Ansible Advanced Topics - Student Guide

Overview

This lesson covers three essential Ansible concepts: Jinja2 templating, roles, and vault functionality. You'll learn how to create dynamic configurations, organize playbooks using roles, and securely manage sensitive data.

Prerequisites

- Basic understanding of Ansible playbooks
- Familiarity with YAML syntax
- Docker and Docker Compose installed
- SSH client

Lab Setup

1. Clone Required Repositories

```bash
git clone git@github.com:spurin/diveintoansible-lab.git
git clone git@github.com:spurin/diveintoansible.git
```

2. Start Lab Environment

```bash
cd diveintoansible-lab
docker-compose up -d
```

3. Access the Lab

- Open your browser and navigate to: `http://localhost:1000/`
- Select **ubuntu-c** container
- Login credentials:
  - Username: `ansible`
  - Password: `password`

Part 1: Jinja2 Templating

Learning Objectives

- Understand Jinja2 template syntax
- Create dynamic configuration files
- Use variables and control structures in templates

Navigate to the Templating Directory

```bash
cd ~/diveintoansible/Ansible\ Playbooks,\ Introduction/Templating\ with\ Jinja2/11
```

Examine the Lab Structure

```bash
ll
```

You should see:

- `ansible.cfg` - Ansible configuration file
- `group_vars/` - Directory for group variables
- `host_vars/` - Directory for host-specific variables
- `hosts` - Inventory file
- `jinja2_playbook.yaml` - Main playbook
- `template.j2` - Jinja2 template file

Key Files to Examine

1. Template File (`template.j2`)

This file contains Jinja2 syntax for creating dynamic content based on host variables and facts.

2. Playbook (`jinja2_playbook.yaml`)

Contains tasks that use the template module to process the Jinja2 template.

Execute the Templating Playbook

```bash
ansible-playbook jinja2_playbook.yaml
```

Verify Template Output

After running the playbook, check the generated files:

On the control node:

```bash
cd /tmp
ll ubuntu-c_template.out
```

On a managed node (CentOS1):

```bash
ssh root@centos1
ls /tmp/
cat /tmp/centos1_template.out
```

Key Concepts

Jinja2 Template Syntax:

- `{{ variable }}` - Variable substitution
- `{% for item in list %}...{% endfor %}` - Loops
- `{% if condition %}...{% endif %}` - Conditionals
- `{{ ansible_hostname }}` - Ansible facts

Template Module:

- Processes Jinja2 templates
- Substitutes variables with actual values
- Creates files on target hosts

Part 2: Ansible Roles

Learning Objectives

- Understand role structure and organization
- Create reusable playbook components
- Implement role dependencies

Navigate to the Roles Directory

```bash
cd ~/diveintoansible/Structuring\ Ansible\ Playbooks/Using\ Roles
```

Examine Role Examples

Example 1 - Directory 07

```bash
cd 07
ll
```

You'll see:

- Standard Ansible files (ansible.cfg, hosts, etc.)
- `nginx/` - Nginx role directory
- `webapp/` - Web application role directory
- `nginx_webapp_playbook.yaml` - Playbook using both roles

Example 2 - Directory 08

```bash
cd ../08
ll
```

Similar structure but with different configurations.

Role Structure

Each role directory (e.g., `nginx/`, `webapp/`) contains:

- `tasks/` - Main tasks for the role
- `handlers/` - Handlers triggered by tasks
- `templates/` - Jinja2 templates
- `files/` - Static files to copy
- `vars/` - Role variables
- `defaults/` - Default variables

- `meta/` - Role metadata and dependencies

Working with Host Variables

Remove a host variable file to see how roles handle missing configurations:

```bash
rm host_vars/centos1
```

Execute the Roles Playbook

```bash
ansible-playbook nginx_webapp_playbook.yaml
```

Key Concepts

Role Benefits:

- Code reusability
- Better organization
- Easier maintenance
- Modular approach

Role Execution Order:

1. Dependencies (if any)
2. Tasks
3. Handlers (when triggered)

Part 3: Ansible Vault

Learning Objectives

- Secure sensitive data in Ansible
- Create and use encrypted strings
- Run playbooks with vault passwords

Navigate to the Vault Directory

```bash
```

```
cd ~/diveintoansible/Ansible\ Playbooks,\ Deep\ Dive/Vault/01
```

Create Encrypted Variables

Encrypt a String

```
bash

ansible-vault encrypt_string --ask-vault-pass --name 'ansible_become_pass' 'password'
```

You'll be prompted to:

1. Enter a new vault password
2. Confirm the vault password

The output will be an encrypted string that can be used in playbooks or variable files.

Test Vault with Ad-hoc Commands

```
bash

ansible --ask-vault-pass -m ping all
```

When prompted, enter the vault password you created earlier.

Working with Vault Playbooks

Navigate to Directory 02

```
bash

cd ../02
```

Create Another Encrypted Variable

```
bash

ansible-vault encrypt_string --ask-vault-pass --name 'ansible_become_pass' 'password'
```

Run Playbook Without Vault Password (This Will Fail)

```
bash
```

```
ansible-playbook vault_playbook.yaml
```

Run Playbook With Vault Password

```bash
ansible-playbook --ask-vault-pass vault_playbook.yaml
```

Key Concepts

Vault Operations:

- `encrypt_string` - Encrypt individual strings
- `encrypt` - Encrypt entire files
- `decrypt` - Decrypt files
- `edit` - Edit encrypted files
- `view` - View encrypted files

Best Practices:

- Never commit vault passwords to version control
- Use different vault passwords for different environments
- Store vault passwords securely (password files, external tools)

Cleanup

When finished with the lab:

```bash
docker-compose down
```

Summary

In this lesson, you learned:

1. **Jinja2 Templating**: How to create dynamic configuration files using variables and control structures
2. **Roles**: How to organize Ansible code into reusable, modular components
3. **Vault**: How to securely handle sensitive data in Ansible playbooks

Additional Practice

1. Create your own Jinja2 template for a configuration file
2. Build a custom role for a service you commonly deploy
3. Experiment with different vault commands and options

Troubleshooting Tips

- Always check file permissions and paths
- Verify inventory file contents
- Use `ansible-config dump` to check configuration
- Use `-vvv` flag for verbose output during debugging
- Ensure vault passwords are entered correctly

Next Steps

- Explore Ansible Galaxy for community roles
- Learn about role dependencies and meta files
- Practice with more complex Jinja2 templates
- Implement vault in production workflows