Monter un lab Linux pour Ansible

Monter un lab Linux pour Ansible

Ce chapitre envisage cinq solutions de lab pour apprendre Ansible pour dans le cadre d'une gestion de Linux. Une première solution montée avec Docker, deux autres avec Libvirt/KVM gérées en Bash ou avec Ansible lui-même et deux dernières avec Vagrant et Virtualbox ou Libvirt/KVM.

Cette documentation décrit des labs sous Linux Centos ou Debian/Ubuntu.

1. Comment monter un lab pour une gestion Ansible ?

Il existe de nombreuse façon d'expérimenter dans un lab une gestion multi-noeuds avec Ansible. Quelles sont ces possibilités ?

1.1. Topologie de lab

On trouvera ici la topologie du lab à monter, constituée de quatre noeuds Linux dont trois à gérer à partir d'un contrôleur avec Ansible. Pour économiser des ressources, il peut être conseillé de laisser l'hôte de virtualisation comme contrôleur et de monter seulement trois machines virtuelles. Le choix du système par défaut est celui de Centos 8 mais d'autres distributions de type Debian/Ubuntu devraient (aussi bien) fonctionner.

Topologie du lab Ansible

1.2. Examen des possibilités

Quelque soient vos choix, vous avez besoin d'un ordinateur, d'images de systèmes d'exploitation mais aussi différents logiciels.

Les logicels à installer sont un "provider" et un "provisionner". Dans une moindre mesure, vous pourriez être tenté de fabriquer vos propres images avec un "builder" :

- un "provider" fournit une solution de virtualisation logicielle (conteneur) ou matérielle (hyperviseur). On pourrait trouver, parmi d'autres choix :
  - Docker.
  - LXD.
  - Libvirt/Qemu/KVM.
  - Virtualbox.
- un "provisionner" crée la topologie et la manipule en configurant de l'authentification par clé SSH, en récupérant ou en configurant les adresses IP des noeuds, en éliminant la toplogie après usage, etc :
  - Bash : le plus simple et natif.
  - Ansible (lui-même) : plus robuste que Bash.
  - Vagrant : plus DevOps, qui s'intègre officiellement avec Virtualbox.
  - Docker-compose : l'outil d'orchestration Docker.
  - Terraform : avec un des "provider" cités, ou même combiné avec Ansible ou Bash, très robuste.
- un "builder" est une procédure (écrite en Bash, un fichier Dockerfile) ou un logiciel qui permet de préparer et fabriquer des images de systèmes d'exploitation pour un "provider".

Il existe bien d'autres possibilités avec les prestataires en nuage (AWS, Azure, GCP, …), avec OpenStack ou encore avec LXD, ou encore des scripts ou des wrappers écrits en Python (comme `kcli`) qui pourraient "approvisionner" la topologie. Mais évoquer ces technologies ici dépasse largement notre objectif.

1.3. Evaluation

Pour être à la hauteur de nos ambitions, il est nécessaire d'établir des critères de choix parmi les combinaisons proposées :

- la légèreté qui évalue les nécessités en ressources

- facilité qui évalue la facilité et la rapidité à monter / démonter le lab, à accéder aux machines, etc.
- réalité qui évalue la proxmité de la solution par rapport au monde réel à gérer avec Ansible

Provider	Provisionner	Légèreté	Facilité	Réalité	Solution
Docker	**Bash**	**+++**	**+++**	**- - -**	goffinet/docker-ansible-lab
Docker	Docker-compose	+++	-	- - -	-
Qemu/KVM	**Bash**	**-**	**+++**	**+++**	goffinet/virt-scripts
Qemu/KVM	**Ansible**	**-**	**++**	**+++**	goffinet/kvm-ansible-lab
Qemu/KVM	Terraform	-	-	+++	-
Virtualbox	**Vagrant**	**-**	**+**	**+++**	goffinet/vagrant-ansible-lab
Docker	Vagrant	+++	-	-	-
Qemu/KVM	Vagrant	-	- - -	-	goffinet/vagrant-libvirt-ansible-lab
Docker	Vagrant	+++	-	- - -	-

Parmi ces multiples solutions, cinq d'entre elles ont été choisies et testées pour les besoins de l'ouvrage. Une première solution avec Docker et un script Bash lance le lab. Cette solution est facile à monter et assez robuste mais les labs auront les limites d'une gestion de interne des conteneurs.

En choix principal, pour se rapprocher de la réalité native à Linux, je conseille vivement d'éprouver la virtualisation matérielle avec Libvirt et Qemu/KVM. On peut approvisionner le lab facilement avec des scripts Bash ou, de manière plus robuste, mais plus lente, avec un livre de jeu Ansible.

Enfin, je ne manquerai pas de citer Vagrant avec Virtualbox dans tous les cas ou Vagrant avec KVM sous Linux.

2. Docker / Bash

Code source du projet : goffinet/docker-ansible-lab

Cette solution a le très grand avantage de fonctionner partout, très vite et au coût le plus réduit. Cette solution convient pour des exercices de gestion multi-noeuds au début de l'apprentissage.

Toutefois, les conteneurs avec Docker connaissent des limites[1] tenant à leur nature notamment concernant le support du lancement des services à chaud. Cela signifie que toute perspective qui touche aux services (démarrage/arrêt) sera difficile à éprouver, de même que les redémarrages du conteneur lui-même à partir de l'espace du conteneur lui-même par exemple. Pour un alignement sur les sujets du RHCE, les conteneurs ne conviennent pas. D'ailleurs, cette solution est la démonstration d'une mauvaise pratique à éviter en environnement de production. Alors pourquoi utiliser des mauvaises idées, même faciles et tentantes ?

2.1. Prérequis

Le seul prérequis est d'installer **Docker**, ici sous Linux[2] :

```
curl -fsSL https://get.docker.com -o get-docker.sh && sh get-docker.sh
```

Vous pouvez aussi passer par http://play-with-docker.com (Cliquez sur "+ ADD NEW INSTANCE").

2.2. Se procurer le script

```
curl -s https://raw.githubusercontent.com/goffinet/docker-ansible-lab/master/startlab.sh > ./startlab.sh
chmod +x startlab.sh
```

2.3. Lancer le lab

```
./startlab.sh
```

Et vous êtes directement dans le contrôleur dans le dossier `/root/lab` avec un inventaire prêt à l'emploi.

2.4. Nettoyage du lab

```
./startlab.sh --remove
```

2.5. Conteneurs

`startlab.sh` démarre quatre conteneurs docker et vous connecte à l'environnement du "controller".

ansible.controller est un conteneur Alpine Linux dans lequel ansible est disponible. On trouve le Dockerfile dans ce même repo. C'est lui qui gère les trois autres noeuds.

node0, **node1** et **node2** sont les conteneurs basés Centos 8 qui agissent comme des noeuds exploitables. Ces noeuds ont déjà été approvisionnés avec la clé ssh du conteneur **ansible.controller**. Ainsi, vous n'avez pas à vous occuper de l'installation des clés. Cette image est disponible sur Registre d'images de Docker et le Dockerfile est dans ce repo.

2.6. Port Mapping

Certains ports des conteneurs sont exposés en tant que ports "exposés" sur l'hôte :

Conteneur	Port du conteneur	Port de l'hôte
node0	80	`$HOSTPORT_BASE`
node1	80	`$HOSTPORT_BASE+1`
node2	80	`$HOSTPORT_BASE+2`
node0	8080	`$HOSTPORT_BASE+3`
node1	30000	`$HOSTPORT_BASE+4`
node2	443	`$HOSTPORT_BASE+5`

La variable `HOSTPORT_BASE` est fixée à la valeur `42726` par défaut et peut être changée en démarrant le lab comme suit :

```
./startlab.sh --remove # Make sure you shut down the previous ones
HOSTPORT_BASE=<some_other_value> ./startlab.sh
```

2.7. Dossier de l'espace de travail Workspace

Un dossier `docker-ansible-lab/lab` sur votre machine locale est monté en tant que `/root/lab` dans le conteneur **ansible.controller**. Ainsi, vous pouvez utiliser votre éditeur favori sur votre machine

locale pour éditer des fichiers.

2.8. Fabriquer les images Docker

Cloner le code source et se rendre dans le dossier images :

```
git clone https://github.com/goffinet/docker-ansible-lab.git
cd docker-ansible-lab/images
make buil_all
```

3. Libvirtd avec les scripts virt-scripts

Code source du projet : goffinet/virt-scripts

Afin de mieux profiter des capacités de virtualisation d'un hôte Linux, il est recommandé d'installer le stack "libvirtd". Dans ce déploiement, on utilise des machines virtuelles plus proches de la réalité.

On propose ici de laisser le contrôleur sur le serveur de virtualisation pour améliorer la performance du lab.

3.1. Installation des pré-requis

Installation des pré-requis :

```
echo "Go to the home folder"
cd
echo "Install Git"
apt-get -y install git || yum -y install git
echo "Clone the virt-script repo on Github"
git clone https://github.com/goffinet/virt-scripts
echo "Go to the virt-scripts folder"
cd virt-scripts
echo "Install the requirements"
```

```
./autoprep.sh
systemctl stop apache2 || systemctl stop httpd
```

3.2. Création de la topologie

Lancer trois invités :

```
echo "include virt-scripts in the PATH"
export PATH=$PATH:~/virt-scripts/ ; echo "PATH=$PATH:~/virt-scripts/" >> ~/.bashrc
echo "Download images Centos and Ubuntu images"
download-images.sh centos8 --force
#download-images.sh focal --force
echo "Launch three Centos 8 guests"
for x in node0 node1 node2 ; do define-guest-image.sh $x centos8 ; done
```

3.3. Récupérer les adresses IP des machines

Récupérer les adresses IP des noeuds avec jq :

```
yum -y install epel-release && yum -y install jq
```

```
export node0=$(jq -r '.[] | select(.hostname=="node0") | ."ip-address"' /var/lib/libvirt/dnsmasq/virbr0.status  | tail -1)
export node1=$(jq -r '.[] | select(.hostname=="node1") | ."ip-address"' /var/lib/libvirt/dnsmasq/virbr0.status | tail -1)
export node2=$(jq -r '.[] | select(.hostname=="node2") | ."ip-address"' /var/lib/libvirt/dnsmasq/virbr0.status | tail -1)
```

3.4. Créer un fichier d’inventaire

Créer un dossier de travail `~/lab` et créer un fichier d’inventaire `inventory` :

```
cd
mkdir lab
cd lab
cat << EOF > inventory
[nodes]
node0 ansible_host=${node0}
node1 ansible_host=${node1}
node2 ansible_host=${node2}

[all:vars]
ansible_connection=ssh
ansible_user=root
ansible_ssh_pass=testtest

EOF
```

Afficher cet inventaire en format JSON :

```
ansible-inventory -i inventory --list
```

3.5. Créer un script d’inventaire dynamique en Bash

Voici un simple `nodes.sh` script d’inventaire dynamique en Bash qui récupère l’adresse IPv4 des trois invités libvirt :

```
#!/bin/bash
#https://stackoverflow.com/questions/56280806/ansible-dynamic-inventory-with-bash-script

if [ "$1" == "--list" ]; then
node0=$(jq -r '.[] | select(.hostname=="node0") | ."ip-address"' /var/lib/libvirt/dnsmasq/virbr0.status  | tail -1)
node1=$(jq -r '.[] | select(.hostname=="node1") | ."ip-address"' /var/lib/libvirt/dnsmasq/virbr0.status | tail -1)
node2=$(jq -r '.[] | select(.hostname=="node2") | ."ip-address"' /var/lib/libvirt/dnsmasq/virbr0.status | tail -1)
cat<<EOF
{
   "_meta": {
```

```
      "hostvars": {
        "node0": {
          "ansible_connection": "ssh",
          "ansible_host": "${node0}",
          "ansible_ssh_pass": "testtest",
          "ansible_user": "root"
        },
        "node1": {
          "ansible_connection": "ssh",
          "ansible_host": "${node1}",
          "ansible_ssh_pass": "testtest",
          "ansible_user": "root"
        },
        "node2": {
          "ansible_connection": "ssh",
          "ansible_host": "${node2}",
          "ansible_ssh_pass": "testtest",
          "ansible_user": "root"
        }
      }
    },
    "all": {
      "children": [
        "nodes",
        "ungrouped"
      ]
    },
    "nodes": {
      "hosts": [
        "node0",
        "node1",
        "node2"
      ]
    }
}
EOF
elif [ "$1" == "--host" ]; then
  echo '{"_meta": {hostvars": {}}}'
else
  echo "{ }"
```

```
fi
```

```
chmod +x nodes.sh
ansible-inventory -i nodes.sh --list
```

3.6. Créer un fichier de configuration

Dans le même dossier créer un fichier de configuration :

```
cat << EOF >> ansible.cfg
[defaults]
inventory = ./nodes.sh
host_key_checking = False
private_key_file = /root/.ssh/id_rsa
EOF
```

3.7. Test de connectivité

Tester Ansible sur tous les hôtes de l’inventaire :

```
ansible -m ping all
```

Résultat :

```
node0 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
node1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
node2 | SUCCESS => {
    "changed": false,
```

```
    "ping": "pong"
}
```

4. Solution de Lab avec KVM et Ansible lui-même

Code source du projet : goffinet/kvm-ansible-lab

4.1. Topologie de lab

Pour reproduire cette topologie avec libvirt/KVM, vous avez donc besoin d'un hôte de virtualisation physique (ou virtuel) avec les instructions de virtualisation activées et une installation Centos8 (ou Ubuntu 20.04) à jour.

Après avoir installé les pré-requis, nous allons créer trois machines virtuelles : controller, node1 et node2 et approvisionner la solution pour assurer une gestion des deux noeuds par la machine de contrôle.

Nous conseillons vivement d'utiliser des instances dans le nuage pour des serveurs "bare-metal"[3] chez Scaleway, Packet, Hetzner ou encore OVH.

Dans un premier temps, il s'agira de configurer l'hôte de virtualisation. Ensuite, on créera les trois machines virtuelles. Enfin, nous configurerons le "controller" en y installant Ansible, en y plaçant un fichier d'inventaire et un fichier de configuration par défaut. Une paire de clés SSH sera générée et permettra d'authentifier le controller auprès des noeuds.

Si vous avez un bon serveur de virtualisation sous la main installé en Centos 8, la configuration "se déroule" en quelques minutes.

Cette topologie est constituée de quatre machines virtuelles installée avec l'image Cloud de Centos 8 et des utilisateurs "root" et "ansible" configurés avec un mot de passe trivial.

4.2. Créer la topologie avec Ansible

S'il fallait créer manuellement cette topologie, il serait nécessaire de procéder par étapes :

1. Installation de quelques pré-requis (Ansible, KVM, Libvirt)

2. Téléchargement des images
3. Création de quatre VM : “controller”, “node0”, “node1” et “node2”

Heureusement un livre de jeu a été préparé pour faciliter ce déploiement.

Sur l’hôte de virtualisation, nous allons “cloner” avec Git le livre de jeux kvm-ansible-lab dans le dossier `~/kvm-ansible-lab` et exécuter celui-ci, il est conseillé d’être ‘root’ sur le serveur de virtualisation. Cela tient en quelques lignes.

```
yum -y install git || apt -y install git
git clone --recursive https://github.com/goffinet/kvm-ansible-lab.git \
~/kvm-ansible-lab
cd ~/kvm-ansible-lab
./run.sh
```

Le script `run.sh` vérifie que Ansible soit installé et vous propose le cas échéant d’y remédier (en Bash). Ensuite, il lance un livre de jeu Ansible qui construit la topologie.

4.3. Prendre connaissance de l’infrastructure créée

Entretemps, vous pouvez vous informer sur ce que vous exécutez en consultant le projet kvm-ansible-lab :

- le script run.sh,
- le livre de jeu virt-infra.yml que le script lance,
- et l’inventaire de la topologie inventory/lab.yml en format YAML.

La documentation nous indique que l’on peut modifier l’état de la topologie en manipulant une variable “`virt_infra_state`”. Ici pour détruire les machines virtuelles et les retirer de la gestion de libvirt, on valorise cette variable `"virt_infra_state=undefined"` sur l’exécution du livre de jeu :

```
./run.sh -e "virt_infra_state=undefined"
```

ou directement avec le livre de jeu :

```
ansible-playbook virt-infra.yml -e "virt_infra_state=undefined"
```

4.4. Gestion Ansible

Dès que le lab sera monté, les machines virtuelles seront directement gérables à partir de l'hôte de virtualisation :

```
ansible -m ping all
localhost | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
node0 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
controller | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
node2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
node1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Aussi, vous pourrez éventuellement vous connecter sur le contrôleur :

```
ssh root@controller
```

et créer un projet de base :

```
dnf -y install epel-release && dnf -y install ansible
PROJECT="lab"
mkdir ~/${PROJECT}
cd ~/${PROJECT}
cat <<  EOF > ~/${PROJECT}/ansible.cfg
```

```
[defaults]
inventory = ./inventory
host_key_checking = False
EOF
cat <<  EOF > ~/${PROJECT}/inventory
[nodes]
node0
node1
node2

[all:vars]
ansible_connection=ssh
ansible_user=ansible
ansible_ssh_pass=testtest
ansible_become=yes
ansible_become_user=root
ansible_become_method=sudo
EOF
ansible -m ping all
```

4.5. Approvissionement du serveur

...

5. Vagrant Up avec Virtualbox

Code source du projet : [goffinet/vagrant-ansible-lab]

Le script Bash `setup.sh` installe Ansible sur un serveur de virtualisation fonctionnant avec Centos 8 ou Ubuntu Focal et lance le livre de jeu `setup.yml` Ansible qui installe et configure Virtualbox, Vagrant et Packer.

Le dossier `./lab` contient ce lab géré par le serveur de virtualisation en tant que contrôleur Ansible.

```
yum -y install git || apt -y install git
git clone --recursive https://github.com/goffinet/vagrant-ansible-lab.git
cd vagrant-ansible-lab
```

```
./setup.sh
cd lab
vagrant up
ansible -m ping all
```

5.1. Démarrage

```
yum -y install git || apt -y install git
git clone --recursive https://github.com/goffinet/vagrant-ansible-lab.git
cd vagrant-ansible-lab
./setup.sh
```

5.2. Une première topologie

```
mkdir test
cd test
```

```
vagrant box add centos/8
vagrant init centos/8
vagrant up
vagrant ssh
```

5.3. Gestion de base

```
vagrant suspend
```

```
vagrant resume
```

```
vagrant halt
```

```
vagrant destroy
```

5.4. Approvisionnement

```
mkdir bootstrap
cd bootstrap
```

```
cat << EOF > bootstrap.sh
#!/usr/bin/env bash

yum -y install httpd
systemctl enable httpd
systemctl start httpd
EOF
```

```
cat << EOF > Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.box = "centos/8"
  config.vm.provision :shell, path: "bootstrap.sh"
end
EOF
```

```
vagrant provision
```

5.5. Déploiement de plusieurs boxes

```
mkdir lab
cd lab
```

```
cat << EOF > ./Vagrantfile
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  # General Vagrant VM configuration.
  config.vm.box = "centos/8"
```

```
  config.ssh.insert_key = false
  config.vm.synced_folder ".", "/vagrant", disabled: true
  config.vm.provider :virtualbox do |v|
    v.memory = 512
    v.linked_clone = true
  end

  # node0.
  config.vm.define "node0" do |app|
    app.vm.hostname = "node0"
    app.vm.network :private_network, ip: "192.168.12.10"
  end

  # node1.
  config.vm.define "node1" do |app|
    app.vm.hostname = "node1"
    app.vm.network :private_network, ip: "192.168.12.11"
  end

  # node2.
  config.vm.define "node2" do |app|
    app.vm.hostname = "node2"
    app.vm.network :private_network, ip: "192.168.12.12"
  end
end
EOF
```

```
cat << EOF > ./ansible.cfg
[defaults]
inventory = inventory
EOF
cat << EOF > ./inventory
[nodes]
node0 ansible_host=192.168.12.10
node1 ansible_host=192.168.12.11
node2 ansible_host=192.168.12.12

[all:vars]
ansible_ssh_user=vagrant
ansible_ssh_private_key_file=~/.vagrant.d/insecure_private_key
```

```
ansible_ssh_common_args='-o StrictHostKeyChecking=no'
ansible_become=yes
ansible_become_user=root
ansible_become_method=sudo
EOF
```

```
vagrant up
```

```
ansible -m ping all
```

6. Vagrant KVM sur Ubuntu Bionic

Code Source : [goffinet/vagrant-libvirt-ubuntu-bionic-ansible-lab]

> Uniquement sur Ubuntu Bionic.

6.1. Installer et configuration Vagrant et Libvirt/KVM

Ce installe différents composants (uniquement sur Ubuntu 18.04 Bionic) :

- Libvirt/Qemu/KVM
- Vagrant et plusieurs plugins
- Packer
- Terraform avec libvirt-provider
- Ansible

```
git clone https://github.com/goffinet/vagrant-libvirt-ubuntu-bionic-ansible-lab.git
cd vagrant-libvirt-ubuntu-bionic-ansible-lab
./setup.sh
reboot
```

6.2. Monter le lab

```
cd lab
vagrant up
```

```
ansible -m ping all
node0 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
node2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
node1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/libexec/platform-python"
    },
    "changed": false,
    "ping": "pong"
}
```

1. LXD serait une piste à explorer. ↩
2. Docker Desktop est aussi disponible pour Windows Install Docker Desktop on Windows et pour Mac Install Docker Desktop on Mac. ↩
3. Un “serveur bare-metal” est un serveur informatique qui est un “serveur physique à locataire unique”. Ce terme est utilisé de nos jours pour le distinguer des formes modernes de virtualisation et d’hébergement en nuage. (source) ↩

Revision #1
Created 3 October 2021 21:05:45 by garfi
Updated 3 October 2021 21:06:47 by garfi