Advanced Ansible

Author: Zayan Ahmed | **Estimated Reading time:** 6 min

Ansible is a powerful automation tool used for configuration management, application deployment, and orchestration. As you move beyond the basics of Ansible, you encounter concepts like roles, variables, and templates that allow you to structure and scale your playbooks more effectively. This document covers these advanced Ansible concepts in detail.

1. Roles in Ansible

What Are Roles?

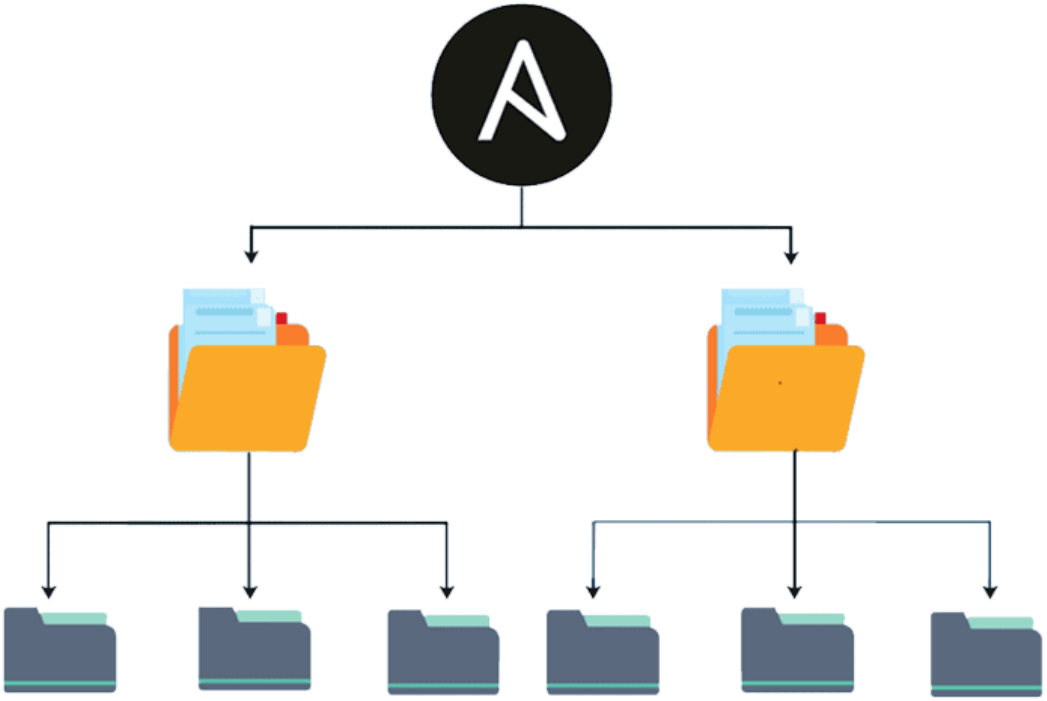
Roles are a method of organizing Ansible content into reusable and modular components. They allow you to separate tasks, variables, files, and templates into a standardized structure that can be shared and maintained independently.

Role Directory Structure

An Ansible role has a specific directory structure:

```
roles/
 <role_name>/
   tasks/
     main.yml          # Contains the main set of tasks
   handlers/
     main.yml          # Defines handlers triggered by tasks
   templates/          # Contains Jinja2 template files
```

```
   files/              # Static files for distribution
   vars/
     main.yml          # Role-specific variables
   defaults/
     main.yml          # Default variables for the role
   meta/
     main.yml          # Metadata about the role (dependencies, author,
etc.)
   tests/
     inventory         # Test inventory
     test.yml          # Test playbook
```



Using Roles in Playbooks

To use a role in a playbook, you include it under the `roles` keyword:

```
- hosts: webservers
  roles:
    - nginx
```

Alternatively, you can use the `include_role` module for more dynamic inclusion:

```
- name: Include nginx role
  include_role:
```

```
  name: nginx
```

Benefits of Roles

- Modular and reusable code.
- Easier to maintain and update.
- Encourages collaboration by isolating functionalities.

2. Variables in Ansible

Types of Variables

Variables in Ansible allow you to define dynamic values. They can be defined in various scopes:

Playbook Variables: Defined directly within a playbook:

```
vars:

 app_port: 8080
```

Host Variables: Defined in the inventory file:

```
webserver ansible_host=192.168.1.10 app_port=8080
```

Group Variables: Defined for groups of hosts:

```
# group_vars/webservers.yml

app_port: 8080
```

1. **Role Variables**: Stored in the `vars/main.yml` or `defaults/main.yml` files of a role.
2. **Facts**: Gathered dynamically during a play using the `setup` module.

Variable Precedence

When multiple variables overlap, Ansible applies them in a specific order of precedence:

1. Role defaults
2. Inventory variables
3. Group variables
4. Host variables
5. Playbook variables
6. Role vars
7. Task variables
8. Extra vars (highest precedence)

Variable Files

Variables can also be stored in separate YAML files and included in playbooks:

```
vars_files:

 - vars/app.yml
```

3. Templates in Ansible

What Are Templates?

Templates in Ansible are files written in the Jinja2 templating language. They allow you to dynamically generate configuration files and scripts based on variables and logic.

Common Use Cases

- Generating configuration files (e.g., `nginx.conf`, `httpd.conf`).
- Creating scripts or manifests dynamically.

Jinja2 Syntax

- **Variable substitution**: `{{ variable_name }}`
- **Control structures**:

```
{% if condition %}

...

{% endif %}
```

```
{% for item in list %}

...

{% endfor %}
```

Example Template

```
server {

   listen {{ app_port }};

   server_name {{ app_name }};



   location / {

       proxy_pass http://{{ backend_server }};

   }

}
```

Using Templates in Playbooks

Use the `template` module to deploy templates:

```
- name: Deploy nginx configuration

 template:

   src: nginx.j2

   dest: /etc/nginx/nginx.conf
```

4. Advanced Concepts

Ansible Vault

Ansible Vault allows you to encrypt sensitive data such as passwords and keys.

Encrypt a file:

```
ansible-vault encrypt secrets.yml
```

Decrypt a file:

```
ansible-vault decrypt secrets.yml
```

Edit an encrypted file:

```
ansible-vault edit secrets.yml
```

Use in a playbook:

```
vars_files:
 - secrets.yml
```

Dynamic Inventories

Dynamic inventories allow you to pull host information from external sources like AWS, Azure, or custom scripts.

- Example AWS inventory plugin:

```
plugin: aws_ec2
regions:
 - us-east-1
keyed_groups:
 - key: tags
```

Custom Modules

You can create custom modules for specific tasks that Ansible does not support natively. Custom modules are written in Python and follow a specific structure.

---

5. Best Practices

1. **Use Roles**: Organize your playbooks into roles for better reusability and structure.
2. **Avoid Hardcoding Variables**: Use variable files or inventory variables to keep your playbooks dynamic.
3. **Use Templates**: Leverage templates for configuration files to reduce redundancy.
4. **Encrypt Sensitive Data**: Use Ansible Vault to handle secrets securely.
5. **Document Your Playbooks**: Use comments to make playbooks easier to understand.

Conclusion:

Ansible's advanced features like roles, variables, and templates make it a versatile and powerful tool for automation. By structuring your playbooks effectively and utilizing these advanced techniques, you can manage complex environments with ease.

Follow me on **LinkedIn** for more