How to create a simple Ansible module

Felix Fontein

June 27, 2019

Roadmap

Action plugins and modules

- Action plugins behave like module
- They are executed on controller node
- They can transfer files to/from remote node
- They can call module

Action plugins and modules

- Action plugins behave like module
- They are executed on controller node
- They can transfer files to/from remote node
- They can call module

- Modules are executed on remote node
- Modules are copied and templated arguments are injected
- Modules cannot access controller's state

Modules

- Modules are executable files
- They receive input in JSON format
- They have to output JSON

Modules

- Modules are executable files
- They receive input in JSON format
- They have to output JSON

- Ansible has a lot of Python support code to make writing modules easy

Modules

- Modules are executable files
- They receive input in JSON format
- They have to output JSON

- Ansible has a lot of Python support code to make writing modules easy

- That's what we want to use today!

General principles

- UNIX philosophy: do only one thing, but do it well!

General principles

- UNIX philosophy: do only one thing, but do it well!
- Modules should be idempotent if somehow possible

General principles

- UNIX philosophy: do only one thing, but do it well!
- Modules should be idempotent if somehow possible
- Different types of modules:
  1. Modules which do something;
  2. `_info` / `_facts` modules

General principles

- UNIX philosophy: do only one thing, but do it well!
- Modules should be idempotent if somehow possible
- Different types of modules:
  1. Modules which do something;
  2. `_info` / `_facts` modules
- You can find *a lot* of material on `https://docs.ansible.com/ansible/latest/dev_guide/index.html`

A very simple module

```
#!/usr/bin/python
from ansible.module_utils.basic import (
  AnsibleModule
)

def main():
  module = AnsibleModule(dict())
  module.exit_json(msg="Hello!")

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

Running the module

1. Create test playbook

```
$ ${EDITOR} example01.yaml
```

2. Create subdirectory «library»

```
$ mkdir library
```

3. Put module into there

```
$ ${EDITOR} library/example01.py
```

4. Execute playbook!

```
$ ansible-playbook example01.yaml
```

A simple module with arguments

```
def main():
    argument_spec = dict(
        name=dict(type='str',
                  required=True),
        some_number=dict(type='int'),
        colors=dict(type='list',
                    elements='str'),
        state=dict(type='str',
                   default='present',
                   choices=['present', 'absent']),
        password=dict(type='str',
                      no_log=True),
    )
    module = AnsibleModule(argument_spec)
    module.exit_json(**module.params)
```

A simple module which can fail

```
1  import os
2
3  def main():
4      argument_spec = dict(
5          file=dict(type='path',
6                    required=True),
7      )
8      module = AnsibleModule(argument_spec)
9
10     fn = module.params['file']
11     if not os.path.exists(fn):
12         module.fail_json(
13             msg='"%s" does not exist!' % fn
14         )
15     module.exit_json(msg='"%s" exists' % fn)
```

A simple module which runs a command

```
1  def main():
2      module = AnsibleModule(dict())
3
4      rc, stdout, stderr = module.run_command(
5          ['ls', '/'],
6          check_rc=True
7      )
8      # Ansible will fail if rc is != 0
9
10     files = [
11         f for f in stdout.split('\n') if f
12     ]
13     module.exit_json(files=files)
```

A simple module which POSTs to a URL

```
from ansible.module_utils.urls import \
                                      fetch_url
def main():
  module = AnsibleModule(dict())
  r, i = fetch_url(
    module, 'https://httpbin.org/',
    method='POST', data='x=42'
  )
  status = i["status"]
  try:
    body = r.read()
  except:
    body = i.get('body')
  body = module.from_json(body)
  module.exit_json(status=status, body=body)
```

What if something goes wrong?

1. If module crashes, `ansible-playbook -vvv` will provide stacktrace

What if something goes wrong?

1. If module crashes, `ansible-playbook -vvv` will provide stacktrace
2. `print()` will not work

What if something goes wrong?

1. If module crashes, `ansible-playbook -vvv` will provide stacktrace
2. `print()` will not work
3. Poor man's debugging:

```
module.exit_json(msg='...')
raise Exception('xxx')
```

What if something goes wrong?

1. If module crashes, `ansible-playbook -vvv` will provide stacktrace
2. `print()` will not work
3. Poor man's debugging:

```
module.exit_json(msg='...')
raise Exception('xxx')
```

4. Use `q` (get via `pip install q`):

```
import q
q.q(value)
```

Meanwhile, do `tail -f /tmp/q`

Documenting modules

- Documentation is included as Python strings
- Keys `DOCUMENTATION` and `RETURN` are YAML
- Key `EXAMPLES` is essentially a list of Ansible tasks

Documenting modules

- Documentation is included as Python strings
- Keys `DOCUMENTATION` and `RETURN` are YAML
- Key `EXAMPLES` is essentially a list of Ansible tasks

- `DOCUMENTATION` contains module name, description, requirements, author, and parameter documentation
- `RETURN` documents returned variables
- `EXAMPLES` gives examples how to use the module

Header

```
#!/usr/bin/python
#
# Copyright 2019 Felix Fontein
# GNU General Public License v3.0+ (see COPYIN
# or https://www.gnu.org/licenses/gpl-3.0.txt)

from __future__ import absolute_import, ...
__metaclass__ = type

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

DOCUMENTATION 1/2

```
1  DOCUMENTATION = r'''
2  ---
3  module: example06
4  short_description: Makes a POST
5  description:
6    - Makes a POST to a URL.
7    - This is really only a test.
8  version_added: "2.9"
9  author:
10   - "Felix Fontein (@felixfontein)"
11 requirements:
12   - mycustommodule >= 1.0
13   - An internet connection
```

DOCUMENTATION 2/2

```
DOCUMENTATION = r'''
---
[...]
options:
  url:
    description:
      - The URL to POST to
    required: yes
    type: str
'''
```

EXAMPLES

```
EXAMPLES = r'''
- name: Do something
    example06:
      url: https://example.com
    register: result
- debug:
      msg: |
        {{ result.status }} --
        {{ result.body }}

# This must be valid YAML. Correct Ansible
# syntax is not checked, better do that
# yourself!
'''
```

RETURN

```
RETURN = r'''
status:
  description:
    - The status code of the POST request
  returned: always
  type: int
  sample: 200
body:
  description:
    - The parsed JSON result
  returned: always
  type: dict
  sample: '{"key": "value"}'
'''
```

Viewing the documentation

1. Using `ansible-doc`:

```
$ export ANSIBLE_LIBRARY=library
$ ansible-doc example06
```

Viewing the documentation

1. Using `ansible-doc`:

```
$ export ANSIBLE_LIBRARY=library
$ ansible-doc example06
```

2. Build HTML documentation with Sphinx
   - Clone `git@github.com:ansible/ansible.git`
   - Put your module into `lib/ansible/modules`
   - In `docs/docsite` run `MODULES=example06 make webdocs` and wait
   - Open `_build/html/modules/example06_module.html`

Testing

- Sanity tests
  - static code analysis (flake8, ...)
  - validate argument spec, compare to documentation
  - try to build documentation
- Unit tests
- Integration tests
  - Essentially Ansible roles

Sanity Testing with ansible-test

- Assume you have your module embedded in the Ansible GIT tree
- Module: `lib/ansible/modules/example06.py`

```
test/runner/ansible-test sanity \
    lib/ansible/modules/example06.py \
    --docker-no-pull --docker
```

Integration Testing with ansible-test

- Module: `lib/ansible/modules/example06.py`
- Tests: `test/integration/targets/example06/`
  - `aliases`: CI options, other modules which have the same tests
  - `meta/main.yml`: dependencies (like `prepare_http_tests`)
  - `tasks/main.yml`: the tests themselves

```
test/runner/ansible-test integration -v \
    --docker-no-pull --docker ubuntu1804 \
    example06
```

Getting Your Module into Ansible (1/2)

- Really read (parts of) the dev guide! `https://docs.ansible.com/ansible/latest/dev_guide/index.html`
- In particular: «Contributing your module to Ansible» checklist (search for «I want to contribute my module or plugin.»)
- Definitely add integration tests, even if marked as `unsupported`

Getting Your Module into Ansible (2/2)

- Make sure sanity and integration tests work before pushing
- Push to GitHub, create PR
- Make sure to fill in the provided form
- Make sure that CI (Shippable) runs through
- Tell people about your PR!

Getting Your Module into Ansible (2/2)

- Make sure sanity and integration tests work before pushing
- Push to GitHub, create PR
- Make sure to fill in the provided form
- Make sure that CI (Shippable) runs through
- Tell people about your PR!
- Now you need reviews...

Getting Your New Module PR Reviewed

- Ask people you know who could use the module to test it
- All kind of user feedback is welcome
  - how it was to use
  - strange/unexpected behavior, missing functionality
  - bugs

Getting Your New Module PR Reviewed

- Ask people you know who could use the module to test it
- All kind of user feedback is welcome
  - how it was to use
  - strange/unexpected behavior, missing functionality
  - bugs
- Try to get people to review the code
- Also try to get people to review all the Ansible details
- Ask nicely in `#ansible-devel` (FreeNode)

Thank You for your attention!

Questions? Comments?