Roles

Roles let you automatically load related vars, files, tasks, handlers, and other Ansible artifacts based on a known file structure. After you group your content in roles, you can easily reuse them and share them with other users.

Role directory structure

An Ansible role has a defined directory structure with eight main standard directories. You must include at least one of these directories in each role. You can omit any directories the role does not use. For example:

```
# playbooks
site.yml
webservers.yml
fooservers.yml
```

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies
        library/          # roles can also include custom modules
        module_utils/     # roles can also include custom module_utils
        lookup_plugins/   # or other types of plugins, like lookup in this case

    webtier/              # same kind of structure as "common" was above, done for the
webtier role
    monitoring/           # ""
    fooapp/               # ""
```

By default Ansible will look in each directory within a role for a `main.yml` file for relevant content (also `main.yaml` and `main`):

- `tasks/main.yml` - the main list of tasks that the role executes.
- `handlers/main.yml` - handlers, which may be used within or outside this role.
- `library/my_module.py` - modules, which may be used within this role (see Embedding modules and plugins in roles for more information).
- `defaults/main.yml` - default variables for the role (see Using Variables (playbooks_variables.html#playbooks-variables) for more information). These variables have the lowest priority of any variables available, and can be easily overridden by any other variable, including inventory variables.

- `vars/main.yml` - other variables for the role (see Using Variables (playbooks_variables.html#playbooks-variables) for more information).
- `files/main.yml` - files that the role deploys.
- `templates/main.yml` - templates that the role deploys.
- `meta/main.yml` - metadata for the role, including role dependencies and optional Galaxy metadata such as platforms supported.

You can add other YAML files in some directories. For example, you can place platform-specific tasks in separate files and refer to them in the `tasks/main.yml` file:

```
# roles/example/tasks/main.yml
- name: Install the correct web server for RHEL
  import_tasks: redhat.yml
  when: ansible_facts['os_family']|lower == 'redhat'

- name: Install the correct web server for Debian
  import_tasks: debian.yml
  when: ansible_facts['os_family']|lower == 'debian'

# roles/example/tasks/redhat.yml
- name: Install web server
  ansible.builtin.yum:
    name: "httpd"
    state: present

# roles/example/tasks/debian.yml
- name: Install web server
  ansible.builtin.apt:
    name: "apache2"
    state: present
```

Roles may also include modules and other plugin types in a directory called `library`. For more information, please refer to Embedding modules and plugins in roles below.

Storing and finding roles

By default, Ansible looks for roles in the following locations:

- in collections, if you are using them
- in a directory called `roles/`, relative to the playbook file
- in the configured roles_path (../reference_appendices/config.html#default-roles-path). The default search path is `~/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles`.
- in the directory where the playbook file is located

If you store your roles in a different location, set the roles_path (../reference_appendices/config.html#default-roles-path) configuration option so Ansible can find your roles. Checking shared roles into a single location makes them easier to use in

multiple playbooks. See Configuring Ansible (../installation_guide/intro_configuration.html#intro-configuration) for details about managing settings in ansible.cfg.

Alternatively, you can call a role with a fully qualified path:

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

Using roles

You can use roles in three ways:

- at the play level with the `roles` option: This is the classic way of using roles in a play.
- at the tasks level with `include_role`: You can reuse roles dynamically anywhere in the `tasks` section of a play using `include_role`.
- at the tasks level with `import_role`: You can reuse roles statically anywhere in the `tasks` section of a play using `import_role`.

Using roles at the play level

The classic (original) way to use roles is with the `roles` option for a given play:

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

When you use the `roles` option at the play level, for each role 'x':

- If roles/x/tasks/main.yml exists, Ansible adds the tasks in that file to the play.
- If roles/x/handlers/main.yml exists, Ansible adds the handlers in that file to the play.
- If roles/x/vars/main.yml exists, Ansible adds the variables in that file to the play.
- If roles/x/defaults/main.yml exists, Ansible adds the variables in that file to the play.
- If roles/x/meta/main.yml exists, Ansible adds any role dependencies in that file to the list of roles.
- 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 you use the `roles` option at the play level, Ansible treats the roles as static imports and processes them during playbook parsing. Ansible executes each play in this order:

- Any `pre_tasks` defined in the play.
- Any handlers triggered by pre_tasks.
- Each role listed in `roles:`, in the order listed. Any role dependencies defined in the role's `meta/main.yml` run first, subject to tag filtering and conditionals. See Using role dependencies for more details.
- Any `tasks` defined in the play.
- Any handlers triggered by the roles or tasks.
- Any `post_tasks` defined in the play.
- Any handlers triggered by post_tasks.

Note

If using tags with tasks in a role, 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. See Tags (playbooks_tags.html#tags) for details on adding and using tags.

You can pass other keywords to the `roles` option:

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

When you add a tag to the `role` option, Ansible applies the tag to ALL tasks within the role.

When using `vars:` within the `roles:` section of a playbook, the variables are added to the play variables, making them available to all tasks within the play before and after the role. This behavior can be changed by DEFAULT_PRIVATE_ROLE_VARS (../reference_appendices/config.html#default-private-role-vars).

Including roles: dynamic reuse

You can reuse roles dynamically anywhere in the `tasks` section of a play using `include_role`. While roles added in a `roles` section run before any other tasks in a play, included roles run in the order they are defined. If there are other tasks before an `include_role` task, the other tasks will run first.

To include a role:

```
---
- hosts: webservers
  tasks:
    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs before the example role"

    - name: Include the example role
      include_role:
        name: example

    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs after the example role"
```

You can pass other keywords, including variables and tags, when including roles:

```
---
- hosts: webservers
  tasks:
    - name: Include the foo_app_instance role
      include_role:
        name: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
      tags: typeA
  ...
```

When you add a tag (playbooks_tags.html#tags) to an `include_role` task, Ansible applies the tag *only* to the include itself. This means you can pass `--tags` to run only selected tasks from the role, if those tasks themselves have the same tag as the include statement. See Selectively running tagged tasks in re-usable files (playbooks_tags.html#selective-reuse) for details.

You can conditionally include a role:

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

Importing roles: static reuse

You can reuse roles statically anywhere in the `tasks` section of a play using `import_role`. The behavior is the same as using the `roles` keyword. For example:

```
---
- hosts: webservers
  tasks:
    - name: Print a message
      ansible.builtin.debug:
        msg: "before we run our role"

    - name: Import the example role
      import_role:
        name: example

    - name: Print a message
      ansible.builtin.debug:
        msg: "after we ran our role"
```

You can pass other keywords, including variables and tags, when importing roles:

```
---
- hosts: webservers
  tasks:
    - name: Import the foo_app_instance role
      import_role:
        name: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
  ...
```

When you add a tag to an `import_role` statement, Ansible applies the tag to *all* tasks within the role. See Tag inheritance: adding tags to multiple tasks (playbooks_tags.html#tag-inheritance) for details.

Role argument validation

Beginning with version 2.11, you may choose to enable role argument validation based on an argument specification. This specification is defined in the `meta/argument_specs.yml` file (or with the `.yaml` file extension). When this argument specification is defined, a new task is

inserted at the beginning of role execution that will validate the parameters supplied for the role against the specification. If the parameters fail validation, the role will fail execution.

Note

Ansible also supports role specifications defined in the role `meta/main.yml` file, as well. However, any role that defines the specs within this file will not work on versions below 2.11. For this reason, we recommend using the `meta/argument_specs.yml` file to maintain backward compatibility.

Note

When role argument validation is used on a role that has defined dependencies, then validation on those dependencies will run before the dependent role, even if argument validation fails for the dependent role.

Specification format

The role argument specification must be defined in a top-level `argument_specs` block within the role `meta/argument_specs.yml` file. All fields are lower-case.

entry-point-name:

- The name of the role entry point.
- This should be `main` in the case of an unspecified entry point.
- This will be the base name of the tasks file to execute, with no `.yml` or `.yaml` file extension.

short_description:

- A short, one-line description of the entry point.
- The `short_description` is displayed by `ansible-doc -t role -l`.

description:

- A longer description that may contain multiple lines.

author:

- Name of the entry point authors.
- Use a multi-line list if there is more than one author.

options:

- Options are often called “parameters” or “arguments”. This section defines those options.
- For each role option (argument), you may include:

option-name:

- The name of the option/argur

description:

- Detailed explanation of what option does. It should be writt full sentences.

type:

- The data type of the option. S Argument spec (../dev_guide/developing_prog

	low_modules.html#argument- for allowed values for `type`. Default is `str`.
	• If an option is of type `list`, `elements` should be specified
required:	• Only needed if `true`. • If missing, the option is not required.
default:	• If `required` is false/missing, `default` may be specified (assumed 'null' if missing). • Ensure that the default value docs matches the default valu the code. The actual default f role variable will always come `defaults/main.yml`. • The default field must not be as part of the description, unl requires additional informatio conditions. • If the option is a boolean valu should use *true/false* if you wa be compatible with *ansible-lin*
choices:	• List of option values. • Should be absent if empty.
elements:	• Specifies the data type for list elements when type is `list`.
options:	• If this option takes a dict or lis dicts, you can define the struc here.

Sample specification

```
# roles/myapp/meta/argument_specs.yml
---
argument_specs:
  # roles/myapp/tasks/main.yml entry point
  main:
    short_description: The main entry point for the myapp role.
    options:
      myapp_int:
        type: "int"
        required: false
        default: 42
        description: "The integer value, defaulting to 42."

      myapp_str:
        type: "str"
        required: true
        description: "The string value"

  # roles/myapp/tasks/alternate.yml entry point
  alternate:
    short_description: The alternate entry point for the myapp role.
    options:
      myapp_int:
        type: "int"
        required: false
        default: 1024
        description: "The integer value, defaulting to 1024."
```

Running a role multiple times in one play

Ansible only executes each role once in a play, even if you define it multiple times, unless the parameters defined on the role are different for each definition. For example, Ansible only runs the role `foo` once in a play like this:

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

You have two options to force Ansible to run a role more than once.

Passing different parameters

If you pass different parameters in each role definition, Ansible runs the role more than once. Providing different variable values is not the same as passing different role parameters. You must use the `roles` keyword for this behavior, since `import_role` and `include_role` do not accept role parameters.

This play runs the `foo` role twice:

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

This syntax also runs the `foo` role twice;

```
---
- hosts: webservers
  roles:
    - role: foo
      message: "first"
    - role: foo
      message: "second"
```

In these examples, Ansible runs `foo` twice because each role definition has different parameters.

Using `allow_duplicates: true`

Add `allow_duplicates: true` to the `meta/main.yml` file for the role:

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

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

In this example, Ansible runs `foo` twice because we have explicitly enabled it to do so.

Using role dependencies

Role dependencies let you automatically pull in other roles when using a role.

Role dependencies are prerequisites, not true dependencies. The roles do not have a parent/child relationship. Ansible loads all listed roles, runs the roles listed under `dependencies` first, then runs the role that lists them. The play object is the parent of all roles, including roles called by a `dependencies` list.

Role dependencies are stored in the `meta/main.yml` file within the role directory. This file should contain a list of roles and parameters to insert before the specified role. For 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
```

Ansible always executes roles listed in `dependencies` before the role that lists them. Ansible executes this pattern recursively when you use the `roles` keyword. For example, if you list role `foo` under `roles:`, role `foo` lists role `bar` under `dependencies` in its meta/main.yml file, and role `bar` lists role `baz` under `dependencies` in its meta/main.yml, Ansible executes `baz`, then `bar`, then `foo`.

Running role dependencies multiple times in one play

Ansible treats duplicate role dependencies like duplicate roles listed under `roles:`: Ansible only executes role dependencies once, even if defined multiple times, unless the parameters, tags, or when clause defined on the role are different for each definition. If two roles in a play both list a third role as a dependency, Ansible only runs that role dependency once, unless you pass different parameters, tags, when clause, or use `allow_duplicates: true` in the role you want to run multiple times. See Galaxy role dependencies (../galaxy/user_guide.html#galaxy-dependencies) for more details.

Note

Role deduplication does not consult the invocation signature of parent roles. Additionally, when using `vars:` instead of role params, there is a side effect of changing variable scoping. Using `vars:` results in those variables being scoped at the play level. In the below example, using `vars:` would cause `n` to be defined as `4` through the entire play, including roles called before it.

In addition to the above, users should be aware that role de-duplication occurs before variable evaluation. This means that Lazy Evaluation (../reference_appendices/glossary.html#term-Lazy-Evaluation) may make seemingly different role invocations equivalently the same, preventing the role from running more than once.

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
```

To use `allow_duplicates: true` with role dependencies, you must specify it for the role listed under `dependencies`, not for the role that lists it. In the example above, `allow_duplicates: true` appears in the `meta/main.yml` of the `tire` and `brake` roles. The `wheel` role does not require `allow_duplicates: true`, because each instance defined by `car` uses different parameter values.

Note

See Using Variables (playbooks_variables.html#playbooks-variables) for details on how Ansible chooses among variable values defined in different places (variable inheritance and scope). Also deduplication happens ONLY at the play level, so multiple plays in the same playbook may rerun the roles.

Embedding modules and plugins in roles

Note

This applies only to standalone roles. Roles in collections do not support plugin embedding; they must use the collection's `plugins` structure to distribute plugins.

If you write a custom module (see Should you develop a module? (../dev_guide/developing_modules.html#developing-modules)) or a plugin (see Developing plugins (../dev_guide/developing_plugins.html#developing-plugins)), you might wish to distribute it as part of a role. For example, if you write a module that helps configure your company's internal software, and you want other people in your organization to use this module, but you do not want to tell everyone how to configure their Ansible library path, you can include the module in your internal_config role.

To add a module or a plugin to a role: Alongside the 'tasks' and 'handlers' structure of a role, add a directory named 'library' and then include the module directly inside the 'library' directory.

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
```

If necessary, you can also embed a module in a role to modify a module in Ansible's core distribution. For example, you can use the development version of a particular module before it is released in production releases by copying the module and embedding the copy in a role. Use this approach with caution, as API signatures may change in core components, and this workaround is not guaranteed to work.

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
```

These filters can then be used in a Jinja template in any role called after 'my_custom_filter'.

Sharing roles: Ansible Galaxy

Ansible Galaxy (https://galaxy.ansible.com) is a free site for finding, downloading, rating, and reviewing all kinds of community-developed Ansible roles and can be a great way to get a jumpstart on your automation projects.

The client `ansible-galaxy` is included in Ansible. The Galaxy client allows you to download roles from Ansible Galaxy and provides an excellent default framework for creating your own roles.

Read the Ansible Galaxy documentation (https://galaxy.ansible.com/docs/) page for more information. A page that refers back to this one frequently is the Galaxy Roles document which explains the required metadata your role needs for use in Galaxy <https://galaxy.ansible.com/docs/contributing/creating_role.html (https://galaxy.ansible.com/docs/contributing/creating_role.html)>.

See also

Galaxy User Guide (../galaxy/user_guide.html#ansible-galaxy)
How to create new roles, share roles on Galaxy, role management

YAML Syntax (../reference_appendices/YAMLSyntax.html#yaml-syntax)
Learn about YAML syntax

Working with playbooks (playbooks.html#working-with-playbooks)
Review the basic Playbook language features

General tips (../tips_tricks/ansible_tips_tricks.html#tips-and-tricks)
Tips and tricks for playbooks

Using Variables (playbooks_variables.html#playbooks-variables)
Variables in playbooks

Conditionals (playbooks_conditionals.html#playbooks-conditionals)
Conditionals in playbooks

Loops (playbooks_loops.html#playbooks-loops)
Loops in playbooks

Tags (playbooks_tags.html#tags)
Using tags to select or skip roles/tasks in long playbooks

Collection Index (../collections/index.html#list-of-collections)
Browse existing collections, modules, and plugins

Should you develop a module? (../dev_guide/developing_modules.html#developing-modules)
Extending Ansible by writing your own modules

GitHub Ansible examples (https://github.com/ansible/ansible-examples)
Complete playbook files from the GitHub project source

Mailing List (https://groups.google.com/group/ansible-project)
Questions? Help? Ideas? Stop by the list on Google Groups