Sommaire

[1. Initiation au déploiement Vagrant 1](#_Toc120701764)

[2. Initiation au provisionning avec Ansible 10](#_Toc120701765)

[2.1) L’arborescence des fichiers ansible 12](#_Toc120701766)

[2.2) Explication des playbooks 13](#_Toc120701767)

[2.3) Explication des tâches effectuées dans chaque dossier roles 14](#_Toc120701768)

[2.4) Explication du contenu des dossiers vars et defaults 16](#_Toc120701769)

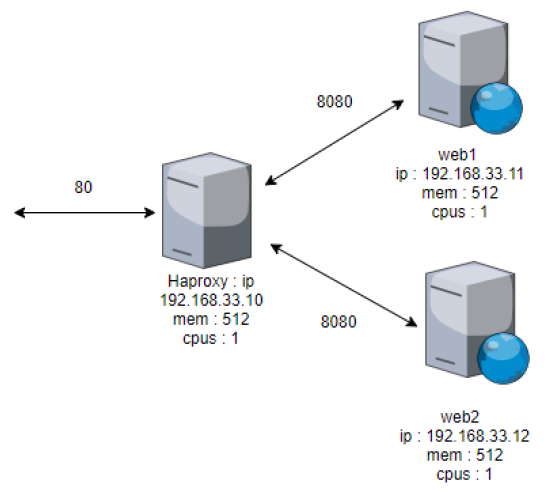
[2.5) Explication du dossier handlers 17](#_Toc120701770)

[2.6) Explication des templates 17](#_Toc120701771)

[2.7) Test du bon fonctionnement du script ansible 18](#_Toc120701772)

# 1. Initiation au déploiement Vagrant

Avant de commencer cette partie nous avons le schéma suivant qui va nous aider pour la suite :



1) Le script suivant permet de déployer 3 VMs haproxy, web1 et web2 avec la box centos/7 qui respecte les critères du schéma ci-dessus :

NODES = [

    {:hostname => "haproxy", :ip => "192.168.33.10", :cpus => 1, :mem => 512 },

    {:hostname => "web1", :ip => "192.168.33.11", :cpus => 1, :mem => 512},

    {:hostname => "web2", :ip => "192.168.33.12", :cpus => 1, :mem => 512}

    ]

Vagrant.configure("2") do |config|

  config.vm.box = "centos/7"

  NODES.each do |node|

    config.vm.define node[:hostname] do |cfg|

        cfg.vm.hostname = node[:hostname]

        cfg.vm.network "private\_network", ip: node[:ip]

            cfg.vm.provider "virtualbox" do |v|

                v.customize ["modifyvm", :id, "--cpus", node[:cpus] ]

                v.customize ["modifyvm", :id, "--memory", node[:mem] ]

                v.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]

                v.customize ["modifyvm", :id, "--natdnsproxy1", "on"]

                v.customize ["modifyvm", :id, "--name", node[:hostname] ]

            end

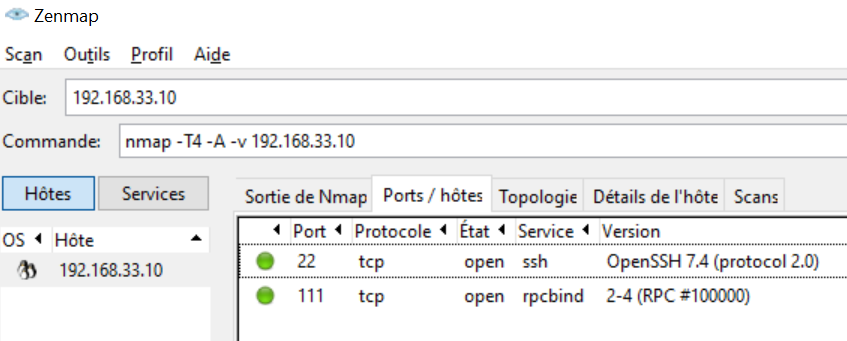
        end

    end

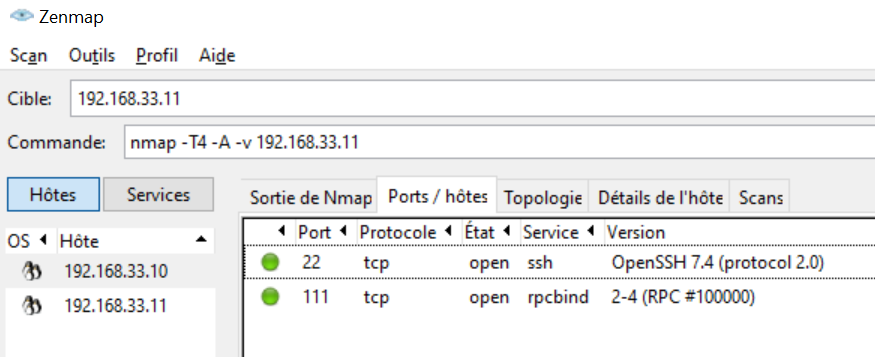
end

2) Maintenant je dois installer la commande nmap sur ma machine physique Windows et je scan les ports de mes différentes VMs :

Machine haproxy :



Machine web1 :



Machine web2 :

Une image contenant texte

Description générée automatiquement

Les 3 machines ont les mêmes ports ouverts par défaut lors de la création de celles-ci

Je dois maintenant protéger mes 3 VMs en installant firewalld dessus. Ce paquet étant déjà installé par défaut je n’ai besoin que d’activer le service et appliquer les règles de protection :

Sur les machines web1 et 2 :

sudo systemctl start firewalld

sudo firewall-cmd --permanent --zone=internal --add-source=192.168.33.10

sudo firewall-cmd --permanent --zone=internal --add-port=8080/tcp

sudo firewall-cmd --permanent --zone=public --add-port=22/tcp

sudo firewall-cmd --reload

Sur la machine haproxy :

sudo systemctl start firewalld

sudo firewall-cmd --permanent --zone=public --add-port=80/tcp

sudo firewall-cmd --permanent --zone=public --add-port=22/tcp

sudo firewall-cmd --reload

Je ne laisse que les ports 80 et 22 ouverts sur ma VM haproxy

Sur les VMs web je ne laisse que la machine haproxy accéder à la machine via le port 8080 ainsi que le port 22 ouvert pour toutes les IPs dans le même réseau

3) Je peux maintenant configurer le SSH sur mes 3 VMs afin qu’elles soient accessibles via la commande ssh [nom\_VM].

Tout d’abord je dois me générer une paire de clé SSH :

ssh-keygen -t rsa -b 4096

Maintenant je peux mettre la paire de clé dans un répertoire spécifique au TP :

C:/Users/samy2/OneDrive/Bureau/MAJEURE VIRTUALISATION/SSH haproxy/public.pub

C:/Users/samy2/OneDrive/Bureau/MAJEURE VIRTUALISATION/SSH haproxy/id\_rsa

Pour finir il faut mettre le code suivant dans le VagrantFile :

config.vm.define "haproxy" do |haproxy|

haproxy.vm.provision "shell" do |s|

# ssh\_pub\_key = File.readlines("C:/Users/samy2/OneDrive/Bureau/MAJEURE VIRTUALISATION/SSH haproxy/public.pub").first.strip

# s.inline = <<-SHELL

# echo #{ssh\_pub\_key} >> /home/vagrant/.ssh/authorized\_keys

# SHELL

# end

# end

Ce code permet d’aller lire le contenu de la clé publique et de l’écrire dans le répertoire des clés autorisées lors de la création de la machine virtuelle

Maintenant je peux me connecter à la machine en mettant la commande suivante :

ssh -i "C:\Users\samy2\OneDrive\Bureau\MAJEURE VIRTUALISATION\SSH haproxy\id\_rsa" USERt@IP\_Machine

Ici le -i sert à spécifier le chemin où se situe la clé privée

Si je veux configurer le SSH sur mes VMs pour qu’elles puissent se connecter entre elles via leur nom il faut faire les étapes suivantes :

Les VMs doivent d’abord savoir quelle IP porte quel nom. Pour cela j’édite le fichier /etc/hosts et dedans je défini les IPs et les noms auxquelles elles sont liées :

192.168.33.10 haproxy

192.168.33.11 web1

192.168.33.12 web2

Je dois aussi éditer le fichier /etc/ssh/sshd\_config pour autoriser chaque VMs à copier sa clé publique sur chaque serveur :

ChallengeResponseAuthentication yes

Puis redémarrer le service sshd :

sudo service sshd reload

Maintenant sur chaque serveur je peux faire les commandes suivantes pour générer une paire de clé SSH et copier la clé publique vers une machine distante :

ssh-keygen -t rsa -b 4096

sudo ssh-copy-id -i ~/.ssh/id\_rsa.pub [Nom\_Machine]

Je peux maintenant accéder à chaque VM grâce à la commande suivante :

ssh [Nom\_Machine]

4) J’installe maintenant le paquet haproxy sur ma VM haproxy et le paquet nginx sur mes VMs web1 et web2 :

Haproxy :

sudo yum update -y

sudo yum install haproxy -y

Sur les serveurs web1 et web2 :

sudo yum update -y

sudo yum install epel-release -y

sudo yum install nginx -y

sudo systemctl start nginx

Nous avons besoin de récupérer le repository epel qui contient le paquet nginx pour centos

5) Pour que les serveurs web répondent sur le port 8080 il faut modifier le fichier **/etc/nginx/nginx.conf** :

server {

listen 8080;

listen [::]:8080;

server\_name \_;

root /usr/share/nginx/html;

Ensuite pour vérifier que le load-balancing fonctionne sur le serveur haproxy bien on peut modifier le fichier **/usr/share/nginx/html/index.html** et faire 2 pages web différentes sur les serveurs web puis relancer le serveur nginx pour prendre en compte les modifications

6) Pour que le haproxy puisse servir à tour de rôle l’un des serveurs web il faut modifier le fichier **/etc/haproxy/haproxy.cfg** :

global

log /dev/log local0

log /dev/log local1 notice

defaults

log global

mode http

option httplog

option dontlognull

timeout connect 5000

timeout client 50000

timeout server 50000

frontend http\_front

bind \*:80

stats uri /haproxy?stats

default\_backend http\_back

backend http\_back

balance roundrobin

server web1 192.168.33.11:8080 check

server web2 192.168.33.12:8080 check

Je peux maintenant tester le si le haproxy fait bien le load-balancing :

Une image contenant texte

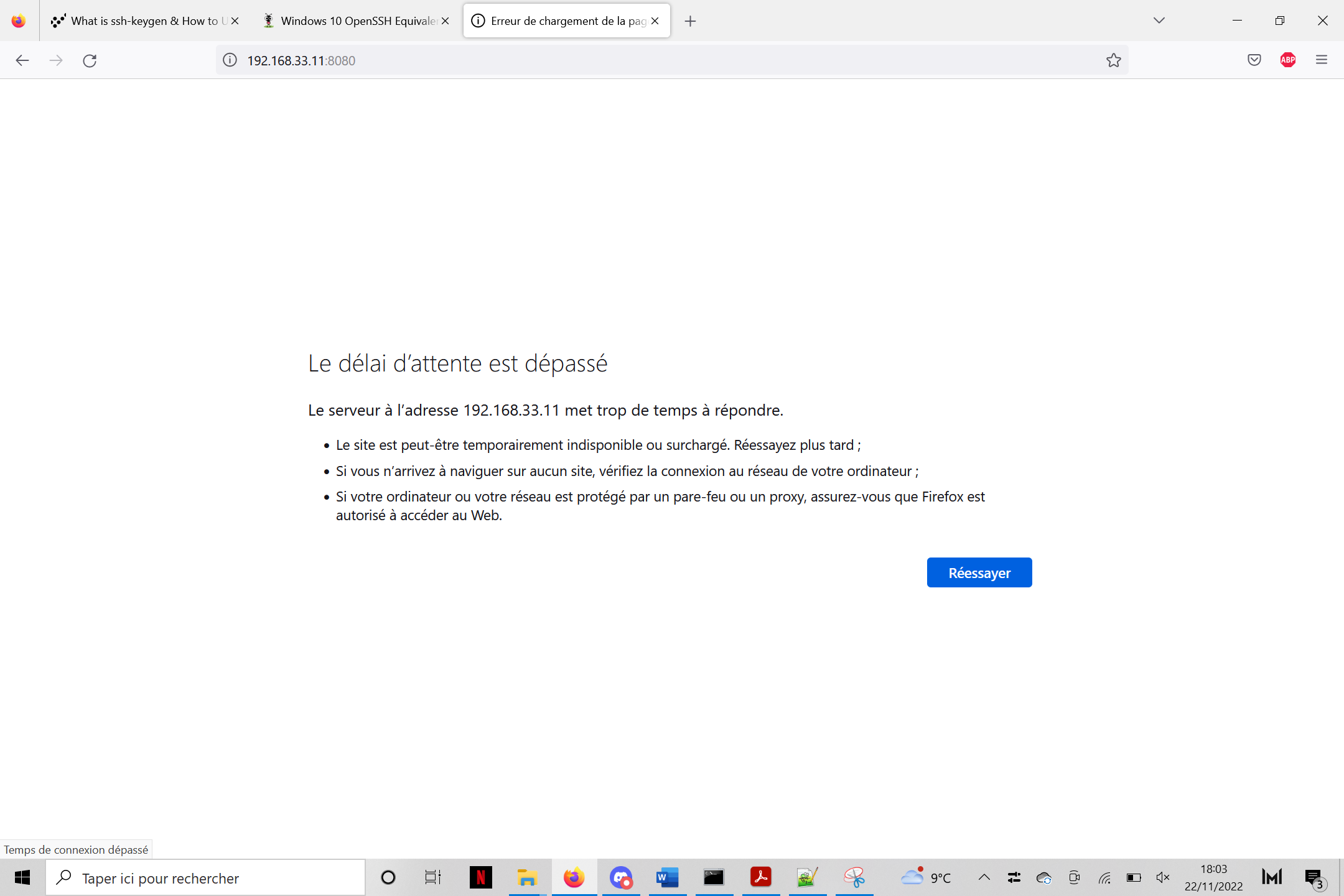
Description générée automatiquement

Maintenant si j’actualise la page je devrais arriver sur la page du serveur web2 :

Une image contenant texte

Description générée automatiquement

Le load-balancing se fait bien grâce au serveur haproxy. Je peux aussi tester d’accéder à la page web d’un des serveurs via son IP directement. Je ne devrais pas pouvoir y arriver car les règles du firewall n’autorisent que le haproxy à se connecter :



7) Pour terminer je peux mettre en place un script qui va déployer automatiquement toute l’architecture que nous venons de voir. Le script est donc le suivant :

NODES = [

    {:hostname => "haproxy", :ip => "192.168.33.10", :cpus => 1, :mem => 512 },

    {:hostname => "web1", :ip => "192.168.33.11", :cpus => 1, :mem => 512},

    {:hostname => "web2", :ip => "192.168.33.12", :cpus => 1, :mem => 512}

    ]

Vagrant.configure("2") do |config|

  config.vm.box = "centos/7"

  NODES.each do |node|

    config.vm.define node[:hostname] do |cfg|

        cfg.vm.hostname = node[:hostname]

        cfg.vm.network "private\_network", ip: node[:ip]

            cfg.vm.provider "virtualbox" do |v|

                v.customize ["modifyvm", :id, "--cpus", node[:cpus] ]

                v.customize ["modifyvm", :id, "--memory", node[:mem] ]

                v.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]

                v.customize ["modifyvm", :id, "--natdnsproxy1", "on"]

                v.customize ["modifyvm", :id, "--name", node[:hostname] ]

            end

        end

    end

    ## CONFIG MACHINE HAPROXY ##

    config.vm.define "haproxy" do |haproxy|

        haproxy.vm.provision "shell", inline: <<-SHELL

            sudo yum update -y

            sudo yum install haproxy sshpass -y

        SHELL

        haproxy.vm.provision "file", source: "haproxy.cfg", destination: "/tmp/haproxy.cfg"

        haproxy.vm.provision "shell", inline: <<-SHELL

            sudo rm /etc/haproxy/haproxy.cfg

            sudo mv /tmp/haproxy.cfg /etc/haproxy/haproxy.cfg

            sudo systemctl start haproxy

            sudo bash -c 'echo "192.168.33.11 web1" >> /etc/hosts'

            sudo bash -c 'echo "192.168.33.12 web2" >> /etc/hosts'

            sudo sed -i 's/ChallengeResponseAuthentication no/ChallengeResponseAuthentication yes/g' /etc/ssh/sshd\_config

            sudo systemctl restart sshd

            ssh-keygen -t rsa -b 4096 -f /home/vagrant/.ssh/haproxy -q -N ""

            sudo systemctl start firewalld

            sudo firewall-cmd --permanent --zone=public --add-port=80/tcp

            sudo firewall-cmd --permanent --zone=public --add-port=22/tcp

            sudo firewall-cmd --reload

        SHELL

    end

    ## CONFIG MACHINE WEB 1 ##

    config.vm.define "web1" do |web|

        web.vm.provision "shell", inline: <<-SHELL

            sudo yum update -y

            sudo yum install epel-release -y

            sudo yum install nginx sshpass -y

            sudo sed -i "s/80/8080/g" /etc/nginx/nginx.conf

            sudo bash -c 'echo "<html><h1>Web 1</h1></html>" > /usr/share/nginx/html/index.html'

            sudo systemctl start nginx

            sudo bash -c 'echo "192.168.33.10 haproxy" >> /etc/hosts'

            sudo bash -c 'echo "192.168.33.12 web2" >> /etc/hosts'

            sudo sed -i 's/ChallengeResponseAuthentication no/ChallengeResponseAuthentication yes/g' /etc/ssh/sshd\_config

            sudo systemctl restart sshd

            ssh-keygen -t rsa -b 4096 -f /home/vagrant/.ssh/web1 -q -N ""

            sudo systemctl start firewalld

            sudo firewall-cmd --permanent --zone=internal --add-source=192.168.33.10

            sudo firewall-cmd --permanent --zone=internal  --add-port=8080/tcp

            sudo firewall-cmd --permanent --zone=public --add-port=22/tcp

            sudo firewall-cmd --reload

        SHELL

    end

    ## CONFIG MACHINE WEB 2 ##

    config.vm.define "web2" do |web|

        web.vm.provision "shell", inline: <<-SHELL

            sudo yum update -y

            sudo yum install epel-release -y

            sudo yum install nginx sshpass -y

            sudo sed -i "s/80/8080/g" /etc/nginx/nginx.conf

            sudo bash -c 'echo "<html><h1>Web 2</h1></html>" > /usr/share/nginx/html/index.html'

            sudo systemctl start nginx

            sudo bash -c 'echo "192.168.33.10 haproxy" >> /etc/hosts'

            sudo bash -c 'echo "192.168.33.11 web1" >> /etc/hosts'

            sudo sed -i 's/ChallengeResponseAuthentication no/ChallengeResponseAuthentication yes/g' /etc/ssh/sshd\_config

            sudo systemctl restart sshd

            ssh-keygen -t rsa -b 4096 -f /home/vagrant/.ssh/web2 -q -N ""

            ## CLE WEB2 vers WEB1 & HA ##

            sudo sshpass -p "vagrant" ssh-copy-id -i /home/vagrant/.ssh/web2.pub -o StrictHostKeyChecking=no vagrant@web1

            sudo sshpass -p "vagrant" ssh-copy-id -i /home/vagrant/.ssh/web2.pub -o StrictHostKeyChecking=no vagrant@haproxy

            ## CLE WEB1 vers WEB2 & HA ##

            sudo ssh -i /home/vagrant/.ssh/web2 -o StrictHostKeyChecking=no vagrant@web1 "sudo sshpass -p 'vagrant' ssh-copy-id -i /home/vagrant/.ssh/web1.pub -o StrictHostKeyChecking=no vagrant@web2 \

            && sudo sshpass -p 'vagrant' ssh-copy-id -i /home/vagrant/.ssh/web1.pub -o StrictHostKeyChecking=no vagrant@haproxy"

            ## CLE HA vers WEB1 & WEB2 ##

            sudo ssh -i /home/vagrant/.ssh/web2 -o StrictHostKeyChecking=no vagrant@haproxy "sudo sshpass -p 'vagrant' ssh-copy-id -i /home/vagrant/.ssh/haproxy.pub -o StrictHostKeyChecking=no vagrant@web1 \

            && sudo sshpass -p 'vagrant' ssh-copy-id -i /home/vagrant/.ssh/haproxy.pub -o StrictHostKeyChecking=no vagrant@web2"

            sudo systemctl start firewalld

            sudo firewall-cmd --permanent --zone=internal --add-source=192.168.33.10

            sudo firewall-cmd --permanent --zone=internal --add-port=8080/tcp

            sudo firewall-cmd --permanent --zone=public --add-port=22/tcp

            sudo firewall-cmd --reload

        SHELL

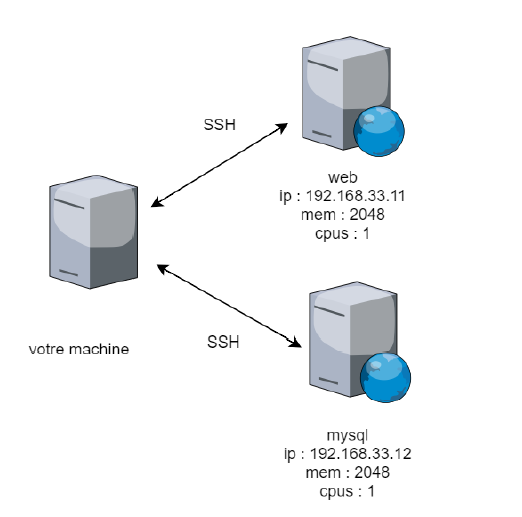
    end

end

Il nous faut créer un fichier haproxy.cfg dans le dossier où se situe le VagrantFile. Ce fichier contient les modifications du haproxy. Le VagrantFile va récupérer ce fichier et l’envoyer vers la VM haproxy afin de remplacer l’ancien fichier haproxy.cfg

# 2. Initiation au provisionning avec Ansible

Durant cette partie je vais devoir reproduire l’architecture suivante :



1) A l’aide de Vagrant je reproduis l’architecture ci-dessus :

NODES = [

    {:hostname => "web", :ip => "192.168.56.2", :cpus => 1, :mem => 2048 },

    {:hostname => "mysql", :ip => "192.168.56.3", :cpus => 1, :mem => 2048}

    ]

Vagrant.configure("2") do |config|

  config.vm.box = "generic/debian11"

  NODES.each do |node|

    config.vm.define node[:hostname] do |cfg|

        cfg.vm.hostname = node[:hostname]

        cfg.vm.network "private\_network", ip: node[:ip]

            cfg.vm.provider "virtualbox" do |v|

                v.customize ["modifyvm", :id, "--cpus", node[:cpus] ]

                v.customize ["modifyvm", :id, "--memory", node[:mem] ]

                v.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]

                v.customize ["modifyvm", :id, "--natdnsproxy1", "on"]

                v.customize ["modifyvm", :id, "--name", node[:hostname] ]

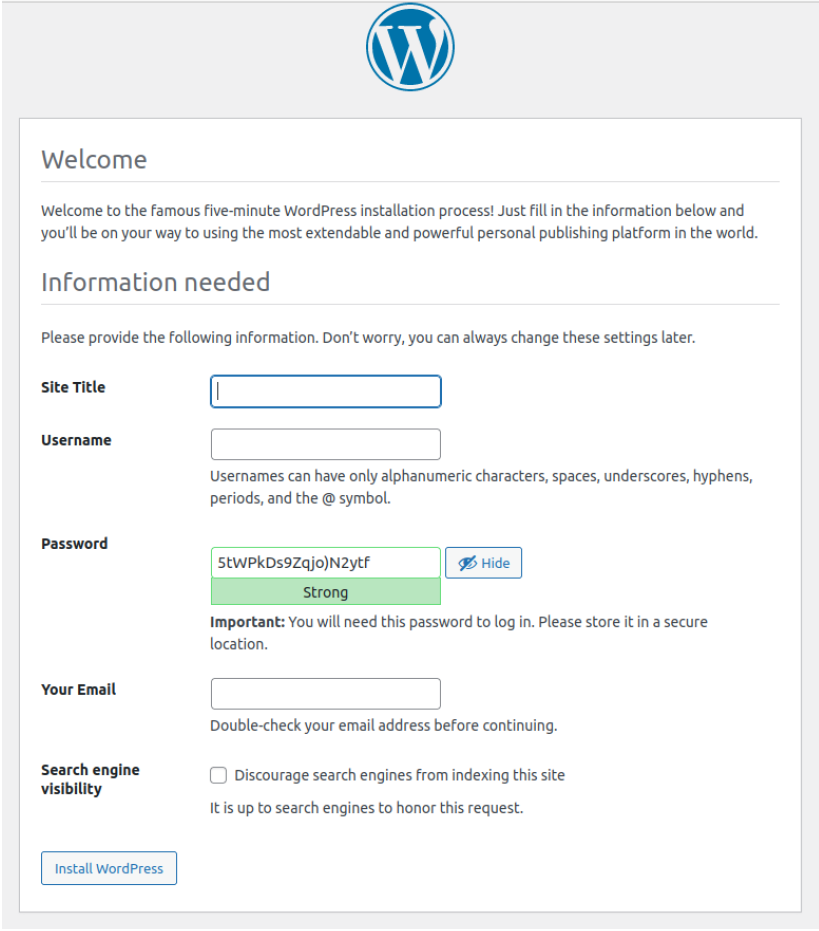
            end

        end

    end

Les IPs ne sont pas les bonnes car Virtualbox n’accepte pas la plage IP 192.168.33.X/24 sous Linux. Il faudrait pour cela que je créer une nouvelle carte réseau en tant que root sur Vbox pour pouvoir ensuite modifier les IPs dans le fichier Vagrantfile

2) A l’aide de l’outil ansible je dois provisionner mes machines afin d’y installer un serveur web, une base de données et un Wordpress. Le résultat de mon script devra m’emmener sur la page suivante lors de ma connexion à ma VM web dans un navigateur (firefox, google chrome…) :



## 2.1) L’arborescence des fichiers ansible

Voici donc l’arborescence de mes scripts :

wordpress

php

mysql

server

wp-config.php.j2

roles

templates

inventory.yml

playbook-bdd.yml

playbook-web.yml

Vagrantfile

Chaque rôle contient différents dossiers avec les tâches qui sont réalisées ainsi que les variables qui sont utilisées dans les rôles

Je peux maintenant modifier mon Vagrantfile pour qu’il exécute chaque playbook sur la machine concernée :

    config.vm.define "web" do |web|

        web.vm.provision "shell", inline: <<-SHELL

        sudo apt update -y

        sudo apt install ansible -y

        SHELL

        web.vm.provision "ansible" do |ansible|

            ansible.playbook = "playbook-web.yml"

        end

    end

    config.vm.define "mysql" do |mysql|

        mysql.vm.provision "shell", inline: <<-SHELL

        sudo apt update -y

        sudo apt install ansible -y

        SHELL

        mysql.vm.provision "ansible" do |ansible|

            ansible.playbook = "playbook-bdd.yml"

        end

    end

Chaque playbook se base sur un fichier inventaire qui va contenir les IP/FQDN des machines sur lesquelles le playbook doit se lancer. Voici le contenu de mon inventaire :

localhost ansible\_connection=local

Cette ligne veut dire que le playbook va devoir s’exécuter sur la machine locale et non pas sur une machine distante

## 2.2) Explication des playbooks

Voici maintenant le contenu de chaque playbook :

**Playbook-web.yml :**

- hosts: all

  roles:

  - server

  - php

  - wordpress

Ce playbook s’exécute sur chaque machine définie dans la partie « all » (ici machine locale) du fichier inventaire et va exécuter les différentes tâches contenues dans les rôles qui sont appelés

**Playbook-bdd.yml :**

- hosts: all

  roles:

  - server

  - mysql

## 2.3) Explication des tâches effectuées dans chaque dossier roles

**Roles/server/tasks/mail.yml :**

---

- name: Installation du package permettant d'installer le paquet mysql-server

  ansible.builtin.apt:

    deb: https://dev.mysql.com/get/mysql-apt-config\_0.8.22-1\_all.deb

  become: true

- name: Mise à jour du serveur (apt update)

  ansible.builtin.apt:

    update\_cache: yes

  become: true

- name: Installation des paquets nécessaires à Wordpress

  ansible.builtin.apt:

    pkg:

    - apache2

    - mysql-server

    - php-mysql

    - php

    - libapache2-mod-php

    - pip

  become: true

- name: Installation d'un paquet python (pymysql)

  ansible.builtin.pip:

    name: pymysql

  become: true

Ce rôle est dédié aux 2 machines et est composé de 4 tâches. Ces 4 tâches vont permettre de télécharger et d’installer tous les paquets nécessaires pour avoir un LAMP fonctionnel. Ici pour avoir le paquet mysql-server disponible au téléchargement il a d’abord fallu que je télécharge et que j’installe le paquet .deb qui va contenir le paquet mysql

**Roles/mysql/tasks/mail.yml :**

---

# tasks file for mysql

- name: Création de la base de données pour Wordpress

  mysql\_db:

    name: "{{ wp\_mysql\_db }}"

    state: present

    login\_unix\_socket: /run/mysqld/mysqld.sock

  become: true

- name: Création utilisateur distant

  mysql\_user:

    name: "{{ wp\_mysql\_user }}"

    password: "{{ wp\_mysql\_password }}"

    priv: '\*.\*:ALL'

    host: "{{ wp\_mysql\_host }}"

    login\_unix\_socket: /run/mysqld/mysqld.sock

  become: true

Ce rôle est dédié à la base de données et est composé de 2 tâches. La 1ère va permettre de créer une base de données mysql qui portera le nom stocké dans mon fichier de variables. La 2ème va permettre de créer un utilisateur distant qui pourra accéder à la base de donnée depuis une autre machine (ici la machine web)

**Roles/php/tasks/mail.yml :**

---

# tasks file for php

- name: Installation des modules php

  apt:

    name:{{ item }}

    state:present

  become: true

  with\_items:

  - php-gd

  - php-ssh2

Ce rôle est dédié au serveur web et est composé d’une seule tâche. Cette tâche permet d’installer des extensions à php afin de faire tourner correctement le wordpress

**Roles/wordpress/tasks/mail.yml :**

---

- name: Téléchargement de Wordpress

  get\_url:

    url: https://wordpress.org/latest.tar.gz

    dest: /tmp/wordpress.tar.gz

    validate\_certs: no

  become: true

- name: Extraction du paquet Wordpress

  unarchive:

    src: /tmp/wordpress.tar.gz

    dest: /var/www/

    copy: no

  become: true

- name: Mise à jour du site par défaut de Apache

  become: true

  lineinfile:

    dest: /etc/apache2/sites-enabled/000-default.conf

    regexp: "(.)+DocumentRoot /var/www/html"

    line: "DocumentRoot /var/www/wordpress"

  notify:

  - restart apache

- name: Configuration du fichier wp-config.php

  template:

    src: templates/wp-config.php.j2

    dest: /var/www/wordpress/wp-config.php

  become: true

Ce rôle est dédié au serveur web et est composé de 4 tâches. La 1ère et la 2ème tâche vont permettre de télécharger et installer les paquets pour avoir Wordpress. La 3ème tâche va modifier le document root du serveur apache pour non pas lancer la page contenue dans /var/www/html mais dans /var/www/wordpress et va relancer le service pour prendre en compte les modifications. Pour finir la dernière tâche va créer le fichier wp-config.php qui va contenir les informations de ma base de données afin d’y avoir accès directement au lancement du Wordpress.

## 2.4) Explication du contenu des dossiers vars et defaults

Les dossiers vars et defaults vont contenir les différentes variables qui seront appelé dans les tasks vu précédemment. Dans les tasks les variables sont appelés entre {{ }}. Lorsqu’une variable est appelé le script va chercher le contenu de cette variable dans le dossier defaults ou vars.

Voici donc le contenu de mes dossiers defaults :

**Roles/mysql/defaults/main.yml :**

---

# defaults file for mysql

wp\_mysql\_db: wordpressdb

wp\_mysql\_user: wordpress

wp\_mysql\_password: wordpress

wp\_mysql\_host: 192.168.56.2

**Roles/wordpress/defaults/main.yml :**

---

# defaults file for wordpress

wp\_mysql\_db: wordpressdb

wp\_mysql\_user: wordpress

wp\_mysql\_password: wordpress

wp\_mysql\_host: 192.168.56.3

## 2.5) Explication du dossier handlers

Le dossier handlers va contenir une ou plusieurs tâches permettant de redémarrer un service par exemple. Ici pour l’installation de Wordpress j’ai besoin d’apporter des modifications sur le serveur apache et donc de redémarrer le service une fois le serveur configuré. Pour cela j’utilise un handler.

Son contenu est le suivant : **roles/wordpress/handlers/main.yml :**

---

# handlers file for wordpress

- name: restart apache

  service:

    name: apache2

    state: restarted

  become: yes

## 2.6) Explication des templates

Les templates permettent de créer des documents qui contiennent des informations de configuration par exemple. Ici j’ai utilisé un template pour me créer un fichier wp-config.php permettant de mettre à jour la liaison de mon Wordpress à ma base de données.

Voici donc son contenu : **templates/wp-config.php.j2 :**

<?php

/\*\* The name of the database for WordPress \*/

define( 'DB\_NAME', '{{ wp\_mysql\_db }}' );

/\*\* Database username \*/

define( 'DB\_USER', '{{ wp\_mysql\_user }}' );

/\*\* Database password \*/

define( 'DB\_PASSWORD', '{{ wp\_mysql\_password }}' );

/\*\* Database hostname \*/

define( 'DB\_HOST', '{{ wp\_mysql\_host }}' );

/\*\* Database charset to use in creating database tables. \*/

define( 'DB\_CHARSET', 'utf8' );

/\*\* The database collate type. Don't change this if in doubt. \*/

define( 'DB\_COLLATE', '' );

define( 'AUTH\_KEY', 'put your unique phrase here' );

define( 'SECURE\_AUTH\_KEY', 'put your unique phrase here' );

define( 'LOGGED\_IN\_KEY', 'put your unique phrase here' );

define( 'NONCE\_KEY', 'put your unique phrase here' );

define( 'AUTH\_SALT', 'put your unique phrase here' );

define( 'SECURE\_AUTH\_SALT', 'put your unique phrase here' );

define( 'LOGGED\_IN\_SALT', 'put your unique phrase here' );

define( 'NONCE\_SALT', 'put your unique phrase here' );

/\*\*#@-\*/

$table\_prefix = 'wp\_';

define( 'WP\_DEBUG', false );

/\*\* Absolute path to the WordPress directory. \*/

if ( ! defined( 'ABSPATH' ) ) {

define( 'ABSPATH', \_\_DIR\_\_ . '/' );

}

/\*\* Sets up WordPress vars and included files. \*/

require\_once ABSPATH . 'wp-settings.php';

## 2.7) Test du bon fonctionnement du script ansible

Pour terminer je peux vérifier le bon fonctionnement de mon script en exécutant la commande « vagrant up ». Une fois les machines crées et configurées je peux lancer un navigateur et vérifier que je tombe bien sur la page de configuration de mon Wordpress :

Une image contenant texte

Description générée automatiquement