Managing Ansible codebase

Mohammed Naser

$ whoami

- Mohammed Naser
- Follow me on Twitter: @_mnaser
- Using Ansible in production to automate all the things for a few years
- Ansible OpenStack modules maintainer

- CEO @ VEXXHOST, Inc.
- OpenStack Technical Committee Chair
- OpenStack Ansible Project Team Lead

We’re hiring.

If you like Ansible and like working in open source upstream projects, let’s chat: **https://vexxhost.com/company/careers/**

Managing Ansible codebase

Using roles

- Role selection criteria:
  - Does it already exist and solve your problem? Use it.
  - Does it exist but partly solves your problem? Fix it upstream.
  - Does it not exist at all and covers open source software? Write and open source it.
- Always pin your roles, never point to tags or branches.
- Make sure to always prefix your variables.
- Vendor your roles .. if you can.
- If someone decides to change their role to be a `shell: rm -rfv /` ...

Use roles_path

- It can be difficult to start identifying site-local roles vs vendored roles

```
roles_path = ./vendor-roles:./roles
```

- By default, Ansible galaxy installs to the first path, so `ansible-galaxy install -r requirements.yaml` will install your roles into `./vendor-roles`

Inventory

- Don’t use static inventory
- If you’re in a cloud, there’s plenty of inventory plugins
- If you’re on bare metal, you can probably link to your DCIM (netbox, etc)

- If you’re managing the list of servers you have in a text file…
- Decouple that.

`shell` is the devil

- Don’t use shell.
- Check if an Ansible module exists first (upstream)
  - If it exists, but not in your release, you can pop it inside `library` in your playbook.
- Check if an Ansible role exists with that module
  - You can simply install the role and include it to make it available
  - You can install that role with ansible-galaxy and use ansible.cfg to point to the `library` folder. **Or Ansible collections?!**
- Don’t like Python? Did you know you can write modules in any language?
  - Ansible just needs to execute it and get JSON output.

`ansible-lint` is your friend

- ansible-lint has improved a lot lately
- It has a lot of good and best practices, follow them.
- Newer releases since the Ansible announcement has resulted in huge improvements in making the roles that pass ansible-lint much more dependable/reliable
- Things like retries when hitting network, etc

`mitogen` is fast, really, really, fast.

- Mitogen is a super interesting tool that helps speed up Ansible
- It runs a small process on the remote host and ‘avoids’ having to load python for *every* single module
- It includes ‘stackable’ connection drivers which allow you to easily use jumpboxes or run Ansible against Docker/LXC/nspawn containers
- Works 99.99% of the time, unless you have some really odd tasks.
- **SUPER beneficial if you have high latency**

`delegate_to` all the time

- Avoid cluttering your target host.
- OpenStack Ansible example:
  - We need to run tasks that depend on openstacksdk being deployed
  - We don’t want to install openstacksdk everywhere.
  - We install Ansible inside venv at /opt/ansible-runtime
  - We deploy all needed Python packages there
  - Delegate those API-calls to local deploy node instead of the remote one

Idempotent roles

- A role should be able to run a million times without changing a thing.
- This goes back to not using shell.
- If you’re doing CI, run the role twice, make sure no changes happen

Molecule all the roles

- TDD is a thing, it’s a really good thing.
- Use TDD with your Ansible roles, ensure a state and make sure it gets there
- Molecule has built-in test framework that can test on a variety of systems
- It also includes idempotency checks.

Playbook per role

- Write one single playbook for every role you have
- Build a `site.yaml` which includes all of your playbooks for all roles
- Run site.yaml when you want to ensure convergence, run individual playbooks when you want to deploy a specific component

`include_role` for DRY-ness

- Don’t repeat code, ever.
- Sometimes, you’ll notice you do something often
- OpenStack Ansible example:
  - We install from source, we have to create a systemd unit
  - We used to have a systemd file for every single project that we had to keep in sync
  - Every time we needed to change, requires 40-50 patches
  - We created systemd_service role, we include_role that and now we can make changes without going back over and over again.

Beware of `check_mode`

- Not all modules support check_mode
- It can result in a destructive behaviour if you run with the assumption that check mode won’t change anything.
- Something to keep in mind.

Gather facts on-demand

- Gathering facts is probably one of the longer things in a playbook run.
- It also isn’t always necessary.
- You can manually run the `setup` module inside your role, filtering specific things to pull up.

`include_vars` at the start, always

- If you support multiple platforms, this is the cleanest way to manage it.

```
- name: Gather variables for each operating system
  include_vars: "{{ item }}"
  with_first_found:
    - "{{ ansible_distribution | lower }}-{{ ansible_distribution_version | lower }}.yml"
    - "{{ ansible_distribution | lower }}-{{ ansible_distribution_major_version | lower }}.yml"
    - "{{ ansible_os_family | lower }}-{{ ansible_distribution_major_version | lower }}.yml"
    - "{{ ansible_distribution | lower }}.yml"
    - "{{ ansible_os_family | lower }}-{{ ansible_distribution_version.split('.')[0] }}.yml"
    - "{{ ansible_os_family | lower }}.yml"
  tags:
    - always
```

`tags` speed things up

- Use proper tagging, it helps your role consumers, helps you speed up the development process.
- Use `always` for any fact collection tasks
- Prefix your tags with your role name and document them
- Example if you had an `ara` role
  - Tags could be: `ara-config, ara-install, etc.`

`with_items` isn't always the best

- When using with_items, Ansible has to re-run that entire task for every single iteration.
- For quite a few modules, it is possible to merge work, such as commonly, for things like package managers
- yum/apt/etc takes a list or string of packages in 'name'. Using with_items will install them *one* by *one*. Using a list will run a single transaction.
- WARNING: Ansible used to 'squash' `with_items` with the `yum` module. It now recommends you just provide a list.

Q&A

We’re hiring.

If you like Ansible and like working in open source upstream projects, let’s chat: **https://vexxhost.com/company/careers/**