Understanding integration tests

> **Note**
>
> Some collections do not have integration tests.

Integration tests are functional tests of modules and plugins. With integration tests, we check if a module or plugin satisfies its functional requirements. Simply put, we check that features work as expected and users get the outcome described in the module or plugin documentation.

There are :ref:`two kinds of integration tests <collections_adding_integration_test>` used in collections:

> **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_integration_about.rst, line 13); *backlink***
>
> Unknown interpreted text role "ref".

- integration tests that use Ansible roles
- integration tests that use `runme.sh`.

This section focuses on integration tests that use Ansible roles.

Integration tests check modules with playbooks that invoke those modules. The tests pass standalone parameters and their combinations, check what the module or plugin reports with the :ref:`assert <ansible_collections.ansible.builtin.assert_module>` module, and the actual state of the system after each task.

> **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_integration_about.rst, line 20); *backlink***
>
> Unknown interpreted text role "ref".

Integration test example

Let's say we want to test the `postgresql_user` module invoked with the `name` parameter. We expect that the module will both create a user based on the provided value of the `name` parameter and will report that the system state has changed. We cannot rely on only what the module reports. To be sure that the user has been created, we query our database with another module to see if the user exists.

```
- name: Create PostgreSQL user and store module's output to the result variable
  postgresql_user:
    name: test_user
  register: result

- name: Check the module returns what we expect
  assert:
    that:
      - result is changed

- name: Check actual system state with another module, in other words, that the user exists
  postgresql_query:
    query: SELECT * FROM pg_authid WHERE rolename = 'test_user'
  register: query_result

- name: We expect it returns one row, check it
  assert:
    that:
      - query_result.rowcount == 1
```

Details about integration tests

The basic entity of an Ansible integration test is a `target`. The target is an :ref:`Ansible role <playbooks_reuse_roles>` stored in the `tests/integration/targets` directory of the collection repository. The target role contains everything that is needed to test a module.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\community\collection_contributors\[ansible-devel][docs][docsite]**

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_integration_about.rst, line 53); *backlink*

Unknown interpreted text role "ref".

The names of targets contain the module or plugin name that they test. Target names that start with `setup_` are usually executed as dependencies before module and plugin targets start execution. See :ref:`collection_creating_integration_tests` for 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_integration_about.rst, line 55); *backlink*

Unknown interpreted text role "ref".

To run integration tests, we use the `ansible-test` utility that is included in the `ansible-core` and `ansible` packages. See :ref:`collection_run_integration_tests` for details. After you finish your integration tests, see to :ref:`collection_quickstart` to learn how to submit a pull request.

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_integration_about.rst, line 57); *backlink*

Unknown interpreted text role "ref".

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_integration_about.rst, line 57); *backlink*

Unknown interpreted text role "ref".

Preparing for integration tests for collections

To prepare for developing integration tests:

1. :ref:`Set up your local environment <collection_prepare_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_integration_about.rst, line 66); *backlink*

   Unknown interpreted text role "ref".

2. Determine if integration tests already exist.

```
ansible-test integration --list-targets
```

If a collection already has integration tests, they are stored in `tests/integration/targets/*` subdirectories of the collection repository.

If you use `bash` and the `argcomplete` package is installed with `pip` on your system, you can also get a full target list.

```
ansible-test integration <tab><tab>
```

Alternately, you can check if the `tests/integration/targets` directory contains a corresponding directory with the same name as the module. For example, the tests for the `postgresql_user` module of the `community.postgresql` collection are stored in the `tests/integration/targets/postgresql_user` directory of the collection repository. If there is no corresponding target there, then that module does not have integration tests. In this case, consider adding integration tests for the module. See :ref:`collection_creating_integration_tests` for 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_integration_about.rst, line 83); *backlink*

Unknown interpreted text role "ref".

Recommendations on coverage

Bugfixes

Before fixing code, create a test case in an :ref:`appropriate test target<collection_integration_prepare>` that reproduces the bug provided by the issue reporter and described in the `Steps to Reproduce` issue section. :ref:`Run <collection_run_integration_tests>` the 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_integration_about.rst, line 94); *backlink*

Unknown interpreted text role "ref".

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_integration_about.rst, line 94); *backlink*

Unknown interpreted text role "ref".

If you failed to reproduce the bug, ask the reporter to provide additional information. The issue may be related to environment settings. Sometimes specific environment issues cannot be reproduced in integration tests, in that case, manual testing by issue reporter or other interested users is required.

Refactoring code

When refactoring code, always check that related options are covered in a :ref:`corresponding test target<collection_integration_prepare>`. Do not assume if the test target exists, everything is covered.

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_integration_about.rst, line 101); *backlink*

Unknown interpreted text role "ref".

Covering modules / new features

When covering a module, cover all its options separately and their meaningful combinations. Every possible use of the module should be tested against:

- Idempotency - Does rerunning a task report no changes?
- Check-mode - Does dry-running a task behave the same as a real run? Does it not make any changes?
- Return values - Does the module return values consistently under different conditions?

Each test action has to be tested at least the following times:

- Perform an action in check-mode if supported. This should indicate a change.
- Check with another module that the changes have `not` been actually made.
- Perform the action for real. This should indicate a change.
- Check with another module that the changes have been actually made.
- Perform the action again in check-mode. This should indicate `no` change.
- Perform the action again for real. This should indicate `no` change.

To check a task:

1. Register the outcome of the task as a variable, for example, `register: result`. Using the :ref:`assert <ansible_collections.ansible.builtin.assert_module>` module, check:

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_integration_about.rst, line 125); *backlink*

Unknown interpreted text role "ref".

1. If `- result is changed` or not.
2. Expected return values.

2. If the module changes the system state, check the actual system state using at least one other module. For example, if the module changes a file, we can check that the file has been changed by checking its checksum with the :ref:`stat <ansible_collections.ansible.builtin.stat_module>` module before and after the test tasks.

> **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_integration_about.rst, line 130); *backlink***
>
> Unknown interpreted text role "ref".

3. Run the same task with `check_mode: yes` if check-mode is supported by the module. Check with other modules that the actual system state has not been changed.
4. Cover cases when the module must fail. Use the `ignore_errors: yes` option and check the returned message with the `assert` module.

Example:

```
- name: Task to fail
  abstract_module:
      ...
  register: result
  ignore_errors: yes

- name: Check the task fails and its error message
  assert:
    that:
      - result is failed
      - result.msg == 'Message we expect'
```

Here is a summary:

- Cover options and their sensible combinations.
- Check returned values.
- Cover check-mode if supported.
- Check a system state using other modules.
- Check when a module must fail and error messages.