Integration tests

- Quick Start
- Configuration
  - ansible-test command
  - integration_config.yml
- Prerequisites
- Non-destructive Tests
- Destructive Tests
- Windows Tests
- Tests in containers
  - Running Integration Tests
  - Container Images
- Other configuration for Cloud Tests
- IAM policies for AWS
  - testing-policies
  - Other Definitions required
- Network Tests
- Where to find out more

Topics

The Ansible integration Test system.

Tests for playbooks, by playbooks.

Some tests may require credentials. Credentials may be specified with credentials.yml.

Some tests may require root.

> Every new module and plugin should have integration tests, even if the tests cannot be run on Ansible CI infrastructure. In this case, the tests should be marked with the `unsupported` alias in `aliases file <integration_aliases>`.

Quick Start

It is highly recommended that you install and activate the `argcomplete` Python package. It provides tab completion in `bash` for the `ansible-test` test runner.

Configuration

ansible-test command

The example below assumes `bin/` is in your `$PATH`. An easy way to achieve that is to initialize your environment with the `env-setup` command:

```
source hacking/env-setup
ansible-test --help
```

You can also call `ansible-test` with the full path:

```
bin/ansible-test --help
```

integration_config.yml

Making your own version of `integration_config.yml` can allow for setting some tunable parameters to help run the tests better in your environment. Some tests (for example, cloud tests) will only run when access credentials are provided. For more information about supported credentials, refer to the various `cloud-config-*.template` files in the `tests/integration/` directory.

Prerequisites

Some tests assume things like `hg`, `svn`, and `git` are installed, and in path. Some tests (such as those for Amazon Web Services) need separate definitions, which will be covered later in this

document.

(Complete list pending)

Non-destructive Tests

These tests will modify files in subdirectories, but will not do things that install or remove packages or things outside of those test subdirectories. They will also not reconfigure or bounce system services.

> Running integration tests within containers
>
> To protect your system from any potential changes caused by integration tests, and to ensure a sensible set of dependencies are available we recommend that you always run integration tests with the `--docker` option, for example `--docker ubuntu`. Get the list of supported container images by running `ansible-test integration --help`. You can find them in the *target docker images* section of the output. The `default` image is used for sanity and unit tests, as well as for platform independent integration tests such as those for cloud modules.

Run as follows for all POSIX platform tests executed by our CI system in a Fedora 34 container:

```
ansible-test integration shippable/ --docker fedora
```

You can exclude a specific test as well, such as for individual modules:

```
ansible-test integration --exclude git
```

You can target a specific test as well, such as for individual modules:

```
ansible-test integration ping
```

You can use the `-v` option to make the output more verbose:

```
ansible-test integration lineinfile -vvv
```

Use the following command to list all the available targets:

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

> Bash users
>
> If you use `bash` with `argcomplete`, obtain a full list by doing: `ansible-test integration <tab><tab>`

Destructive Tests

These tests are allowed to install and remove some trivial packages. You will likely want to devote these to a virtual environment, such as Docker. They won't reformat your filesystem:

```
ansible-test integration destructive/ --docker fedora
```

Windows Tests

These tests exercise the `winrm` connection plugin and Windows modules. You'll need to define an inventory with a remote Windows Server to use for testing, and enable PowerShell Remoting to continue.

Running these tests may result in changes to your Windows host, so don't run them against a production/critical Windows environment.

Enable PowerShell Remoting (run on the Windows host by a Remote Desktop):

```
Enable-PSRemoting -Force
```

Define Windows inventory:

```
cp inventory.winrm.template inventory.winrm
${EDITOR:-vi} inventory.winrm
```

Run the Windows tests executed by our CI system:

```
ansible-test windows-integration -v shippable/
```

Tests in containers

If you have a Linux system with Docker or Podman installed, running integration tests using the same containers used by the Ansible continuous integration (CI) system is recommended.

> Podman
>
> By default, Podman will only be used if the Docker CLI is not installed. If you have Docker installed but want to use Podman, you can change this behavior by setting the environment variable `ANSIBLE_TEST_PREFER_PODMAN`.
>
> Docker on non-Linux
>
> Using Docker Engine to run Docker on a non-Linux host (such as macOS) is not recommended. Some tests may fail, depending on the image used for testing. Using the `--docker-privileged` option when running `integration` (not `network-integration` or `windows-integration`) may resolve the issue.

Running Integration Tests

To run all CI integration test targets for POSIX platforms in a Ubuntu container:

```
ansible-test integration shippable/ --docker ubuntu
```

You can also run specific tests or select a different Linux distribution. For example, to run tests for the `ping` module on a Ubuntu container:

```
ansible-test integration ping --docker ubuntu
```

Container Images

Container images are updated regularly. To see the current list of container images:

```
ansible-test integration --help
```

The list is under the **target docker images and supported python version** heading.

Other configuration for Cloud Tests

To run some tests, you must provide access credentials in a file named `cloud-config-aws.yml` or `cloud-config-cs.ini` in the tests/integration directory. Corresponding `.template` files are available for syntax help. The newer AWS tests now use the file `tests/integration/cloud-config-aws.yml`.

IAM policies for AWS

Ansible needs fairly wide ranging powers to run the tests in an AWS account. These rights can be provided to a dedicated user. These need to be configured before running the test.

testing-policies

The GitHub repository mattclay/aws-terminator contains two sets of policies used for all existing AWS module integration tests. The hacking/aws_config/setup_iam.yml playbook can be used to setup two groups:

> - ansible-integration-ci will have the policies applied necessary to run any integration tests not marked as unsupported and are designed to mirror those used by Ansible's CI.
> - ansible-integration-unsupported will have the additional policies applied necessary to run the integration tests marked as unsupported including tests for managing IAM roles, users and groups.

Once the groups have been created, you'll need to create a user and make the user a member of these groups. The policies are designed to minimize the rights of that user. Please note that while this policy does limit the user to one region, this does not fully restrict the user (primarily due to the limitations of the Amazon ARN notation). The user will still have wide privileges for viewing account definitions, and will also be able to manage some resources that are not related to testing (for example, AWS lambdas with different names). Tests should not be run in a primary production account in any case.

Other Definitions required

Apart from installing the policy and giving it to the user identity running the tests, a lambda role ansible_integration_tests has to be created which has lambda basic execution privileges.

Network Tests

For guidance on writing network test see `testing_resource_modules`.

Where to find out more

If you'd like to know more about the plans for improving testing Ansible, join the Ansible community forum

---

□□□ #2
□ □□□ □□□ 2026-10-31 14:36:28 UTC
□ □□□ □□□ 2026-10-31 14:36:29 UTC