1. What are Ansible Roles?

- Roles are a way to **organize** Ansible playbooks into reusable components.
- They allow **modularization** by grouping tasks, variables, handlers, templates, and files.
- Helps in **maintaining** and **scaling** automation.

---

2. Directory Structure of a Role

A role follows a structured directory format:

```
my_role/
|—— tasks/        # Main tasks to execute (main.yml)
|—— handlers/     # Handlers for notifying services (main.yml)
|—— templates/    # Jinja2 templates (.j2) for configurations
|—— files/        # Static files (e.g., binaries, scripts)
|—— vars/         # Role-specific variables (main.yml)
|—— defaults/     # Default variables (main.yml)
|—— meta/         # Metadata about the role (dependencies, author
info)
|—— README.md     # Documentation for the role
```

---

3. Creating a Role

You can create a role manually or using the `ansible-galaxy` command:

```
ansible-galaxy init my_role
```

This will generate the standard role structure.

---

4. Using Roles in a Playbook

To include a role in a playbook:

```
- name: Apply Web Server Role
```

```
  hosts: web_servers
  roles:
    - webserver
```

This will execute everything inside the `webserver` role.

---

5. Tasks in a Role

The `tasks/main.yml` file defines the main execution flow:

```
- name: Install Apache
  apt:
    name: apache2
    state: present

- name: Start Apache Service
  service:
    name: apache2
    state: started
    enabled: yes
```

---

6. Using Variables in a Role

Define default values inside `defaults/main.yml`:

```
apache_port: 80
```

Use the variable inside `tasks/main.yml`:

```
- name: Configure Apache to Listen on Port {{ apache_port }}
  template:
    src: apache.conf.j2
    dest: /etc/apache2/apache2.conf
```

---

7. Role Dependencies

Define dependencies inside `meta/main.yml`:

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

When `webserver` runs, `common` and `security` will be executed first.

---

8. Reusing Roles Across Playbooks

Instead of duplicating tasks, create common roles (e.g., `security`, `monitoring`) and reuse them in multiple playbooks.

---

9. Downloading Prebuilt Roles

You can install roles from **Ansible Galaxy**:

```
ansible-galaxy install geerlingguy.nginx
```

Then use it in a playbook:

```
- name: Deploy Nginx
  hosts: web_servers
  roles:
    - geerlingguy.nginx
```

---

10. Best Practices for Roles

✅ **Keep roles modular** – One role should perform one function.
✅ **Use variables wisely** – Place defaults in `defaults/` and allow overrides.

✅ **Use handlers for idempotency** – Restart services only when necessary.
✅ **Document your role** – Include a `README.md` for future reference.
✅ **Test roles independently** – Use `ansible-playbook -i inventory test_role.yml` for validation.

Conclusion

- **Ansible Roles** help in organizing and reusing automation code.
- They follow a **structured format** making them scalable and maintainable.
- Roles can be **shared**, **reused**, and **downloaded** from Ansible Galaxy.

Ansible Collections

1. What are Ansible Collections?

- **Ansible Collections** are a way to package and distribute **roles, modules, plugins, and playbooks** together.
- They help in **organizing** and **sharing** automation code more efficiently.
- Collections can be published and downloaded from **Ansible Galaxy** or private repositories.

---

2. Why Use Ansible Collections?

✅ **Modular and Organized:** Collections allow managing large projects easily.
✅ **Reusable Components:** Includes roles, plugins, and modules that can be reused.
✅ **Versioning & Updates:** Collections can be versioned, making updates easier.
✅ **Offline Usage:** Collections can be installed locally and used without an internet connection.

---

3. Structure of an Ansible Collection

A collection has a predefined directory structure:

```
my_collection/
```

```
|── plugins/      # Custom modules, filters, lookups, etc.
|── roles/        # Reusable Ansible roles
|── playbooks/    # Playbooks within the collection
|── docs/         # Documentation files
|── README.md     # Collection documentation
|── meta/         # Metadata (dependencies, author, version)
|── tests/        # Automated tests for the collection
```

---

4. How to Install an Ansible Collection?

a) Install from Ansible Galaxy

You can install collections from **Ansible Galaxy** using:

```
ansible-galaxy collection install community.general
```

To install a specific version:

```
ansible-galaxy collection install community.general:1.2.0
```

b) Install from a requirements file

Create a `requirements.yml` file:

```
collections:
```

```
  - name: community.general
    version: 1.2.0
  - name: ansible.windows
```

Then install:

```
ansible-galaxy collection install -r requirements.yml
```

c) Install from a Local File

If you have a `.tar.gz` collection file:

```
ansible-galaxy collection install my_collection.tar.gz
```

---

5. How to Use an Ansible Collection in a Playbook?

After installing a collection, you can reference its modules and roles.

a) Using Modules from a Collection

```
- name: Use a Module from a Collection
  hosts: localhost
  tasks:
    - name: Fetch System Information
      community.general.hostname:
```

```
        name: my-server
```

Here, `community.general.hostname` refers to the **hostname module** inside the **community.general collection**.

b) Using Roles from a Collection

```
- name: Use a Role from a Collection
  hosts: all
  roles:
    - name: my_namespace.my_collection.my_role
```

---

6. How to Create an Ansible Collection?

Step 1: Create a Collection Structure

Use the `ansible-galaxy collection init` command:

```
ansible-galaxy collection init my_namespace.my_collection
```

This will generate:

```
my_namespace-my_collection/
|—— plugins/
|—— roles/
|—— playbooks/
```

```
|── meta/
|── README.md
```

Step 2: Add Modules, Roles, or Plugins

Place your modules inside `plugins/modules/`, roles inside `roles/`, and playbooks inside `playbooks/`.

Step 3: Define Metadata in `meta/runtime.yml`

```
requires_ansible: ">=2.10"
dependencies:
  - ansible.builtin
  - community.general
```

Step 4: Build the Collection

```
ansible-galaxy collection build
```

This generates a `.tar.gz` package.

Step 5: Publish the Collection

To upload to **Ansible Galaxy**:

```
ansible-galaxy collection publish
my_namespace-my_collection-1.0.0.tar.gz
```

---

7. How to List Installed Collections?

To see all installed collections:

```
ansible-galaxy collection list
```

---

8. Best Practices for Using Ansible Collections

✅ **Use versioned collections** to avoid breaking changes.
✅ **Keep collections modular** and focused on specific tasks (e.g., `aws`, `security`).
✅ **Use `ansible.builtin` for core modules** to avoid unnecessary dependencies.
✅ **Use private collections** for company-specific automation needs.

---

9. Popular Ansible Collections

Here are some commonly used collections:

Collection Name	Purpose
`community.general`	General utilities (file management, networking, etc.)
`ansible.windows`	Windows automation
`amazon.aws`	AWS cloud automation

`kubernetes.core`	Kubernetes automation
`fortinet.fortios`	Fortinet firewall management

---

10. Summary

- **Ansible Collections** package **roles, modules, plugins, and playbooks** together.
- They help in **organizing** and **sharing** automation content.
- Collections can be installed from **Ansible Galaxy** or **private sources**.
- They make automation **scalable, modular, and reusable**.