5. Defining roles

5. Defining roles

Roles are ways of automatically loading certain vars_files, tasks, and handlers based on a known file structure. Grouping content by roles also allows easy sharing of roles with other users.

5.1 Role directory structure

Example project structure:

```
|site.yml
|webservers.yml
|fooservers.yml
|roles
|  |common
|  |  |tasks
|  |  |handlers
|  |  |files
|  |  |templates
|  |  |vars
|  |  |defaults
|  |  |meta
|  |webservers
|  |  |tasks
|  |  |defaults
|  |  |meta
```

Roles expect files to be in certain directory names. Roles must include at least one of these directories, however it is possible to exclude any directory which is not in use. When a directory is used, each directory must contain a main.yml file, which contains the following:

- tasks - contains the main list of tasks to be executed by the role.
- handlers - contains handlers, which may be used by this role or even outside this role.
- defaults - default variables for the role.
- vars - other variables for the role.
- files - contains files which can be deployed using this role.
- templates - contains templates which can be deployed using this role.
- meta - defines some metadata for this role.

Other YAML files may be included in certain directories. For example, it is common practice to have platform-specific tasks included from the tasks/main.yml file:

```
# roles/example/tasks/main.yml
- name: added in 2.4, previously 'include' was used
  import_tasks: redhat.yml
  when: ansible_facts['os_family']|lower == 'redhat'
- import_tasks: debian.yml
  when: ansible_facts['os_family']|lower == 'debian'

# roles/example/tasks/redhat.yml
- yum:
    name: "httpd"
    state: present

# roles/example/tasks/debian.yml
- apt:
    name: "apache2"
    state: present
```

Roles may also include modules and other plugin types.

5.2 Using roles

The classic (original) way to use roles is via the *r*oles: option for a given play:

```
---
- hosts: webservers
  roles:
    - common
    - webservers
```

This designates the following behaviors, for each role ‘x’:

- If roles/x/tasks/main.yml exists, tasks listed therein will be added to the play.
- If roles/x/handlers/main.yml exists, handlers listed therein will be added to the play.
- If roles/x/vars/main.yml exists, variables listed therein will be added to the play.
- If roles/x/defaults/main.yml exists, variables listed therein will be added to the play.
- If roles/x/meta/main.yml exists, any role dependencies listed therein will be added to the list of roles (1.3 and later).
- Any copy, script, template or include tasks (in the role) can reference files in roles/x/{files,templates,tasks}/ (dir depends on task) without having to path them relatively or absolutely.

When used in this manner, the order of execution for your playbook is as follows:

- Any pre_tasks defined in the play.
- Any handlers triggered so far will be run.
- Each role listed in roles will execute in turn. Any role dependencies defined in the roles meta/main.yml will be run first, subject to tag filtering and conditionals.
- Any tasks defined in the play.
- Any handlers triggered so far will be run.
- Any post_tasks defined in the play.
- Any handlers triggered so far will be run.

Note

When using tags with tasks, unsure that you also tag your pre_tasks, post_tasks, and role dependencies. Pass this along as well, especially if the pre/post tasks and role dependencies are used for monitoring outage window control or load balancing.

As of v2.4, you can now use roles in line with any other tasks using import_role or include_role:

```yaml
---
- hosts: webservers
  tasks:
    - debug:
        msg: "before we run our role"
    - import_role:
        name: example
    - include_role:
        name: example
    - debug:
        msg: "after we ran our role"
```

When roles are defined in the classic manner, they are treated as static imports and processed during playbook parsing.
The name used for the role can be a simple name, or it can be a fully qualified path:

```yaml
---
- hosts: webservers
  roles:
    - role: '/path/to/my/roles/common'
```

Roles can also accept other keywords:

```yaml
---
- hosts: webservers
  roles:
    - common
    - role: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
    - role: foo_app_instance
      vars:
        dir: '/opt/b'
        app_port: 5001
```

Or, using the newer syntax:

```
---
- hosts: webservers
  tasks:
    - include_role:
        name: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
```

You can conditionally import a role and execute its tasks:

```
---
- hosts: webservers
  tasks:
    - include_role:
        name: some_role
      when: "ansible_facts['os_family'] == 'RedHat'"
```

Finally, you may wish to assign tags to the tasks inside the roles you specify. You can do that as follows:

```
---
- hosts: webservers
  roles:
    - role: foo
      tags:
        - bar
        - baz
    # using YAML shorthand, this is equivalent to the above:
    - { role: foo, tags: ["bar", "baz"] }
```

Or, again, using the newer syntax:

```
---
- hosts: webservers
  tasks:
    - import_role:
        name: foo
      tags:
        - bar
        - baz
```

Note

This tags all of the tasks in that role with the tags specified, appending to any tags that are specified inside the role.

If you want to tag just the import of the role itself:

```
---
- hosts: webservers
  tasks:
    - include_role:
        name: bar
      tags:
        - foo
```

Note

The tags in this example will not be added to tasks inside an include_role, you can use a surrounding block directive to do both.

Warning

It is not possible to import a role while specifying a subset of tags to execute. If you want to construct a role with many tags and you want to call subsets of the role at different times, you should consider just splitting that role into multiple roles.

5.3 Role default variables

Role default variables allow you to set default variables for included or dependent roles (see below). To create defaults, simply add a code[defaults/main.yml] file in your role directory. These variables will have the lowest priority of any variables available, and can be easily overridden by any other variable, including inventory variables.

5.4 Role dependencies

Role dependencies allow you to automatically pull in other roles when using a role. Role dependencies are stored in the meta/main.yml file within the role directory, as noted above. This file should contain a list of roles and parameters to insert before the specified role, such as the following example roles/myapp/meta/main.yml:

```
---
dependencies:
  - role: common
    vars:
      some_parameter: 3
  - role: apache
    vars:
      apache_port: 80
  - role: postgres
    vars:
      dbname: blarg
      other_parameter: 12
```

Note

Role dependencies must use the classic role definition style. Role dependencies are always executed before the role that includes them, and may be recursive. Dependencies also follow the duplication rules specified above. If another role also lists it as a dependency, it will not be run again based on the same rules given above.

Always remember that when using allow_duplicates: true, it needs to be in the dependent role's meta/main.yml, not the parent.
For example, a role named car depends on a role named wheel as follows:

```
---
dependencies:
  - role: wheel
    vars:
      n: 1
  - role: wheel
    vars:
      n: 2
  - role: wheel
    vars:
      n: 3
  - role: wheel
    vars:
      n: 4
```

And the wheel role depends on two roles: tire and brake. The meta/main.yml for wheel would then contain the following:

```
---
dependencies:
  - role: tire
  - role: brake
```

And the meta/main.yml for tire and brake would have to contain the following:

```
 ---
allow_duplicates: true
```

The resulting execution order would be as follows:

```
tire(n=1)
brake(n=1)
wheel(n=1)
tire(n=2)
brake(n=2)
wheel(n=2)
...
car
```

Note that we did not have to use allow_duplicates: true for wheel, because each instance defined by car uses different parameter values.

5.5 Role duplication and execution

Ansible will only allow one role to execute once, even if defined multiple times, if the parameters defined on the role are not different for each definition. For example:

```
---
- hosts: webservers
  roles:
    - foo
    - foo
```

Given the above, the role foo will only be run once.

To make roles run more than once, there are two options:

- Pass different parameters in each role definition.
- Add allow_duplicates: true to the meta/main.yml file for the role.

Example 1 - passing different parameters:

```
---
- hosts: webservers
  roles:
    - role: foo
      vars:
        message: "first"
    - { role: foo, vars: { message: "second" } }
```

In this example, because each role definition has different parameters, foo will run twice.

Example 2 - using allow_duplicates: true:

```
# playbook.yml
---
- hosts: webservers
  roles:
    - foo
    - foo

# roles/foo/meta/main.yml
---
allow_duplicates: true
```

In this example, foo will run twice because we have explicitly enabled it to do so.

5.6 Role search path

Ansible will search for roles in the following way:

- A roles/ directory, relative to the playbook file.
- By default, in /etc/ansible/roles

In Ansible 1.4 and later you can configure an additional roles_path to search for roles. Use this to check all your common roles out to one location and share them easily between multiple playbook projects. See Configuring Ansible for details about how to set this up in ansible.cfg.