Ansible Module Development Walkthrough

In this section, we will walk through developing, testing, and debugging an Ansible module.

What's covered in this section:

Environment setup

1. Clone the Ansible repository: `$ git clone https://github.com/ansible/ansible.git`
2. Change directory into the repository root dir: `$ cd ansible`
3. Create a virtual environment: `$ python3 -m venv venv` (or for Python 2 `$ virtualenv venv`. Note, this requires you to install the virtualenv package: `$ pip install virtualenv`)
4. Activate the virtual environment: `$ . venv/bin/activate`
5. Install development requirements: `$ pip install -r requirements.txt`
6. Run the environment setup script for each new dev shell process: `$ . hacking/env-setup`

> **Note**
> After the initial setup above, every time you are ready to start developing Ansible you should be able to just run the following from the root of the Ansible repo: `$ . venv/bin/activate && . hacking/env-setup`

New module development

If you are creating a new module that doesn't exist, you would start working on a whole new file. Here is an example:

- Navigate to the directory that you want to develop your new module in. E.g. `$ cd lib/ansible/modules/cloud/azure/`
- Create your new module file: `$ touch my_new_test_module.py`
- Paste this example code into the new module file: (explanation in comments)

Search this site

```
#!/usr/bin/python

ANSIBLE_METADATA = {
    'metadata_version': '1.1',
    'status': ['preview'],
    'supported_by': 'community'
}

DOCUMENTATION = '''
---
module: my_sample_module

short_description: This is my sample module

version_added: "2.4"

description:
    - "This is my longer description explaining my sample module"

options:
    name:
        description:
            - This is the message to send to the sample module
        required: true
    new:
        description:
            - Control to demo if the result of this module is changed or not
        required: false

extends_documentation_fragment:
    - azure

author:
    - Your Name (@yourhandle)
'''

EXAMPLES = '''
# Pass in a message
- name: Test with a message
  my_new_test_module:
    name: hello world

# pass in a message and have changed true
- name: Test with a message and changed output
  my_new_test_module:
    name: hello world
    new: true

# fail the module
- name: Test failure of the module
  my_new_test_module:
    name: fail me
'''

RETURN = '''
original_message:
    description: The original name param that was passed in
    type: str
message:
    description: The output message that the sample module generates
'''

from ansible.module_utils.basic import AnsibleModule

def run_module():
    # define the available arguments/parameters that a user can pass to
    # the module
    module_args = dict(
        name=dict(type='str', required=True),
        new=dict(type='bool', required=False, default=False)
    )

    # seed the result dict in the object
    # we primarily care about changed and state
    # change is if this module effectively modified the target
    # state will include any data that you want your module to pass back
    # for consumption, for example, in a subsequent task
    result = dict(
        changed=False,
        original_message='',
        message=''
    )

    # the AnsibleModule object will be our abstraction working with Ansible
    # this includes instantiation, a couple of common attr would be the
```

Search this site

```
    # args/params passed to the execution, as well as if the module
    # supports check mode
    module = AnsibleModule(
        argument_spec=module_args,
        supports_check_mode=True
    )

    # if the user is working with this module in only check mode we do not
    # want to make any changes to the environment, just return the current
    # state with no modifications
    if module.check_mode:
        return result

    # manipulate or modify the state as needed (this is going to be the
    # part where your module will do what it needs to do)
    result['original_message'] = module.params['name']
    result['message'] = 'goodbye'

    # use whatever logic you need to determine whether or not this module
    # made any modifications to your target
    if module.params['new']:
        result['changed'] = True

    # during the execution of the module, if there is an exception or a
    # conditional state that effectively causes a failure, run
    # AnsibleModule.fail_json() to pass in the message and the result
    if module.params['name'] == 'fail me':
        module.fail_json(msg='You requested this to fail', **result)

    # in the event of a successful module execution, you will want to
    # simple AnsibleModule.exit_json(), passing the key/value results
    module.exit_json(**result)

def main():
    run_module()

if __name__ == '__main__':
    main()
```

Local/direct module testing

You may want to test the module on the local machine without targeting a remote host. This is a great way to quickly and easily debug a module that can run locally.

- Create an arguments file in `/tmp/args.json` with the following content: (explanation below)

```
{
    "ANSIBLE_MODULE_ARGS": {
        "name": "hello",
        "new": true
    }
}
```

- If you are using a virtual environment (highly recommended for development) activate it: `$ . venv/bin/activate`
- Setup the environment for development: `$ . hacking/env-setup`
- Run your test module locally and directly: `$ python ./my_new_test_module.py /tmp/args.json`

This should be working output that resembles something like the following:

```
{"changed": true, "state": {"original_message": "hello", "new_message": "goodbye"}, "invocation": {"module_args": {"name": "hello", "new": true}}}
```

The arguments file is just a basic json config file that you can use to pass the module your parameters to run the module it

Playbook module testing

If you want to test your new module, you can now consume it with an Ansible playbook.

- Create a playbook in any directory: `$ touch testmod.yml`
- Add the following to the new playbook file:

Search this site

```
- name: test my new module
  connection: local
  hosts: localhost
  tasks:
  - name: run the new module
    my_new_test_module:
      name: 'hello'
      new: true
    register: testout
  - name: dump test output
    debug:
      msg: '{{ testout }}'
```

- Run the playbook and analyze the output: `$ ansible-playbook ./testmod.yml`

Debugging (local)

If you want to break into a module and step through with the debugger, locally running the module you can do:

- Set a breakpoint in the module: `import pdb; pdb.set_trace()`
- Run the module on the local machine: `$ python -m pdb ./my_new_test_module.py ./args.json`

Debugging (remote)

In the event you want to debug a module that is running on a remote target (i.e. not localhost), one way to do this is the following:

- On your controller machine (running Ansible) set *ANSIBLE_KEEP_REMOTE_FILES=1* (this tells Ansible to retain the modules it sends to the remote machine instead of removing them)
- Run your playbook targetting the remote machine and specify `-vvvv` (the verbose output will show you many things, including the remote location that Ansible uses for the modules)
- Take note of the remote path Ansible used on the remote host
- SSH into the remote target after the completion of the playbook
- Navigate to the directory (most likely it is going to be your ansible remote user defined or implied from the playbook: `~/.ansible/tmp/ansible-tmp-...`)
- Here you should see the module that you executed from your Ansible controller, but this is the zipped file that Ansible sent to the remote host. You can run this by specifying `python my_test_module.py` (not necessary)
- To debug, though, we will want to extract this zip out to the original module format: `python my_test_module.py explode` (Ansible will expand the module into `./debug-dir`)
- Navigate to `./debug-dir` (notice that unzipping has caused the generation of `ansible_module_my_test_module.py`)
- Modify or set a breakpoint in the unzipped module
- Ensure that the unzipped module is executable: `$ chmod 755 ansible_module_my_test_module.py`
- Run the unzipped module directly passing the args file: `$ ./ansible_module_my_test_module.py args` (args is the file that contains the params that were originally passed. Good for repro and debugging)

Unit testing

Unit tests for modules will be appropriately located in `./test/units/modules`. You must first setup your testing environment. In this example, we're using Python 3.5.

- Install the requirements (outside of your virtual environment): `$ pip3 install -r ./test/runner/requirements/units.txt`
- To run all tests do the following: `$ ansible-test units --python 3.5` (you must run `. hacking/env-setup` prior to this)

> **Note**
> Ansible uses pytest for unit testing.

To run pytest against a single test module, you can do the following (provide the path to the test module appropriately):

```
$ pytest -r a --cov=. --cov-report=html --fulltrace --color yes test/units/modules/.../test/my_new_test_module.py
```

Going Further

Search this site

If you are starting new development or fixing a bug, create a new branch:

`$ git checkout -b my-new-branch`.

If you are planning on contributing back to the main Ansible repository, fork the Ansible repository into your own GitHub account and develop against the new non-devel branch in your fork. When you believe you have a good working code change, submit a pull request to the Ansible repository.

If you want to submit a new module to the upstream Ansible repo, be sure to run through sanity checks first. For example:

`$ ansible-test sanity -v --docker --python 2.7 MODULE_NAME`

Note that this example requires docker to be installed and running. If you'd rather not use a container for this, you can choose to use `--tox` instead of `--docker`.

Communication and development support

Join the IRC channel `#ansible-devel` on freenode for discussions surrounding Ansible development.

For questions and discussions pertaining to using the Ansible product, use the `#ansible` channel.

Credit

Thank you to Thomas Stringer (@tstring) for contributing source material for this topic.

Previous (developing_modules.html)

Next (developing_modules_general_windows.html)

Search this site