Ansible Workshop - Exercises

Basics

Get to know Ansible and learn to write your first Ansible Playbooks.

9 - Reusability with Roles

Objective

While it is possible to write a playbook in one file as we've done throughout this workshop, eventually you'll want to reuse files and start to organize things.

Ansible Roles are the way we do this. When you create a role, you deconstruct your playbook into parts and those parts sit in a directory structure. This is explained in more details in Ansible documentation in Roles or in Sample Ansible setup.

This exercise will cover:

- the folder structure of an Ansible Role
- how to build an Ansible Role
- creating an Ansible Play to use and execute a role
- using Ansible to create a Apache VirtualHost on node2

Guide

Step 1 - Understanding the Ansible Role Structure

Roles follow a defined directory structure; a role is named by the top level directory. Some of the subdirectories contain YAML files, named `main.yml`. The files and templates subdirectories can contain objects referenced by the YAML files.

An example project structure could look like this, the name of the role would be "apache":

```
apache/
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

The various `main.yml` files contain content depending on their location in the directory structure shown above. For instance, `vars/main.yml` references variables, `handlers/main.yaml` describes handlers, and so on. Note that in contrast to playbooks, the `main.yml` files only contain the specific content and not additional playbook information like hosts, `become` or other keywords.

> **Tip**
>
> There are actually two directories for variables: `vars` and `default`. Default variables, `defaults/main.yml`, have the lowest precedence and usually contain default values set by the role authors and are often used when it is intended that their values will be overridden. Variables set in `vars/main.yml` are for variables not intended to be modified.

Using roles in a Playbook is straight forward:

```
---
- name: Launch roles
  hosts: web
  roles:
    - role1
    - role2
```

For each role, the tasks, handlers and variables of that role will be included in the Playbook, in that order. Any copy, script, template, or include tasks in the role can reference the relevant files, templates, or tasks *without absolute or relative path names*. Ansible will look for them in the role's files, templates, or tasks respectively, based on their use.

Step 2 - Create a Basic Role Directory Structure

Ansible looks for roles in a subdirectory called `roles` in the project directory. This can be overridden in the Ansible configuration. Each role has its own directory. To ease creation of a new role the tool `ansible-galaxy` can be used.

> **Tip**
>
> Ansible Galaxy is your hub for finding, reusing and sharing the best Ansible content. `ansible-galaxy` helps to interact with Ansible Galaxy. For now we'll just using it as a helper to build the directory structure.

Okay, lets start to build a role. We'll build a role that installs and configures Apache to serve a virtual host. Run these commands in your `~/ansible-files` directory:

```
[student@ansible-1 ansible-files]$ mkdir roles
[student@ansible-1 ansible-files]$ ansible-galaxy init --offline roles/apache-webserver
```

Have a look at the role directories and their content:

```
[student@ansible-1 ansible-files]$ tree roles
```

```
roles/
└── apache-webserver
    ├── defaults
    │   └── main.yml
    ├── files
    ├── handlers
    │   └── main.yml
    ├── meta
    │   └── main.yml
    ├── README.md
    ├── tasks
    │   └── main.yml
    ├── templates
    ├── tests
    │   ├── inventory
    │   └── test.yml
    └── vars
        └── main.yml
```

Step 3 - Create the Tasks File

The `main.yml` file in the tasks subdirectory of the role should do the following:

- Make sure httpd is installed
- Make sure httpd is started and enabled
- Put HTML content into the Apache document root
- Install the template provided to configure the vhost

> **Note**
>
> The `main.yml` (and other files possibly included by main.yml) can **only contain tasks**, **not** complete playbooks!

Edit the `roles/apache-webserver/tasks/main.yml` file:

```yaml
---
- name: Install httpd package
  ansible.builtin.package:
    name: httpd
    state: present

- name: Start and enable httpd service
  ansible.builtin.service:
    name: httpd
    state: started
    enabled: true
```

Note that here just tasks were added. The details of a playbook are not present.

The tasks added so far do:

- Install the httpd package using the package module
- Use the service module to enable and start httpd

Next we add two more tasks to ensure a *vhost* directory structure on the managed nodes and copy HTML content:

```
- name: Ensure vhost directory is present
  ansible.builtin.file:
    path: "/var/www/vhosts/{{ ansible_hostname }}"
    state: directory
    mode: "0755"
    owner: apache
    group: apache

- name: Deliver html content
  ansible.builtin.copy:
    src: web.html
    dest: "/var/www/vhosts/{{ ansible_hostname }}/index.html"
    mode: "0644"
    owner: apache
    group: apache
```

Note that the *vhost* directory is created/ensured using the `file` module.

Info

The term *Virtual Host* refers to the practice of running more than one web site (such as *company1.example.com* and *company2.example.com*) on a single machine. The fact that they are running on the same physical server is not apparent to the end user.

The last task we add uses the template module to create the vhost configuration file from a j2-template:

```
- name: Deploy vhost template
  ansible.builtin.template:
    src: vhost.conf.j2
    dest: /etc/httpd/conf.d/vhost.conf
    owner: root
    group: root
    mode: "0644"
  notify:
    - Restart_httpd
```

Note it is using a handler to restart httpd after an configuration update.

The full `tasks/main.yml` file is:

```
---
- name: Install httpd package
  ansible.builtin.package:
    name: httpd
    state: present

- name: Start and enable httpd service
  ansible.builtin.service:
    name: httpd
    state: started
    enabled: true

- name: Ensure vhost directory is present
  ansible.builtin.file:
    path: "/var/www/vhosts/{{ ansible_hostname }}"
    state: directory
    mode: "0755"

- name: Deliver html content
  ansible.builtin.copy:
    src: web.html
    dest: "/var/www/vhosts/{{ ansible_hostname }}/index.html"
    mode: "0644"

- name: Deploy vhost template
  ansible.builtin.template:
    src: vhost.conf.j2
    dest: /etc/httpd/conf.d/vhost.conf
    owner: root
    group: root
    mode: "0644"
  notify:
    - Restart_httpd
```

Step 4 - Create the handler

Create the handler in the file `roles/apache-webserver/handlers/main.yml` to restart httpd when notified by the template task:

```
---
# handlers file for roles/apache-webserver
- name: Restart_httpd
  ansible.builtin.service:
    name: httpd
    state: restarted
```

Step 5 - Create the web.html and vhost configuration file template

Create the HTML content that will be served by the webserver.

- Create an `web.html` file in the "src" directory of the role, the `files` folder. Add a simple HTML content to the file:

```
<body>
    <h1>The virtual host configuration works!</h1>
</body>
```

- Create the `vhost.conf.j2` template file in the role's `templates` subdirectory.

The contents of the `vhost.conf.j2` template file are found below.

```
# {{ ansible_managed }}

<VirtualHost *:8080>
    ServerAdmin webmaster@{{ ansible_fqdn }}
    ServerName {{ ansible_fqdn }}
    ErrorLog logs/{{ ansible_hostname }}-error.log
    CustomLog logs/{{ ansible_hostname }}-common.log common
    DocumentRoot /var/www/vhosts/{{ ansible_hostname }}/

    <Directory /var/www/vhosts/{{ ansible_hostname }}/>
  Options +Indexes +FollowSymlinks +Includes
  Order allow,deny
  Allow from all
    </Directory>
</VirtualHost>
```

Warning

The *vhost* configuration expects that the webserver announces on Port 8080, the configuration was adjusted in a previous exercise.

Step 6 - Test the role

You are ready to test the role against `node2`. But since a role cannot be assigned to a node directly, first create a playbook which connects the role and the host. Create the file `test_apache_role.yml` in the directory `~/ansible-files`:

```
---
- name: Use apache-webserver role
  hosts: node2
  become: true
  pre_tasks:
    - name: Output info before any role execution
      ansible.builtin.debug:
        msg: "Beginning web server configuration."
  post_tasks:
    - name: Output info after all roles are executed
      ansible.builtin.debug:
        msg: "Web server has been configured."
  roles:
    - apache-webserver
```

Note the `pre_tasks` and `post_tasks` keywords. Normally, the tasks of *roles* execute before the *tasks* of a playbook. To control order of execution `pre_tasks` are performed before any roles are applied. The `post_tasks` are performed after all the roles have completed. Here we just use them to better highlight when the actual role is executed.

Info

In most use cases, you should not mix/use *roles* **and** *tasks* in your play together. If you need to have single tasks in your play, why not create another role and include the tasks there?!

Now you are ready to run your playbook:

Ansible

```
[student@ansible-1 ansible-files]$ ansible-playbook test_apache_role.yml
```

Navigator

```
[student@ansible-1 ansible-files]$ ansible-navigator run test_apache_role.yml -m stdout
```

Run a curl command against `node2` to confirm that the role worked or use the `check_httpd.yml` playbook (you may need to adjust the variable in it to `node2:8080`):

```
[student@ansible-1 ansible-files]$ curl -s http://node2:8080
<body>
<h1>The virtual host configuration works!</h1>
</body>
```

Warning

If you are using the local development environment, remember, you are using containers instead of actual VMs! You need to **append the correct port** (e.g. `curl http://node1:8003"` for Port 8080 of *node1*, or `curl http://node2:8006"` for Port 8080 of *node2*).
Take a look at the table with the ports overview or execute `podman ps` and check the output.

You can also use the IP address of node2 (copy it from your inventory) and paste it into the browser (as well as adding `:8080`).

Congratulations! You have successfully completed this exercise!

Troubleshooting problems

Did the final curl work?
You can see what ports the web server is running on by using the netstat command, connect to the managed node via SSH:

```
#> sudo netstat -tulpn
```

If *netstat* is not present, install it with this command:

```
#> sudo dnf install -y net-tools
```

There should be a line like this:

```
tcp6       0      0 :::8080                 :::*                    LISTEN      25237/httpd
```

If it is not working, make sure that `/etc/httpd/conf/httpd.conf` has `Listen 8080` in it. This should have been changed by Exercise 7.