Red Hat Enterprise Linux Automation with Ansible

FEEDBACK TRANSLATIONS CERTIFICATE OF ATTENDANCE

Creating Roles

Objectives

After completing this section, you should be able to create a role in a playbook's project directory and run it as part of one of the plays in the playbook.

The Role Creation Process

Creating roles in Ansible requires no special development tools. Creating and using a role is a three step process:

1. Create the role directory structure.
2. Define the role content.
3. Use the role in a playbook.

Creating the Role Directory Structure

By default, Ansible looks for roles in a subdirectory called `roles` in the directory containing your Ansible Playbook. This allows you to store roles with the playbook and other supporting files.

If Ansible cannot find the role there, it looks at the directories specified by the Ansible configuration setting `roles_path`, in order. This variable contains a colon-separated list of directories to search. The default value of this variable is:

```
~/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles
```

This allows you to install roles on your system that are shared by multiple projects. For example, you could have your own roles installed your home directory in the `~/.ansible/roles` subdirectory, and the system can have roles installed for all users in the `/usr/share/ansible/roles` directory.

Each role has its own directory with a standardized directory structure. For example, the following directory structure contains the files that define the `motd` role.

```
[user@host ~]$ tree roles/
roles/
└── motd
    ├── defaults
    │   └── main.yml
    ├── files
    ├── handlers
    ├── meta
    │   └── main.yml
    ├── README.md
    ├── tasks
    │   └── main.yml
    └── templates
        └── motd.j2
```

The `README.md` provides a basic human-readable description of the role, documentation and examples of how to use it, and any non-Ansible requirements it might have in order to work. The `meta` subdirectory contains a `main.yml` file that specifies information about the author, license, compatibility, and dependencies for the module. The `files` subdirectory contains fixed-content files and the `templates` subdirectory contains templates that can be deployed by the role when it is used. The other subdirectories can contain `main.yml` files that define default variable values, handlers, tasks, role metadata, or variables, depending on the subdirectory they are in.

If a subdirectory exists but is empty, such as `handlers` in this example, it is ignored. If a role does not use a feature, the subdirectory can be omitted altogether. For example, the `vars` subdirectory has been omitted from this example.

Creating a Role Skeleton

You can create all the subdirectories and files needed for a new role using standard Linux commands. Alternatively, command line utilities exist to automate the process of new role creation.

The `ansible-galaxy` command line tool (covered in more detail later in this course) is used to manage Ansible roles, including the creation of new roles. You can run `ansible-galaxy init` to create the directory structure for a new role. Specify the name of the role as an argument to the command, which creates a subdirectory for the new role in the current working directory.

```
[user@host playbook-project]$ cd roles
[user@host roles]$ ansible-galaxy init my_new_role
- my_new_role was created successfully
[user@host roles]$ ls my_new_role/
defaults  files  handlers  meta  README.md  tasks  templates  tests  vars
```

Defining the Role Content

Once you have created the directory structure, you must write the content of the role. A good place to start is the *`ROLENAME`*`/tasks/main.yml` task file, the main list of tasks run by the role.

The following `tasks/main.yml` file manages the `/etc/motd` file on managed hosts. It uses the `template` module to deploy the template named `motd.j2` to the managed host. Because the `template` module is configured within a role task, instead of a playbook task, the `motd.j2` template is retrieved from the role's `templates` subdirectory.

```
[user@host ~]$ cat roles/motd/tasks/main.yml
---
# tasks file for motd

- name: deliver motd file
  template:
    src: motd.j2
    dest: /etc/motd
    owner: root
    group: root
    mode: 0444
```

The following command displays the contents of the `motd.j2` template of the `motd` role. It references Ansible facts and a `system_owner` variable.

```
[user@host ~]$ cat roles/motd/templates/motd.j2
This is the system {{ ansible_facts['hostname'] }}.

Today's date is: {{ ansible_facts['date_time']['date'] }}.

Only use this system with permission.
You can ask {{ system_owner }} for access.
```

The role defines a default value for the `system_owner` variable. The `defaults/main.yml` file in the role's directory structure is where this value is set.

The following `defaults/main.yml` file sets the `system_owner` variable to `user@host.example.com`. This will be the email address that is written in the `/etc/motd` file of managed hosts that this role is applied to.

```
[user@host ~]$ cat roles/motd/defaults/main.yml
---
system_owner: user@host.example.com
```

Recommended Practices for Role Content Development

Roles allow playbooks to be written modularly. To maximize the effectiveness of newly developed roles, consider implementing the following recommended practices into your role development:

- Maintain each role in its own version control repository. Ansible works well with `git`-based repositories.
- Sensitive information, such as passwords or SSH keys, should not be stored in the role repository. Sensitive values should be parameterized as variables with default values that are not sensitive. Playbooks that use the role are responsible for defining sensitive variables through Ansible Vault variable files, environment variables, or other `ansible-playbook` options.
- Use `ansible-galaxy init` to start your role, and then remove any directories and files that you do not need.
- Create and maintain `README.md` and `meta/main.yml` files to document what your role is for, who wrote it, and how to use it.
- Keep your role focused on a specific purpose or function. Instead of making one role do many things, you might write more than one role.
- Reuse and refactor roles often. Resist creating new roles for edge configurations. If an existing role accomplishes a majority of the required configuration, refactor the existing role to integrate the new configuration scenario. Use integration and regression testing techniques to ensure that the role provides the required new functionality and also does not cause problems for existing playbooks.

Defining Role Dependencies

Role dependencies allow a role to include other roles as dependencies. For example, a role that defines a documentation server may depend upon another role that installs and configures a web server. Dependencies are defined in the `meta/main.yml` file in the role directory hierarchy.

The following is a sample `meta/main.yml` file.

```
---
dependencies:
  - role: apache
    port: 8080
  - role: postgres
    dbname: serverlist
    admin_user: felix
```

By default, roles are only added as a dependency to a playbook once. If another role also lists it as a dependency it will not be run again. This behavior can be overridden by setting the `allow_duplicates` variable to `yes` in the `meta/main.yml` file.

> **IMPORTANT**
>
> Limit your role's dependencies on other roles. Dependencies make it harder to maintain your role, especially if it has many complex dependencies.

Using the Role in a Playbook

To access a role, reference it in the `roles:` section of a play. The following playbook refers to the `motd` role. Because no variables are specified, the role is applied with its default variable values.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  remote_user: devops
  become: true
  roles:
    - motd
```

When the playbook is executed, tasks performed because of a role can be identified by the role name prefix. The following sample output illustrates this with the `motd :` prefix in the task name:

```
[user@host ~]$ ansible-playbook -i inventory use-motd-role.yml

PLAY [use motd role playbook] ************************************************

TASK [setup] *******************************************************************
ok: [remote.example.com]

TASK [motd: deliver motd file] ***********************************************
changed: [remote.example.com]

PLAY RECAP *********************************************************************
remote.example.com         : ok=2    changed=1    unreachable=0    failed=0
```

The above scenario assumes that the `motd` role is located in the `roles` directory. Later in the course you will see how to use a role that is remotely located in a version control repository.

Changing a Role's Behavior with Variables

A well-written role uses default variables to alter the role's behavior to match a related configuration scenario. This helps make the role more generic and reusable in a variety of contexts.

The value of any variable defined in a role's `defaults` directory will be overwritten if that same variable is defined:

- in an inventory file, either as a host variable or a group variable.
- in a YAML file under the `group_vars` or `host_vars` directories of a playbook project
- as a variable nested in the `vars` keyword of a play
- as a variable when including the role in `roles` keyword of a play

The following example shows how to use the `motd` role with a different value for the `system_owner` role variable. The value specified, `someone@host.example.com`, will replace the variable reference when the role is applied to a managed host.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  remote_user: devops
  become: true
  vars:
    system_owner: someone@host.example.com
  roles:
    - role: motd
```

When defined in this way, the `system_owner` variable replaces the value of the default variable of the same name. Any variable definitions nested within the `vars` keyword will not replace the value of the same variable if defined in a role's `vars` directory.

The following example also shows how to use the `motd` role with a different value for the `system_owner` role variable. The value specified, `someone@host.example.com`, will replace the variable reference regardless of being defined in the role's `vars` or `defaults` directory.

```
[user@host ~]$ cat use-motd-role.yml
---
- name: use motd role playbook
  hosts: remote.example.com
  remote_user: devops
  become: true
  roles:
    - role: motd
      system_owner: someone@host.example.com
```

IMPORTANT

Variable precedence can be confusing when working with role variables in a play.

- Almost any other variable will override a role's default variables: inventory variables, play `vars`, inline *role parameters*, and so on.
- Fewer variables can override variables defined in a role's `vars` directory. Facts, variables loaded with `include_vars`, registered variables, and role parameters are some variables that can do that. Inventory variables and play `vars` cannot. This is important because it helps keep your play from accidentally changing the internal functioning of the role.
- However, variables declared inline as role parameters, like the last of the preceding examples, have very high precedence. They can override variables defined in a role's `vars` directory. If a role parameter has the same name as a variable set in play `vars`, a role's `vars`, or an inventory or playbook variable, the role parameter overrides the other variable.

REFERENCES

Using Roles — Ansible Documentation (https://docs.ansible.com/ansible/2.9/user_guide/playbooks_reuse_roles.html#using-roles)

Using Variables — Ansible Documentation (https://docs.ansible.com/ansible/2.9/user_guide/playbooks_variables.html)

← PREVIOUS (/ROL/APP/COURSES/RH294-8.4/PAGES/CH07S04) → NEXT (/ROL/APP/COURSES/RH294-8.4/PAGES/CH07S06)