DevOps Shack

Ansible Roles Documentation

Introduction to Ansible Roles

Ansible roles are a way to organize and structure playbooks into reusable components. Roles enable you to split complex playbooks into smaller, more manageable units that are easier to maintain and reuse. This modular approach allows you to encapsulate related tasks, variables, files, templates, and handlers into a single role, making it simpler to share and reuse across different projects and environments.

Roles are typically used in larger projects to manage configurations across multiple hosts and environments. With Ansible Galaxy, roles can also be shared publicly or privately and used as dependencies in other projects.

Why Use Roles?

- **Modularity:** Break down tasks into smaller, reusable components.
- **Organization:** Group related tasks, files, and templates, making them easier to manage and maintain.
- **Reusability:** Easily reuse roles across different projects or environments.
- **Collaboration:** Encourage sharing of roles between teams and projects, improving collaboration.
- **Separation of Concerns:** Keep your playbooks clean by offloading specific tasks into roles.

Structure of an Ansible Role

Each role follows a standardized directory structure. This helps Ansible know where to find certain files (like tasks, handlers, variables, etc.) when executing the role.

Role Directory Structure

The following is the default directory layout of a role:

```
roles/
  my_role/
    tasks/
    handlers/
    files/
    templates/
    vars/
    defaults/
    meta/
```

Here's a breakdown of each directory and its purpose:

1. **tasks/:**
    - This directory contains the main list of tasks to be executed by the role. The tasks are written in the main.yml file. Tasks are the primary actions, such as installing packages, creating files, or starting services.

Example:

```
# roles/my_role/tasks/main.yml
---
- name: Install NGINX
  apt:
    name: nginx
    state: present
```

2. **handlers/:**
    - Handlers are tasks that only run if they are triggered by another task. These are used for tasks that need to run after a certain change, like restarting a service after a configuration file is modified.

Example:

```
# roles/my_role/handlers/main.yml
---
- name: Restart NGINX
  service:
    name: nginx
    state: restarted
```

3. **files/:**
    - The files directory stores static files that will be copied to the managed nodes. These could be configuration files, scripts, or binaries.

Example:

```
# Usage in task
- name: Copy configuration file
  copy:
    src: myconfig.conf
    dest: /etc/nginx/nginx.conf
```

4. **templates/:**
    - This directory contains Jinja2 templates. Templates are used when you need to create a file dynamically based on variables or conditions.

Example:

```
# roles/my_role/templates/nginx.conf.j2
server {
    listen 80;
    server_name {{ server_name }};
}

# Usage in task
- name: Deploy NGINX config from template
  template:
    src: nginx.conf.j2
    dest: /etc/nginx/nginx.conf
```

5. **vars/:**
    - The vars directory contains variables that are specific to the role. These variables can be defined in main.yml and accessed within tasks, handlers, and templates.

Example:

```
# roles/my_role/vars/main.yml
---
server_name: example.com
```

6. **defaults/:**
    - This directory contains the default values for variables. Variables defined in defaults/main.yml have the lowest precedence, which means they can be easily overridden by playbooks or command-line arguments.

Example:

```
# roles/my_role/defaults/main.yml
---
nginx_port: 80
```

7. **meta/:**
    - The meta directory defines metadata about the role, such as dependencies on other roles. You can specify what other roles must be installed before this role can run.

Example:

```
# roles/my_role/meta/main.yml
---
dependencies:
 - { role: another_role }
```

Creating a Role

Step 1: Initialize a New Role

You can create a new role manually by creating the directory structure or by using the ansible-galaxy command:

```
ansible-galaxy init my_role
```

This command will generate the complete directory structure for you in the roles/my_role/ directory.

Step 2: Add Tasks

The heart of any role lies in its tasks. You add tasks to the tasks/main.yml file.

Example:

```
# roles/my_role/tasks/main.yml
---
- name: Install Apache
  apt:
    name: apache2
    state: present
  notify: Restart Apache
```

In this task, we are installing the Apache web server using the apt module. Additionally, the notify directive triggers a handler when the task causes a change.

Step 3: Add Handlers

Now, add the handler that will be triggered by the task in the handlers/main.yml file.

```
# roles/my_role/handlers/main.yml
---
- name: Restart Apache
  service:
    name: apache2
    state: restarted
```

This handler will restart the Apache service whenever the Install Apache task results in a change (e.g., installation of the package).

Step 4: Define Variables

Variables allow you to customize your role's behavior dynamically. Add variables in vars/main.yml:

```
# roles/my_role/vars/main.yml
---
document_root: /var/www/html
```

You can use this variable in tasks and templates. For example:

yaml

Copy code

```
# roles/my_role/tasks/main.yml
---
- name: Ensure the document root directory exists
  file:
    path: "{{ document_root }}"
    state: directory
```

Step 5: Add Templates

If you need to use templates, add them to the templates/ directory. Here's an example of an NGINX configuration file template:

```
# roles/my_role/templates/nginx.conf.j2
server {
    listen 80;
    server_name {{ server_name }};
    root {{ document_root }};
}
```

You can reference the template in your task:

yaml

Copy code

```
# roles/my_role/tasks/main.yml
---
- name: Configure NGINX with template
  template:
    src: nginx.conf.j2
    dest: /etc/nginx/nginx.conf
```

Step 6: Define Role Metadata

If your role has dependencies on other roles, you can define them in meta/main.yml:

```
# roles/my_role/meta/main.yml
---
dependencies:
  - role: geerlingguy.nginx
```

This ensures that the required role geerlingguy.nginx is installed and executed before your role.

Using Roles in Playbooks

Once your role is created, you can use it in a playbook like this:

```
---
- hosts: webservers
  roles:
    - my_role
```
If your role has variables, you can pass them in the playbook as well:
yaml
Copy code
```
---
- hosts: webservers
  roles:
    - role: my_role
      vars:
        document_root: /var/www/my_site
```

Role Dependencies

Roles can depend on other roles to function properly. You can define role dependencies in the meta/main.yml file under the dependencies section.

Example:

```
# roles/my_role/meta/main.yml
---
dependencies:
  - role: geerlingguy.apache
  - role: geerlingguy.mysql
```
This will ensure that the geerlingguy.apache and geerlingguy.mysql roles are installed and run before the my_role role.

Best Practices for Using Roles

1. **Modularize Playbooks**: Break down playbooks into logical roles to promote reuse and maintainability.
2. **Encapsulate Logic**: Avoid defining too many variables or tasks outside of the role. Keep logic encapsulated within the role.
3. **Use Defaults**: Place default variables in the defaults/ directory to allow for easy overrides.
4. **Document Roles**: Always include a README.md file in your role directory explaining its purpose, usage, and variables.
5. **Follow Conventions**: Use the standard Ansible role directory structure. This ensures that your roles are easily understandable by others.
6. **Idempotence**: Ensure that your roles are idempotent, meaning running the role multiple times should result in the same state without making unnecessary changes.

Sharing Roles via Ansible Galaxy

Ansible Galaxy is a platform for sharing Ansible roles. Once you've created a role, you can share it on Ansible Galaxy for others to use.

Step 1: Create a Galaxy Account

Go to Ansible Galaxy and create an account or log in with your GitHub credentials.

Step 2: Share a Role

You can upload roles from your GitHub repository to Ansible Galaxy by following the instructions on the Galaxy website.

Advanced Role Features

Role Dependencies with Versions

You can specify the version of a role to ensure that the correct version is used.

```
# roles/my_role/meta/main.yml
---
dependencies:
  - { role: geerlingguy.apache, version: "1.2.3" }
```

Role with Multiple Tasks Files

You can break down your tasks/ into multiple files for better organization.

Example:

```
# roles/my_role/tasks/main.yml
---
- include_tasks: setup.yml
- include_tasks: install.yml
- include_tasks: configure.yml
```

Conclusion

Ansible roles are a powerful way to structure and modularize your playbooks. By using roles, you can organize your configurations, share them with others, and reuse them across different projects. Following the standard directory structure, defining proper handlers, variables, and templates, and adhering to best practices will make your roles scalable, maintainable, and easy to understand.

Roles simplify large Ansible projects and make complex systems easier to manage and automate. With roles, you can also leverage the power of Ansible Galaxy to reuse community-contributed roles and focus on solving your specific use cases.