Using shared resources in collections

Although developing Ansible modules contained in collections is similar to developing standalone Ansible modules, you use shared resources like documentation fragments and module utilities differently in collections. You can use documentation fragments within and across collections. You can use optional module utilities to support multiple versions of ansible-core in your collection. Collections can also depend on other collections.

Using documentation fragments in collections

To include documentation fragments in your collection:

1. Create the documentation fragment: `plugins/doc_fragments/fragment_name`.
2. Refer to the documentation fragment with its FQCN.

```
extends_documentation_fragment:
  - kubernetes.core.k8s_name_options
  - kubernetes.core.k8s_auth_options
  - kubernetes.core.k8s_resource_options
  - kubernetes.core.k8s_scale_options
```

:ref:`module_docs_fragments` covers the basics for documentation fragments. The kubernetes.core collection includes a complete example.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\(ansible-devel)(docs)(docsite)(rst)(dev_guide)developing_collections_shared.rst, line 32); *backlink***
>
> Unknown interpreted text role "ref".

If you use FQCN, you can use documentation fragments from one collection in another collection.

Leveraging optional module utilities in collections

Optional module utilities let you adopt the latest features from the most recent ansible-core release in your collection-based modules without breaking your collection on older Ansible versions. With optional module utilities, you can use the latest features when running against the latest versions, while still providing fallback behaviors when running against older versions.

This implementation, widely used in Python programming, wraps optional imports in conditionals or defensive *try/except* blocks, and implements fallback behaviors for missing imports. Ansible's module payload builder supports these patterns by treating any module_utils import nested in a block (e.g., *if*, *try*) as optional. If the requested import cannot be found during the payload build, it is simply omitted from the target payload and assumed that the importing code will handle its absence at runtime. Missing top-level imports of module_utils packages (imports that are not wrapped in a block statement of any kind) will fail the module payload build, and will not execute on the target.

For example, the *ansible.module_utils.common.respawn* package is only available in Ansible 2.11 and higher. The following module code would fail during the payload build on Ansible 2.10 or earlier (as the requested Python module does not exist, and is not wrapped in a block to signal to the payload builder that it can be omitted from the module payload):

```
from ansible.module_utils.common.respawn import respawn_module
```

By wrapping the import statement in a `try` block, the payload builder will omit the Python module if it cannot be located, and assume that the Ansible module will handle it at runtime:

```
try:
    from ansible.module_utils.common.respawn import respawn_module
except ImportError:
    respawn_module = None
...
if needs_respawn:
    if respawn_module:
        respawn_module(target)
    else:
        module.fail_json('respawn is not available in Ansible < 2.11, ensure that foopkg is installed')
```

The optional import behavior also applies to module_utils imported from collections.

Listing collection dependencies

Listing collection dependencies

We recommend that collections work as standalone, independent units, depending only on ansible-core. However, if your collection must depend on features and functionality from another collection, list the other collection or collections under `dependencies` in your collection's :file:`galaxy.yml` file. For more information on the :file:`galaxy.yml` file, see :ref:`collections_galaxy_meta`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\(ansible-devel)(docs)(docsite)(rst)(dev_guide)developing_collections_shared.rst, line 73); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\(ansible-devel)(docs)(docsite)(rst)(dev_guide)developing_collections_shared.rst, line 73); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\(ansible-devel)(docs)(docsite)(rst)(dev_guide)developing_collections_shared.rst, line 73); *backlink*

Unknown interpreted text role "ref".

You can use git repositories for collection dependencies during local development and testing. For example:

```
dependencies: {'git@github.com:organization/repo_name.git': 'devel'}
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\dev_guide\(ansible-devel)(docs)(docsite)(rst)(dev_guide)developing_collections_shared.rst, line 85)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`collections`
       Learn how to install and use collections.
   :ref:`contributing_maintained_collections`
       Guidelines for contributing to selected collections
   `Mailing List <https://groups.google.com/group/ansible-devel>`_
       The development mailing list
   :ref:`communication_irc`
       How to join Ansible chat channels
```