

This is why `--syntax-check` is critical before production runs.

Step 2: Inventory Resolution

- Ansible loads the inventory file.
- It finds the host group defined in:

```
hosts: web
```

- Only those hosts are selected for execution.

Important concept:

> Playbooks never run on “all servers” by default.
> They only run on hosts resolved from the inventory.

Step 3: SSH Connection (Agentless Communication)

- Ansible connects to each target host using SSH.
- Authentication is done using:
  - SSH keys (recommended)
  - Passwords (not recommended)

Key point:

> Ansible does not require any agent or daemon on managed nodes.

Step 4: Privilege Escalation (become)

If the play contains:

```
become: true
```

- Ansible logs in as the remote user
- Uses `sudo` to gain root privileges
- Executes tasks as root

Without `becoming`, tasks like package installation will fail.

Step 5: Module Transfer to Managed Node

This is a critical concept.

- Ansible modules are **small programs**, usually written in Python.
- For each task:
    - The required module is copied to the remote host (temporary location)
    - The module is executed
    - The module is removed after execution

Important:

> Ansible does not run raw shell commands by default.
> It runs modules.
>
> When you run a task, Ansible is not typing `apt install nginx` into the shell. Instead, it packages a small piece of **Python code**, sends it to the target server, and asks the target's Python installation to run it.

Why is this better than raw Shell?

If Ansible just typed commands into the shell, it would be "dumb."

- **Shell:** If you run `mkdir /data` twice, the second time it crashes ("File exists").
- **Python Module:** The Python script checks *first*: "Does this directory exist?" If yes, it does nothing and reports "Success." If no, it creates it. This is how **Idempotency** works.

Step 6: Module Execution and Idempotency

The module:

- Checks the current state of the system
- Compares it with the desired state
- Makes changes only if required

Example:

```
apt:
  name: nginx
  state: present
```

Internally:

- If nginx is already installed → no action
- If not installed → install it

This logic is what provides **idempotency**.

Step 7: Result Collection and Output

- The module returns results in JSON format.
- The control node interprets the result.
- Output is displayed as:
    - ok
    - changed
    - failed

Verbose modes (`-v`, `-vv`, `-vvv`) show increasing levels of detail from this step.