No test, no fear!

Testando playbooks com molecule

Caio Volpato

Chapter de computação visual

28 de setembro de 2022

Texto original

Essa palestra foi originada de um texto do mesmo autor:
ansible localhost -a "/bin/echo Primeiros passos com o Ansible"

Somos um grupo de divulgação cientifica no medium, escrevemos sobre Computação, Matemática, Estatística e Ciência de Dados.

- Link: medium.com/computando-arte/
- Fundado em Nov/2020 🎂
- A informação quer ser livre: Licenciado sob CC BY-SA 4.0
- /join: Por que e como fazer um blog técnico

Recapitulando

Na talk anterior falamos um pouco sobre o ansible:

- Como funciona (arquitetura)
- Como montar o arquivo inventario
- Como armazenar segredos de forma segura (ansible-vault)
- Rodamos alguns comandos de forma adhoc
- Fizemos um playbook simples como exemplo (webserver apache)

Uma provocação

Um playbook bem escrito não precisa de testes

Agenda de hoje

- As dificuldades de testar manualmente playbooks
- Ferramenta Vagrant
- Ferramenta Molecule
- Evoluir nosso playbook de exemplo
- Setup do CI (gitlab-ci)
- Dicas e truques

Motivação

Quando preciso testar um playbook, preciso fazer:

1. Criar uma VM
2. Habilitar o servidor ssh para conectar
3. Apontar para a VM no inventario
4. Rodar o playbook
5. Logar na VM e testar manualmente se funcionou corretamente
6. Deletar a VM

Em resumo: É muito custoso fazer testes manuais

Vagrant

O vagrant é uma ferramenta que cria VMs de forma fácil e reprodutível, com o intuito de criar uma infra de testes o mais próxima de produção possível.

Vagrantfile

```
Vagrant.configure("2") do |config|
  config.vm.box = "geerlingguy/rockylinux8"
  config.vm.synced_folder ".", "/vagrant", disabled: true

  config.vm.define "debvm" do |deb| # debian vm
    deb.vm.hostname = "debian"
    deb.vm.box = "debian/bullseye64"
    deb.vm.network :private_network, ip: "192.168.60.7"
  end

  config.vm.define "rockylinux" do |rocky| # rockylinux vm
    rocky.vm.hostname = "rockylinux"
    rocky.vm.box = "geerlingguy/rockylinux8"
    rocky.vm.network :private_network, ip: "192.168.60.6"
  end
end
```

Comandos do Vagrant

- `vagrant up` criar as VMs
- `vagrant ssh debvm` para acessar a vm debvm
- `vagrant halt` desliga as VMs (sem destruir)
- `vagrant destroy`

Provisioner ansible no vagrant

O playbook do ansible não precisa ser disparado manualmente.

O vagrant tem um provisiover do ansible que dispara o ansible ao criar as VMs.

```
Vagrant.configure("2") do |config|
   config.vm.box = "geerlingguy/rockylinux8"
   config.vm.synced_folder ".", "/vagrant", disabled: true

+  config.vm.provision "ansible" do |ansible|
+    ansible.playbook = "playbook.yml"
+  end
end
```

Molecule

Molecule é uma ferramenta projetada para testar playbooks

Por padrão utiliza o docker

Além de aplicar o playbook permite executar testes automatizados

Testa idempotência =)

Ansible roles

role é uma forma de organizar os playbooks, para facilitar a componentização.

Ansible Galaxy (galaxy.ansible.com) é um repositório publico de roles

Estrutura dos roles

```
roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies
    webtier/              # same kind of structure as "common" was above, done for the webtier role
```

Exemplo do uso do galaxy

```yaml
# playbook.yaml
---
- hosts: machine # or group or all
  become: yes
  vars:
    security_ssh_password_authentication: "no"
    security_ssh_permit_root_login: "no"
    security_autoupdate_enabled: true
    security_autoupdate_reboot: false

  roles:
    - geerlingguy.security
```

Para executar: `ansible-playbook -i inventario.ini playbook.yaml`

Estrutura molecule

Para inicializar um role com molecule:

```
molecule init role company.role_name -d docker
```

Serão criados 3 yamls em `molecule/default/`

- molecule.yml: Configurações do molecule, como qual imagem docker será utilizada.
- converge.yml : Importa role a ser testado
- verify.yml : Codigo do teste.

Comandos molecule

Para "subir" o contêiner, aplicando o playbook: `molecule converge`

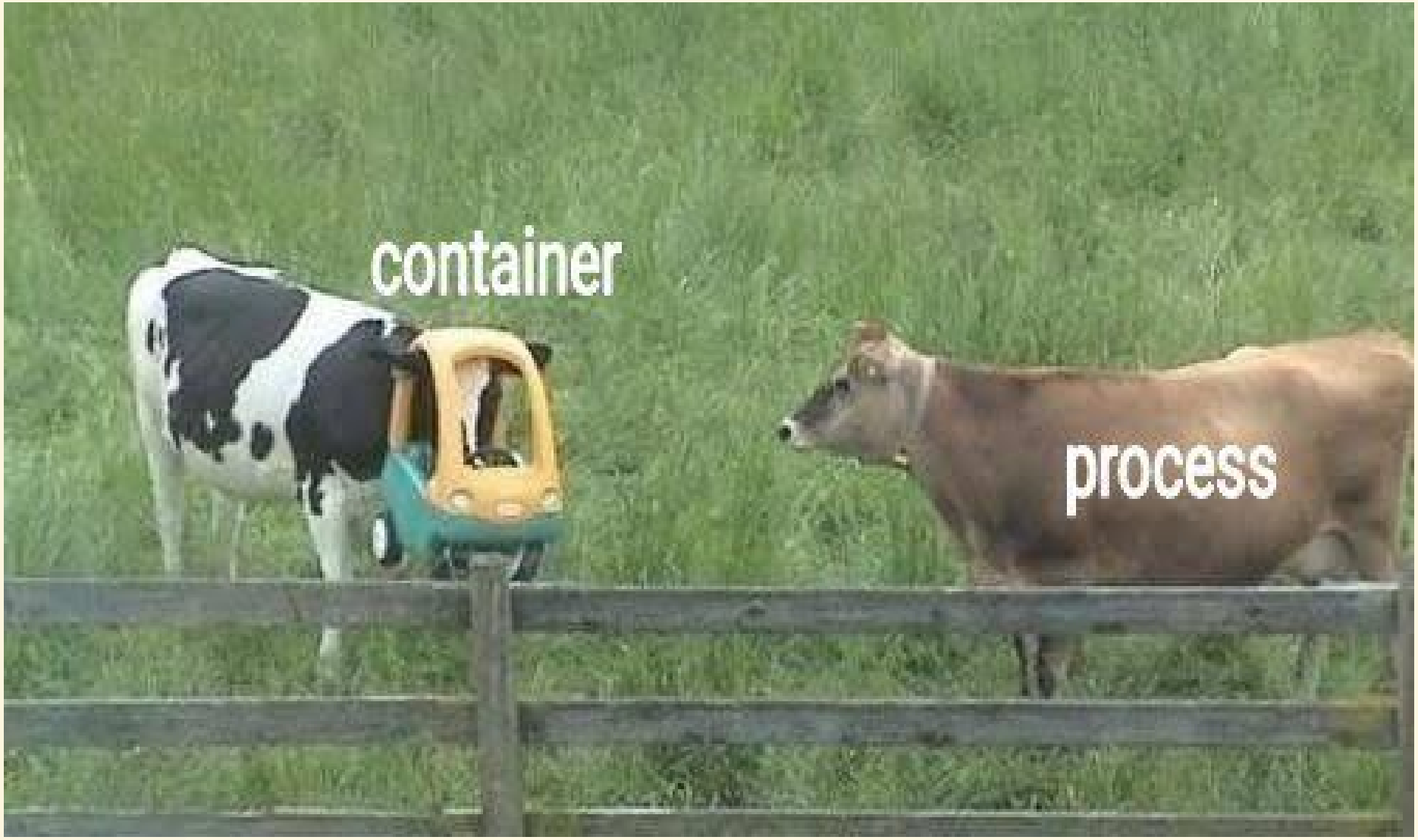
Para acessar o docker: `molecule login`

Executar somente o teste: `molecule verify`

Para destruir: `molecule destroy`

Para fazer tudo (subir, verificar idempotência, executar o teste e destruir no final): `molecule test`

cuidado com essas imagens com systemd



Exemplo: Hospedar site hugo com nginx

1. Instala o hugo.
2. Instala o nginx.
3. Clona o repositório do site.
4. Build do site.
5. Configura o nginx, apontando para o site.
6. Reinicia o nginx, para colocar em efeito a nova config.

Este exemplo está disponivel em: gitlab.com/caioau/ansible_demo

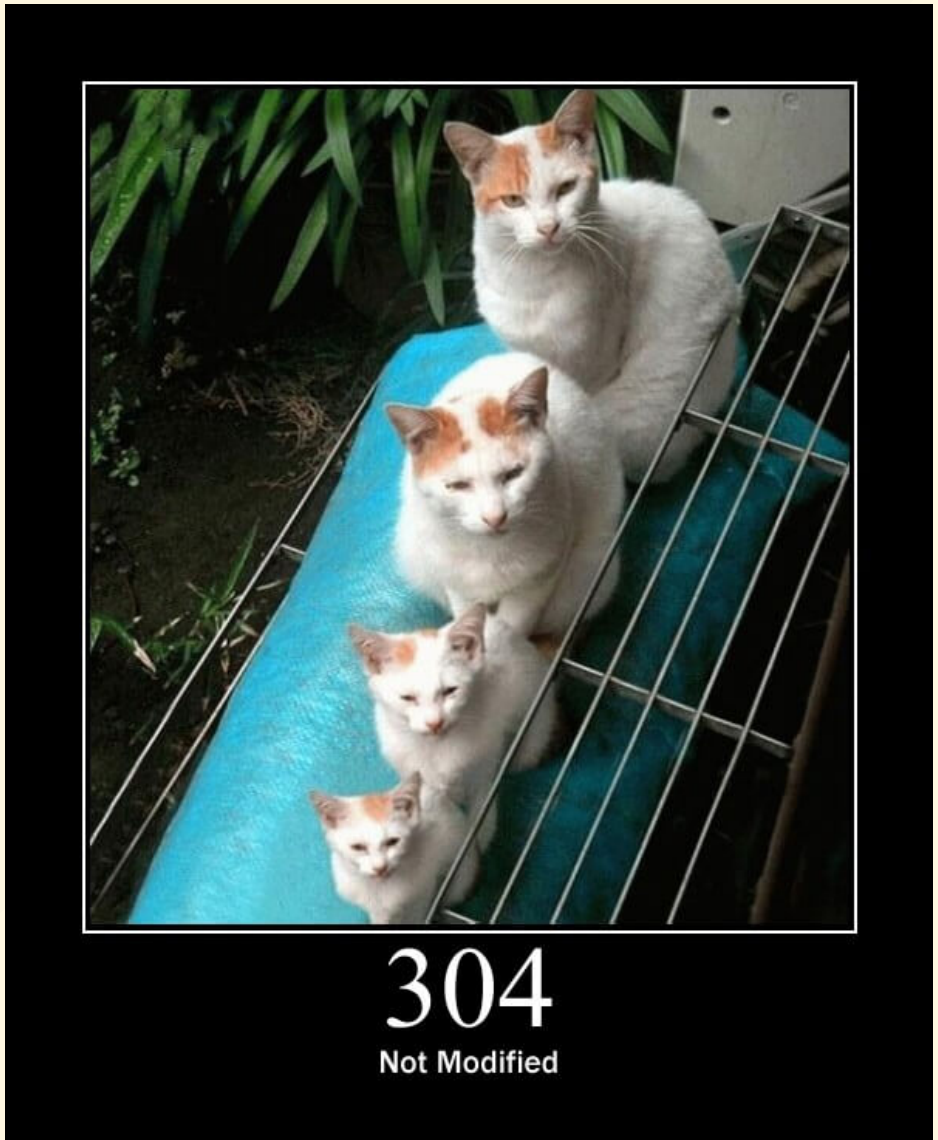
```yaml
- hosts: all
  become: yes

- import_playbook: playbook.yml
  vars:
    site_repo_version: HEAD
    hugo_version: 0.102.1
```

Os testes

Fazemos alguns requests, verificando se:

1. O request retorna 200-ok E o title é o esperado, atestando que é o site que esperamos.
2. Testa o cache: Fazendo um segundo request, passando o header `if-modified-since` com o valor `last_modified` obtido no primeiro e verifica se retorna 304.
3. Verifica se a pagina 404 é a correta, não a padrão do nginx :)

http.cat



O que acontece quando rodamos um playbook?

1. É a primeira vez, a aplicação não está instalada ainda -> instala
2. A aplicação já está instalada e é a mesma versao do playbook -> nada acontece (idempotencia)
3. Ta instalada, mas a versão no playbook é diferente -> decidir entre:
    1. atualiza
    2. ignora e mantem a versão antiga

exemplo: validate do template

Quando o arquivo de conf do nginx é renderizado, através do template, executamos o nginx com a opção de validar o mesmo, antes de colocar em efeito.

Como nem sempre vamos rodar o playbook para instalar do zero a aplicação , em alguns momentos para atualizar também corremos o risco de quebrar a aplicação.

alternativamente podemos detectar que a aplicação já esta instalada e optar por não atualizar.

Dicas e truques

- Use e abuse de tags do git, marque onde prod está, facilitando testar se a migração terá sucesso
- Use o cache da pipeline, salvando o venv pra buildar mais rapido na raspi. Também tem o piwheels.org
- problema do cgroupv2: issue se utilizar e cgroupv2 precisa subir o daemon do docker com algumas configurações.
- Além de rodar os linters no pipeline use local com pre-commit
- Execute o ansible, alterando sua infra, também no pipeline.

Voltando a nossa provocação inicial

Eai, um playbook bem escrito precisa de testes?

Um platbook bem escrito não precisa de testes **externos** ;)

Os testes mais importantes estão presentes no próprio playbook, que checa pré condições que podem levar a falhas e aborta a execução se necessário evitando maiores danos.

E os testes no `verify.yml` ficam reservados para testes funcionais e testes de integração.

Referências

- Livro Ansible for DevOps do Jeff Geerling.
  - lives: Ansible 101 playlist.
  - repo: github.com/geerlingguy/ansible-for-devops
- Livro Engenharia de Software Moderna: engsoftmoderna.info