🧪Lab 15 — Role Dependencies & Advanced Role Composition (Deep Concept + Easy Hands-On)

Author: *Sandeep Kumar Sharma*

🕗Learning Objectives

In this lab you will learn: - What role dependencies are in Ansible - Why dependencies help build modular, layered automation - How `meta/main.yml` controls role relationships - The difference between implicit vs explicit dependency loading - How to override dependent role variables - Best practices for structuring multi-role automation projects - How to test roles with dependencies easily

---

👙Learning Outcomes

After completing this lab, you will: - Understand advanced role composition - Use role dependencies inside `meta/main.yml` - Build multi-layered automation stacks - Organize roles into clean, reusable architecture - Pass variables into both parent and dependent roles

---

🧰What Are Role Dependencies?

Role dependencies allow one role to automatically pull in and execute **other roles** before itself.

A dependency says:

> "Before running this role, run the following roles first."

This helps break large automation into logical layers.

Example use cases: - Installing common packages before application deployment - Preparing OS hardening before installing Apache - Ensuring firewall is configured before webserver role runs - Setting up Python/Java before running an app role

Dependencies are defined inside:

```
roles/<role_name>/meta/main.yml
```

🧱How Dependencies Work (Deep Explanation)

Dependencies are loaded in the following manner: 1. When a role is invoked in a playbook, Ansible reads its **meta/main.yml**. 2. If dependency roles exist, they are executed **in the order defined**. 3. All dependency roles complete before the parent role begins. 4. Variable precedence applies as usual (parent playbook vars override dependency defaults).

Dependency execution is **idempotent** — roles run only if tasks require changes.

📦Dependency Syntax

Inside `meta/main.yml`:

```
---
dependencies:
  - role: common
  - role: firewall
    firewall_ports: [80, 443]
```

You can pass variables to dependency roles directly under each entry.

🧰Dependency Types — Conceptual Breakdown

1. Static dependencies (via `meta/main.yml`)

- Loaded at parse time
- Cannot be controlled by `when` conditions
- Best for strict requirements (package prerequisites, kernel settings)

2. Dynamic role inclusion (`include_role`)

- Loaded at runtime
- Supports conditions
- Best for optional functionality (e.g., “enable backup only if flag=true”)

3. Importing roles (`import_role`)

- Static like dependencies
- Evaluated early
- Allows splitting large playbooks

Example dynamic usage:

```
- include_role:
    name: backup_role
  when: backup_enabled
```

Where To Place Variables in a Dependency Tree

Parent playbook vars override everything

Variables flow in the following direction:

```
play vars
   ↓
role vars
   ↓
dependencies
```

Use `defaults/main.yml` in roles for low-priority values. Use `vars/main.yml` sparingly.

🧱Best Practices for Role Dependency Architecture

1. Keep roles independent and minimal.
2. Use dependencies only when sequencing is mandatory.
3. Do not create circular dependencies (A depends on B, B depends on A).
4. Document role dependencies in README.md.
5. Split large roles into sub-roles (e.g., `apache_install`, `apache_config`).
6. Use dynamic includes only when conditional execution is required.
7. Pass configuration parameters explicitly to dependent roles.

🛠️ Hands-On Exercise (Very Easy)

This hands-on demonstrates the concept using simple roles.

We will create: - **parent_role** → The main role - **common_role** → A dependency role that installs basic utilities

Step 1 — Create the dependency role

```
ansible-galaxy init common_role
```

Edit tasks:

```
nano common_role/tasks/main.yml
```

Add:

```
---
- name: Install common utilities
  package:
    name: "{{ item }}"
    state: present
  loop:
    - git
    - curl
    - wget
```

Step 2 — Create the parent role

```
ansible-galaxy init parent_role
```

Step 3 — Define dependency in parent role

Edit:

```
nano parent_role/meta/main.yml
```

Add:

```
---
dependencies:
  - role: common_role
```

Step 4 — Add a very simple task in parent role

```
nano parent_role/tasks/main.yml
```

Add:

```
---
- name: Display message from parent role
  debug:
    msg: "Parent role executed after dependency role"
```

Step 5 — Create test playbook

```
nano test-role-deps.yml
```

Add:

```
---
- hosts: dev
  become: yes
  roles:
    - parent_role
```

Step 6 — Run the playbook

```
ansible-playbook test-role-deps.yml
```

Expected Behavior

1. **common_role** runs first and installs basic utilities.
2. Then **parent_role** runs and prints the debug message.

This confirms the dependency chain works correctly.

🧪 Hands-On Checklist

- [ ] Create dependency role
- [ ] Create parent role
- [ ] Add dependency in meta file
- [ ] Run playbook to observe dependency ordering
- [ ] Validate common_role runs before parent_role

👙 Lab Summary

This lab provided a deep conceptual understanding of role dependencies, their usage, variable flow, and best practices — while keeping the hands-on portion simple and easy to execute.