Ansible Roles – Explained in Detail

Ansible Roles help you **organize** and **structure** your automation content. They are a **standard way** to package and reuse configuration — great for team collaboration and large projects.

🎯 Why Use Roles?

- ✅ Makes playbooks **clean and readable**
- 🔄 Encourages **reuse** of code
- 🔒 Separates **logic** from **data**
- 📁 Provides **directory structure** for better organization

🧱 Basic Role Directory Structure

When you create a role, it follows this standard layout:

```
my_role/
├── defaults/
│   └── main.yml        # Default variables
├── files/
│   └── <static files>   # Files to copy (e.g. config files)
├── handlers/
│   └── main.yml        # Handlers like restart service
├── meta/
│   └── main.yml        # Role dependencies
├── tasks/
│   └── main.yml        # Main list of tasks
```

```
├── templates/
│   └── <jinja2 templates>
├── vars/
│   └── main.yml        # Variables with higher priority
```

📘 Ansible Roles – Detailed Documentation

1. Introduction

When managing multiple applications—such as **HTML app**, **Angular app**, or **PHP app**—we often repeat many common tasks:

- Install HTTPD
- Start & enable HTTPD
- Install Git
- Copy code to the server
- Apply configurations
- Deploy the application

Instead of rewriting these tasks again and again for every application, **Ansible Roles** allow us to organize and reuse automation components in a clean structure.

Roles help us:

- Remove repetition

- Improve reusability
- Maintain clean playbooks
- Follow a modular automation style

A role contains reusable logic that can be used in any playbook by simply calling the role name.

---

2. Why Use Ansible Roles?

Without Roles

Every app would define the same steps repeatedly:

Install httpd
Start httpd
Enable httpd
Install git
Copy code
Apply config

This increases:

- Human errors
- Duplication
- Difficult maintenance

With Roles

We move repeated tasks into a reusable component called a **role**:

```
Role:
  - Install httpd
  - Start & enable httpd
  - Install git
  - Deploy code
```

Then for each app (HTML, Angular, PHP), we simply call the role:

```
roles:
  - httpd
  - angApp
  - phpApp
```

This makes playbooks highly modular.

---

3. Creating an Ansible Role

To create a role, we use:

```
ansible-galaxy init role1
```

This creates the structure:

```
role1/
 ├── tasks/
 ├── handlers/
 ├── templates/
 ├── files/
 ├── vars/
 ├── defaults/
 ├── meta/
```

Each directory has a purpose:

Folder	Purpose
tasks	Core execution logic
handlers	Restart/Reload actions
templates	Jinja2 templates
files	Static files
vars	Hard variables
defaults	Default variables
meta	Role metadata

---

4. Defining Tasks (tasks/main.yml)

Example: Install HTTPD, start service, enable service.

Create a file: install.yml

```
- name: Install apache2
  package:
    name: apache2
    state: present
```

```
- name: Start apache2
  service:
    name: apache2
    state: started

- name: Enable apache2
  service:
    name: apache2
    enabled: yes
```

Create config task file:

configs.yml

```
- name: Ensure SELinux Permissive
  selinux:
    policy: targeted
    state: permissive

- name: Ensure config updated
  template:
    src: httpd.j2
    dest: /etc/httpd/conf/httpd.conf

- name: Deploy sample page
  copy:
    src: info.html
    dest: /var/www/html/info.html
```

---

5. Templates (templates/httpd.j2)

Go to templates directory:

```
cd templates
```

Download a base config:

```
sudo yum -y install wget
wget https://raw.githubusercontent.com/ansible/ansible/master/lib/httpd.conf
```

Edit:

Replace:

```
Listen 80
```

with:

```
Listen {{ http_port }}
```

---

6. Variables (vars/main.yml)

```
http_port: 8080
```

This makes the service listen on port 8080 dynamically.

---

7. Adding Static Files

Go to `/files` folder:

cd files

Create file:

`info.html`

<h1>This is a static page</h1>

Call it from config task file:

```
copy:
  src: info.html
  dest: /var/www/html/info.html
```

---

8. Creating Handlers

Go to handlers folder:

cd handlers

Create:

main.yml

```
- name: Restart HTTPD
  service:
    name: httpd
    state: restarted
```

---

9. Linking Handlers to Tasks

Edit tasks/configs.yml:

```
- name: Update config
  template:
    src: httpd.j2
    dest: /etc/httpd/conf/httpd.conf
  notify: Restart HTTPD
```

---

10. Main Task File

In `tasks/main.yml`:

```
- import_tasks: install.yml
- import_tasks: configs.yml
```

This executes tasks in order.

---

11. Calling the Role From Playbook

Create a playbook:

`web-setup.yml`

```
- hosts: all
  become: yes
  roles:
    - role1
```

Run the playbook:

```
ansible-playbook web-setup.yml
```

---

12. Creating More Roles (Example: Angular App)

```
ansible-galaxy init angular-App
```

Then create tasks, templates, handlers same as role1.

Create a combined playbook:

```
- hosts: all
  become: yes
  roles:
    - role1
    - angular-App
```

You can add multiple roles based on requirement.

---

13. Updating Variables

Edit:

```
role1/vars/main.yml
```

Change:

```
http_port: 8080
```

to:

```
http_port: 80
```

Run playbook again:

```
ansible-playbook web-setup.yml
```

The role picks updated variables.

---

14. Final Notes

- Roles allow you to break a large playbook into **modular components**.
- Roles are reusable across:
  - Apache apps
  - Node apps
  - Angular apps
  - Java apps
  - PHP apps

- You can add new roles anytime.
- Just plug the role into your playbook.

---

15. Summary

Feature	Benefit
Multiple task files	Cleaner structure
Templates	Dynamic configs
Handlers	Automated restart
Vars	Application customization
Roles	Reusability & modular automation
