

Custom Ansible Module Hello World

09 FEBRUARY 2016 on ansible, tutorial, devops, ansible module

What is an ansible module?

Ansible modules are the building blocks for building ansible playbooks. They are small pieces of `python` code that can be triggered from the `yaml` in a playbook.

Ansible provides a bunch of modules which cover most of your needs .. but sometimes not all of them!

When and why create a custom module?

Most of the time there is no need to create a custom module. However, sometimes it's the best (or even only way) to interact with a certain piece of software.

Often: I find the modules are a nice way to interact more fluently with services that provide a RESTful API. For example Github or Pivotal. You can, of course, interact with these services with the URI module, but it can sometimes be a little clumsy!

Creating a simple custom module

Luckily creating Ansible Modules is ridiculously easy!

We'll create a quick little module that can create or delete a repository on github.

Let's start with the very least that we can do in order to see some output.

Crete the following file structure:

```
play.yml
[library]
  |_ github_repo.py
  |_ test_github_repo.py
```

library/github_repo.py

```
#!/usr/bin/python

from ansible.module_utils.basic import *

def main():
```

Python

```
module = AnsibleModule(argument_spec={})
response = {"hello": "world"}
module.exit_json(changed=False, meta=response)

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

Notes

- `main()` is the entrypoint into your module.
- `#!/usr/bin/python` is required. Leave it out and you've got some hard debugging time on your hands!
- `AnsibleModule` comes from `from ansible.module_utils.basic import *`. It has to be imported with the `*`
- `AnsibleModule` helps us handle incoming parameters and exiting the program (`module.exit_json()`). We'll add some parameters shortly

play.yml

```
- hosts: localhost
  tasks:
    - name: Test that my module works
      github_repo:
        register: result

    - debug: var=result
```

YAML

A simple playbook which runs our module and will dump the output to `debug`

We can run our playbook:

```
$ ansible-playbook play.yml
```

(note: I haven't specified an inventory. As of Ansible 2 (perhaps before), this assumes localhost). You may need to supply one with `-i`.

Output:

```
PLAY *****

TASK [setup] *****
ok: [localhost]

TASK [Test that my module works] *****
ok: [localhost]

TASK [debug] *****
ok: [localhost] => {
  "result": {
    "changed": false,
```

Bash

```

    "meta": {
        "hello": "world"
    }
}

PLAY RECAP *****
localhost           : ok=3    changed=0    unreachable=0    failed=0

```

Cool. it worked! From here-on it's just Python! I don't know about you, but that was a lot easier than I expected it to be!

Processing input

Of course, for our module to be any use at all, we'll need some inputs. Since we're creating a Github repo, we'll just borrow [the example off the docs](#). Let's allow for the following fields

```

- name: Create a github Repo
  github_repo:
    github_auth_key: "..."
    name: "Hello-World",
    description: "This is your first repository",
    private: yes
    has_issues: no
    has_wiki: no
    has_downloads: no
    state: present
    register: result

```

YAML

In our code, we can specify the inputs like so:

```

def main():

    fields = {
        "github_auth_key": {"required": True, "type": "str"},
        "name": {"required": True, "type": "str" },
        "description": {"required": False, "type": "str"},
        "private": {"default": False, "type": "bool" },
        "has_issues": {"default": True, "type": "bool" },
        "has_wiki": {"default": True, "type": "bool" },
        "has_downloads": {"default": True, "type": "bool" },
        "state": {
            "default": "present",
            "choices": ['present', 'absent'],
            "type": 'str'
        },
    },

    module = AnsibleModule(argument_spec=fields)
    module.exit_json(changed=False, meta=module.params)

```

Python

notes:

- The expected inputs are defined as a dictionary.

- Available types (that I am aware of) are: str, bool, dict, list, ...
- You can specify if it's `required`, a `default` value, and limit possible inputs with `choices`.

In the above code, we simply accept the inputs and pipe the parsed inputs (`module.params`) to `exit_json`.

Now, if you run the playbook (`ansible-playbook play.yml`), debug should spit out a nice dictionary of the inputs we passed in.

Cool. So now we know how to handle the inputs, let's start actually executing some code.

To the top of the file add:

```
def github_repo_present(data):
    has_changed = False
    meta = {"present": "not yet implemented"}
    return (has_changed, meta)

def github_repo_absent(data=None):
    has_changed = False
    meta = {"absent": "not yet implemented"}
```

Python

These are the functions that will actually do the work. Back in our `main()` method, let's add some code to call the desired function:

```
def main():
    fields = {...}
    choice_map = {
        "present": github_repo_present,
        "absent": github_repo_absent,
    }
    module = AnsibleModule(argument_spec=fields)
    has_changed, result = choice_map.get(module.params['state'])(module.params)
    module.exit_json(changed=has_changed, meta=result)
```

Python

We use a map to map the `state` provided to a function that will handle that state.

The last thing to do is to write the code to for `github_repo_present` and `github_repo_absent`:

```
def github_repo_present(data):

    api_key = data['github_auth_key']

    del data['state']
    del data['github_auth_key']

    headers = {
```

Python

```

        "Authorization": "token {}".format(api_key)
    }
    url = "{}{}".format(api_url, '/user/repos')
    result = requests.post(url, json.dumps(data), headers=headers)

    if result.status_code == 201:
        return False, True, result.json()
    if result.status_code == 422:
        return False, False, result.json()

    # default: something went wrong
    meta = {"status": result.status_code, 'response': result.json()}
    return True, False, meta

def github_repo_absent(data=None):
    headers = {
        "Authorization": "token {}".format(data['github_auth_key'])
    }
    url = "{}repos/{}/{}".format(api_url, "toast38coza", data['name'])
    result = requests.delete(url, headers=headers)

    if result.status_code == 204:
        return False, True, {"status": "SUCCESS"}
    if result.status_code == 404:
        result = {"status": result.status_code, "data": result.json()}
        return False, False, result
    else:
        result = {"status": result.status_code, "data": result.json()}
        return True, False, result

```

Notes:

- Notice how we do our best to work out if the state has changed or not. (in the delete method it's a little tricky, cause github will also return a 404 if you are not properly authenticated).
- It's useful to return the response in the `meta` .. makes it easier to debug.
- With Ansible modules, it's important that

We can add a block to delete the API just after we created it. Our final playbook might look like this:

```

- hosts: localhost
  vars:
    - github_token: "..."
  tasks:
    - name: Create a github Repo
      github_repo:
        github_auth_key: "{{github_token}}"
        name: "Hello-World"
        description: "This is your first repository"
        private: yes
        has_issues: no
        has_wiki: no
        has_downloads: no

    - name: Delete that repo

```

YAML

```
github_repo:
  github_auth_key: "{{github_token}}"
  name: "Hello-World"
  state: absent
```

You can run this to create and then delete a repo with the name `Hello-World`.

Last step, we need to add our documentation to the module

Ansible likes us to include strings for `DOCUMENTATION` and `EXAMPLES` at the top of our module. It should explain how the module can be used. We can pretty much just use our playbook from above:

```
DOCUMENTATION = '''
---
module: github_repo
short_description: Manage your repos on Github
'''

EXAMPLES = '''
- name: Create a github Repo
  github_repo:
    github_auth_key: "..."
    name: "Hello-World"
    description: "This is your first repository"
    private: yes
    has_issues: no
    has_wiki: no
    has_downloads: no
  register: result

- name: Delete that repo
  github_repo:
    github_auth_key: "..."
    name: "Hello-World"
    state: absent
  register: result
'''
```

Python

Conclusion

And that's it. Congrats! You've successfully created a module that can create and remove a Github Repo with Ansible.

Next steps:

There's a lot of room for improvement in this module. You could:

- Update the `present` state so that if the repo already exists, it will update the existing repo.
- This plugin currently only handles creating repos for users. Extend it to also handle creating repos for users.
- You *could* handle *all* the possible inputs for `CREATE` ..

Here is the final code for our module:

```
1  #!/usr/bin/python
2
3  DOCUMENTATION = '''
4  ---
5  module: github_repo
6  short_description: Manage your repos on Github
7  '''
8
9  EXAMPLES = '''
10 - name: Create a github Repo
11     github_repo:
12         github_auth_key: "...
13         name: "Hello-World"
14         description: "This is your first repository"
15         private: yes
16         has_issues: no
17         has_wiki: no
18         has_downloads: no
19     register: result
20
21 - name: Delete that repo
22     github_repo:
23         github_auth_key: "...
24         name: "Hello-World"
25         state: absent
26     register: result
27 '''
28
29 from ansible.module_utils.basic import *
30 import requests
31
32 api_url = "https://api.github.com"
33
34
35 def github_repo_present(data):
36
37     api_key = data['github_auth_key']
38
39     del data['state']
40     del data['github_auth_key']
41
42     headers = {
43         "Authorization": "token {}".format(api_key)
44     }
45     url = "{}{}".format(api_url, '/user/repos')
46     result = requests.post(url, json.dumps(data), headers=headers)
47
48     if result.status_code == 201:
49         return False, True, result.json()
50     if result.status_code == 422:
51         return False, False, result.json()
52
53     # default: something went wrong
54     meta = {"status": result.status_code, 'response': result.json()}
55     return True, False, meta
56
57
58 def github_repo_absent(data=None):
59     headers = {
60         "Authorization": "token {}".format(data['github_auth_key'])
61     }
62     url = "{}repos/{}/{}".format(api_url, "toast38coza", data['name'])
63     result = requests.delete(url, headers=headers)
```

2/5/2018

Build a custom Ansible module in 10 minutes

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

```
        if result.status_code == 204:
            return False, True, {"status": "SUCCESS"}

        if result.status_code == 404:
            result = {"status": result.status_code, "data": result.json()}
            return False, False, result

        else:
            result = {"status": result.status_code, "data": result.json()}
            return True, False, result

def main():


    fields = {
        "github_auth_key": {"required": True, "type": "str"},
        "name": {"required": True, "type": "str"},
        "description": {"required": False, "type": "str"},
        "private": {"default": False, "type": "bool"},
        "has_issues": {"default": True, "type": "bool"},
        "has_wiki": {"default": True, "type": "bool"},
        "has_downloads": {"default": True, "type": "bool"},
        "state": {
            "default": "present",
            "choices": ['present', 'absent'],
            "type": 'str'
        },
    },
}

choice_map = {
    "present": github_repo_present,
    "absent": github_repo_absent,
}

module = AnsibleModule(argument_spec=fields)
is_error, has_changed, result = choice_map.get(
    module.params['state'])(module.params)

if not is_error:
    module.exit_json(changed=has_changed, meta=result)
else:
    module.fail_json(msg="Error deleting repo", meta=result)

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

github_repo.py hosted with  by GitHub

view raw

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

Name

- Data Janitor

a year ago

where does this github_repo.py file is saved to? This has to be saved in one of the library directories of Ansible to facilitate the call within playbook correct?

1

• Reply

• Share
- toast38coza

Mod

Data Janitor

a year ago

Either the library directory of ansible, or simply in a directly called library in the same location as your main playbook. for example:

...

. playbook.yml

|_ library

. github_repo.py

...

• Reply

• Share
- Jose Diaz-Gonzalez

Data Janitor

a year ago

Yep, that is correct!

• Reply

• Share
- Mohit Pant

2 months ago

hey can i use os.system command inside the ansible module. I am trying to invoke azure cli commands through my ansible module by os.system. It works for provisioning on azure but throws warning [WARNING]: Module invocation had junk after the JSON data: and then this error The error was: TypeError: list indices must be integers, not str fatal: [localhost]: FAILED! => {"failed": true, "msg": "Unexpected failure during module execution.", "stdout": ""}

• Reply

• Share
- Chandrasekhar Ravulapalli

3 months ago

One of the best tutorials for creating Ansible modules!!! Great work!!!

• Reply

• Share
- Mumshad Mannambeth

6 months ago

Please checkout <https://www.ansible-playabl...> to easily develop custom modules and documentation around it.

• Reply

• Share
- nnyan

10 months ago

Would you be available to create a custom ansible module?

• Reply

• Share
- yellowbird

a year ago

This is awesome! Thank you for the great tutorial! I was wondering though how can I automate the creation of repos to an organization?

• Reply

• Share
- toast38coza

Mod

yellowbird

a year ago

Hi @yellowbird to create a repo for an organization, the endpoint looks like this: `POST /orgs/:org/repos`. Check the docs for more: <https://developer.github.co...>

• Reply

• Share

yellowbird

toast38coza

a year ago

Thanks for replying @toast38coza

• Reply

• Share

toast38coza

Mod

yellowbird


a year ago

so does that mean that we swap '/user/repos' in line 45 with '/orgs/:org/repos'

• Reply

• Share
- <http://blog.toast38coza.me/custom-ansible-module-hello-world/>

9/11



Yup. that should do the trick :)


^

|

▼

• Reply

• Share ›



Brenda Bell

• a year ago

How do you implement a custom module for a python virtual environment? Is there a way to have the module use the interpreter in use by the task that triggered it?


^

|

▼

• Reply

• Share ›



ZillaYT .

• a year ago

This is now obsolete with Python 2.x? If so how is this done in Python 2.x? I can't discern from this page: <http://docs.ansible.com/ans...>


^

|

▼

• Reply

• Share ›



toast38coza

Mod

➔

ZillaYT .

• a year ago

Hi @ZillaYT . this should work fine with python 2.x.


^

|

▼

• Reply

• Share ›



technocrattobe

• 2 years ago

fix the file name. spend 20 minutes figuring out error

The offending line appears to be:

tasks:

- name: Test that my module works

^ here


^

|

▼

• Reply

• Share ›



toast38coza

Mod

➔

technocrattobe

• a year ago

Hi @technocrattobe I've fixed the naming mistake. Thanks for pointing it out!


^

|

▼

• Reply

• Share ›



Matthew Morgan

• 2 years ago

Thanks so much for writing this. I used this base to make a fully working module. I have a really tough time following ansible's doc so it was so great to find this! Keep 'em coming plz!


^

|

▼

• Reply

• Share ›



toast38coza

Mod

➔

Matthew Morgan

• a year ago

@Matthew Morgan It's a pleasure. Glad it was useful!


^

|

▼

• Reply

• Share ›



toast38coza

Mod

• 2 years ago

Hi Ashutosh. That is correct. I'll fix it up shortly. Thanks for letting me know!


^

|

▼

• Reply

• Share ›



Ashutosh Kaul

• 2 years ago

github_repository.py should have been github_repo.py or did I miss something ?

^

|

▼

• Reply

• Share ›

ALSO ON TOAST38COZA. LEARNING AS I GO

Part 1: Up and running with Kong and Docker Compose

5 comments • 2 years ago

toast38coza

— Cool. Thanks for your input. I've updated the compose file for V 0.8.* I've included a gist for either Postgres or ...

Part 2: Defining our Kong API Gateway with Ansible

3 comments • 2 years ago

belajar program

— Nice tutorial, one question. how to handle api-keys dinamically for example using one of programming language?

Make your vuex store globally available by registering it as a plugin

6 comments • a year ago

Thibaut Roskam

— Hey,How does your solution compare to "this.\$store" from vuex itself ? see <https://vuex.vuejs.org/en/s...>Thanks!

Validating a Spring MVC @Controller with a custom validation class - the easy way

1 comment • 2 years ago

Roman Hrytsyshyn

— Instead of binder.setValidator(new PersonFormValidator());you have to use ...

Read [more posts](#) by this author.

