Ansible Collection & Role Best Practices

[Document Revision History 2](#_Toc118442957)

[Overview 2](#_Toc118442958)

[Ansible Roles 2](#_Toc118442959)

[Structure 2](#_Toc118442960)

[Defaults 3](#_Toc118442961)

[Files 3](#_Toc118442962)

[Handlers 4](#_Toc118442963)

[Meta 4](#_Toc118442964)

[Tasks 4](#_Toc118442965)

[Templates 4](#_Toc118442966)

[Tests 4](#_Toc118442967)

[Vars 5](#_Toc118442968)

[Best Practices 5](#_Toc118442969)

[Real-world examples 5](#_Toc118442970)

[Ansible Collections 5](#_Toc118442971)

[Structure 6](#_Toc118442972)

[Galaxy.yml 6](#_Toc118442973)

[Docs 6](#_Toc118442974)

[Meta 7](#_Toc118442975)

[Playbooks 7](#_Toc118442976)

[Plugins 7](#_Toc118442977)

[Roles 8](#_Toc118442978)

[Tests 8](#_Toc118442979)

[Best Practices 8](#_Toc118442980)

[Real-world examples 9](#_Toc118442981)

[Takeaways 9](#_Toc118442982)

# Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Changes** |
| 1.0 | 11/04/2022 | Stephen Zuk | Document Origination |

# 

# Overview

The goal of this document is to provide guidance both from the documentation and from our consultants' experience on how to structure roles and collections.

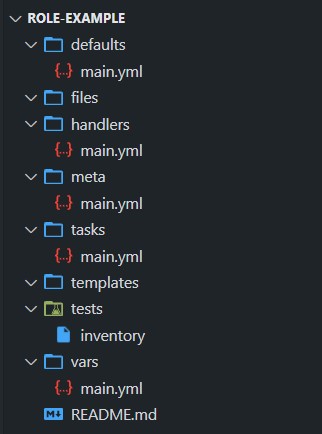
Keeping your Ansible code organized and well defined from the start will help alleviate struggles to understand the code as time goes on. Whenever you are writing code, you should always ***remember the next guy***. Someone else in the future will likely be building on top of your code, or just trying to understand it when it breaks. Having good structure, documentation, and proper coding standards helps create a level playing field that everyone can understand.

Also, keep in mind that while Ansible/RedHat has standards which in most cases should be followed, every customer has their own ways of coding, and building out solutions that make sense to them in their environment. No one way is always right, but as long as you keep a general structure and syntax (variables, file names, code comments, etc.) it will help allow people to quickly scan your code, and gain an understanding of what’s happening, regardless of the content.

# Ansible Roles

## Structure

Provided below is an example role structure including most of the important directories you will be using on a daily basis. Refer to the [Ansible Role](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html#role-directory-structure) [Directory Structure](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html#role-directory-structure) documentation for all available options.



### defaults

The defaults directory is for defining the variable defaults. The variables in default have the lowest priority thus becoming easy to override. If the definition of a variable is not defined anywhere else in code, the variable in defaults/main.yml will be used.

This is a great place to start with all variables that are ***needed*** for the role to function. Typically I will place every single variable that is assumed to be defined in code here, and then overwrite the contents with **Extra Vars** through Ansible Tower Surveys, or just through **Vars** files within the code itself. Personally, I like to use the *defaults/main.yml* as my documentation of variables.

### files

We use the files directory to add files that are needed by the target machine, without modification. Mostly, we use copy tasks for referencing files in the files directory. Ansible does not require a path for resources stored in files directory when working in the role.

Note, that you don’t need to refer to the exact location of the file in question. Ansible knows that if you’re trying to use the copy module, or like module, and you specify a file name it will try to find it in this ***files*** directory. Similar to the ***Templates*** directory.

### handlers

The *handlers* directory is used for storing Ansible handlers. Handlers are tasks that may be flagged during a play to run at the play’s completion. We can have as many and as few handlers as we need.

Note, that handlers will run at the ***end*** of the Ansible runtime. If you want to force handlers to run sooner or in the middle of your playbook, you’d want to use the ***meta*** plugin built-in to Ansible to force the handlers to refresh. See the [ansible.builtin.meta](https://docs.ansible.com/ansible/latest/collections/ansible/builtin/meta_module.html) documentation for more information around this.

### meta

We use the *meta* directory to store authorship information which is useful if we choose to publish our role on *galaxy.ansible.com or your private automation hub*. The metadata of an Ansible role consists of author, supported platforms, and dependencies. While not required it’s recommended you have a meta/ directory as there are some instances where Ansible requires it.

### tasks

The *task* directory is where we write most of the logic of our roles which includes each task our role will perform. We write each series of tasks in a separate file and include them into the *main.yml* file in the *tasks* directory.

It’s recommended that, if it’s only a few tasks in a role, you could put them all in main.yml. However, if you’re seeing that your main.yml starts to grow past a couple hundred lines or over a dozen tasks, you should look to break those tasks apart into smaller tasks files for easier legibility by other engineers.

### templates

We use the *template* directory to also add files to our machine(similar to *files* directory). Only difference between *template* and *files* directories is that the *template* directory supports alteration (modification). [Jinja2](https://jinja.palletsprojects.com/en/master/) language is used to create these alterations. Most software configuration files become templates.

### tests

We can use the *tests* directory if we have built an automated testing process around our role. This directory contains a sample inventory and a *test.yml* file. This would be useful if you wanted to use Jenkins to run automated tests.

### vars

This is where we create variable files that define necessary variables for our role. The variables defined in this directory are meant for role internal use only. Also, it is a good idea to namespace our role variable names, to prevent potential naming conflicts with variables outside of our role.

## Best Practices

* Always include a README.md in your roles, provide a general description of what this role does, what platforms it supports, and any required variables/collections. This can sometimes be tedious to continuously update, but it’s vital for those who want to take a look and understand your role at a high level quickly.
* Once your tasks/main.yml grows past ~100-200 lines or past ~12 tasks, you should look to split your code into multiple tasks files which are grouped by like tasks. For example, if you were deploying a Web application, and you were installing package dependencies, configuring the environment, installing an application, configuring an application, and then starting the application. Those would be good task files for each grouped set of tasks.
* Knowing when to create a role is important, and even more important to know when to put that role in a collection or leave it standalone. Creating a role should typically only be done when you have a use-case that has grown past a few simple tasks. My rule of thumb is once my initial development playbook has grown past 10 tasks It’s typically time for a role. Another immediate use-case for a role, is if you have other content that needs to be packaged such as templated files, or binaries to distribute.

## Real-world examples

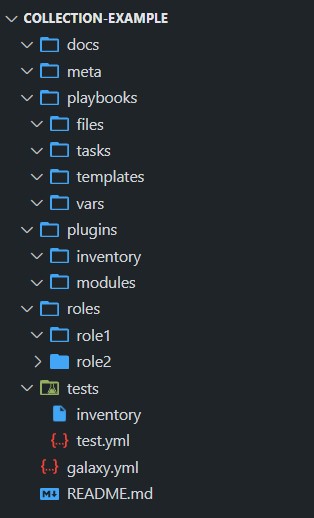
* <https://github.com/geerlingguy/ansible-role-ansible>
* <https://github.com/geerlingguy/ansible-role-gitlab>

-

# Ansible Collections

Provided below is an example role structure including most of the important directories you will be using on a daily basis. Refer to the [Ansible Collection](https://docs.ansible.com/ansible/latest/dev_guide/developing_collections_structure.html) [Directory Structure](https://docs.ansible.com/ansible/latest/dev_guide/developing_collections_structure.html) documentation for all available options.

## Structure



### Galaxy.yml

A collection must have a galaxy.yml file that contains the necessary information to build a collection artifact. See [Collection Galaxy metadata](https://docs.ansible.com/ansible/latest/dev_guide/collections_galaxy_meta.html#collections-galaxy-meta) [structure](https://docs.ansible.com/ansible/latest/dev_guide/collections_galaxy_meta.html#collections-galaxy-meta) for details. This is a ***required*** file in order for your collection to build and publish correctly. This is often missed the first few times creating collections.

### Docs

Use the docs folder to describe how to use the roles and plugins the collection provides, role requirements, and so on. I typically will put all my module, role, and playbook documentation in

this directory. You can then link to the individual ***.md*** files on your main *README.md* in the root of your repository. This is common practice and is really recommended if you’re developing large collections which may have multiple roles all requiring different variables to be defined.

### Meta

A collection can store some additional metadata in a runtime.yml file in the collection’s meta directory. Typically, in most cases especially internally you won’t likely use this feature very much, but it can be useful. Check out our official documentation around, what is supported here: [https://docs.ansible.com/ansible/latest/dev\_guide/developing\_collections\_structure.html#meta -directory](https://docs.ansible.com/ansible/latest/dev_guide/developing_collections_structure.html#meta-directory)

### Playbooks

In prior releases, you could reference playbooks in this directory using the full path to the playbook file from the command line. In ansible-core 2.11 and later, you can use the FQCN, namespace.collection.playbook (with or without extension), to reference the playbooks from the command line or from import\_playbook. This will keep the playbook in ‘collection context’, as if you had added collections: [ namespace.collection ] to it.

You can have most of the subdirectories you would expect, such files/, vars/ or templates/ but no roles/ since those are handled already in the collection.

### Plugins

Add a ‘per plugin type’ specific subdirectory here, including module\_utils which is usable not only by modules, but by most plugins by using their FQCN. This is a way to distribute modules, lookups, filters, and so on without having to import a role in every play.

Vars plugins are unsupported in collections. Cache plugins may be used in collections for fact caching but are not supported for inventory plugins.

Something else to keep in mind, is that if you decide to develop your own ***module\_utils***, you will need to refer to them by their FQDN inside your python code. More information, around that can be found on the official docs:

Page [https://docs.ansible.com/ansible/latest/dev\_guide/developing\_collections\_structure.html#mod ule-utils](https://docs.ansible.com/ansible/latest/dev_guide/developing_collections_structure.html#module-utils)

### Roles

Collection roles are mostly the same as existing roles, but with a couple of limitations:

* Role names are now limited to contain only lowercase alphanumeric characters, plus \_ and start with an alpha character.
* Roles in a collection cannot contain plugins any more. Plugins must live in the collection plugins directory tree. Each plugin is accessible to all roles in the collection.

The directory name of the role is used as the role name. Therefore, the directory name must comply with the above role name rules. The collection import into Galaxy will fail if a role name does not comply with these rules.

### Tests

Ansible Collections are tested much like Ansible itself, by using the ansible-test utility which is released as part of Ansible, version 2.9.0 and newer. Because Ansible Collections are tested using the same tooling as Ansible itself, via ansible-test, all Ansible developer documentation for testing is applicable for authoring Collections Tests with one key concept to keep in mind.

See [Testing collections](https://docs.ansible.com/ansible/latest/dev_guide/developing_collections_testing.html#testing-collections) for specific information on how to test collections with ansible-test.

Likely you will be performing these tests on pushes to your repository in Jenkins. You would need test target infrastructure in place for Jenkins to run a good majority of tests. For example, if you wanted to ensure that you could operate a role to deploy a Tomcat application, there would need to either be an EC2 instance spun up on demand for testing, or have existing infrastructure in place to run the automated tests against.

## Best Practices

* Just like knowing when to create a role, knowing when to package content into a collection is equally important. You don’t want to default to one option or the other or both, there are use-cases for each. My rule of thumb is, if you have custom modules, such as inventory plugins or action plugins, then a collection likely makes sense. Other use-cases for collections are when you’re grouping like content together, such as roles, and playbooks that you want to be able to share to other teams/organizations to consume. Collections can be brought in automatically using ***requirements.yml*** file in the *collections* directory.

## Real-world examples

* <https://github.com/ansible-collections/ansible.utils>
* <https://github.com/ansible-collections/community.vmware>
* <https://github.com/ansible-collections/ansible.netcommon>
* <https://github.com/ansible-collections/community.general>
* <https://github.com/ansible-collections/cisco.ios>
* <https://github.com/ansible-collections/community.mysql>

# Takeaways

* Naming conventions are crucial, when it comes to naming **collections**, and **roles**. It’s also equally as important when it comes to naming **variables** in your code. Having a very rigid structure means that you can almost assume where or what a variable is used for just by the name of it.
* Roles and collections both have a pretty set-in-stone structure, as Ansible only supports certain directories out of the box. As long as you keep with this structure, almost all roles/collections will look the same in your environment.
* Writing good documentation is time consuming, difficult and oftentimes obsolete by the time you update the code at some point. But it’s what 99% of people are going to be looking for when they are trying to understand what the code is doing, or just trying to use the content as a consumer.
* Start small with your use-cases first. Try writing a single playbook, then expand it into a role, and further into a collection should you need it to exist in that format. Typically roles are only necessary when you have more than a dozen or so tasks in a single playbook, or you start needing the extra features like **vars, templates, files,** etc.
* Don’t just start with a role, or collection out of the box unless you are 100% certain it needs those extra features. A lot of times, ansible playbooks can stay just as ansible playbooks and get the job done.