🧪Lab 14.2 — Ansible Roles (Deep Dive & Best Practices)

Author: *Sandeep Kumar Sharma*

🕗Learning Objectives

In this lab you will learn: - Full role anatomy and each directory purpose in detail - Variable precedence and where to put variables (`defaults` vs `vars`) - How handlers, files, templates, and tasks interact inside a role - Role parameters and how to pass variables to roles - Using `meta/main.yml` for role dependencies and metadata - Importing/including roles dynamically (`include_role`, `import_role`) - Best practices for authoring reusable roles - How to test a role quickly using a small playbook

---

👙Learning Outcomes

After completing this lab you will be able to: - Create production-ready roles with clear separation of concerns - Decide where to place variables and why - Use handlers and templates correctly inside roles - Compose roles with dependencies and reuse them across projects - Apply tags and limits to execute parts of a role

---

🧱Role Anatomy — Detailed Explanation

A role is a filesystem layout which Ansible understands. A typical role looks like:

```
my_role/
├── defaults/
│   └── main.yml          # Lowest precedence variables (safe to override)
├── vars/
│   └── main.yml          # Higher precedence variables (harder to override)
├── tasks/
│   └── main.yml          # Task list executed by the role
├── handlers/
│   └── main.yml          # Handlers triggered by notify
├── templates/            # Jinja2 templates (.j2)
├── files/                # Static files to copy with the copy module
├── meta/
│   └── main.yml          # Role metadata and dependencies
```

```
├── tests/
│   └── test.yml          # Optional small playbook used for manual testing
└── README.md             # Role documentation (highly recommended)
```

📌Purpose of Each Directory (Expanded)

- **defaults/** — Variables here have the *lowest* precedence. Ideal for configurable defaults that users can override via inventory, extra vars, or group_vars.
- **vars/** — Variables here have *higher* precedence than defaults and are generally used for internal values that should not be easily overridden.
- **tasks/** — The actions the role performs. Keep `main.yml` concise and break complex logic into multiple task files included with `include_tasks`.
- **handlers/** — Actions that run only when notified by tasks (e.g., restart service after config change).
- **templates/** — Dynamic configuration files using Jinja2. Use `template` module to render them.
- **files/** — Static files you want to copy as-is with the `copy` module.
- **meta/** — Contains `main.yml` with metadata (author, license) and `dependencies` — other roles that must be applied first.
- **tests/** — Small playbooks to exercise the role manually; recommended before adding automated tests.
- **README.md** — Document variables, examples, and usage instructions.

Variable Precedence — Quick Reference

From lowest to highest precedence (only relevant items shown): 1. defaults role variables (defaults/main.yml) 2. inventory group_vars 3. inventory host_vars 4. playbook `vars_files` 5. playbook `vars` 6. extra vars (`ansible-playbook -e`) — highest priority

Note: `vars/main.yml` inside a role has higher precedence than `defaults/main.yml` and will typically override defaults. Use `defaults` for values you expect users to change.

⚙️Role Variables — Where to put what

- Use **defaults/** for tunable settings (package name, ports, paths).
- Use **vars/** for internal constants or computed values that you do not want overridden easily.
- Do NOT place secrets in `defaults` or `vars` — use Ansible Vault or external secret managers.

🔦Handlers, Templates & Files Interaction

- Tasks modify or deploy a file/template and `notify` a handler.
- Handlers are defined in `handlers/main.yml` inside the role.
- Templates live under `templates/` and are deployed with the `template` module.
- Files that must be copied untouched go to `files/` and are used with the `copy` module.

Example sequence inside a role: 1. Install package 2. Deploy template to config path (template) 3. Notify handler to restart service (notify) 4. Handler runs once at the end of the role if any notify triggered

---

🧰Passing Variables to Roles (Role Parameters)

You can pass variables to a role from the playbook:

```
- hosts: dev
  roles:
    - role: my_role
      var1: value1
      var2: value2
```

These variables behave like play-level vars and will override `defaults` but can be overridden by extra vars.

📦Role Dependencies (`meta/main.yml`)

Define dependencies in `meta/main.yml` to ensure roles are applied in the correct order. Example:

```
# meta/main.yml
---
dependencies:
  - { role: common, some_var: 10 }
  - role: firewall
```

When the role with dependencies is applied, Ansible will run the dependent roles first.

🕐Including & Importing Roles Dynamically

- `import_role` — static, evaluated during playbook parsing

- `include_role` — dynamic, evaluated during runtime and supports `when` conditions

Example `include_role` with condition:

```
- name: Include role only when needed
  include_role:
    name: backup_role
  when: backup_enabled | default(false)
```

Use `include_role` when you want runtime decision making.

🧰Best Practices for Authoring Roles

1. **Keep roles small and single-purpose.** One role should do one job (e.g., `webserver`, `database`).
2. **Name variables with the role prefix** to avoid clashes: `webserver_port`, `webserver_user`.
3. **Provide sensible defaults** in `defaults/main.yml` and document them in README.md.
4. **Avoid hard-coded values**; prefer variables and defaults.
5. **Use handlers** for service restarts and ensure they are idempotent.
6. **Keep tasks idempotent** — consider `creates`, `removes`, or `when` checks.
7. **Use** `tags` in tasks to allow selective execution during development.
8. **Document the role**: variables, example usage, supported platforms, author.
9. **Store secrets outside the role** and use Vault when necessary.
10. **Use** `tests/` **playbooks** to validate role behavior quickly.

🧪Example — Full Role: `webserver_role` (Files & Snippets)

defaults/main.yml

```
---
webserver_package: "{{ 'apache2' if ansible_facts['os_family'] == 'Debian' else
'httpd' }}"
webserver_service: "{{ webserver_package }}"
webserver_listen_port: 80
```

tasks/main.yml

```yaml
---
- name: Install web package
  package:
    name: "{{ webserver_package }}"
    state: present

- name: Deploy web config
  template:
    src: web.conf.j2
    dest: "/etc/{{ 'apache2' if ansible_facts['os_family'] == 'Debian' else
'httpd' }}/conf.d/web.conf"
  notify: restart webserver

- name: Ensure document root exists
  file:
    path: /var/www/html
    state: directory
    mode: '0755'

- name: Deploy test index.html
  copy:
    src: index.html
    dest: /var/www/html/index.html
```

handlers/main.yml

```yaml
---
- name: restart webserver
  service:
    name: "{{ webserver_service }}"
    state: restarted
```

templates/web.conf.j2

```
Listen {{ webserver_listen_port }}
ServerName {{ inventory_hostname }}

<VirtualHost *:{{ webserver_listen_port }}>
  DocumentRoot /var/www/html
</VirtualHost>
```

files/index.html

```
<html>
  <body>
    <h1>Deployed by Ansible Role</h1>
  </body>
</html>
```

meta/main.yml

```
---
dependencies:
  - role: common
```

tests/test.yml (simple test playbook)

```
---
- hosts: dev
  roles:
    - role: webserver_role
```

🔗 How to run the role quickly

From the repository root:

```
ansible-playbook tests/test.yml
```

To pass custom variables:

```
ansible-playbook tests/test.yml -e "webserver_listen_port=8080"
```

🧾 Role Versioning & Distribution

- Use semantic versioning in your role repository tags (v1.0.0).
- Publish reusable roles to Ansible Galaxy or an internal Galaxy server for team reuse.
- Add `meta/main.yml` fields like `galaxy_info` to improve discovery.

🧰Testing Roles (Manual & Automated)

- Manual: use `tests/test.yml` as above.
- Automated: use Molecule (optional) for scenario-based tests and linting.

📚Documentation — What to include in README.md

- Role purpose
- Supported OS/families
- Variables (defaults + description)
- Example usage
- Dependencies
- Author and license

👙 Lab Summary

This lab covered a deep, practical view of Ansible roles: structure, variable placement, handlers, templates, dependencies, best practices, and testing.

Author: Sandeep Kumar Sharma