DEVELOPING ANSIBLE MODULES FOR FOREMAN AND KATELLO
$ WHOAMI
Evgeni Golov
Senior Software Engineer at Red Hat
ex-Consultant at Red Hat
Debian Developer
♥ FOSS ♥
♥ automation ♥
FOREMAN + ANSIBLE = ♥

- Foreman has an API
- Everyone loves writing YAML instead of clicking in a GUI
- So we wrote modules to allow that
- They have bugs, missing features or we miss whole modules
- This is how everyone can help
FOREMAN ANSIBLE
MODULES
FOREMAN ANSIBLE MODULES

• A collection of Ansible modules to interact with the Foreman API
• Also supports Foreman plugins like Katello, Remote Execution, SCC
• Provide an abstraction layer, so you don't have to repeat yourself
- name: Create ACME Organization
  foreman_organization:
    username: admin
    password: changeme
    server_url: https://foreman.example.com
  name: ACME
  state: present
UNDER THE HOOD

- Connect to the API
- Search for an entity (usually by name)
- Create/Update/Delete depending on current state and user input
- Report to the user
WRITING FOREMAN
ANSIBLE MODULES
class ForemanOrganizationModule(ForemanEntityAnsibleModule):
    pass

module = ForemanOrganizationModule(
    entity_spec=dict(
        name=dict(required=True),
        description=dict(),
        label=dict(),
    ),
)

with module.api_connection():
    module.run()
REFERENCE AN ORGANIZATION (LIST) FROM ANOTHER MODULE

module = ForemanLocationAnsibleModule(
    entity_spec=dict(
        ...,
        organizations=dict(type='entity_list'),
        ),
    )

with module.api_connection():
    module.run()
class ForemanDomainModule(ForemanTaxonomicEntityAnsibleModule):
    pass

module = ForemanDomainModule(
    entity_spec=dict(  
        name=dict(required=True),  
        ...  
    ),
)

with module.api_connection():
    module.run()
RENAMING ENTITIES

```python
module = ForemanDomainModule(
    argument_spec=dict(
        updated_name=dict(),
    ),
    entity_spec=dict(
        name=dict(required=True),
        ...,
    ),
)

with module.api_connection():
    module.run()
```
entity_dict = module.clean_params()
with module.api_connection():
    entity_dict, scope =
        module.handle_organization_param(entity_dict)
    entity = module.find_resource_by_name(
        'content_credentials', name=entity_dict['name'],
        params=scope, failsafe=True)

module.ensure_entity('content_credentials',
    entity_dict, entity, params=scope)
entity_dict = module.clean_params()
with module.api_connection():
    params = {'id': entity_dict['name']}
    power_state = module.resource_action('hosts', 'power_status', params=params)
    if module.state == 'state':
        module.exit_json(power_state=power_state['state'])
    elif (module.state == power_state['state']):
        module.exit_json()
    else:
        params['power_action'] = module.state
        module.resource_action('hosts', 'power', params=params)
AVAILABLE HELPERS

- `list_resource`
- `show_resource`
- `find_resource/
  find_resource_by_{name,title,id}`
- `find_resources/
  find_resources_by_{name,title,id}`
- `ensure_entity`
- `resource_action`
TESTING FOREMAN
ANSIBLE MODULES
OUR TEST SUITE

• Ansible playbooks for each module
  ▪ Handle setup, tests, teardown
  ▪ Ensure idempotency by checking the changed property

• VCRpy is used to record API interaction
  ▪ Tests can be run in test or record mode
  ▪ CI always runs test mode
  ▪ Developers need record mode when API requests change
TEST EXECUTION

- `make test` runs tests for *ALL* modules
- `make test_<module>` only for that one module
- `make record_<module>` when a new recording is needed
EXAMPLE:
ORGANIZATION.YML

- include: tasks/organization.yml
  vars:
    organization_state: present
    expected_change: true

- include: tasks/organization.yml
  vars:
    organization_state: present
    expected_change: false
EXAMPLE: 
TASKS/ORGANIZATION.YML

- name: "Testing organization"
  vars:
    - organization_name: "Test Organization"
    - organization_description: "A test organization"
  foreman_organization:
    name: "{{ organization_name }}"
    description: "{{ organization_description }}"
    state: "{{ organization_state }}"
  register: result
- assert:
  fail_msg: "Testing organization failed"
  that:
    - result.changed == expected_change
  when: expected_change is defined
DEVELOPMENT ENVIRONMENT
PYTHON ENVIRONMENT FOR USERS

- Modules and dependencies are available as RPM
- And from Ansible Galaxy (modules) / PyPI (dependencies)
A devel setup has more dependencies
Using a virtualenv is highly recommended!
  • `ansible_python_interpreter = "/usr/bin/env python"
• The tests also require a configuration file

```bash
python3 -m venv ./venv
source ./venv/bin/activate
make test-setup
```
(re-)running existing tests (make test) uses recorded fixtures
- this is great to ensure API requests didn't change after refactoring
- real behavior changes will yield "cannot match request" errors

behavior changes require new recording
- need to run make record_<testname>
- requires running Foreman/Katello
FOREMAN/KATELLO ENVIRONMENT

- on Linux, the easiest way is forklift
- any instance that can be destroyed is fine
- set URL and admin credentials in tests/test_playbooks/vars/server.yml
DEBUGGING MODULES

If you're used to print-based debugging, Ansible will hide all interesting information from you and you'll need a different approach.
RAISE EXCEPTION AND MODULE.WARN

- raise Exception("the message")
- module.warn("the message")
- not nice, but gets the job done
\textbf{Q}

\textit{q} is the Quick-and-dirty debugging output for tired programmers.

- \texttt{q("the message")}
- output goes to /tmp/q
A REAL DEBUGGER

- pdb is the default Python debugger, but doesn't play nice with Ansible
  - mostly because Ansible forks another Python process
- a debugger with remote debugging feature is useful: epdb, remote-pdb
DEMO
let's fix #586 together
THANKS!

✉ evgeni@golov.de
🌐 die-welt.net
🐦 @zhenech
✉️ @zhenech@chaos.social
🗣️ @evgeni