





Remerciements





Nos prochain événements:

REX Prometheus
animé par
Alexandre Taveau et Nicolas Macquet
le 16 Mars 2020



Nous suivre

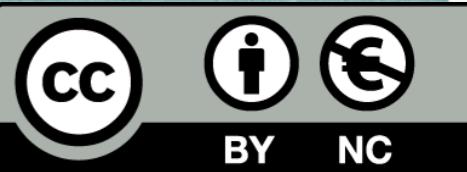
<https://cafedevops.org/>

<https://twitter.com/CafeDevOpsLyon>

<https://www.meetup.com/fr-FR/cafe-devops-lyon/>

contact@cafedevops.org

Molecule, le testeur de rôle Ansible. CI/CD experiment



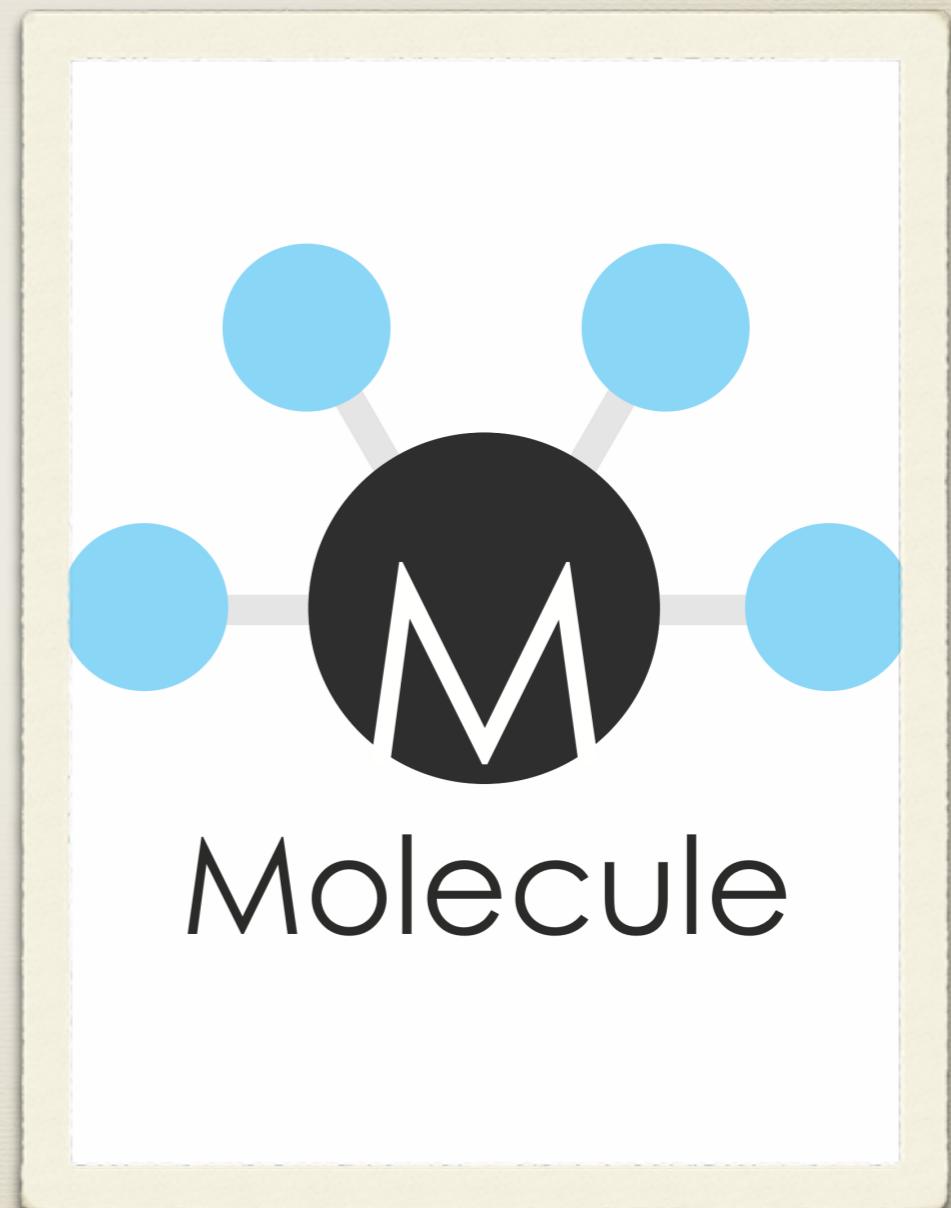
Sommaire 1/2

* **Introduction**

- * Qu'est ce que molécule ?
- * Pourquoi l'utiliser ?

* **Comment ça marche ?**

- * Pré requis
- * Intégration dans un rôle
- * Structure de molécule
- * Détail du fichier molecule.yml
- * Usage molecule
- * Les tests avec Testinfra



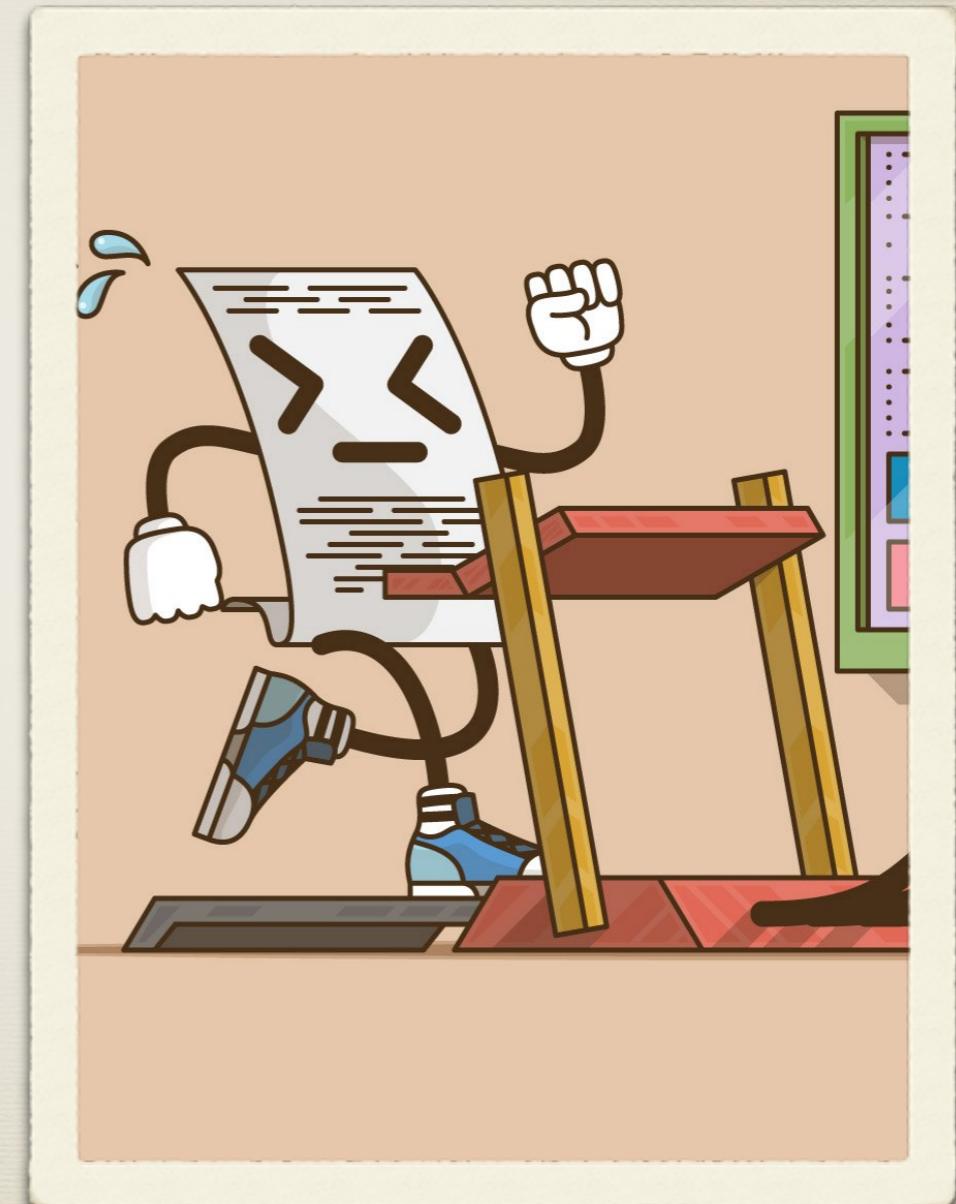
Sommaire 2/2

* **Intégration continue**

- * Architecture
- * Bitbucket
- * Jenkins
 - Pipeline
 - Multibranch-pipeline
 - Jenkinsfile

* **Sujets transverse**

- * Marmo
- * Convertisseur docker => openstack
- * Openstack -> Creation/suppression de l'environnement de test



Première Partie



Introduction

Qu'est ce que molécule ?

Définition :

Molécule est un outil développé en python dont le but est de tester des rôles ansible. Dans les grandes lignes celui-ci permet d'initialiser un environnement, déployer le rôle, exécuter des tests sur la machine cible et enfin supprimer l'environnement.

Cette suite d'action est appelé scenario et peut contenir des étapes supplémentaires comme par exemple le test de l'idempotence du rôle.

Fonctionnalités :

- * Permet de créer un environnement de test avec différents drivers : docker, vagrant, openstack ...
- * Permet de tester son rôle sur différents OS : redhat-6.6, redhat-7.3 ...
- * Permet de tester la syntaxe, l'idempotence et le checkmode d'un rôle.
- * Permet de tester l'état final de la machine à l'aide d'un framework de test : testinfra, goss ...

Pourquoi l'utiliser ?

Isolation du développement d'un rôle :

- * Test des rôles en local avec docker (centos 6.6 et 7.3)
- * Evite de solliciter un environnement du projet
- * Isole le développement et évite tout impact sur les environnements projets

Robustesse du rôle :

- * Limite les régressions sur un role (en augmentant le nombre de tests)
- * Test from scratch d'un déploiement
- * Test unitaire du role
- * Améliore la qualité du développement



Comment ça marche ?

Pré requis

Utilisation de molécule :

- * Disposer d'une machine Linux
- * Avoir accès à un repo pip
- * Installer les paquets python :
 - ansible==2.8
 - molecule==2.20

Utilisation du driver docker :

- * Installer docker-ce
- * Avoir accès à un repo docker
- * Installer le paquet python :
 - docker-py

Intégration dans un rôle

Fonctionnement :

- * Molécule vient se greffer directement rôle à tester
- * Un répertoire « molecule » sera créé à la racine du rôle
- * Il contiendra l'ensemble des scenarios de tests

Mise en place molécule

- * Se positionner à la racine d'un rôle
- * Lancer la commande d'initialisation :
 - molecule init scenario -r mon_role -s default -d docker
- * Un scenario « default » sera créé dans le répertoire molécule et sera configuré pour utiliser le driver docker.

Structure de molécule

Structure par défaut :

- * Répertoire molécule à la racine du rôle
- * Un répertoire par scenario
- * Un répertoire de test qui sera exécuté par le scenario (verify)
- * Le playbook.yml qui sera lancé par le scenario (converge)
- * Le molecule.yml qui contient la description du scenario
 - Le driver utilisé
 - Les instances de tests
 - Les tests qui seront exécutés
 - La séquence à lancer (idempotence, check, syntax ...)

```
[root@ansible mon_role]# tree
.
├── defaults
│   └── main.yml
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
└── molecule
    ├── default
    │   ├── Dockerfile.j2
    │   ├── INSTALL.rst
    │   ├── molecule.yml
    │   ├── playbook.yml
    │   └── tests
    │       └── test_default.py
    └── install
        ├── Dockerfile.j2
        ├── INSTALL.rst
        ├── molecule.yml
        ├── playbook.yml
        └── tests
            └── test_default.py
    └── README.md
    └── tasks
        └── main.yml
```

Structure de molécule

Structure alternative :

- * Répertoire molécule à la racine du rôle
- * Un répertoire par scenario
- * Un répertoire « resources » pour partager les données entre les scenarios

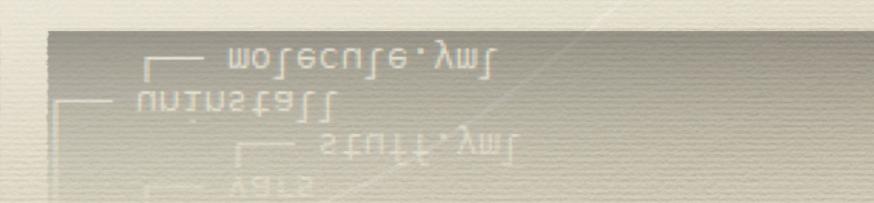
Avantages :

- * Structure plus lisible
- * Factorisation de certains fichier (Dockerfile, vars, tests ...)

Inconvénients :

- * Molécule ne permet pas de créer nativement cette arborescence

```
[root@molecule molecule]# tree
.
├── install
│   └── molecule.yml
└── resources
    ├── dockerfiles
    │   ├── Dockerfile.j2
    │   └── Dockerfile-systemd.j2
    ├── playbooks
    │   ├── playbook-install.yml
    │   └── playbook-uninstall.yml
    └── tests
        ├── install
        │   └── test_default.py
        └── uninstall
            └── test_uninstall.py
    └── vars
        └── stuff.yml
└── uninstall
    └── molecule.yml
```



Molecule.yml

Ce fichier est un descriptif du scenario. Il contient le driver utilisé, la plateforme de test, la séquence du scenario, les tests à exécuter ...

Dependency :

- * Correspond à ansible-galaxy. Permet de récupérer un rôle externe.

Driver :

- * Détermine comment sera créé l'environnement de test.

Platforms :

- * Contient les instances où seront exécutés les tests
- * Test le déploiement sur des images centos 6.6 et 7.3
- * Dockerfile-systemd utilisé pour le démarrage des services

```
---
```

```
dependency:
  name: galaxy
```

```
driver:
  name: docker
```

```
lint:
  name: yamllint
```

```
platforms:
  - name: centos-6.6
    image: "lsp-centos:6.6"
    registry:
      url: "https://index.docker.io/v1/"
    disable_cache: true
    privileged: true
    volume_mounts:
      - "/sys/fs/cgroup:/sys/fs/cgroup:rw"
    dockerfile: ../resources/dockerfiles/Dockerfile.j2
```

```
  - name: centos-7.3
    image: "lsp-centos:7.3"
    registry:
      url: "https://index.docker.io/v1/"
    disable_cache: true
    privileged: true
    volume_mounts:
      - "/sys/fs/cgroup:/sys/fs/cgroup:rw"
    command: "/usr/sbin/init"
    dockerfile: ../resources/dockerfiles/Dockerfile-systemd.j2
```

```
dockerfile: '../resources/dockerfiles/Dockerfile-systemd.j2'
command: "/usr/sbin/init"
volume_mounts:
  - "/sys/fs/cgroup:/sys/fs/cgroup:rw"
```

Molecule.yml

Provisioner:

- * options => options ansible
 - extra_vars, vault_password_file ...
- * env => variable système
 - ANSIBLE_LIBRARY, ANSIBLE_ROLE_PATH ...
- * playbooks :
 - * prepare : playbook exécuté avant le converge (optionnel)
 - * converge: playbook exécuté lors du scenario

Scénario:

- * name => doit être identique au nom du répertoire du scénario
- * test_sequence => l'ensemble des actions molécule.
 - converge => déploiement ansible
 - Idempotence => seconde exécution du déploiement
 - verify => exécution des tests

Verifier:

- * Directory => répertoire où se trouve les tests
- * Additional_files_or_dirs => tests supplémentaires à exécuter (liste de répertoires ou fichiers)

```
provisioner:  
  name: ansible  
  options:  
    vvv: true  
  lint:  
    name: ansible-lint  
  playbooks:  
    converge: ../resources/playbooks/playbook-install.yml  
  env:  
    ANSIBLE_VAULT_PASSWORD_FILE: "~/vault.key"  
  
scenario:  
  name: install  
  test_sequence:  
    - lint  
    - syntax  
    - create  
    - prepare  
    - converge  
    - check  
    - verify  
    - destroy  
  
verifier:  
  name: testinfra  
  directory: ../resources/tests/common  
  additional_files_or_dirs:  
    - ../install/  
  options:  
    verbose: true  
  lint:  
    name: flake8
```

```
name: flake8  
true:  
  Ansible: none  
  converge:
```

Usage molécule

Initialisation d'un nouveau rôle ansible + molecule :

- * molecule init role -r <role_name> -d docker

Initialisation d'un ou plusieurs scenario molecule :

- * Exécuter la commande à la racine d'un rôle
- * molecule init scenario -r <role_name> -s <scenario_x> ... -s <scenario_y>

Exécution des scenarios :

- * Exécuter la commande à la racine d'un rôle
- * molecule test => exécution de tous les scenarios :
- * molecule test -s <scenario_x> ... -s <scenario_y>
- * molecule <action> => verify, converge, syntax ...
- * *Options :*
 - molecule --debug test : mode ultra verbose
 - molecule test --destroy=never : ignore la suppression de l'environnement

Usage molécule

Affichage des instances :

- * molecule list
- * molecule list -s <scenario_x>

Connexion à une instance :

- * molecule login -h <instance_name> -s <scenario_x>
/!\ Pour pouvoir s'y connecter, l'instance doit être up (utiliser l'option --no-destroy après test de molecule pour ne pas supprimer les instances)

Les tests avec Testinfra

Définition

- * Framework de test python
- * Utilise les assert pour valider si le test est en succès ou non :

```
def test_installed_pkg(host):
    assert host.package('haproxy18').is_installed
```

- * Exécution en fin de scenario (verify) :

```
test_sequence:
  - create
  - converge
  - verify
  - destroy
```

- * Permet de vérifier que le déploiement est conforme à ce qu'on attend
- * Permet de limiter les régressions lors des modifications d'un rôle
Pour activer le mode debug :
- * Ajouter l'option « s » dans le bloc « verifier » du molecule.yml

```
verifier:
  name: testinfra
  directory: ../resources/tests/install
  options:
    verbose: true
    s: true
  lint:
    name: flake8
```

Les tests avec Testinfra

Exemple d'un fichier de test :

```
import os
import pytest
import testinfra.utils.ansible_runner

testinfra_hosts = testinfra.utils.ansible_runner.AnsibleRunner(
    os.environ['MOLECULE_INVENTORY_FILE']).get_hosts('all')


@pytest.mark.parametrize('s', [
    'grafana-server',
    'influxdb',
    'jmxtrans',
    'collector'
])

def test_hosts_service(host, s):
    service = host.service(s)
    assert service.is_running

def test_hosts_file(host):
    f = host.file('/etc/hosts')

    assert f.exists
    assert f.user == 'root'
    assert f.group == 'root'
```

Celui-ci permet de vérifier que les services grafana, influxdb ... sont bien up.

Les tests avec Testinfra

Documentation :

- * <https://testinfra.readthedocs.io/en/latest/modules.html>

Exemple :

- * host.file("/etc/passwd").exists
- * oct(host.file("/etc/password").mode) == '0600'
- * host.package("nginx").is_installed
- * host.socket("tcp://0.0.0.0:80").is_listening
- * ...

“Demo”

Récapitulatif

Fonctionnement :

- * Molécule vient se greffer directement rôle à tester
- * Un répertoire « molecule » sera créé à la racine du rôle
- * Il contiendra l'ensemble des scenarios de tests

Structure de molécule :

- * Répertoire molécule à la racine du rôle
- * Un répertoire par scenario
- * Un répertoire de test qui sera exécuté par le scenario (verify)
- * Le playbook.yml qui sera lancé par le scenario (converge)
- * Le molecule.yml qui contient la description du scenario
 - Le driver utilisé (docker, vagrant, openstack ...)
 - Les instances de tests (centos-6.6, centos-7.3 ...)
 - Les tests qui seront exécutés (tests unitaires python)
 - La séquence à lancer (idempotence, check, syntax ...)

```
[root@ansible mon_role]# tree
.
├── defaults
│   └── main.yml
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
└── molecule
    ├── default
    │   ├── Dockerfile.j2
    │   ├── INSTALL.rst
    │   ├── molecule.yml
    │   ├── playbook.yml
    │   └── tests
    │       └── test_default.py
    │           └── test_default.pyc
    └── install
        ├── Dockerfile.j2
        ├── INSTALL.rst
        ├── molecule.yml
        ├── playbook.yml
        └── tests
            └── test_default.py
            └── test_default.pyc
└── README.md
└── tasks
    └── main.yml
└── vars
    └── main.yml
```

Deuxième Partie

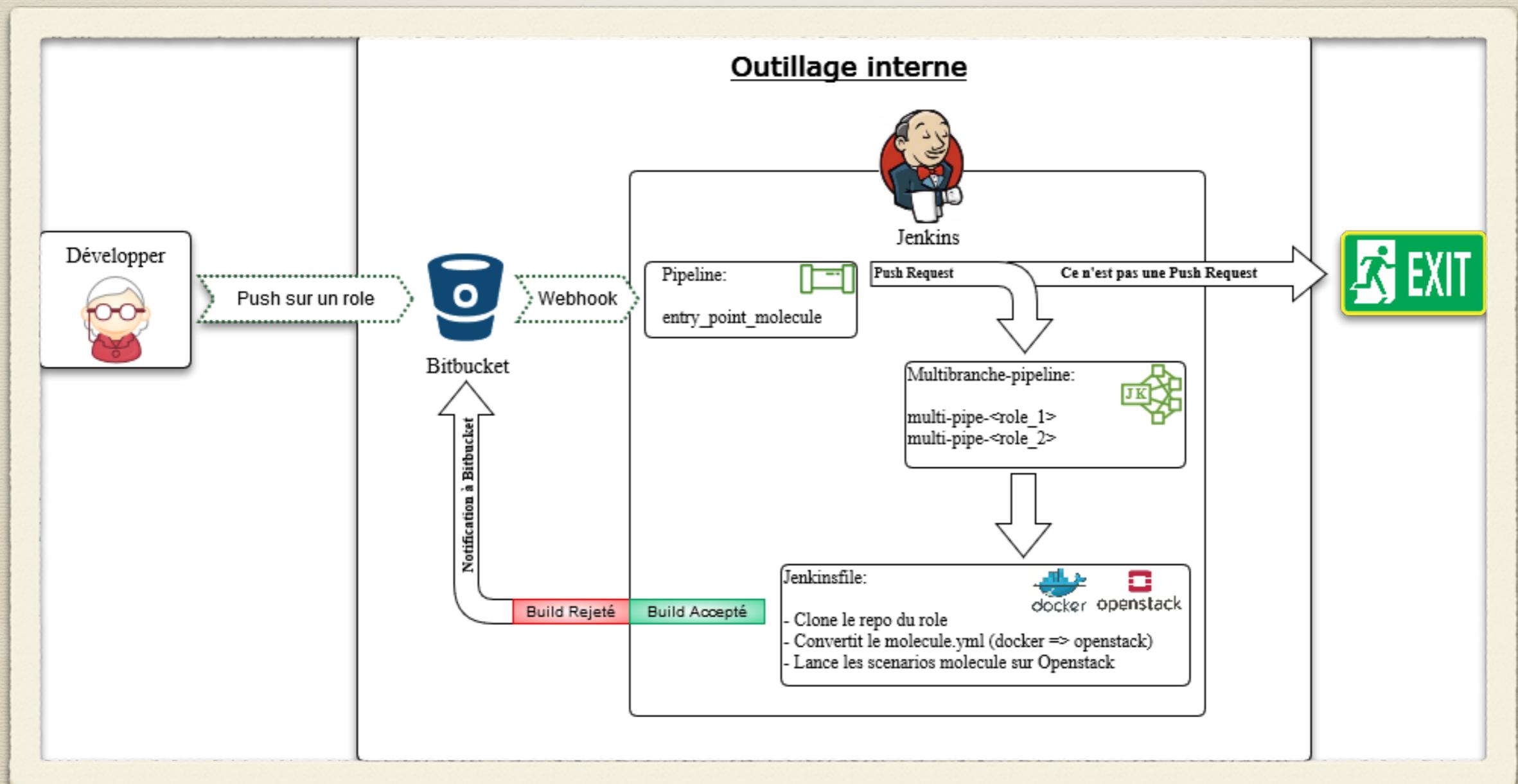
Local responsibility syndrome



Intégration continue de molecule

Intégration continue de molecule

L'objectif est de lancer l'ensemble des scenarios molécule d'un rôle sur openstack afin de vérifier le déploiement du rôle sur un environnement plus proche de la réalité contrairement à docker.



Bitbucket

Chaque rôle dispose d'une structure spécifique afin de s'intégrer à toute la chaîne de test automatique

La structure classique d'une role Ansible

- * Defaults, handlers, meta, tasks, templates, vars

Un répertoire molecule

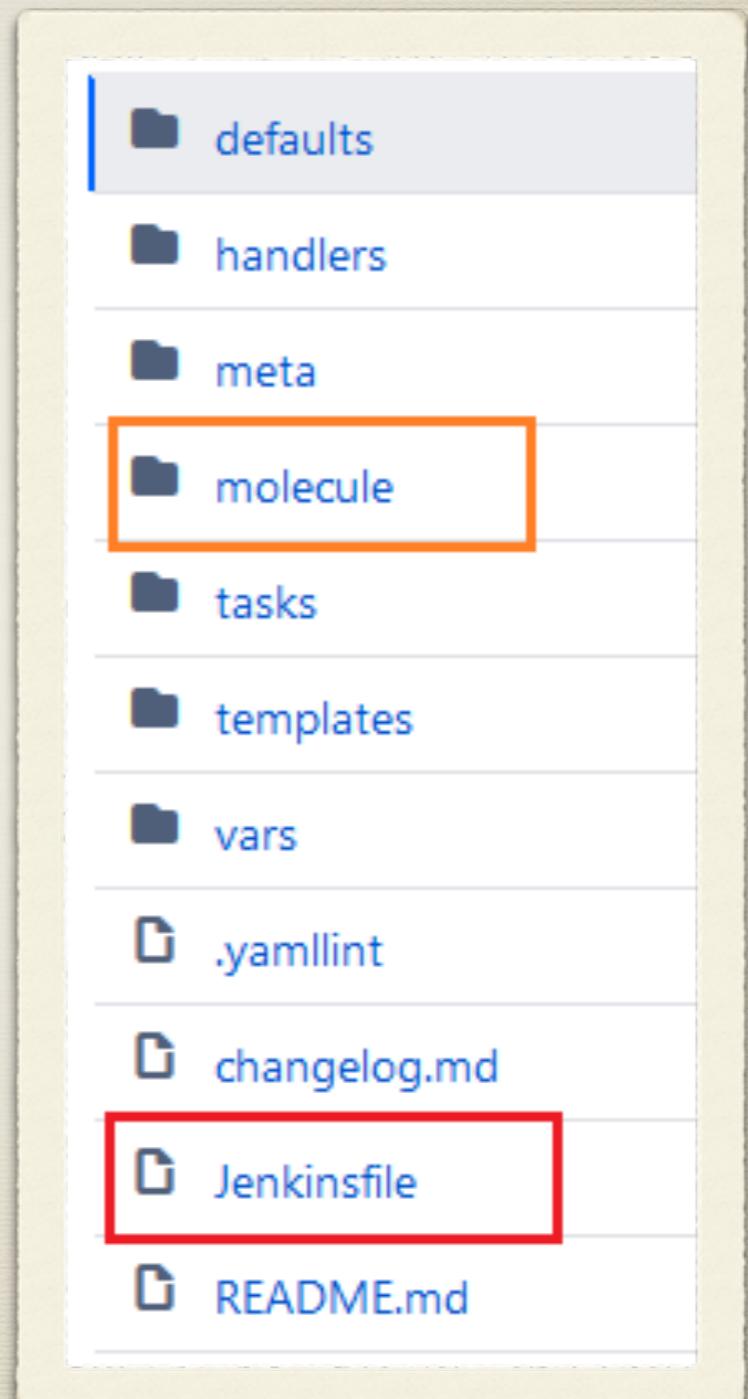
- * Contient les différents scénarios de test

Une documentation pour expliquer le role

- * README.md

Un Jenkinsfile

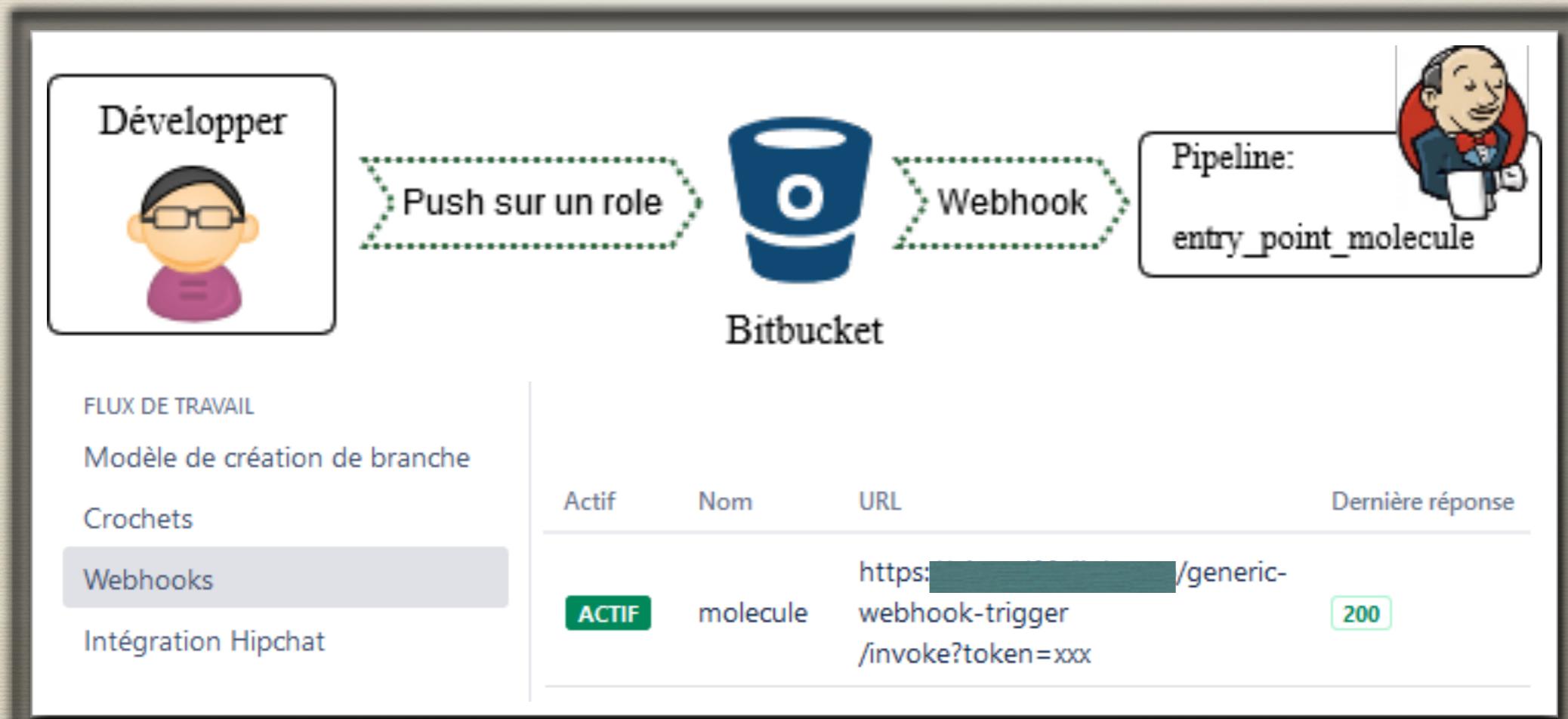
- * Clone le repo du role
- * Convertit le molecule.yml docker en openstack
- * Lance les scenarios molecule sur l'openstack
- * Notifie bitbucket du résultat du job:



Bitbucket

Pour être testé automatiquement, chaque rôle dispose d'un webhook bitbucket :

- * Déclenche le pipeline jenkins « entry_point_molecule »
- * Se déclenche lors d'un push/PR



Jenkins

- * Outil open source d'intégration continue
- * S'interface avec des systèmes de gestion de versions (Git)
- * Vocabulaire :
 - * job : suite d'actions à exécuter
 - * pipeline : type de job

Jenkins / Pipeline

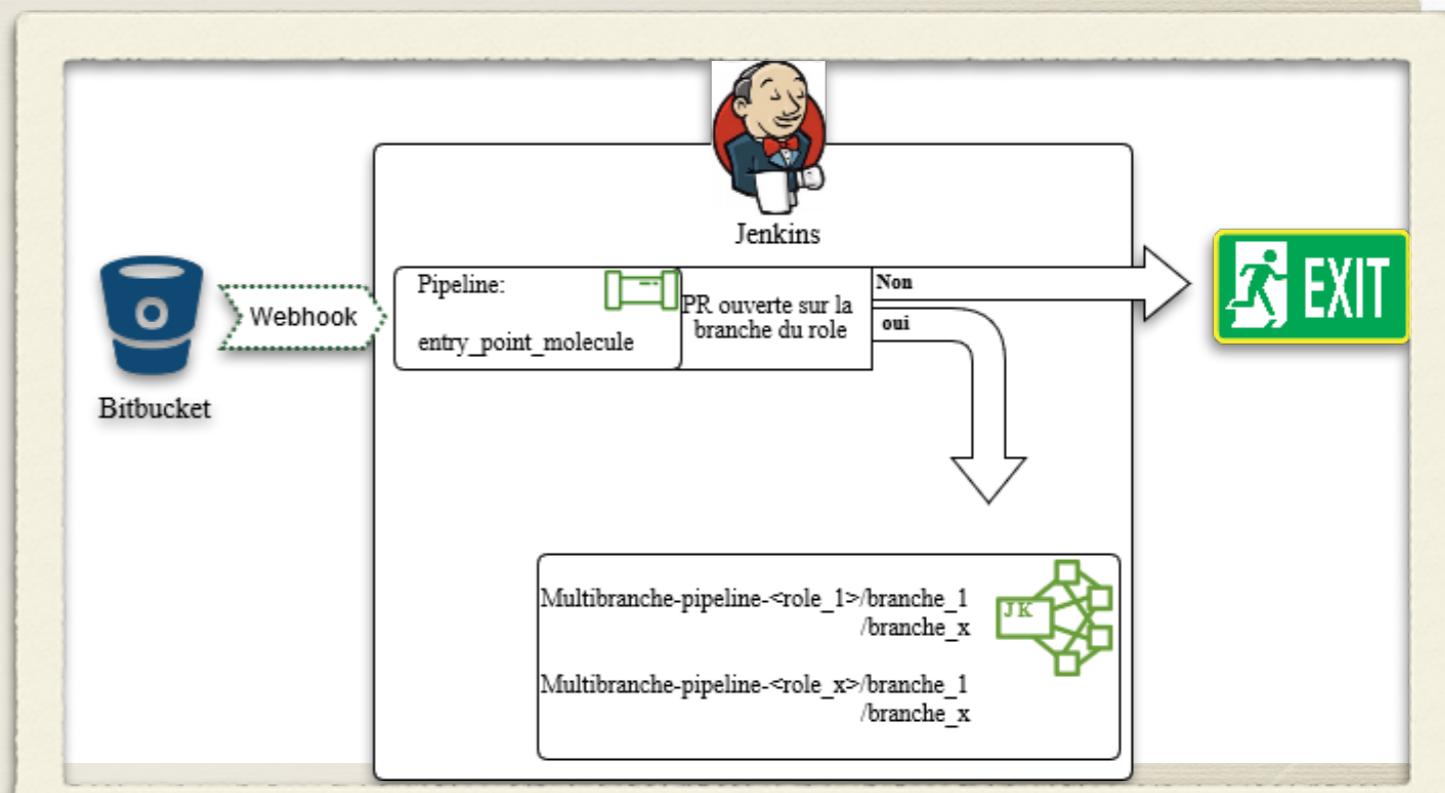
Coté Jenkins nous avons deux types de jobs :

* Un pipeline entry_point_molecule :

- Sert de point d'entrée pour le webhook
- Est déclenché par le webhook bitbucket lors d'un push
- Vérifie s'il y a une PR ouverte sur la branche du rôle
- Si oui, lance le job correspondant au multibranch-pipeline

* Le multibranch-pipeline est déclenché uniquement lors d'une PR ouverte pour alléger le jenkins et openstack.

Les tests sont exécutés uniquement lors de la validation étant donné qu'ils sont lancés en local avec docker lors du développement du rôle.



Jenkins / Multibranch-pipeline

Un multibranch-pipeline pour chaque rôle :

- * Scan toutes les branches du rôle
- * Crée un job pour chaque branche du rôle disposant d'un Jenkinsfile
- * Récupère le Jenkinsfile de la branche du rôle et l'exécute

Détail d'un multibranch-pipeline avec deux branches disposant d'un Jenkinsfile:

Liste des multibranch-pipeline :

- [pipeline-linkysup-flume](#)
- [pipeline-linkysup-grafana](#)
- [pipeline-linkysup-haproxy_keepalived](#)
- [pipeline-linkysup-module_delete_old_files](#)

← Up

i Status

⚙ Configure

▶ Scan Multibranch Pipeline Now

▶ Scan Multibranch Pipeline Log

📈 Multibranch Pipeline Events

✖ Delete Multibranch Pipeline

pipeline-linkysup-flume

Branches (2)						
S	M	Name ↓	Dernier succès	Dernier échec	Dernière durée	Built On Fav
		master	5 j 9 h - #6	6 j 1 h - #5	34 mn	▶ ★
		sw-i-tle -amelioration -role	6 j 1 h - #1	s. o.	29 mn	▶ ★

Jenkins / Jenkinsfile

Chaque rôle dispose d'un Jenkinsfile :

- * Contient les étapes pour que jenkins exécute les tests molecule sur openstack
- * Celui-ci est récupéré par le multibranch-pipeline du rôle qui a déclenché le webhook
- * C'est le point centrale de l'infra. Il contient toute la logique pour exécuter les tests molecules.

Détail des étapes du Jenkinsfile :

- * Clonage du repo du role à tester
- * Création de l'environnement virtuel python (dl des lib python ...) => workspace
- * Conversion du molecule.yml pour utiliser le driver openstack (script python custom)
- * Lancement des tests molecule (utilisation du driver openstack de molecule)
- * Notification de bitbucket de l'état du job

En cas d'échec, bitbucket est notifié et bloque la fusion de la branche jusqu'à ce que le test soit finalement en succès.

Intégration Jenkins/Openstack

L'idée est de lancer l'ensemble des scenarios molécule d'un rôle sur openstack afin de vérifier le déploiement du rôle sur un environnement plus proche de la réalité contrairement à docker.

Pour cela, chaque rôle dispose à sa racine d'un Jenkinsfile :

- Cloner le repo du role
- Convertit le molecule.yml docker en openstack
- Lance les scenarios molecule sur l'openstack
- Notifie bitbucket du résultat du job

Un multibranch pipeline est créé pour chaque rôle :

- Exécute pour les branches concernées le Jenkinsfile du role (càd lance les tests molécule sur openstack)

Le job entry_point_molecule déclenche le multibranch pipeline du role :

- Si et seulement si il y a un PR ouverte sur cette branche
- Inutile de surcharger le Jenkins/openstack sur une branche sans PR

Un webhook bitbucket pour chaque rôle est configuré :

- Déclenche un job jenkins « entry_point_molecule »
- Se déclenche lors d'un push/PR



Sujets transverse

Outils custom:

Pour faciliter l'utilisation de molecule nous avons mis en place un utilitaire :
Marmo.

- * Surcharge certaines commandes molecule
- * Gère la création de l'arborescence alternative molecule
- * Facilite l'usage des options debug et no-destroy
 - marmo -t -s <scenario_x> ... -s <scenario_y> (-d -n)
- * Permet de convertir un molecule.yml avec le driver **docker** en driver **openstack**
- * Permet de mapper le nom des images docker avec le nom des images openstack

Conversion du molecule.yml

Development local:



Jenkins CI:



```
openstack:  
  lsp-centos:6.6:  
    image: 'RHEL-6.6-N1-custom'  
    flavor: 'Small_Linux'  
    provisioner:  
      create: ../resources/openstack/create.yml  
      destroy: ../resources/openstack/destroy.yml  
      prepare: ../resources/openstack/prepare.yml  
  lsp-centos:7.3:  
    image: 'RHEL-7.3-N1-custom'  
    flavor: 'Small_Linux'  
    provisioner:  
      create: ../resources/openstack/create.yml  
      destroy: ../resources/openstack/destroy.yml  
      prepare: ../resources/openstack/prepare.yml
```

```

-----
dependency:
  name: galaxy
driver:
  name: docker
lint:
  name: yamllint
platforms:
  - name: "centos_6.6"
    image: "lsp-centos:6.6"
    registry:
      url: [REDACTED]
    disable_cache: true
    dockerfile: ../resources/Dockerfile.j2
  - name: "centos_7.3"
    image: "lsp-centos:7.3"
    registry:
      url: [REDACTED]
    disable_cache: true
    dockerfile: ../resources/Dockerfile.j2
provisioner:
  name: ansible
  options:
    vvv: false
  lint:
    name: ansible-lint
playbooks:
  converge: ../resources/playbooks/playbook-download.yml
env:
  ANSIBLE_VAULT_PASSWORD_FILE: "~/vault.key"
scenario:
  name: download
  test_sequence:
    - create
    - prepare
    - converge
    - check
    - verify
    - syntax
    - destroy
verifier:
  name: testinfra
  options:
    verbose: true
  directory: ../resources/test-file

```

1

Detection de l'image

```

docker:
  RHEL-6.6-N1:
    image: "lsp-centos:6.6"
    disable_cache: true
  RHEL-7.3-N1:
    image: "lsp-centos:7.3"
    disable_cache: true
openstack:
  >lsp-centos:6.6:
    image: 'RHEL-6.6-N1-custom'
    flavor: 'Small_Linux'
    provisioner:
      create: ../resources/openstack/create.yml
      destroy: ../resources/openstack/destroy.yml
      prepare: ../resources/openstack/prepare.yml
  lsp-centos:7.3:
    image: 'RHEL-7.3-N1-custom'
    flavor: 'Small_Linux'
    provisioner:
      create: ../resources/openstack/create.yml
      destroy: ../resources/openstack/destroy.yml
      prepare: ../resources/openstack/prepare.yml
# TODO variabiliser le volume_size (par defaut 40)
SPEC_lsp-centos:6.6:
  image: 'RHEL-6.6-N1-custom'
  flavor: 'HighCPU_Linux'
  volume_size: 40
  provisioner:
    create: ../resources/openstack/create.yml
    destroy: ../resources/openstack/destroy.yml
    prepare: ../resources/openstack/prepare.yml
# TODO variabiliser le volume_size
SPEC_lsp-centos_volume_100:6.6:
  image: 'RHEL-6.6-N1-custom'
  flavor: 'HighCPU_Linux'
  volume_size: 100
  provisioner:
    create: ../resources/openstack/create.yml
    destroy: ../resources/openstack/destroy.yml
    prepare: ../resources/openstack/prepare.yml

```

2

Conversion en molecule.yml
compatible Openstack

“Demo”

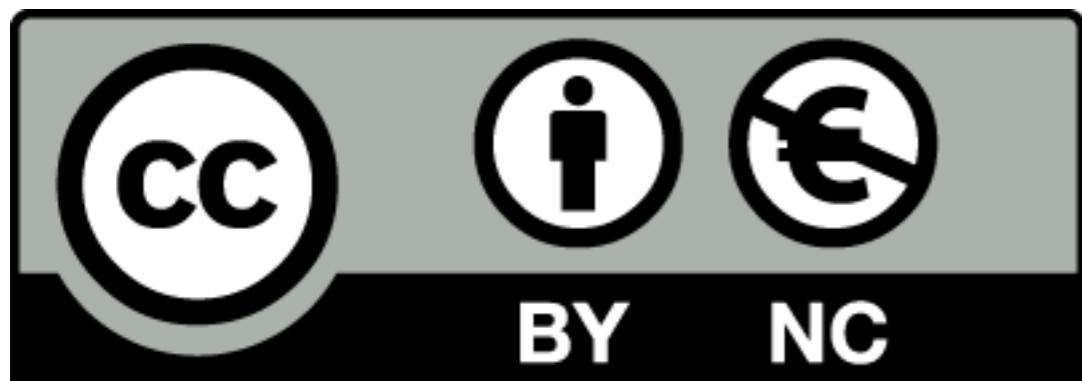


Questions ?

Github:

- <https://github.com/Hachebourin>

Auteurs: Thibault LECOQ et Quentin LE BARON



Ce support est sous licence Creative Commons:

- Citer les auteurs.
- Interdiction de tirer un profit commercial de l'œuvre sans autorisation de l'auteur.
- Partage de l'œuvre, avec obligation de rediffuser selon la même licence.