Ansible Roles

Ansible Advanced

Agenda

- Ansible Roles
- Using Roles
- Ansible Galaxy
- Sharing Roles
- Authoring Roles

Ansible Roles

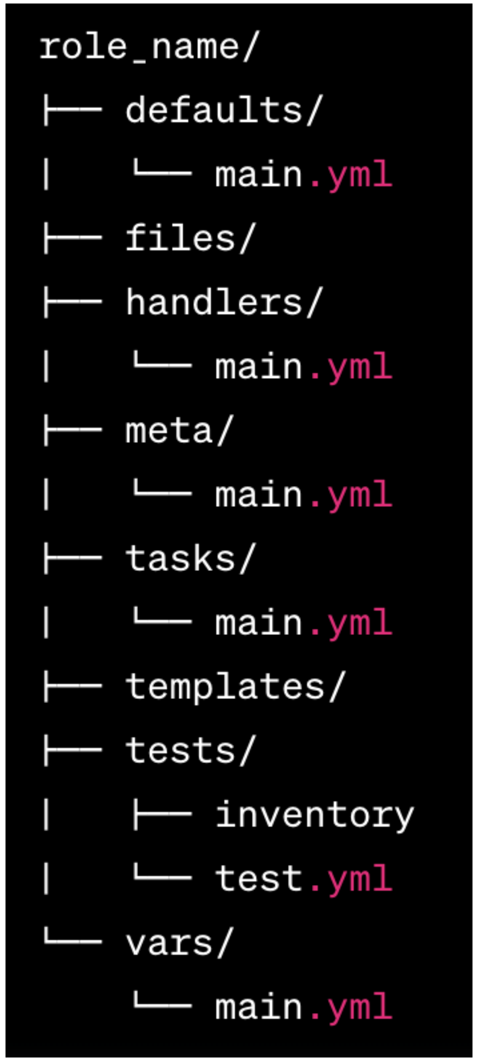
Ansible Advanced

What are Ansible Roles

- Ansible role is a pre-defined structure for organizing automation tasks in a way that allows for code reuse, ease of testing, and modular organization
- Roles encapsulate a specific piece of functionality or configuration, making it easier to drop that functionality into multiple playbooks or share with other users
- They are a key aspect of Ansible's playbook organization, enabling the reuse of code and modularization of configuration tasks
- These building blocks are easily shared using Ansible Galaxy

Role directory structure

- **`defaults/`**: Default variables for the role. These have the lowest priority
- **`files/`**: Contains files that the role can deploy onto the target system
- **`handlers/`**: Contains handlers, which are tasks that respond to a "notify" directive from other tasks
- **`meta/`**: Contains metadata about the role, such as role dependencies.
- **`tasks/`**: This directory contains the main list of tasks that the role will execute.
- **`templates/`**: Contains template files, which use the Jinja2 templating engine, and can be deployed using the template module
- **`tests/`**: Contains files for testing the role, often using tools like Molecule or simple playbooks.
- **`vars/`**: Variables for the role with higher priority than defaults.



```
role_name/
├── defaults/
|    └── main.yml
├── files/
├── handlers/
|    └── main.yml
├── meta/
|    └── main.yml
├── tasks/
|    └── main.yml
├── templates/
├── tests/
|    ├── inventory
|    └── test.yml
└── vars/
     └── main.yml
```

Benefits

- Modularity
  - Roles allow you to break down complex playbooks into smaller, reusable components
  - Each role focuses on a specific task or service, making it easier to understand and manage.
- Reusability
  - Once you've created a role, you can reuse it across multiple playbooks or even different projects
  - This reduces duplication and ensures consistency across your infrastructure.
- Sharing and Collaboration
  - Roles can be easily shared with others, either within your organization or with the broader community through platforms like Ansible Galaxy
  - This promotes collaboration and allows teams to benefit from the work of others.
- Version Control
  - Roles can be versioned, allowing you to track changes over time, roll back to previous versions if needed, and ensure that you're always using the correct version of a role.
- Separation of Concerns
  - By encapsulating specific functionalities into roles, you can separate the logic of your configuration from the data
  - This means you can use the same role in different environments (e.g., staging, production) with different variables.

Benefits

- Testing
  - Roles can be individually tested, ensuring that a specific piece of functionality works as expected
  - Tools like Molecule can be used to test roles in isolation, improving the reliability of your automation.
- Organization
  - Roles provide a standardized directory structure, making it easier to find where specific tasks, templates, files, and variables are defined
  - This structure improves readability and maintainability.
- Flexibility
  - Roles can be conditionally included or excluded in playbooks, and they can also depend on other roles
  - This allows for the creation of layered and flexible automation workflows.
- Role Dependencies
  - Roles can have dependencies on other roles, ensuring that prerequisites are always met
  - For example, a role that configures a web application might depend on another role that installs and configures the web server.
- Isolation
  - If a role is updated or modified, it won't affect other roles or playbooks unless they specifically depend on it
  - This isolation reduces the risk of unintended side effects when making changes

Using Roles

Ansible Advanced

How to use Roles

- First step is to get role files to execute on your playbook
- The default way to get files is to integrate with Ansible Galaxy
- After downloading the role you can use it directly on your playbook, like a task or na handler

PHP Role

- Get from Ansible Galaxy

```
ansible-galaxy install geerlingguy.php
```

- Add the role to playbook

```
- hosts: your_target_host
  become: yes  # Use elevated privileges
  vars:
    php_packages:
      - php
      - php-cli
      - php-fpm
      - php-json
      - php-common
      - php-mysql
      - php-zip
      - php-gd
      - php-mbstring
      - php-curl
      - php-xml
      - php-pear
      - php-bcmath
    php_memory_limit: "256M"
    php_max_execution_time: "60"
    php_upload_max_filesize: "32M"
    php_post_max_size: "32M"
  roles:
    - geerlingguy.php
```

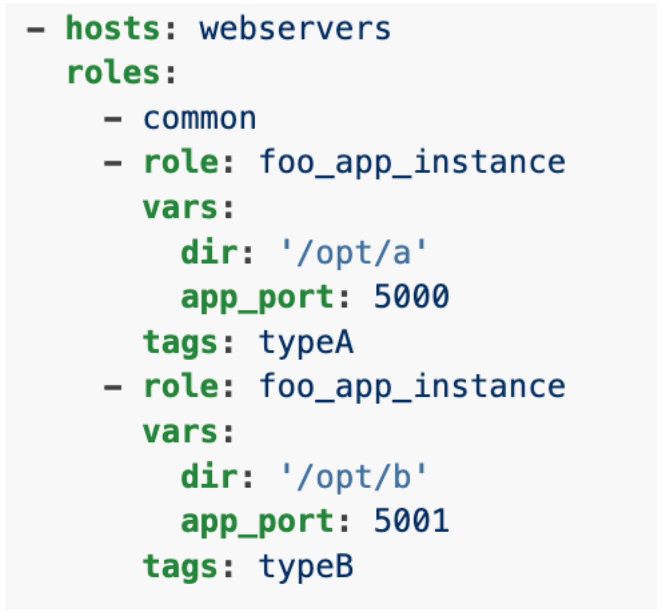
Execution order

- You can have roles, tasks and handlers on your playbook
- The playbook will execute on the following order
  - Each role listed in **`roles:`** in the order listed.
    - Any role dependencies defined in the role's **`meta/main.yml`** run first
  - Any tasks defined in the play.
  - Any handlers triggered by the roles or tasks.

```yaml
- hosts: your_target_host
  become: yes

  vars:
    php_memory_limit: "256M"
    php_max_execution_time: "60"

  roles:
    - geerlingguy.php

  tasks:
    - name: Ensure git is installed
      ansible.builtin.package:
        name: git
        state: present

    - name: Clone a repository
      ansible.builtin.git:
        repo: 'https://github.com/example/repo.git'
        dest: '/path/to/destination'
        clone: yes
        update: yes

    - name: Another task
      ...
```

Use same role with different variables

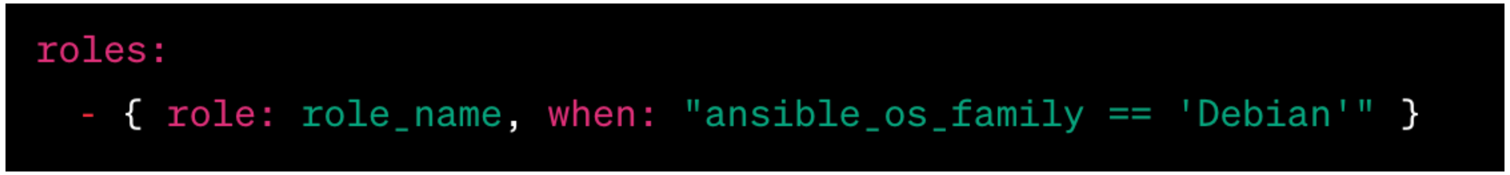
- Every role have a set of variables (with defaults)
- Depending the variables value on your playbook, you can get diferente outcome
- Variables values to send to role can be set as playbook variables or directly on role block in playbook



```yaml
- hosts: webservers
  roles:
    - common
    - role: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
      tags: typeA
    - role: foo_app_instance
      vars:
        dir: '/opt/b'
        app_port: 5001
      tags: typeB
```

Conditional execution

- You may execute your role based on a condition
- For the condition, you may follow all rules defined for conditional on playbook tasks



```
roles:
  - { role: role_name, when: "ansible_os_family == 'Debian'" }
```

Role Dependencies

- Role dependencies let you automatically pull in other roles when using a role
- Role dependencies are prerequisites, not true dependencies
- The roles do not have a parent/child relationship
- Ansible loads all listed roles, runs the roles listed under dependencies first, then runs the role that lists them
- For exemple, if you list role **`foo`** under **`roles:`**, role **`foo`** lists role **`bar`** under dependencies in its **`meta/main.yml`** file, and role **`bar`** lists role **`baz`** under dependencies in its **`meta/main.yml`**, Ansible executes **`baz`**, then **`bar`**, then **`foo`**.

Ansible Galaxy

Ansible Advanced

Ansible Galaxy

- Ansible Galaxy is a community hub for sharing and discovering Ansible roles and collections
- It provides a centralized platform for Ansible users to find reusable content and contribute their own
- Interacting with Galaxy:
  - Web Interface: Accessible at https://galaxy.ansible.com, where you can search, download, and rate roles/collections.
  - CLI: Ansible Galaxy has a command-line interface bundled with Ansible, accessed using ansible-galaxy.

Ansible Galaxy CLI

- Install Roles: **`ansible-galaxy install username.role_name`**
- List Installed Roles: **`ansible-galaxy list`**
- Remove Roles: **`ansible-galaxy remove role_name`**
- Search Roles: **`ansible-galaxy search "search_term"`**
- Create Role Skeleton: **`ansible-galaxy init role_name`**
- Import Roles/Collections: **`ansible-galaxy import`**
- Setup Configuration: **`ansible-galaxy setup`**

Authentication

- Ansible Galaxy uses GitHub for authentication
- Users need a GitHub account to log in to Ansible Galaxy
- When you first log in to Galaxy via the web interface, you'll be asked to authorize the application with GitHub, granting access to your public repositories.

Sharing Roles

Ansible Advanced

Sharing Roles

- Ansible Galaxy is the usual way to share roles but make them public
- If you want to share privately you only need a git repository
- You can reference roles using folder path only but using git repositor is the recommended way

Reference private role

- Create a requirements.yml with following contente

```
- name: custom_role_name
  src: <repository_url>
  scm: git
  version: master
```

- Then install your role like a Ansible Galaxy Role

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

Authoring Roles

Ansible Advanced

Create directory structure

- Ansible Galaxy help you creating role directory structure

```
ansible-galaxy init user_setup
```

- This command automatically creates all folders and empty YAML files inside that folders

Author your code

- Now you can start to edit the generated files
- Some considerations
  - File **`meta/main.yml`**, you should specify role dependencies but you should add additional details like role version, author, etc.
  - File **`README.md`** contains a template to document role. This information is crucial if you want to share your role
  - Mandatory files to be changed are **`defaults`** and **`tasks`**. **`Defaults`** to define your role parameters and **`tasks`** to define your role execution

Demo: Using custom role

Ansible Advanced

Lab: Author your role

Ansible Advanced