Discovering the oVirt Ansible Collection

Luca Andrea Fusè
Red Hat Instructor @ Extraordy

Stefano Stagnaro
Red Hat Instructor @ Extraordy
What are we going to talk about?

- **Why Ansible?**
- **What is a “Collection”?**
  - Why Ansible Collections?
  - oVirt Ansible Collection
- **What can be done with it?**
  - Creation of virtual machine
  - oVirt Inventory Plugin
  - oVirt Roles
- **TL;DR**
Why Ansible?
Why Ansible?

- Ansible allows the use of oVirt API in a sane and declarative manner
- We are able to easily read and modify an Ansible playbook (vs long and complicated bash/Python scripts).
- Actually, we may easily reuse the playbook that we are writing.
- We can create some roles to perform complex actions on our infrastructure
- We can manage our oVirt in a DevOps-ish way using Ansible Tower
What is a “Collection”?
What is a “Collection”?

“Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins.”

Ok, nice. But **why** do we need collections?
Why Ansible Collections?

- Collections let product owners manage modules/plugins externally of Ansible Core.
- This means that we may have new features added faster to our module/plugins.
- Actually, that also means that we can package the module, the plugins, and the roles all together, in a single project.
- **Downside:** Collections are supported from ansible 2.9 and from ansible 2.10 collections will be the default.
We can use the community version on Ansible Galaxy or the one on Red Hat Automation Hub. (They are both the same)

To install one of them it’s possible to use ansible-galaxy collection install ovirt.ovirt

Or, we may configure Ansible Tower to import one of them.
What can be done with it?
Create a disk/upload an Iso

- name: copy images on ovirt
  ovirt_disk:
  auth: "{{ ovirt_auth }}"
  name: "{{ name }}"
  storage_domain: "{{ storage }}"
  content_type: iso
  wait: true
  format: raw
  upload_image_path: "{{ path }}"

- name: creation of storage
  ovirt_disk:
  auth: "{{ ovirt_auth }}"
  name: "{{ name }}"
  size: "{{ disk_size }}"
  state: present
  storage_domain: "{{ storage }}"
Create a virtual machine

- name: creation of a vm
  ovirt_vm:
    auth: "{{ ovirt_auth }}"
    name: "{{ name }}"
    cluster: "{{ cluster_name }}"
    state: running
    cd_iso: "{{ iso }}"
    memory: "{{ memory }}"
    cpu_cores: "{{ number_core }}"
    boot_devices:
      - cdrom
  nics:
    - profile_name: "{{ profile }}"
      name: "{{ nic }}"
Using the ip of guest VM for Ansible inventory

- Using the *inventory plugin* of oVirt Collections, we can dynamically get the ip of the guest VM on our platform.
- We can configure it to split the different VMs in different groups.
oVirt Inventory Plugin

```python
plugin: ovirt.ovirt.ovirt
ovirt_url: url
ovirt_username: user
ovirt_password: password
keyed_groups:
  - key: cluster
    prefix: 'cluster'
groups:
  test: "'test' in tags"
  prod: "'prod' in tags"
```
The Ansible Roles for oVirt are not deprecated
We can find them inside the ovirt collection
We can use them to:
  ○ Do a cluster upgrade
  ○ Create a template
  ○ Do Infrastructure operations
  ○ ecc
- hosts: localhost
collections:
  - ovirt.ovirt
vars:
  engine_fqdn: "{{ engine_fqdn }}"
  engine_user: "{{ user }}"
  cluster_name: "{{ cluster_name }}"
  stop_non_migratable_vms: true
  host_statuses:
    - up
  host_names:
    - "{{ nodes }}"
roles:
  - cluster_upgrade
TL;DR
● Ansible Collections are cool and easy to use
● oVirt API are cool, but Ansible oVirt Collection is cooler
● We can use them to manage our platform in an easy, declarative and idempotent way.
Thank you!

Email:

lafuse@extraordy.com

sstagnaro@extraordy.com