ANSIBLE ET VAGRANT

ABDENNADHER Raed DELUCINGE Jean-Etienne LIATTI Steven RINGOT Gaëtan

Cours : Virtualisation des SI, 2017-2018

Enseignant : Brero Massimo Damiano Assistant : Sebastien Chassot

Table des matières :

Vagrant

Nous (Raed et Steven) avons commencé à explorer Vagrant avec KVM sous l'impulsion du groupe Cluster KVM (Sylvain, Loïc, Nathanaël et Mayron). Ils aimeraient pouvoir déployer et configurer leurs VMs automatiquement.

Qu'est-ce que Vagrant ?

Vagrant est un outil de gestion du cycle de vie de machines virtuelles. Il permet de créer et de lancer une ou plusieurs machines virtuelles pré-configurées selon des critères définis à l'avance dans des fichiers de configuration (Vagrantfile).

Recherches et tests avec Vagrant

Le problème initial est que Vagrant peut déployer des machines virtuelles uniquement pour VirtualBox, Hyper-V et VMware par défaut. Nous avons alors cherché un moyen de déployer pour KVM.

Un plugin/librairie existe, vagrant-libvirt (provider pour Vagrant). Nous avons commencé la lecture de cette doc. Nous avons utilisé la version 1.9.1 de Vagrant, qui n'est pas la dernière en date et qui est compatible avec libvirt. Nous avons réalisé toutes ces manipulations sur nos machines personnelles car la plupart du temps les droits *root* étaient requis et qu'il n'y avait pas forcément les bonnes versions des softs.

Nous avons eu de la peine à utiliser Vagrant avec libvirt et KVM car Vagrant n'est pas compatible par défaut avec KVM. Nous avons cherché plusieurs tutoriels et sommes tombés sur cet unique script qui avait l'air prometteur. Avec ce script nous voulions transformer notre image `client1` existante pour la donner au groupe KVM (voir ligne 12 dans le code suivant, `centos-7-client1-disk1.vmdk`).

```
BOX_NAME=vagrant-build
BASE_DIR="`pwd`/machines"
BOX_DIR="${BASE_DIR}/${BOX_NAME}"

mkdir -p ${BASE_DIR}

VBoxManage createvm --name "${BOX_NAME}" --ostype RedHat_64 --basefolder
${BASE_DIR}
VBoxManage registervm "${BOX_DIR}/${BOX_NAME}.vbox"
```

```
mkdir -p tmp
rm -rf tmp/clone.vdi
VBoxManage clonehd centos-7-client1-disk1.vmdk tmp/clone.vdi --format vdi
VBoxManage modifyhd tmp/clone.vdi --resize 20480
VBoxManage clonehd tmp/clone.vdi "${BOX_DIR}/${BOX_NAME}.vmdk" --format
vmdk
VBoxManage -q closemedium disk tmp/clone.vdi
rm -f tmp/clone.vdi

VBoxManage storagectl "${BOX_NAME}" --name LsiLogic --add scsi --controller
LsiLogic
VBoxManage storageattach "${BOX_NAME}" --storagectl LsiLogic --port 0 --
device 0 --type hdd --medium "${BOX_DIR}/${BOX_NAME}.vmdk"

VBoxManage setextradata "${BOX_NAME}"
"VBoxInternal/Devices/e1000/0/LUN#0/Config/SSH/Protocol" TCP
VBoxManage setextradata "${BOX_NAME}"
"VBoxInternal/Devices/e1000/0/LUN#0/Config/SSH/GuestPort" 22
VBoxManage setextradata "${BOX_NAME}"
"VBoxInternal/Devices/e1000/0/LUN#0/Config/SSH/HostPort" 22222

VBoxManage modifyvm "${BOX_NAME}" --usb on --usbehci on
VBoxManage modifyvm "${BOX_NAME}" --memory 512

VBoxManage startvm "${BOX_NAME}" #--type headless

echo "Sleeping to give machine time to boot"
sleep 240

echo "Uploading ssh key & creating vagrant user"
cat ~/.ssh/id_rsa.pub | ssh -p 22222 root@localhost "umask 077; test -d
.ssh || mkdir .ssh ; cat >> .ssh/authorized_keys"
ssh -p 22222 root@localhost <<EOT
  useradd vagrant
  echo vagrant | passwd vagrant --stdin
  umask 077
  test -d /home/vagrant/.ssh || mkdir -p /home/vagrant/.ssh
  cp ~/.ssh/authorized_keys /home/vagrant/.ssh
  chown -R vagrant:vagrant /home/vagrant/.ssh
EOT
scp -P 22222 templates/sudoers root@localhost:/etc/sudoers

echo -n "Waiting for machine to shutdown"
VBoxManage controlvm ${BOX_NAME} acpipowerbutton
while [ `VBoxManage showvminfo --machinereadable ${BOX_NAME} | grep
VMState=` != 'VMState="poweroff"' ]; do
  echo -n .
  sleep 1
done
```

```
echo "Done"
vagrant package --base ${BOX_NAME} --output ${BOX_NAME}.box
```

Ce script a fonctionné partiellement, car il ne prenait pas l'image qu'on lui a fournit, mais une image par défaut. **C'est pour cela que nous avons abandonné l'idée d'utiliser Vagrant avec KVM et libvirt.**

Nous avons toutefois remarqué (**27.11.2017**) que la librairie a été mise à jour ces derniers jours et, qu'apparemment, elle est compatible avec la dernière version de Vagrant (2.0.1).

Commandes et scripts pour CentOS 7

Vagrant utilise des "box" comme images de base, qu'on peut lui fournir en local (typiquement des fichiers iso ou .img) ou depuis Vagrant Cloud. Dans notre cas, nous avons utilisé l'image `centos/7`. Voici une marche à suivre pour utiliser Vagrant avec VirtualBox.

Tout d'abord, nous créons un dossier pour Vagrant :

```
$ mkdir vagrant_folder
$ cd vagrant_folder
$ vagrant init
```

`vagrant init` crée un fichier de configuration nommé `Vagrantfile`, dont voici le contenu initial :

```
Vagrant.configure("2") do |config|
  config.vm.box = "base"
end
```

Ensuite, nous exécutons cette commande :

```
$ vagrant box add centos/7
```

Ceci télécharge la box en question et nous permet de choisir le provider avec lequel nous allons travailler. Une fois cette tâche terminée (téléchargement), devons normalement obtenir le message suivant :

```
==> box: Successfully added box 'centos/7' (v1710.01) for 'VirtualBox'!
```

Nous pouvons maintenant commencer à éditer notre fichier `Vagrantfile` comme ceci :

```
Vagrant.configure("2") do |config|
  config.vm.box = "centos/7"
end
```

Finalement, nous lançons la machine virtuelle avec la commande suivante :

```
$ vagrant up
```

Voici sa sortie :

```
Bringing machine 'default' up with 'VirtualBox' provider...
==> default: Importing base box 'centos/7'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'centos/7' is up to date...
==> default: Setting the name of the VM:
vagrant_default_1511781695307_34658
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default:
    default: Vagrant insecure key detected. Vagrant will automatically
replace
    default: this with a newly generated keypair for better security.
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
    default: Key inserted! Disconnecting and reconnecting using new SSH
key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: No guest additions were detected on the base box for this VM!
Guest
    default: additions are required for forwarded ports, shared folders,
host only
    default: networking, and more. If SSH fails on this machine, please
install
    default: the guest additions and repackage the box to continue.
    default:
    default: This is not an error message; everything may continue to work
properly,
    default: in which case you may ignore this message.
==> default: Rsyncing folder: /home/raed/Bureau/vagrant/ => /vagrant
```

Nous constatons que par défaut le port pour SSH est le 2222, que l'IP est la 127.0.0.1 et que `username = vagrant` et utilise une paire de clé pour l'authentification. Finalement nous pouvons nous connecter à la machine virtuelle avec la commande suivante :

```
$ vagrant ssh
```

Décembre 2017 - Janvier 2018

Déploiement et configuration

Nous avons continué nos expériences avec Vagrant avec VirtualBox comme provider. Nous avons commencé par essayer avec une image CentOS, mais arrivés à l'étape de la configuration du serveur SSH, nous n'avons pas réussi à nous connecter.

D'après l'erreur survenue (`ssh_exchange_identification: read: connection reset by peer`) et nos recherches sur le web, il semblerait que ce soit une incompatibilité entre le client SSH des machines hepia et du serveur SSH CentOS. Nous avons finalement réussi à installer, configurer et lancer une machine virtuelle sous Debian (Jessie) et pouvoir s'y connecter en SSH avec l'utilisateur `vagrant`. Voici les fichiers de configuration et les scripts.

Arborescence des fichiers

```
.
├── debian
|   ├── bootstrap.sh
|   ├── conf_files_vm
|   |   ├── apt.conf
|   |   ├── interfaces
|   |   ├── proxy
|   |   ├── pub_key_server_ansible
|   |   ├── resolv.conf
|   |   └── sshd_config
|   ├── create.sh
|   └── Vagrantfile
└── vagrant.sh
```

Pour lancer toute la séquence, il faut exécuter `vagrant.sh`. Nous allons maintenant décrire la séquence complète et dans l'ordre d'exécution.

vagrant.sh

Ce fichier initial va copier le dossier `<template>` de l'OS voulu pour la VM dans le dossier `<folder_name>`. En effet, si on désire créer plusieurs machines virtuelles, il faut créer un dossier pour vagrant par machine virtuelle. Ce script va appeler le script suivant, `create.sh`, dans le dossier `<folder_name>`, en lui passant le nom de la VM, son IP, le nombre de CPU et la quantité de RAM voulus.

Finalement, une fois que la VM a été configurée et redémarrée, on lance le script Ansible depuis la VM serveur Ansible (cf. doc Ansible). Cette dernière étape est nécessaire uniquement car nous n'avons pas les droits pour installer Ansible sur la même machine physique de l'école où est installé Vagrant.

```
#!/bin/bash

if [[ $# != 6 ]]; then
    echo "Usage: ./vagrant.sh <template> <folder_name> <machine_name> <ip>
<cpu> <ram>"
    echo "<folder_name> is like: debian_apache"
    echo "<machine_name> is like: vagrant_debian_client"
    echo "<ip> is like: 10.194.184.196"
    echo "<cpu> is like: 2"
    echo "<ram> is like: 512"
    exit 1
fi

TEMPLATE=$1
FOLDER_NAME=$2
MACHINE_NAME=$3
IP=$4
CPU=$5
RAM=$6

rm -rf $FOLDER_NAME
mkdir $FOLDER_NAME
cp -r debian/* $FOLDER_NAME
cd $FOLDER_NAME
./create.sh $MACHINE_NAME $IP $CPU $RAM
cd ..
rm -rf $FOLDER_NAME


##### Ansible part #####
ssh root@10.194.184.190 "ansible-playbook
/etc/ansible/roles/main_service.yml -e \"service=Web port_http=80
domain=$IP\" --limit 'Debian'"
```

create.sh

Script de création : on commence par modifier le fichier `interfaces` et `Vagrantfile` avec les arguments fournis (IP, nom, CPU et RAM) avec l'utilitaire `sed` (lignes 21 à 24). On continue par supprimer si besoin une éventuelle image ayant le même nom (ligne 26), puis on lance l'installation (appel à `Vagrantfile`) (ligne 27) et une fois l'installation terminée, nous arrêtons la machine (ligne 28).

Les deux dernières lignes concernent VirtualBox, la 1ère permet à la machine virtuelle d'accéder au LAN (mode bridge) et la dernière redémarre la VM.

```
#!/bin/bash

if [[ $# != 4 ]]; then
    echo "Usage: ./create.sh <machine_name> <ip> <cpu> <ram>"
    echo "<machine_name> is like: vagrant_debian_client"
    echo "<ip> is like: 10.194.184.196"
    echo "<cpu> is like: 2"
    echo "<ram> is like: 512"
    exit 1
fi

NICTYPE="82540EM"
ADAPTER=enp0s31f6
FOLDER=conf_files_vm

MACHINE_NAME=$1
IP=$2
CPU=$3
RAM=$4

sed -i -e "s/\(address \).*/\1$IP/" $FOLDER/interfaces
sed -i -e "s/\(vb.name = \).*/\1\"$MACHINE_NAME\"/" Vagrantfile
sed -i -e "s/\(vb.memory = \).*/\1\"$RAM\"/" Vagrantfile
sed -i -e "s/\(vb.cpus = \).*/\1\"$CPU\"/" Vagrantfile

vagrant destroy
vagrant up
vagrant halt
VBoxManage modifyvm $MACHINE_NAME --nic1 bridged --nictype1 $NICTYPE --
bridgeadapter1 $ADAPTER
VBoxManage startvm $MACHINE_NAME --type headless
```

Vagrantfile

Ce fichier est le fichier de configuration d'une VM par Vagrant. Sont définis le nom de l'image (téléchargée sur Vagrant Cloud) (ligne 2), le script shell qui sera exécuté à la fin de l'installation (ligne 3), le provider et le nom de la VM (lignes 3 et 4) et la RAM et le CPU (lignes 6 et 7).

À noter que tout le contenu du dossier où se trouve ce fichier `Vagrantfile` sera copié à la racine de la VM, dans le dossier `/vagrant`.

```
Vagrant.configure("2") do |config|
  config.vm.box = "debian/jessie64"
  config.vm.provision :shell, path: "bootstrap.sh"
  config.vm.provider :virtualbox do |vb|
    vb.name = "vagrant_debian_client"
    vb.memory = 1024
    vb.cpus = 2
  end
end
```

bootstrap.sh

Script exécuté lorsque l'installation de la machine virtuelle est terminée, par elle-même. Nous copions tous les fichiers de configuration pré-écrits et mettons à jour les applications installées, ceci faisant également office de test de connectivité internet. La clé publique est celle du serveur Ansible.

```
#!/bin/bash

FOLDER=conf_files_vm

cp /vagrant/$FOLDER/interfaces /etc/network/interfaces
cat /vagrant/$FOLDER/proxy >> /etc/profile
cp /vagrant/$FOLDER/apt.conf /etc/apt/apt.conf
cp /vagrant/$FOLDER/resolv.conf /etc/resolv.conf
cat /vagrant/$FOLDER/pub_key_server_ansible >>
/home/vagrant/.ssh/authorized_keys

sudo apt update
sudo apt -y upgrade
sudo apt -y install net-tools nano htop

mv /etc/ssh/sshd_config /etc/ssh/sshd_config_old
cp /vagrant/$FOLDER/sshd_config /etc/ssh/sshd_config
```

interfaces

Fichier de configuration des interfaces réseau pour Debian.

```
source /etc/network/interfaces.d/*

auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
    address 10.194.184.196
    netmask 255.255.255.0
    gateway 10.194.184.1
```

proxy

Configuration du proxy (contrainte du réseau de l'hepia).

```
MY_PROXY="http://129.194.185.57:3128/"

HTTP_PROXY=$MY_PROXY
HTTPS_PROXY=$MY_PROXY
FTP_PROXY=$MY_PROXY
http_proxy=$MY_PROXY
https_proxy=$MY_PROXY
ftp_proxy=$MY_PROXY

export HTTP_PROXY HTTPS_PROXY FTP_PROXY http_proxy https_proxy ftp_proxy
```

apt.conf

Définition du proxy pour `apt` également.

```
Acquire::http::proxy "http://129.194.185.57:3128/";
Acquire::https::proxy "https://129.194.185.57:3128/";
Acquire::ftp::proxy "ftp://129.194.185.57:3128/";
```

resolv.conf

Fichier de configuration DNS (mauvaise adresse par défaut).

```
nameserver 9.9.9.9
```

sshd_config

Configuration du serveur SSH sur Debian. Les lignes en commentaires ont été enlevées pour plus de lisibilité. Le seul changement qui a été fait par rapport à la version par défaut inclue dans cette image Debian se trouve à la dernière ligne, `PasswordAuthentication yes` au lieu de `PasswordAuthentication no`, pour autoriser la connexion SSH avec mot de passe.

```
Port 22
Protocol 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
HostKey /etc/ssh/ssh_host_ed25519_key
UsePrivilegeSeparation yes
KeyRegenerationInterval 3600
ServerKeyBits 1024
SyslogFacility AUTH
LogLevel INFO
LoginGraceTime 120
PermitRootLogin without-password
StrictModes yes

RSAAuthentication yes
PubkeyAuthentication yes
AuthorizedKeysFile  %h/.ssh/authorized_keys
IgnoreRhosts yes
RhostsRSAAuthentication no
HostbasedAuthentication no
PermitEmptyPasswords no
ChallengeResponseAuthentication no

X11Forwarding yes
X11DisplayOffset 10
PrintMotd no
PrintLastLog yes
TCPKeepAlive yes
AcceptEnv LANG LC_*
Subsystem sftp /usr/lib/openssh/sftp-server
```

```
UsePAM yes
UseDNS no
PasswordAuthentication yes
```

Connexion à la VM

Une fois que toutes ces étapes sont terminées, nous pouvons nous connecter à cette machine virtuelle (depuis le réseau hepia), par cette commande et avec mot de passe **vagrant**.

```
ssh vagrant@10.194.184.196
```

Ansible

Introduction

Qu'est-ce que Ansible ?

Ansible est un logiciel libre d'automatisation des applications et de l'infrastructure informatique. Déploiement d'application, gestion de Configuration et livraison en continue.

But final :

Nous (Gaetan et Jean-Etienne) avons commencé à explorer Ansible et ces possibilités pour pouvoir déployer des installations, mises à jour, ... automatiquement dans un réseau pour pouvoir l'intégrer avec Vagrant (Read et Steven). Au final nous devons déployer une machine virtuelle avec un service et des paramètres donner.

Version utilisée (Ansible et OS) :

Pour notre projet nous avons installés un CentOS 7 avec une configuration minimale, dans laquelle nous avons par la suite ajouté :

- EPEL release
- Ansible (version 2.4.1.0)
- Python (version 2.7.5)
- Nano

Attention, la mise à jour d'Ansible peut fortement créer des soucis de compatibilités.

Format a respecté impérativement en cas de création d'un fichier YAML (.yml) :

· Chaque fichier doit commencer par « **---** » (3 tirets normaux) hors rôle

· L'indentation est très importante, utiliser uniquement des **ESPACES**.

· Suivant l'OS, il est important de les différencier, car les commandes ne sont pas multiplateformes et entraine la non-fonctionnalité du script ansible.

Accès SSH sur les machines :

Si des problèmes de connexion en ssh arrivent , utiliser les commandes suivantes :

```
ssh-add
ssh-keygen -t rsa -C "ansible@ip_ansible"
ssh-copy-id user@ipduclient
```

Permettent de régler les problèmes en créant puis copiant des clés SSH afin de par la suite, ne pas avoir de problèmes d'identification.

A noter : la génération de clés n'est pas nécessaires si déjà effectué une fois.

Principe des groupes d'utilisateurs/machines :

Il est à noter que le fichier se situant dans le fichier :

```
/etc/ansible/hosts
```

Se trouve les différents groupes de machines et utilisateurs pour ansible.

un utilisateur PEUT-être dans PLUSIEURS groupes différents.

Lors d'une commande ansible, il est possible de faire appel à un certain groupe, en rajoutant celui-ci à la fin de la commande ansible. Exemple avec le groupe **Users** :

```
ansible -m ping Users
```

Afin de rajouter un groupe, il faut reprendre la syntaxe suivante :

```
[nom_groupe]
nom_host1 ansible_ssh_host= addresse ip 1
nom_host2 ansible_ssh_host= addresse ip 2
nom_host3 ansible_ssh_host= addresse ip 3
```

Mais attention, il faut penser à créer le fichier

```
/etc/ansible/group_vars/nom_groupe
```

Il contient toutes les informations de configuration de connexion, comme ansible_ssh_user, qui n'est autre que le nom d'utilisateur utilisé pour se connecter au groupe en question.

Utilisation de playbooks :

Pour lancer un playbook dans Ansible il faut exécuter la commande suivante :

```
ansible-playbook chemin_du_fichier/Nom_du_fichier -e
"nom_variable=valeur_var"
```

Valeurs importantes à mettre dans le fichier en question :

- hosts : Qui indique le groupe ou la machine qui exécutera nos futures tâches/services
- remote_user : nom d'utilisateur utilisé pour exécuter les tâches en question.
- tasks : Indiquant le début de la liste des tâches qui vont être exécutées par la suite.

/!\ hosts a priorité sur remote_user !

Exemple simple avec un ping :

```
---
- hosts: Users
  remote_user: root

  tasks:
    - ping:
```

Dans le cadre d'un service, celui-ci doit être appelé grâce à la façon suivante :

```
- service:
     name: Nom_du_service_à_exécuter
     state: Etat_futur_du_service (started, stopped, restarted, reloaded)
```

Il est possible de récupérer des variables mises avec l'option -e (voir plus haut) pour ce faire le formatage du fichier se fait de la façon suivante :

- Nom_variable: "{{ Nom_variable }}" = peut recuperer une liste/tableau
- Nom_variable: "Nom_variable" = recuperer une string

```
---
  vars:
      Nom_variable: "{{ Nom_variable }}"

  tasks:
- include: tasks1.yml
when: 1

- include: tasks2.yml
when: 2
```

Condition When dans ansible

Avec une condition when on peut faire un "if" avec une condition, ce qui permet d'appelle des roles/playbook , de faire des tâches et etc selon une condition.

Pourquoi on l'utilise ?

Nous avons utilisé la condition "ansible_os_family" (intégrer dans les services d'ansible) pour différencier par rapport à la famille de l'OS pour utiliser les bonnes commandes liées à celui-ci. Nous avons aussi utilisé pour notre variable "service" qui permet de savoir quel service installer (Web,dhcp,..).

Rôles (extrait de buzut.fr):

Les rôles représentent une manière d'abstraire les directives includes. Grâce aux rôles, il n'est plus utile de préciser les divers includes dans le playbook, ni les paths des fichiers de variables etc. Le playbook n'a qu'à lister les différents rôles à appliquer. En outre, depuis les tasks du rôle, l'ensemble des chemins sont relatifs. Inutile donc de préciser Le nom du fichier suffit, Ansible s'occupe du reste.

Pourquoi un/des rôle(s)

Car cela nous permet de généraliser une installation par service (web,dhcp,..) plus facilement et réutilisable par d'autres personnes !

Génération du rôle

Pour générer un rôle , il faut utiliser "ansible-galaxy + nom + init". Il va générer les dossiers/fichiers de base pour notre rôle. Notre dossier contiendra les dossiers/fichiers suivants :

```
- Roles
    - "Nom du role"
        - Defaults : les variables par défaut qui seront à disposition du rôle.
            - main.yml
        - Vars : Variables à disposition du rôle cependant elles ont vocation à être modifiées par l'utilisateur et elles prennent le dessus sur celle du dossier « defaults » si elles sont renseignées.
            - main.yml
        - Tasks : ici on renseigne nos taches (comme dans un playbooks normal).
            - main.yml
        - Meta : Sert à renseigner les dépendances liées à nos rôles (ssl, et etc).
            - main.yml
        - README : renseigne sur comment utilisé les rôles, variables à définir et etc.
```

On appelle un rôle de la façon suivante:

```
---
  tasks:
- include_role: role.yml
```

Bibliographie :

https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-ansible-on-centos-7 https://github.com/ansible/ansible/issues/19584
https://docs.ansible.com/ansible/latest/yum_module.html
https://docs.ansible.com/ansible/latest/service_module.html
https://docs.ansible.com/ansible/latest/apt_module.html
https://docs.ansible.com/ansible/latest/apt_repository_module.html
https://serversforhackers.com/c/an-ansible-tutorial https://buzut.fr/tirer-toute-puissance-dansible-roles/