Add unit tests to a collection

This section describes all of the steps needed to add unit tests to a collection and how to run them locally using the `ansible-test` command.

See :ref:`testing_units_modules` for more details.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\collection_contributors\[ansible-devel][docs][docsite][rst][community][collection_contributors]collection_unit_tests.rst, line 10); *backlink***
>
> Unknown interpreted text role "ref".

Understanding the purpose of unit tests

Unit tests ensure that a section of code (known as a `unit`) meets its design requirements and behaves as intended. Some collections do not have unit tests but it does not mean they are not needed.

A `unit` is a function or method of a class used in a module or plugin. Unit tests verify that a function with a certain input returns the expected output.

Unit tests should also verify when a function raises or handles exceptions.

Ansible uses pytest as a testing framework.

See :ref:`testing_units_modules` for complete details.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\collection_contributors\[ansible-devel][docs][docsite][rst][community][collection_contributors]collection_unit_tests.rst, line 27); *backlink***
>
> Unknown interpreted text role "ref".

Inclusion in the Ansible package requires integration and/or unit tests You should have tests for your collection as well as for individual modules and plugins to make your code more reliable To learn how to get started with integration tests, see :ref:`collection_integration_tests`.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\collection_contributors\[ansible-devel][docs][docsite][rst][community][collection_contributors]collection_unit_tests.rst, line 29); *backlink***
>
> Unknown interpreted text role "ref".

See :ref:`collection_prepare_local` to prepare your environment.

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

Determine if unit tests exist

Ansible collection unit tests are located in the `tests/units` directory.

The structure of the unit tests matches the structure of the code base, so the tests can reside in the `tests/units/plugins/modules/` and `tests/units/plugins/module_utils` directories. There can be sub-directories, if modules are organized by module groups.

If you are adding unit tests for `my_module` for example, check to see if the tests already exist in the collection source tree with the path `tests/units/plugins/modules/test_my_module.py`.

Example of unit tests

Example of unit tests

Let's assume that the following function is in `my_module` :

```
def convert_to_supported(val):
    """Convert unsupported types to appropriate."""
    if isinstance(val, decimal.Decimal):
        return float(val)

    if isinstance(val, datetime.timedelta):
        return str(val)

    if val == 42:
        raise ValueError("This number is just too cool for us ;)")

    return val
```

Unit tests for this function should, at a minimum, check the following:

- If the function gets a `Decimal` argument, it returns a corresponding `float` value.
- If the function gets a `timedelta` argument, it returns a corresponding `str` value.
- If the function gets `42` as an argument, it raises a `ValueError`.
- If the function gets an argument of any other type, it does nothing and returns the same value.

To write these unit tests in collection is called `community.mycollection`:

1. If you already have your local environment :ref:`prepared <collection_prepare_local>`, go to the collection root directory.

   System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\collection_contributors\[ansible-devel][docs][docsite][rst][community][collection_contributors]collection_unit_tests.rst, line 74); *backlink*

   Unknown interpreted text role "ref".

   ```
   cd ~/ansible_collection/community/mycollection
   ```

2. Create a test file for `my_module`. If the path does not exist, create it.

   ```
   touch tests/units/plugins/modules/test_my_module.py
   ```

3. Add the following code to the file:

   ```
   # -*- coding: utf-8 -*-

   from __future__ import (absolute_import, division, print_function)
   __metaclass__ = type

   from datetime import timedelta
   from decimal import Decimal

   import pytest

   from ansible_collections.community.mycollection.plugins.modules.my_module import (
       convert_to_supported,
   )

   # We use the @pytest.mark.parametrize decorator to parametrize the function
   # https://docs.pytest.org/en/latest/how-to/parametrize.html
   # Simply put, the first element of each tuple will be passed to
   # the test_convert_to_supported function as the test_input argument
   # and the second element of each tuple will be passed as
   # the expected argument.
   # In the function's body, we use the assert statement to check
   # if the convert_to_supported function given the test_input,
   # returns what we expect.
   @pytest.mark.parametrize('test_input, expected', [
       (timedelta(0, 43200), '12:00:00'),
       (Decimal('1.01'), 1.01),
       ('string', 'string'),
       (None, None),
       (1, 1),
   ])
   def test_convert_to_supported(test_input, expected):
       assert convert_to_supported(test_input) == expected

   def test_convert_to_supported_exception():
       with pytest.raises(ValueError, match=r"too cool"):
   ```

```
        convert_to_supported(42)
```

See :ref:`testing_units_modules` for examples on how to mock `AnsibleModule` objects, monkeypatch methods (`module.fail_json`, `module.exit_json`), emulate API responses, and more.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\collection_contributors\[ansible-devel][docs][docsite][rst][community][collection_contributors]collection_unit_tests.rst, line 127); *backlink*
>
> Unknown interpreted text role "ref".

4. Run the tests using docker:

```
ansible-test units tests/unit/plugins/modules/test_my_module.py --docker
```

Recommendations on coverage

Use the following tips to organize your code and test coverage:

- Make your functions simple. Small functions that do one thing with no or minimal side effects are easier to test.
- Test all possible behaviors of a function including exception related ones such as raising, catching and handling exceptions.
- When a function invokes the `module.fail_json` method, passed messages should also be checked.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\collection_contributors\[ansible-devel][docs][docsite][rst][community][collection_contributors]collection_unit_tests.rst, line 147)
>
> Unknown directive type "seealso".
>
> ```
> .. seealso::
>
>   :ref:`testing_units_modules`
>      Unit testing Ansible modules
>   :ref:`developing_testing`
>      Ansible Testing Guide
>   :ref:`collection_integration_tests`
>      Integration testing for collections
>   :ref:`testing_integration`
>      Integration tests guide
>   :ref:`testing_collections`
>      Testing collections
>   :ref:`testing_resource_modules`
>      Resource module integration tests
>   :ref:`collection_pr_test`
>      How to test a pull request locally
> ```