

# 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

# 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

# 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 a handler
- After we'll check how to author your own role and check that we can host the role in different places

# Where Ansible search for Roles

---

- Ansible will use the following list in that sequence to try to find a role to be used
  1. In a directory called **roles/**, relative to the playbook file
  2. In the configured **roles\_path**. The default paths are:
    - `~/.ansible/roles`
    - `/usr/share/ansible/roles`
    - `/etc/ansible/roles`
  3. In the directory where the playbook file is located



# 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`.

# 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.

# Ansible Collections

---

- Collections are a newer concept in Ansible and provide a more flexible and powerful way to organize and distribute content
- Collections can contain roles, modules, plugins, playbooks, and other content types
- Collections allow for better organization and distribution of content beyond just roles, making it easier to share and reuse Ansible content
- Collections are versioned, making it easier to manage dependencies and ensure consistency across different environments

# Demo: Review Ansible Galaxy

Ansible Advanced

# ansible-galaxy CLI

---

- To interact with Ansible Galaxy you should use **ansible-galaxy** command
- This command allow you to interact both with collections and roles
- Every command should follow the following structure



```
$ ansible-galaxy role <COMMAND>
```

```
$ ansible-galaxy collection <COMMAND>
```

# ansible-galaxy CLI

---

- Install a role



```
$ ansible-galaxy role install <role_name>
```

- Remove a role



```
$ ansible-galaxy role remove <role_name>
```

# ansible-galaxy CLI

---

- Search for a role



```
$ ansible-galaxy role search <role_name>
```

- List locally available roles



```
$ ansible-galaxy role list
```



# Using Roles

Ansible Advanced

# Role 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
```

# 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

# Role Dependencies - Example


---

- If you list role **role1** under **roles:**, on your playbook
- The role **role1** lists role **role2** under dependencies in its **meta/main.yml** file
- The role **role2** lists role **role3** under dependencies in its **meta/main.yml**
- Ansible executes **role3**, then **role2**, then **role1**.

# Install roles

---

- You can install directly using CLI command



```
$ ansible-galaxy role install <ROLE_NAME>
```

- Or, using a **requirements.yml** listing all your dependencies



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

# Demo: Using roles

Ansible Advanced

# Execution order

---

- You can have roles, tasks and handlers on your playbook following this order:
  1. Each role listed in **roles:** in the order listed.
  2. Any tasks defined in the play.
  3. Any handlers triggered by the roles or tasks.
- Roles execution follows the same parallelism strategy as the tasks

# 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 different outcome
- Variables values to send to role can be set as playbook variables or directly on role block in playbook

```
- 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'" }
```

# Authoring Roles

Ansible Advanced

# 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
```

# defaults folder

---

- This directory and file are intended for defining default variables for the role
- Variables set in defaults are the most "default" or the least priority in the hierarchy of variable precedence
- This means that these values are easily overridden by almost any other method of setting variables.
- Use defaults for variables that you expect users of the role to frequently override
- These are typically parameters that customize the role's behavior but have sensible defaults that work out of the box for most users
- For instance, a role that installs and configures a web server might have the HTTP port as a default variable, assuming the standard port 80 but allowing users to change it if needed

# vars folder

---

- The **vars** directory is for variables that should not be easily overridden
- Variables defined in **vars/main.yml** have a higher precedence than those in **defaults** and most other variable sources, except for inventory variables set at the host level, playbook variables, command-line variables, and a few others
- **vars** are suitable for variables that are essential to the role's operations and should only be overridden in specific circumstances
- These might include variables that depend on the system's architecture, paths to specific system resources, or values critical to the role's internal logic.
- For example, a role might use a variable in vars to define the path to a software binary that is expected to be the same across all deployments where the role is applicable.

# Create directory structure

---

- Ansible Galaxy help you creating role directory structure



```
$ ansible-galaxy role init <ROLE_NAME>
```

- 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



# 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 repository is the recommended way

# Reference private role


---

- Create a **requirements.yml** with following content



```
- src: https://github.com/tasb/ansible-role-tasb-nginx.git  
  version: main  
  name: tasb.nginx
```

- Then install your role like a Ansible Galaxy Role



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

# Version Property

---

- **version** property on Git repository properties allow you to run and test different versions off the same role
- This property can assume 3 different meanings:
  1. Branch name, getting last commit from that branch
  2. Git Tag, using that specific tag
  3. Commit SHA, using a specific commit in any branch

# Demo: Using custom role

Ansible Advanced

# Lab: Author your role

Ansible Advanced

