DevOps Shack

Ansible Collections Documentation

Ansible **Collections** are a distribution format for Ansible content, including roles, modules, plugins, playbooks, and documentation. Collections make it easier to manage and share reusable content across different projects or teams. Instead of managing individual modules, roles, or plugins, Ansible Collections group all related content in a structured way, allowing users to share and reuse them efficiently.

Here's a detailed guide on how to use Ansible Collections, from installation to practical usage.

---

1. What Are Ansible Collections?

- **Ansible Collections**: A collection is a bundle of Ansible content, including:
    - **Roles**
    - **Modules**
    - **Plugins** (e.g., filter, lookup, callback, or connection plugins)
    - **Playbooks**
    - **Documentation**
    - **Test cases**

Collections are designed to make it easier to manage and distribute Ansible content as a single unit.

Why Use Ansible Collections?

- **Modularity**: Collections bundle content together, making it easier to manage.
- **Reuse**: They allow the reuse of common Ansible content (e.g., modules or roles) across different projects.
- **Sharing**: They facilitate sharing content within teams or via platforms like Ansible Galaxy and Automation Hub.

---

2. Installing and Using Collections

Step 1: Install a Collection

To install a collection, you use the ansible-galaxy command-line tool. Collections can be installed from:

- **Ansible Galaxy** (the community platform for Ansible content).
- **Automation Hub** (a private or enterprise repository for certified content).

Example: Install a Collection from Ansible Galaxy

The basic syntax for installing a collection is as follows:

```
ansible-galaxy collection install <namespace>.<collection_name>
```

For instance, to install the popular **community.general** collection:

```
ansible-galaxy collection install community.general
```

This command downloads the collection and installs it in the default path: ~/.ansible/collections/ansible_collections/.

Specifying a Version

You can also specify a particular version of a collection using ==<version>:

```
ansible-galaxy collection install community.general==3.4.0
```

Installing from Automation Hub

If you are using a private Automation Hub (e.g., Red Hat Automation Hub), you need to authenticate first, and then install the collection:

```
ansible-galaxy collection install my_company_namespace.my_collection_name
```

Step 2: List Installed Collections

To view all the collections installed on your system, run:

```
ansible-galaxy collection list
```

This will list all installed collections, along with their versions and paths.

---

3. Using Collections in Playbooks

Once you've installed a collection, you can reference its content (modules, roles, plugins) in your Ansible playbooks. To use content from a collection, you need to prefix the name of the module or role with the collection name.

Using Modules from a Collection

If you have a collection that contains modules, you need to specify the fully qualified collection name (FQCN) when referencing a module.

Example: Using a Module from a Collection

For example, the **community.general** collection contains the ping module:

```
---
- name: Test connection using a module from the collection
  hosts: localhost
  tasks:
    - name: Run ping module from the community.general collection
      community.general.ping:
```

In this case:

- community.general is the collection name.
- ping is the module from that collection.

If a module is available in multiple collections, it's essential to use the **fully qualified collection name (FQCN)** to avoid ambiguity.

Using Roles from a Collection

Roles inside a collection can be used similarly. You need to specify the full name of the role, including the collection name.

Example: Using a Role from a Collection

Assume there's a role named nginx inside the collection my_company.webserver:

```
---
- name: Apply nginx role from my_company.webserver collection
  hosts: webservers
  roles:
    - my_company.webserver.nginx
```

In this example:

- my_company.webserver is the collection name.
- nginx is the role within that collection.

---

4. Managing Collections with requirements.yml

When working with multiple collections in a project, it's often useful to manage dependencies using a requirements.yml file. This file allows you to define the collections that should be installed automatically for your project.

Creating a requirements.yml File

The requirements.yml file is used to specify collections that need to be installed.

Example of requirements.yml:

```
collections:
 - name: community.general
   version: "3.4.0"

 - name: ansible.posix
   version: "1.2.0"

 - name: my_company_namespace.my_collection_name
```

This file specifies three collections:

- community.general version 3.4.0
- ansible.posix version 1.2.0
- my_company_namespace.my_collection_name with no specific version (it will install the latest available version).

Installing Collections from requirements.yml

To install all collections listed in a requirements.yml file:

```
ansible-galaxy collection install -r requirements.yml
```

This will install all the collections and their dependencies as specified.

---

5. Writing and Creating Your Own Collection

Ansible Collections make it easy for developers to package, distribute, and reuse Ansible content. You can create your own collection with roles, modules, and plugins.

Step 1: Initialize a New Collection

You can use the ansible-galaxy command to initialize a new collection structure:

```
ansible-galaxy collection init <namespace>.<collection_name>
```

For example:

```
ansible-galaxy collection init my_company.my_collection
```

This command creates the following directory structure:

```
my_company/
└── my_collection/
    ├── docs/
    ├── files/
    ├── plugins/
    ├── playbooks/
    ├── roles/
    └── tests/
```

- **docs/**: Documentation for the collection.
- **files/**: Static files used by roles or modules.
- **plugins/**: Custom Ansible plugins (e.g., action, lookup, or filter plugins).
- **playbooks/**: Any playbooks packaged in the collection.
- **roles/**: Roles included in the collection.
- **tests/**: Test cases and testing configurations.

Step 2: Add Roles or Modules

Populate the roles/, playbooks/, or plugins/ directories with your Ansible content.

For example, to add a role:

- Navigate to roles/ and create your role using the standard role directory structure (tasks, handlers, templates, files, etc.).

```
cd my_company/my_collection/roles/
```

```
ansible-galaxy init my_role
```

You can also add modules or custom plugins in the plugins/ directory.

Step 3: Build and Share Your Collection

Once your collection is complete, you can package it and share it on platforms like **Ansible Galaxy** or **Automation Hub**.

Building a Collection:

To build your collection into a tarball package, navigate to the root of your collection and run:

```
ansible-galaxy collection build
```

This command creates a .tar.gz file containing all the collection's content. This package can be uploaded to Ansible Galaxy or shared privately.

Publishing to Ansible Galaxy:

To share your collection on Ansible Galaxy, you can use the following command after building it:

```
ansible-galaxy collection publish <path_to_tar_gz>
```

Make sure you have a Galaxy account linked to your GitHub profile for seamless publication.

---

6. Using Collections from Git Repositories

You can also use collections stored in Git repositories. This is useful for developing or testing new collections that are not yet published to Ansible Galaxy.

Install a Collection from a Git Repository

You can specify the source as a Git URL when installing a collection:

```
ansible-galaxy collection install git+https://github.com/<user>/<collection>.git
```

If you want to install a specific branch or tag:

```
ansible-galaxy collection install
git+https://github.com/<user>/<collection>.git,branch=<branch_name>
```

---

7. Best Practices for Using Collections

- **Use Fully Qualified Names:** Always use fully qualified collection names (FQCN) to ensure compatibility and avoid conflicts between modules and roles in different collections.
- **Pin Versions in requirements.yml:** Always pin the version of collections in your requirements.yml file to avoid compatibility issues in the future.
- **Share Reusable Code via Collections:** If you have content (roles, modules, or plugins) that is reused across multiple projects, consider packaging it into a collection for easier management and distribution.
- **Namespace Collections Appropriately:** If creating your own collection, use meaningful namespaces and names to avoid collisions with existing content.

Conclusion

Ansible Collections provide a powerful way to package, distribute, and reuse Ansible content, including roles, modules, and plugins. Whether you are managing infrastructure at scale or sharing reusable content with your team, collections help improve modularity, reusability, and collaboration within the Ansible ecosystem. By following the steps outlined in this guide, you can install, use, and even create your own collections for various automation use cases.