**Roles**

**Topics**

* Roles
  + Role Directory Structure
  + Using Roles
  + Role Duplication and Execution
  + Role Default Variables
  + Role Dependencies
  + Embedding Modules and Plugins In Roles
  + Role Search Path
  + Ansible Galaxy

New in version 1.2.

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.

**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 perfectly fine to exclude any which are not being used. When in use, each directory must contain a main.yml file, which contains the relevant content:

* 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 anywhere outside this role.
* defaults - default variables for the role (see Variables for more information).
* vars - other variables for the role (see Variables for more information).
* files - contains files which can be deployed via this role.
* templates - contains templates which can be deployed via this role.
* meta - defines some meta data for this role. See below for more details.

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, previouslly you used 'include'

import\_tasks: redhat.yml

when: ansible\_os\_platform|lower == 'redhat'

- import\_tasks: debian.yml

when: ansible\_os\_platform|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. For more information, please refer to the Embedding Modules and Plugins In Roles section below.

**Using Roles**

The classic (original) way to use roles is via the roles: 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**

See below for more information regarding role dependencies.

**Note**

If using tags with tasks (described later as a means of only running part of a playbook), be sure to also tag your pre\_tasks, post\_tasks, and role dependencies and pass those along as well, especially if the pre/post tasks and role dependencies are used for monitoring outage window control or load balancing.

As of Ansible 2.4, you can now use roles inline with any other tasks using import\_role or include\_role:

---

- 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.

**Note**

The include\_role option was introduced in Ansible 2.3. The usage has changed slightly as of Ansible 2.4 to match the include (dynamic) vs. import (static) usage. See Dynamic vs. Static for more details.

The name used for the role can be a simple name (see Role Search Path below), or it can be a fully qualified path:

---

- hosts: webservers

roles:

- { role: '/path/to/my/roles/common' }

Roles can accept parameters:

---

- hosts: webservers

roles:

- common

- { role: foo\_app\_instance, dir: '/opt/a', app\_port: 5000 }

- { role: foo\_app\_instance, 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 execute a role. This is not generally recommended with the classic syntax, but is common when using import\_role or include\_role:

---

- hosts: webservers

tasks:

- include\_role:

name: some\_role

when: "ansible\_os\_family == 'RedHat'"

Finally, you may wish to assign tags to the roles you specify. You can do so inline:

---

- hosts: webservers

roles:

- { 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. The tags in this example will *not* be added to tasks inside an include\_role. Tag the include\_role task directly in order to apply tags to tasks in included roles. If you find yourself building a role with lots of tags and you want to call subsets of the role at different times, you should consider just splitting that role into multiple roles.

**Role Duplication and Execution**

Ansible will only allow a 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:

1. Pass different parameters in each role definition.
2. Add allow\_duplicates: true to the meta/main.yml file for the role.

Example 1 - passing different paramters:

---

- hosts: webservers

roles:

- { role: foo, message: "first" }

- { role: foo, 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.

**Role Default Variables**

New in version 1.3.

Role default variables allow you to set default variables for included or dependent roles (see below). To create defaults, simply add a defaults/main.ymlfile 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.

**Role Dependencies**

New in version 1.3.

Role dependencies allow you to automatically pull in other roles when using a role. Role dependencies are stored in the meta/main.yml file contained 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 in an example roles/myapp/meta/main.yml:

---

dependencies:

- { role: common, some\_parameter: 3 }

- { role: apache, apache\_port: 80 }

- { role: postgres, 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.

**Note**

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, n: 1 }

- { role: wheel, n: 2 }

- { role: wheel, n: 3 }

- { role: wheel, 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 contain the following:

---

allow\_duplicates: true

The resulting order of execution 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.

**Note**

Variable inheritance and scope are detailed in the Variables.

**Embedding Modules and Plugins In Roles**

This is an advanced topic that should not be relevant for most users.

If you write a custom module (see Developing Modules) or a plugin (see Developing Plugins), you may wish to distribute it as part of a role. Generally speaking, Ansible as a project is very interested in taking high-quality modules into ansible core for inclusion, so this shouldn’t be the norm, but it’s quite easy to do.

A good example for this is if you worked at a company called AcmeWidgets, and wrote an internal module that helped configure your internal software, and you wanted other people in your organization to easily use this module – but you didn’t want to tell everyone how to configure their Ansible library path.

Alongside the ‘tasks’ and ‘handlers’ structure of a role, add a directory named ‘library’. In this ‘library’ directory, then include the module directly inside of it.

Assuming you had this:

roles/

my\_custom\_modules/

library/

module1

module2

The module will be usable in the role itself, as well as any roles that are called *after* this role, as follows:

- hosts: webservers

roles:

- my\_custom\_modules

- some\_other\_role\_using\_my\_custom\_modules

- yet\_another\_role\_using\_my\_custom\_modules

This can also be used, with some limitations, to modify modules in Ansible’s core distribution, such as to use development versions of modules before they are released in production releases. This is not always advisable as API signatures may change in core components, however, and is not always guaranteed to work. It can be a handy way of carrying a patch against a core module, however, should you have good reason for this. Naturally the project prefers that contributions be directed back to github whenever possible via a pull request.

The same mechanism can be used to embed and distribute plugins in a role, using the same schema. For example, for a filter plugin:

roles/

my\_custom\_filter/

filter\_plugins

filter1

filter2

They can then be used in a template or a jinja template in any role called after ‘my\_custom\_filter’

**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