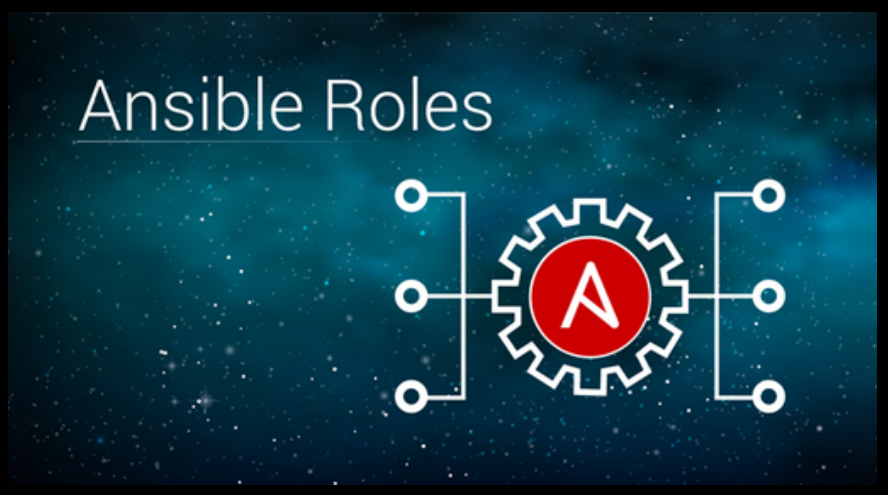
ANSIBLE

Roles

Introduction to Ansible:



Ansible is an open-source automation tool that simplifies IT orchestration, allowing you to automates tasks such as configuration management, application deployment, and infrastructure provisioning.

At its core, Ansible relies on YAML-formatted files called playbooks, which define a series of tasks to be executed on remote hosts.

Ansible Roles:

Ansible Roles provide a well-defined framework and structure for setting your tasks, variables, handlers, metadata, templates, and other files. They enable us to reuse and share our Ansible code efficiently. This way, we can reference and call them in our playbooks with just a few lines of code while we can reuse the same roles over many projects without the need to duplicate our code.

When starting with Ansible, it's pretty common to focus on writing playbooks to automate repeating tasks quickly. As new users automate more and more tasks with playbooks, they reach a point where using just Ansible playbooks is limiting. So here in this case Ansible Roles come into picture.

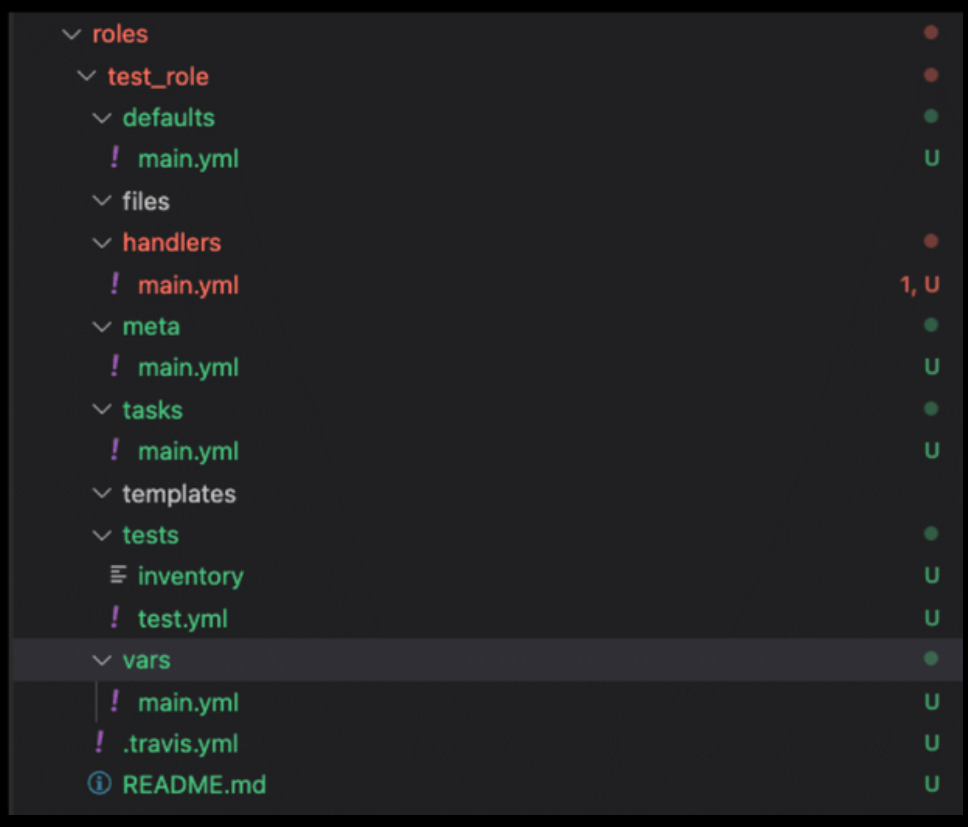
Organizing our Ansible content into roles provides us with a structure that is more manageable than just using playbooks. As our projects get bigger with more playbooks, they become more complicated.

When we place our Ansible code into roles, it helps us organize our automation projects into logical groups and follow the separation of concerns design principle. Collaboration and speed are boosted because various users can work on different roles at the same time without changing the same playbooks together.

In this detailed explanation, we'll dive into the concept of roles in Ansible, covering their structure, advantages, recommended practices, and examples.

Structure of a Role:

defaults – Default variable values are defined in files within this directory. These variables serve as the fallback values if no other value is provided, promoting flexibility and ease of customization.



files – Contains static and custom files that the role uses to perform various tasks.

handlers – A set of handlers that are triggered by tasks of the role.

meta – Includes metadata information for the role, its dependencies, the author, license, available platform, etc.

tasks – A list of tasks to be executed by the role. This part could be considered similar to the task section of a playbook.

templates – Contains Jinja2 template files used by tasks of the role. (Read more about how to create an Ansible template.)

tests – Includes configuration files related to role testing.

vars – Variable files in this directory define variables specific to the role. These variables can be used within tasks, templates, and other files to customize the behavior of the role.

Benefits of Roles:

Roles offer several benefits for managing Ansible automation projects:

Modularity: Roles encourage modularity by containing related functionality within independent units. This modular method simplifies code organization, improves readability, and enables code reuse across different projects.

Code Reuse: By defining reusable roles, you can avoid duplicating code and leverage existing automation logic across multiple projects. This reduces development effort, minimizes errors, and improves consistency in automation workflows.

Maintainability: Roles enhance maintainability by offering a clear and structured framework for automation code. Updates and maintenance tasks can be performed within individual roles without affecting other aspects of the automation project.

Scalability: Roles facilitate scalable automation projects by enabling the assembly of complex workflows from smaller, reusable components. This simplifies the management of large-scale infrastructure configurations and deployments.

Collaboration: Roles facilitate collaboration among team members by standardizing code organization and promoting best practices. Roles can be shared, version-controlled, and distributed via Ansible Galaxy or other repositories, fostering a collaborative development environment.

Examples of Roles:

Let's consider a few examples of roles to illustrate their usage in Ansible automation projects:

Web Server Role: This role installs and configures a web server (e.g., Nginx or Apache) on target hosts. It includes tasks for installing the web server software, managing configuration files, and starting the service.

Database Server Role: This role sets up a database server (e.g., MySQL or PostgreSQL) on target hosts. It includes tasks for installing the database software, configuring database users and permissions, and starting the database service.

Application Deployment Role: This role deploys an application to target hosts, including tasks for fetching the application code from a version control system (e.g., Git), installing dependencies, configuring environment variables, and restarting the application server.

Here is the **example of web server role**:

Once we have defined all the necessary parts of our role, it's time to use it in plays. The classic and most obvious way is to reference a role at the play level with the roles option:

With this option, each role defined in our playbook is executed before any other tasks defined in the play.

This is an example play to try out our new webserver role. Let's go ahead and execute this play.

```
- hosts: all
  become: true
  roles:
    - webserver
```

```
→  ansible-roles git:(master) ✗ ansible-playbook main_playbook.yml

PLAY [all] *********************************************************************************************

TASK [Gathering Facts] *********************************************************************************
ok: [host1]

TASK [webserver : Update and upgrade apt] **************************************************************
changed: [host1]

TASK [webserver : Install Nginx to version 1.18.0-0ubuntu1.3] ******************************************
changed: [host1]

TASK [webserver : Copy the Nginx configuration file to the host] ***************************************
changed: [host1]

TASK [webserver : Create link to the new config to enable it] ******************************************
ok: [host1]

TASK [webserver : Create Nginx directory] **************************************************************
changed: [host1]

TASK [webserver : Copy index.html to the Nginx directory] **********************************************
changed: [host1]

RUNNING HANDLER [webserver : Restart the Nginx service] ************************************************
changed: [host1]

PLAY RECAP *********************************************************************************************
host1                      : ok=8    changed=6    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Great! All the tasks have been completed successfully. Let's also validate that we have configured our Ngnix web server correctly Now connect your master via ssh and execute-

systemctl status nginx

to verify that Nginx service is up and running. Finally, run the command curl localhost to check if the web server responds with the custom page that we configured.

```
*** System restart required ***
Last login: Sat May 21 08:05:27 2022 from 10.0.2.2
      @ubuntu-focal:~$ systemctl status nginx
● nginx.service - A high performance web server and a reverse proxy server
     Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
     Active: active (running) since Sat 2022-05-21 08:05:27 UTC; 2min 8s ago
       Docs: man:nginx(8)
    Process: 18528 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
    Process: 18529 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
   Main PID: 18530 (nginx)
      Tasks: 3 (limit: 1131)
     Memory: 3.5M
     CGroup: /system.slice/nginx.service
             ├─18530 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
             ├─18531 nginx: worker process
             └─18532 nginx: worker process
      @ubuntu-focal:~$ curl localhost
<html>
  <head>
    <title>Hello from Ngnix </title>
  </head>
  <body>
  <h1>This is our test webserver</h1>
  <p>This Nginx web server was deployed by Ansible.</p>
  </body>
</html>
```

When using the roles option at the play level, we can override any of the default role's variables or pass other keywords, like tags. Tags are added to all tasks within the role.

```
- hosts: all
  become: true
  roles:
    - role: webserver
      vars:
        nginx_version: 1.17.10-0ubuntu1
      tags: example_tag
```

Apart from defining roles at the play level with the roles option, we can use them also at the tasks level with the options include_role dynamically and import_role statically. These are useful when we would like to run

our role tasks in a specific order and not necessarily before any other playbook tasks. This way, roles run in the order they are defined in plays.


```yaml
- hosts: all
  tasks:
    - name: Print a message
      ansible.builtin.debug:
        msg: "This task runs first and before the example role"

    - name: Include the example role and run its tasks
      include_role:
        name: example

    - name: Print a message
      ansible.builtin.debug:
        msg: "This task runs after the example role"

    - name: Include the example_2 role and run its tasks in the end
      include_role:
        name: example_2
```

Even if we define a role multiple times, Ansible will execute it only once. Occasionally, we might want to run a role multiple times but with different parameters. Passing a different set of parameters to the same roles allows executing the role more than once Example of executing the role test three times.

```yaml
- hosts: all
  roles:
    - role: test
      message: "First time"
    - role: test
      message: "Second time"
    - role: test
      message: "Third time"
```

Sharing Roles with Ansible Galaxy:

Sharing roles with Ansible Galaxy allows you to distribute your roles to the broader Ansible community or within your organization. Ansible Galaxy is an online open-source, public repository of Ansible content. You can use Ansible Galaxy to browse for roles that fit your use case and save time by using them instead of writing everything from scratch included with Ansible . a command-line tool share roles, deploy them, and more. For each role, you can see its code repository, documentation, and even a rating from other users.

Before running any role, check its code repository to ensure it's safe and does what you expect. Here's how you can share roles using Ansible Galaxy:

- Create a Role by using command

ansible-galaxy init my_role

- **Add Metadata by** editing the **meta/main.yr**

file within your role directory. This metadata includes information like the author, description, dependencies, and supported platforms.

```yaml
galaxy_info:
  author: Your Name
  description: Your role description
  license: license (e.g., MIT)
  min_ansible_version: 2.9
  platforms:
    - name: Ubuntu
      versions:
        - bionic
        - focal
  galaxy_tags:
    - tag1
    - tag2
```

- ***Commit Your Role***: *Once your role is ready, commit it to a version control system like Git. Ensure that all necessary files are included, especially the* **meta/main.yml** *file.*
- ***Share Your Role***: *You can share your role with the Ansible Galaxy community or within your organization by publishing it to Galaxy. To do this, you'll first need to create an account on Ansible Galaxy's website (https://galaxy.ansible.com/).*
- ***Upload Your Role:*** *You can upload your role using the* **ansible-galaxy** *command-line tool. Run the following command within your role directory:*

  ansible-galaxy login
- Follow the prompts to log in to your Ansible Galaxy account. Once logged in, you can publish your role:

  ansible-galaxy publish my_role.tar.gz
- **Verify Your Role**: After uploading your role, you can verify its presence on Ansible Galaxy by searching for it on the website or using the **ansible-galaxy** command-line tool.

Ansible Roles Tips & Tricks:

- *Always use descriptive names for your roles, tasks, and variables. Document the intent and the purpose of your roles thoroughly and point out any variables that the user has to set. Set sane defaults and simplify your roles as much as possible to allow users to get onboarded quickly.*
- *Never place secrets and sensitive data in your roles YAML files. Secret values should be passed to the role at execution time by the play as a variable and should never be stored in any code repository.*
- *At first, it might be tempting to define a role that handles many responsibilities. For instance, we could create a role that installs multiple components, a common anti-pattern. Try to follow the separation of concerns design principle as much as possible and separate your roles based on different functionalities or technical components.*
- *Try to keep your roles as loosely coupled as possible and avoid adding too many dependencies.*
- *To control the execution order of roles and tasks, use the import_role or Include_role tasks instead of the classic roles keyword.*
- *When it makes sense, group your tasks in separate task files for improved clarity and organization.*