CRÉER UN CLUSTER KUBERNETES MULTI-NŒUD AVEC VAGRANT ET ANSIBLE

Introduction

Dans le chapitre précédent, nous avons vu comment mettre en place très facilement un cluster Kubernetes avec l'outil Minikube. Cependant, cette méthode connaît quelques limites, notamment le fait qu'on ne peut déployer qu'un nœud unique qui est à la fois de type master et worker.

Pour éviter ce problème, nous allons donc dans cet article **mettre en place un cluster Kubernetes multi-nœud** afin de s'approcher le plus d'un cluster k8s digne d'un environnement de production .

Je le dis et le redis chaque jour : "Un bon devopsien, est un informaticien qui automatise ses tâches !". Et comme nous sommes ingénieux , nous allons **automatiser l'aménagement de notre cluster Kubernetes multi-nœud**. Pour cela, nous utiliserons l'outil **Vagrant** et **Ansible**.

D'un côté, vagrant sera utilisé pour **créer et provisionner nos machines virtuelles** à l'aide de l'hyperviseur Virtualbox et d'un autre côté, Ansible sera utilisé pour **installer et configurer l'environnement Kubernetes**.

Vous récupérerez le projet complet dans mon Github ici.

Sans plus attendre, commençons les explications !

Structure et explication pas à pas du projet

Arborescence

Voici à quoi ressemble l'arborescence du projet, une fois téléchargé :

```
|__ roles
|   |___ common
|   |   |___ defaults
|   |   |
|___ main.yml
|   |   |___ handlers
|   |   |
|___ main.yml
|   |
|___ tasks
|   |
|___ main.yml
|   |___ main.yml
|   |___ master
|   |   |___ meta
|   |   |
|___ main.yml
|   |
|___ tasks
|   |
|___ main.yml
|
|___ worker
|       |___ meta
|       |
|___ main.yml
|
|___ tasks
|
|___ main.yml

|___ Vagrantfile
```

Nous avons donc en notre possession, trois rôles Ansible et un Vagrantfile. Je vais par la suite, vous expliquer les différents fichiers existant dans notre projet afin de comprendre les différentes étapes utilisées qui mènent à la création d'un cluster kubernetes multi-nœud solide.

Le Vagrantfile

C'est à partir du fichier **Vagrantfile** que toute la procédure débute. Ce fichier décrit comment nos nouvelles machines seront configurées et provisionnées.

J'ai mis un maximum de commentaires, histoire de comprendre la finalité de chaque instruction. Voici déjà à quoi ressemble donc notre fichier **Vagrantfile** :

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
# ##########################################################################
# ################### CONFIGURATION VARIABLES ##############################
# ##########################################################################
IMAGE_NAME = "bento/ubuntu-18.04"   # Image to use
MEM = 2048                          # Amount of RAM
CPU = 2                             # Number of processors (Minimum value of 2 otherwi
MASTER_NAME="master"                # Master node name
WORKER_NBR = 1                      # Number of workers node
NODE_NETWORK_BASE = "192.168.50"    # First three octets of the IP address that will b
POD_NETWORK = "192.168.100.0/16"    # Private network for inter-pod communication



Vagrant.configure("2") do |config|
    config.ssh.insert_key = false

    # RAM and CPU config
    config.vm.provider "virtualbox" do |v|
        v.memory = MEM
        v.cpus = CPU
    end

    # Master node config
    config.vm.define MASTER_NAME do |master|

        # Hostname and network config
        master.vm.box = IMAGE_NAME
        master.vm.network "private_network", ip: "#{NODE_NETWORK_BASE}.10"
        master.vm.hostname = MASTER_NAME

        # Ansible role setting
        master.vm.provision "ansible" do |ansible|

            # Ansbile role that will be launched
            ansible.playbook = "roles/main.yml"

            # Groups in Ansible inventory
            ansible.groups = {
                "masters" => ["#{MASTER_NAME}"],
                "workers" => ["worker-[1:#{WORKER_NBR}]"]
            }

            # Overload Anqible variables
            ansible.extra_vars = {
```

```
                node_ip: "#{NODE_NETWORK_BASE}.10",
                node_name: "master",
                pod_network: "#{POD_NETWORK}"
            }
        end
    end

    # Worker node config
    (1..WORKER_NBR).each do |i|
        config.vm.define "worker-#{i}" do |worker|

            # Hostname and network config
            worker.vm.box = IMAGE_NAME
            worker.vm.network "private_network", ip: "#{NODE_NETWORK_BASE}.#{i + 10}"
            worker.vm.hostname = "worker-#{i}"

            # Ansible role setting
            worker.vm.provision "ansible" do |ansible|

                # Ansbile role that will be launched
                ansible.playbook = "roles/main.yml"

                # Groups in Ansible inventory
                ansible.groups = {
                    "masters" => ["#{MASTER_NAME}"],
                    "workers" => ["worker-[1:#{WORKER_NBR}]"]
                }

                # Overload Anqible variables
                ansible.extra_vars = {
                    node_ip: "#{NODE_NETWORK_BASE}.#{i + 10}"
                }
            end
        end
    end
end
```

Vous pouvez personnaliser votre cluster Kubernetes depuis le fichier **Vagrantfile** à partir des variables de configuration, par exemple vous pouvez agrandir le nombre de nœuds en changeant la variable `WORKER_NBR`, voire attribuer davantage de ressources à vos nœuds en revalorisant les variables `CPU` et/ou `RAM`, etc ...

Cependant, si on exécute le contenu actuel de notre **Vagrantfile**, nous obtiendrons la configuration suivante sur nos VMs :

- Création de deux nœuds (1 Master et 1 Worker) tournant sous la distribution

Ubuntu en version 18.04 avec 2 CPU et 2048 en RAM pour chaque nœud.

- Création d'un réseau Virtualbox HOST ONLY, utilisé pour accéder au master et worker Kubernetes depuis notre machine hôte, avec notamment :
  - Le master nommé `master` et possédant l'ip `192.168.50.10`
  - Le worker nommé `worker-1` et possédant l'ip `192.168.50.11`
- Les connexions internes entre les PODs Kubernetes se passeront depuis un réseau privé sur la plage IP `192.168.100.0/16`. Ces adresses IPs ne seront pas accessibles de l'extérieur du cluster Kubernetes et changeront lorsque les PODs seront détruits et créés.
- Les logiciels indispensables pour le cluster Kubernetes seront installés et façonnés par le playbook Ansible **main.yml**.

Les rôles Ansible

Nous avons alors en notre disposition trois rôles Ansible, dont :

- `roles/common` : installe et configure les packages communs pour le maître et les workers qui sont nécessaires pour le bon fonctionnement du cluster Kubernetes.
- `roles/master` : spécifie la configuration spéciale au maître Kubernetes.
- `roles/node` : spécifie la configuration spéciale aux workers Kubernetes.

Comme vu précédemment, le **Vagrantfile** fait appel au playbook **main.yml**, voici son contenu :

```
- hosts: masters
  become: yes
  roles:
```

```
    - { role: master}

- hosts: workers
  become: yes
  roles:
    - { role: worker}
```

Ainsi, soit la machine est de type master, dans ce cas on lance le rôle `roles/master` , dans le cas contraire on lance le rôle `roles/worker` .

Ces deux rôles inclus tous les deux dans leur dossier, plutôt dans les fichiers **roles/worker/meta/main.yml** et **roles/master/meta/main.yml** le rôle `roles/commun` en tant que dépendance.

```
dependencies:
  - role: common
```

Le rôle commun

Au sein du rôle `roles/commun` , plus précisément dans le fichier **roles/common/defaults/main.yml** , on peut retrouver les multiples clés gpg, dépôts et paquets qui seront ajoutés et installés sur les nœuds du cluster, soit :

```
gpg_keys:
- key: https://download.docker.com/linux/ubuntu/gpg
- key: https://packages.cloud.google.com/apt/doc/apt-key.gpg

repositories:
- repo: "deb [arch=amd64] https://download.docker.com/linux/ubuntu {{ansible_distribut
- repo: "deb https://apt.kubernetes.io/ kubernetes-xenial main" #k8s not available yet

https_packages:
- name: apt-transport-https
- name: curl

docker_packages:
- name: docker-ce
- name: docker-ce-cli
- name: containerd.io


k8s_packages:
- name: kubeadm
```

```
- name: kubelet
- name: kubectl
```

J'ai déjà expliqué l'utilité des différents paquets kubernetes dans ce chapitre, cependant je n'ai pas encore décrit l'outil `kubeadm`, cet outil va nous permettre tout simplement d'**initialiser notre cluster Kubernetes** et de **joindre des machines à notre cluster k8s**.

Toujours dans le rôle `roles/common`, ça sera le fichier **roles/common/tasks/main.yml** qui se chargera de l'exécution des tâches collectives, voici son contenu :

```
- name: Install packages that allow apt to be used over HTTPS
  apt:
    name='{{ item.name }}'
    state=present
    update_cache=yes
  with_items: "{{ https_packages | default([]) }}"


- name: Add new repositories keys
  apt_key:
    url='{{item.key}}'
    state=present
  with_items: "{{ gpg_keys | default([]) }}"


- name: Add new apt repositories
  apt_repository:
    repo='{{item.repo}}'
    state=present
  with_items: "{{ repositories | default([]) }}"


- name: Install docker
  apt:
    name="{{ item.name }}"
    state=present
    update_cache=yes
  with_items: "{{ docker_packages | default([]) }}"
  notify:
    - docker status


- name: Add vagrant user to docker group
  user:
    name: vagrant
    group: docker
```

```
- name: Remove swapfile from /etc/fstab
  mount:
    name: "{{ item }}"
    fstype: swap
    state: absent
  with_items:
    - swap
    - none


- name: Disable swap
  command: swapoff -a
  when: ansible_swaptotal_mb > 0


- name: Install Kubernetes binaries
  apt:
    name="{{ item.name }}"
    state=present
    update_cache=yes
  with_items: "{{ k8s_packages | default([]) }}"

- name: Configure node ip
  lineinfile:
    path: '/etc/systemd/system/kubelet.service.d/10-kubeadm.conf'
    line: 'Environment="KUBELET_EXTRA_ARGS=--node-ip={{ node_ip }}"'
    regexp: 'KUBELET_EXTRA_ARGS='
    insertafter: '\[Service\]'
    state: present
  notify:
    - restart kubelet
```

Il commence d'abord par télécharger les paquets indispensables pour l'utilisation du protocole https pour l'utilitaire APT. Par la suite il rajoute les clés gpg et les dépôts définis précédemment. Postérieurement il installe l'environnement Docker et le démarre ensuite. Vous remarquerez aussi dans la liste des tâches, qu'à un moment donné, il désactive le swap, cette phase est indispensable pour le démarrage de `kubectl`. Une fois le swap désactivé, l'étape suivante est d'installer les paquets k8s et plus tard de rajouter l'IP du nœud en question dans la config `kubectl`.

Le rôle master

À ce stade, nous avons fini les étapes universelles des nœuds. Se suivent alors les tâches spécifiques à chaque type de nœud.

Pour le master, les tâches sont définies dans le fichier **roles/master/tasks/main.yml** :

```
- name: Initialize the Kubernetes cluster using kubeadm
  command: kubeadm init --apiserver-advertise-address="{{ node_ip }}" --apiserver-cert

- name: Setup kubeconfig for vagrant user
  command: "{{ item }}"
  with_items:
    - mkdir -p /home/vagrant/.kube
    - cp -i /etc/kubernetes/admin.conf /home/vagrant/.kube/config
    - chown vagrant:vagrant /home/vagrant/.kube/config

- name: Install flannel pod network
  become: false
  command: kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/master/Do

- name: Generate join command
  command: kubeadm token create --print-join-command
  register: join_command

- name: Copy join command to local file
  become: false
  local_action: copy content="{{ join_command.stdout_lines[0] }}" dest="./join-commanc
```

La première tâche exécutée "`kubeadm init`", initialise le nœud maître Kubernetes ou il est spécifié en tant que paramètres, l'IP du serveur API, le nom du cluster, et la plage IP des pods.

Une fois le nœud master initialisé, l'étape suivante consiste à gérer la partie réseau du cluster de manière à connecter les divers modules sur les différents nœuds du cluster. Pour cela, nous devons inclure un plugin CNI (Container Network Interface), "Hein quoi CNI, kezako ? "

CNI signifie "Container Networking Interface" c'est un projet de la CNCF (Cloud Native Computing Foundation), qui est une organisation qui gère également le projet Kubernetes. L'objectif de ce projet est de créer une norme conçue pour faciliter la configuration réseau des conteneurs que ça soit pendant leurs créations

ou lors de leurs destructions, le tout basée sur un plugin.

Ces plugins permettent de s'assurer que les exigences réseau de Kubernetes sont satisfaites et fournissent les fonctionnalités réseau requises pour assurer le bon fonctionnement du cluster. Différents plugins existent dont le plugin flannel, qui est celui utilisé dans notre rôle.

> **Information**
>
> CNI est défini par une spécification disponible ici , je vous laisse le soin de la lire.

L'étape finale réside sur la génération d'un token, qui sera utilisé plus tard par les workers afin de les autoriser à rejoindre le cluster. Ce token est par la suite copié dans un fichier qui est ensuite transféré sur notre machine locale dans le but d'être utilisé dans la suite par le rôle `roles/workers`.

Le rôle worker

Les tâches dans le rôle du worker n'ont rien de compliqué, et sont disponibles dans le fichier **roles/worker/tasks/main.yml** :

```
- name: Copy the join command to server location
  copy: src=join-command dest=/tmp/join-command.sh mode=0777

- name: Join the node to cluster
  command: sh /tmp/join-command.sh
```

Ici, il ne fait que récupérer et exécuter le fichier généré par le master contenant le token permettant de rejoindre notre cluster kubernetes.

Voilà, on en a fini avec les explications, exécutons maintenant notre projet !

Lancement du projet pour créer le cluster Kubernetes

Prérequis

Avant d'effectuer l'exécution du projet, il faut au préalable installer Vagrant et Ansible.

Installation de Vagrant

Rendez-vous sur la page d'installation officielle de vagrant et téléchargez le package correspondant à votre système d'exploitation et à votre architecture. Dans mon cas, ma machine est sous la distribution Fedora 30, je vais donc choisir le package 64 bits sous Centos, soit :

```
sudo rpm -Uvh https://releases.hashicorp.com/vagrant/2.2.5/vagrant_2.2.5_x86_64.rpm
```

Résultat :

```
Récupération de https://releases.hashicorp.com/vagrant/2.2.5/vagrant_2.2.5_x86_64.rpm
Verifying...                          ################################# [100%]
Préparation...                        ################################# [100%]
Mise à jour / installation...
   1:vagrant-1:2.2.5-1                ################################# [100%
```

Testez ensuite le bon déroulement de votre installation, en vérifiant la version de vagrant :

```
vagrant --version
```

Résultat :

```
Vagrant 2.2.5
```

Installation d'Ansible

Après avoir installé l'utilitaire vagrant, il faut dorénavant installer l'outil Ansible, référez-vous à la page d'installation officielle d'Ansible. Dans mon côté, sur ma distribution Fedora 30, voici comment j'ai installé Ansible :

```
sudo dnf -y install ansible
```

Comme d'habitude, on va tester le bon déroulement de notre installation, en vérifiant la version d'ansible :

```
ansible --version
```

Résultat :

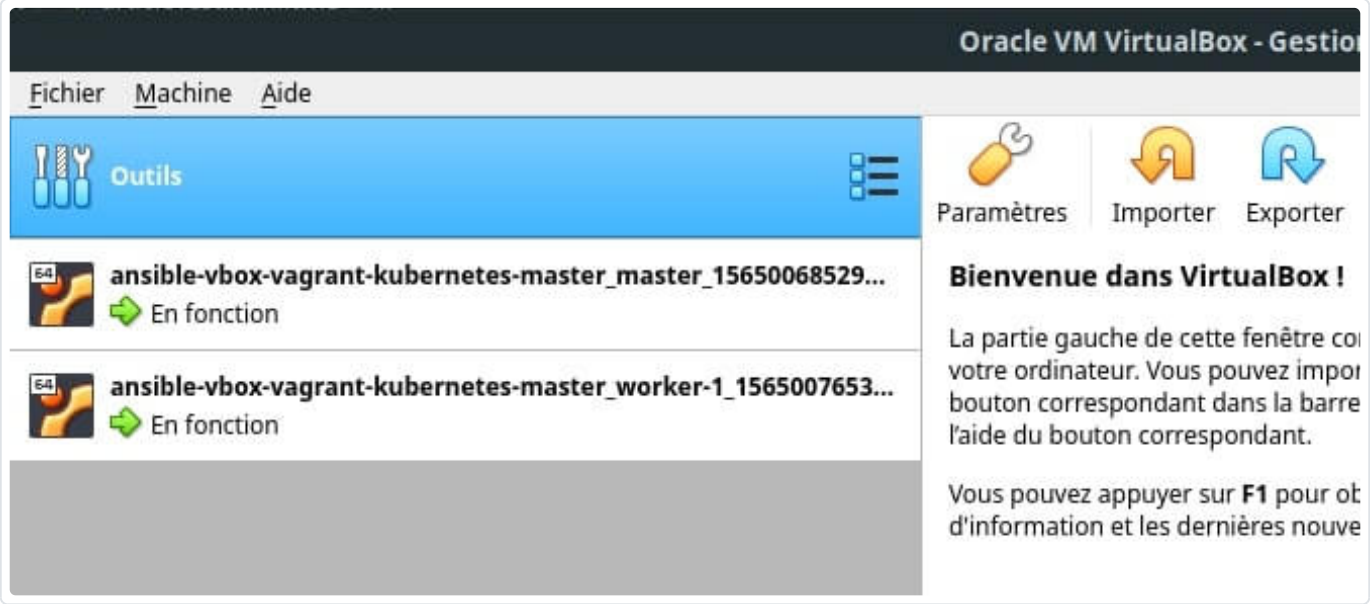
```
ansible 2.8.2
config file = /etc/ansible/ansible.cfg
configured module search path = ['/home/hatim/.ansible/plugins/modules', '/usr/share/a
ansible python module location = /usr/lib/python3.7/site-packages/ansible
executable location = /usr/bin/ansible
python version = 3.7.4 (default, Jul  9 2019, 16:32:37) [GCC 9.1.1 20190503 (Red Hat 9
```

Exécution du projet

Une fois tous les prérequis satisfaits, on peut dorénavant commencer par lancer notre projet. Placez vous d'abord dans la racine du projet au même niveau que le fichier **Vagrantfile** et lancez ensuite la commande suivante :

```
vagrant up
```

Après la fin de l'exécution, si on retourne sur Virtualbox, on peut visualiser les deux machines :



Si vous souhaitez ouvrir une connexion ssh, soit vous êtes au même niveau que votre **Vagrantfile**, dans ce cas, vous exécutez la commande suivante :

```
vagrant ssh master
```

Soit vous êtes dans un autre dossier :

```
ssh -r vagrant@192.168.50.10
...
password = vagrant
```

Si vous souhaitez communiquer avec l'API de votre cluster Kubernetes en utilisant l'outil **`kubectl`** depuis votre machine hôte, voici comment ça se passe :

Vous récupérer le dossier **.kube** de votre master, en le positionnant dans votre dossier utilisateur, comme suit :

```
scp -r vagrant@192.168.50.10:/home/vagrant/.kube $HOME/
...
password = vagrant
```

Dès à présent votre `kubectl` est configuré pour communiquer à votre API k8s, récupérerons alors la liste des nœuds disponibles dans notre cluster :

```
kubectl get nodes
```

Résultat :

```
NAME       STATUS   ROLES    AGE   VERSION
master     Ready    master   28m   v1.15.1
worker-1   Ready    <none>   20m   v1.15.1
```

Cool, comme prévu, en notre présence, nous avons nos deux nœuds, un master et un worker !

Conclusion

Vous l'aurez sans doute remarqué, la combinaison entre l'outil Vagrant et Ansible est vraiment extraordinaire. Elle nous a permis d'automatiser facilement l'installation et la configuration de notre cluster. On peut même le personnaliser en modifiant les variables de configuration disponibles dans le Vagrantfile.

Je vous conseille grandement de vous intéresser, un peu plus en profondeur à ces deux outils afin d'automatiser le provisionnement pour vos futurs projets.