

Cluster kubernetes HA

VagrantFile

```
VAGRANT_NODES = [
  { :hostname => "k8s-master1", :ip => "192.168.56.11" },
  { :hostname => "k8s-master2", :ip => "192.168.56.12" },
  { :hostname => "k8s-master3", :ip => "192.168.56.13" },
  { :hostname => "k8s-lb1",    :ip => "192.168.56.14" },
  { :hostname => "k8s-lb2",    :ip => "192.168.56.15" },
  { :hostname => "k8s-nfs",    :ip => "192.168.56.16" },
  { :hostname => "k8s-worker1", :ip => "192.168.56.17" }
]

Vagrant.configure("2") do |config|
  config.vm.box = "ubuntu/jammy64"
  config.vm.boot_timeout = 600

  VAGRANT_NODES.each do |node|
    config.vm.define node[:hostname] do |node_config|
      node_config.vm.hostname = node[:hostname]
      node_config.vm.network "private_network", ip: node[:ip]

      node_config.vm.provider "virtualbox" do |vb|
        if node[:hostname].start_with?("k8s-lb", "k8s-nfs")
          vb.memory = 512
          vb.cpus = 1
        elsif node[:hostname].start_with?("k8s-master")
          vb.memory = 4096
          vb.cpus = 2
        else
          vb.memory = 2048
          vb.cpus = 2
        end
      end
    end
  end
end
```

```
    vb.gui = false
  end
end
end
end
```

Nom de la VM	IP	Rôle	RAM	vCPU
k8s-master1	192.168.56.11	Master node	4096 MB	2
k8s-master2	192.168.56.12	Master node	4096 MB	2
k8s-master3	192.168.56.13	Master node	4096 MB	2
k8s-lb1	192.168.56.14	Load balancer	512 MB	1
k8s-lb2	192.168.56.15	Load balancer	512 MB	1
k8s-nfs	192.168.56.16	NFS Server	512 MB	1
k8s-worker1	192.168.56.17	Worker node	2048 MB	2

Chaque machine a un **nom d'hôte** et une **adresse IP privée** dans le réseau

192.168.56.x .

✓ Commandes nécessaires :

1. Créer et démarrer toutes les VMs :

```
vagrant up
```

2. Accéder à une VM (ex. master1) :

```
vagrant ssh k8s-master1
```

3. Voir l'état de toutes les VMs :

```
vagrant status
```

4. Arrêter toutes les VMs :

```
vagrant halt
```

5. Supprimer toutes les VMs (définitivement avec leurs disques) :

```
vagrant destroy -f
```

Voici une documentation claire et structurée de toutes les **étapes nécessaires pour préparer les nœuds** `k8s-master1` et `k8s-worker1` à rejoindre un cluster Kubernetes avec `containerd` comme runtime :

Déploiement Manuel d'un Cluster Kubernetes (v1.28)



Désactiver le Firewall (temporairement)

Permet de ne pas bloquer les communications entre les nœuds durant l'installation.

```
sudo ufw disable
```



Mise à jour du système & synchronisation de l'horloge

Kubernetes exige une synchronisation précise de l'heure entre les nœuds pour fonctionner correctement.

```
sudo apt update && sudo apt full-upgrade -y  
sudo apt install -y systemd-timesyncd  
sudo timedatectl set-ntp true
```



Désactiver le Swap

Kubernetes **n'autorise pas** le swap. Il faut le désactiver temporairement et définitivement.

```
# Désactiver temporairement :
```

```
sudo swapoff -a
```

```
# Désactiver définitivement :
```

```
sudo sed -i.bak -r 's/(.+ swap .+)/#\1/' /etc/fstab
```



Activer les modules kernel requis

Ces modules permettent à Kubernetes de gérer correctement le réseau entre pods.

```
echo -e "overlay\nbr_netfilter" | sudo tee /etc/modules-load.d/k8s.conf  
sudo modprobe overlay  
sudo modprobe br_netfilter
```



Configurer les paramètres réseau

Ces options permettent à `iptables` de bien gérer le trafic réseau inter-pods.

```
cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf  
net.bridge.bridge-nf-call-ip6tables = 1  
net.bridge.bridge-nf-call-iptables = 1  
net.ipv4.ip_forward = 1  
EOF  
  
sudo sysctl --system
```



Installer les paquets nécessaires

Préparer le système pour ajouter des dépôts tiers (Kubernetes, Docker...).

```
sudo apt install -y apt-transport-https ca-certificates curl gpg software-prope
```

Configuration du Load Balancer (k8s-lb1)

Objectif :

- Offrir une IP virtuelle stable (`192.168.56.14`) exposée aux workers et aux masters pour l'API Kubernetes.
- Répartir la charge entre `k8s-master1` , `k8s-master2` , `k8s-master3` .

Installer HAProxy

```
sudo apt update
sudo apt install -y haproxy
```

Configurer `/etc/haproxy/haproxy.cfg`

Ajoute **en bas du fichier** la section suivante :

```
frontend kubernetes-api
  bind *:6443
  mode tcp
  option tcplog
  default_backend kubernetes-masters

backend kubernetes-masters
  mode tcp
  balance roundrobin
  option tcp-check
  default-server inter 3s fall 3 rise 2
  server master1 192.168.56.11:6443 check
  server master2 192.168.56.12:6443 check
  server master3 192.168.56.13:6443 check
```

Redémarrer HAProxy :

```
sudo systemctl restart haproxy  
sudo systemctl enable haproxy
```

Installer Kubernetes (kubelet, kubeadm, kubectl)

➤ Ajouter la clé GPG et le dépôt Kubernetes :

```
sudo mkdir -p /etc/apt/keyrings  
  
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key | \  
sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg  
  
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://  
pkgs.k8s.io/core:/stable:/v1.28/deb/ /' | \  
sudo tee /etc/apt/sources.list.d/kubernetes.list
```

➤ Installer les binaires Kubernetes :

```
sudo apt update  
sudo apt install -y kubelet kubeadm kubectl  
sudo apt-mark hold kubelet kubeadm kubectl
```

kubeadm : pour initialiser ou rejoindre un cluster

kubelet : agent sur chaque nœud

kubectl : outil en ligne de commande pour gérer le cluster

Installer containerd (container runtime)

Kubernetes a besoin d'un runtime pour exécuter les conteneurs (ici, **containerd**).

➤ Ajouter le dépôt Docker :

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

➤ Installer containerd :

```
sudo apt update
sudo apt install -y containerd.io
```

➤ Générer et modifier la configuration pour utiliser **SystemdCgroup** :

```
sudo mkdir -p /etc/containerd
sudo containerd config default | sudo tee /etc/containerd/config.toml > /dev/null
sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/' /etc/containerd/config.toml
```

➤ Redémarrer containerd :

```
sudo systemctl restart containerd
sudo systemctl enable containerd
```



Configurer crictl (outil de debug pour containerd)

crictl est utile pour déboguer les conteneurs sans passer par Docker.

```
sudo apt install -y cri-tools
```

```
cat <<EOF | sudo tee /etc/crictl.  
runtime-endpoint: unix:///run/containerd/containerd.sock  
image-endpoint: unix:///run/containerd/containerd.sock  
timeout: 2  
debug: false  
pull-image-on-create: false  
EOF
```

Sur **chaque nœud** (master1, master2, master3, worker1), éditer :

```
sudo nano /etc/default/kubelet
```

Ajouter :

```
KUBELET_EXTRA_ARGS=--node-ip=192.168.56.11 # changer l'IP selon le nœu  
d
```

Puis recharger le service :

```
sudo systemctl daemon-reexec  
sudo systemctl restart kubelet
```



Télécharger les images Kubernetes

Sur tous les nœuds / LoadBalancer :

```
kubeadm config images pull --cri-socket unix:///run/containerd/containerd.so  
ck
```



Initialisation du Master1

Sur `k8s-master1` :

```
kubeadm init \  
  --apiserver-advertise-address=192.168.56.11 \  
  --pod-network-cidr=10.244.0.0/16 \  
  --control-plane-endpoint=192.168.56.14:6443 \  
  --upload-certs \  
  --cri-socket unix:///run/containerd/containerd.sock
```

À noter après l'exécution :

- Le **token** pour rejoindre les nœuds
- Le **hash** `-discovery-token-ca-cert-hash`
- La **clé** `-certificate-key` pour les masters

Configuration de `kubectl` sur `master1`

```
mkdir -p $HOME/.kube  
sudo cp /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Installation du réseau CNI (Weave Net)

Sur `master1` :

```
kubectl apply -f https://github.com/weaveworks/weave/releases/download/v  
2.8.1/weave-daemonset-k8s.
```

Ajouter les autres masters au cluster

Sur `k8s-master2` (192.168.56.12) :

```
kubeadm join 192.168.56.14:6443 \  
--token <YOUR_TOKEN> \  
--discovery-token-ca-cert-hash sha256:<HASH> \  
--control-plane \  
--certificate-key <CERT_KEY> \  
--apiserver-advertise-address=192.168.56.12 \  
--cri-socket unix:///run/containerd/containerd.sock
```

Sur `k8s-master3` (192.168.56.13) :

```
kubeadm join 192.168.56.14:6443 \  
--token <YOUR_TOKEN> \  
--discovery-token-ca-cert-hash sha256:<HASH> \  
--control-plane \  
--certificate-key <CERT_KEY> \  
--apiserver-advertise-address=192.168.56.13 \  
--cri-socket unix:///run/containerd/containerd.sock
```



Ajouter le worker

Sur `k8s-worker1` (192.168.56.17) :

```
kubeadm join 192.168.56.14:6443 \  
--token <YOUR_TOKEN> \  
--discovery-token-ca-cert-hash sha256:<HASH> \  
--cri-socket unix:///run/containerd/containerd.sock
```



Vérification Finale sur Master1

```
kubectl get nodes -o wide  
kubectl get pods -A -o wide
```

Documentation – Automatisation de l'installation d'un cluster Kubernetes avec Ansible


Structure Générale

```
k8s-cluster-ansible/  
├── ansible.cfg  
├── group_vars/  
│   └── all.yml  
├── inventory.yml  
├── keys/  
├── playbooks/  
└── roles/
```

1. **ansible.cfg**

Ce fichier configure le **comportement global d'Ansible**.

```
[defaults]  
inventory = inventory.yml    # Fichier d'inventaire personnalisé  
remote_user = root           # Utilisateur par défaut (souvent utilisé en root via sudo)  
host_key_checking = False    # Désactive la vérification du fingerprint SSH  
roles_path = ./roles         # Chemin vers les rôles
```


 **Remarque :** Ce fichier est global mais est **surpassé** par les valeurs définies dans `inventory.yml` (comme `ansible_user` et `ansible_ssh_private_key_file`).

2. `group_vars/all.yml`

Ce fichier contient les **variables globales** à tous les hôtes.

```
k8s_version: "1.28.15"                # Version de Kubernetes à installer
k8s_apt_key: https://pkgs.k8s.io/core:/stable:/v1.28/deb/Release.key
k8s_apt_repo: "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gp
g] https://pkgs.k8s.io/core:/stable:/v1.28/deb/ /"

docker_apt_key: https://download.docker.com/linux/ubuntu/gpg
docker_apt_repo: "deb [arch=amd64 signed-by=/etc/apt/keyrings/docker.gp
g] https://download.docker.com/linux/ubuntu {{ ansible_distribution_release | l
ower }} stable"
```

 Utilisées dans les rôles comme `containerd` et `kubernetes` pour ajouter les dépôts.


3. `inventory.yml`

C'est le fichier d'inventaire **statique**, qui déclare tous les nœuds avec leurs **adresses IP, utilisateurs SSH et chemins de clés privées**.

```
all:
  vars:
    ansible_user: vagrant
    ansible_ssh_private_key_file: ~/.vagrant.d/insecure_private_key

  children:
    masters:
      hosts:
        k8s-master1:
          ansible_host: 192.168.56.11
          ansible_user: vagrant
          ansible_ssh_private_key_file: keys/master1
          kubeadm_token: ...
          discovery_hash: ...
          certificate_key: ...
```

```
k8s-master2:
...
k8s-master3:
...
workers:
  hosts:
    k8s-worker1:
...
loadbalancers:
  hosts:
    k8s-lb1:
...
```

 **Les variables spécifiques (`token` , `hash` , `cert_key`)** sont ajoutées uniquement sur `k8s-master1` car c'est lui qui initie le cluster.

4. `keys/`

Ce dossier contient les **clés privées SSH Vagrant** copiées depuis :

```
C:\vagrant-k8s-cluster\.vagrant\machines<vm>\virtualbox\private_key
```

 Ces clés permettent à Ansible d'accéder à chaque VM sans mot de passe.

5. `playbooks/`

Ce dossier contiendra les **playbooks Ansible** pour automatiser chaque étape du déploiement :

- `setup_cluster.yml` : préparation commune (disable swap, modules kernel, etc.)
- `loadbalancer.yml` : configuration de HAProxy sur `k8s-lb1`
- `init_master.yml` : initialisation du premier master
- `join_master.yml` / `join_worker.yml` : ajout des nœuds au cluster
- `deploy_network.yml` : installation du CNI (ex. Weave Net)

- `deploy_cluster.yml` : exécution de tous les playbooks en une seule commande
- `configure_crictl.yml` : configuration de `crictl`
- `reset_cluster.yml` : suppression complète du cluster (utile en dev)

6. `roles/`

Répertoire contenant les **rôles Ansible** pour chaque composant :

Rôle	Fonction principale
<code>common</code>	Mise à jour système, désactivation swap, modules kernel, etc.
<code>containerd</code>	Installation et configuration du runtime containerd
<code>kubernetes</code>	Installation de kubelet, kubeadm, kubectl + sources APT
<code>init_master</code>	Initialise <code>k8s-master1</code> , génère la commande <code>kubeadm join</code>
<code>join_master</code>	Fait rejoindre <code>k8s-master2/3</code> au cluster en mode control-plane
<code>join_worker</code>	Fait rejoindre les workers
<code>loadbalancer</code>	Installe et configure <code>haproxy</code> (via <code>haproxy.cfg.j2</code>)
<code>reset_cluster</code>	Réinitialise complètement les nœuds Kubernetes

Rôle `common`

 **Cible** : Tous les nœuds (masters, workers, loadbalancer)

 **Objectif** : Appliquer toutes les prérequis système communs à Kubernetes.

 **Tâches exécutées** :

Ordre	Tâche	Description
1	Désactiver le firewall (ufw)	Évite les blocages de ports entre les nœuds
2	Mise à jour complète	Mise à jour du cache et des paquets (<code>apt full-upgrade</code>)
3	Installer <code>systemd-timesyncd</code>	Synchronisation automatique de l'horloge

4	Activer NTP	Active la synchronisation de l'heure
5	Désactiver le swap	Requis par Kubernetes (temporaire + permanent via <code>/etc/fstab</code>)
6	Charger les modules kernel requis	<code>overlay</code> et <code>br_netfilter</code>
7	Configurer <code>sysctl</code>	Active le forwarding IP et règles de pont réseau
8	Installer les paquets de base	curl, gpg, etc.

✅ **Résultat attendu :** Le système est prêt pour Kubernetes : pas de swap, réseau activé, modules chargés, heure à jour.

Rôle `containerd`

📌 **Cible :** Masters & Workers uniquement

🔧 **Objectif :** Installer et configurer `containerd` (le runtime conteneur).


🔧 **Tâches exécutées :**

Ordre	Tâche	Description
1	Ajouter la clé GPG Docker	Sécurise l'accès au dépôt
2	Ajouter le dépôt Docker	Permet l'installation de <code>containerd</code> depuis Docker
3	Installer <code>containerd</code>	Le runtime container utilisé par Kubernetes
4	Générer le fichier <code>config.toml</code>	Avec <code>containerd config default</code>
5	Configurer <code>SystemdCgroup = true</code>	Requis pour compatibilité avec kubelet
6	Redémarrer et activer <code>containerd</code>	Appliquer la config dès le boot

✅ **Résultat attendu :** `containerd` fonctionne avec les bons paramètres pour Kubernetes (`SystemdCgroup` activé).


Rôle **kubernetes**

 **Cible : Masters & Workers uniquement**

 **Objectif :** Installer les outils de base Kubernetes : **kubelet** , **kubeadm** , **kubectl** .

 **Tâches exécutées :**

Ordre	Tâche	Description
1	Créer /etc/apt/keyrings	Dossier pour stocker les clés
2	Télécharger la clé du repo Kubernetes	Signature de confiance
3	Ajouter le dépôt APT de Kubernetes	Accès à kubeadm , kubelet , etc.
4	Mettre à jour les paquets APT	Intègre le nouveau dépôt
5	Installer kubelet , kubeadm , kubectl	Les outils essentiels
6	Hold des versions	Empêche leur mise à jour automatique (stabilité)

 **Résultat attendu :** Kubernetes est installé et prêt à être initialisé ou rejoint via **kubeadm** .

Playbooks de test

Ces playbooks permettent d'exécuter **chaque rôle séparément** pour s'assurer qu'ils fonctionnent bien.

 **test_common.yml**

```
- name: Test common role
  hosts: all
  become: yes
  roles:
    - role: common
  tags: common
```


➡ Applique le rôle `common` à **tous les nœuds**.

▶ `test_containerd.yml`

```
- name: Test containerd role
  hosts: masters,workers
  become: true
  tags: containerd
  roles:
    - containerd
```

➡ Applique le rôle `containerd` uniquement aux **masters et workers**.

▶ `test_kubernetes.yml`

```
- name: Test kubernetes role
  hosts: masters,workers
  become: true
  tags: kubernetes
  roles:
    - kubernetes
```

➡ Applique le rôle `kubernetes` uniquement aux **masters et workers**.

Rôle `loadbalancer`

📌 **Cible :** `k8s-lb1`

🎯 **Objectif :** Installer et configurer **HAProxy** pour équilibrer la charge sur les masters.

📄 **Fichier** `haproxy.cfg.j2`

Contient la configuration HAProxy pour écouter sur le port `6443` et répartir les requêtes vers les 3 masters :

```
frontend kubernetes-frontend
  bind *:6443
  mode tcp
  option tcplog
  default_backend kubernetes-backend

backend kubernetes-backend
  mode tcp
  balance roundrobin
  option tcp-check
  default-server inter 10s downinter 5s rise 2 fall 3
  server k8s-master1 192.168.56.11:6443 check
  server k8s-master2 192.168.56.12:6443 check
  server k8s-master3 192.168.56.13:6443 check
```

Tâches `loadbalancer/tasks/main.yml`

1. Installe HAProxy
2. Déploie le fichier de config via un **template Jinja**
3. Redémarre et active le service

Rôle `init_master`

 **Cible :** `k8s-master1`


 **Objectif :** Initialiser le cluster Kubernetes.

Étapes :

1. Tirer les images nécessaires avec `kubeadm config images pull`
2. Lancer `kubeadm init` avec :
 - IP de l'API : `192.168.56.14` (Load Balancer)
 - CIDR du réseau Pod : `10.244.0.0/16` (Weave)

- Activation du certificat partagé `-upload-certs`
3. Sauvegarde la sortie dans `/root/kubeadm-init-output.txt`
 4. Configure `kubectrl` pour root (`~/.kube/config`)
 5. Génère un script `gen_join_cmd.sh` via le template Jinja :
 - Extrait et nettoie les commandes `kubeadm join` du fichier texte
 - Génère `/root/join-master.sh` automatiquement exécutable
-

Rôle `join_master`

 **Cible :** `k8s-master2` & `k8s-master3`


 **Objectif :** Faire rejoindre les autres masters au cluster en mode **control-plane**.

Étapes :

1. Exécute `kubeadm join` avec :
 - Token et hash fournis depuis `k8s-master1`
 - Clé du certificat (`-certificate-key`)
 2. Configure `kubectrl` (`~/.kube/config`) pour debug/commande locale
-

Rôle `join_worker`

 **Cible :** `k8s-worker1`

 **Objectif :** Ajouter les nœuds workers au cluster.

Étapes :

1. Exécute `kubeadm join` avec :
 - Token et hash depuis `k8s-master1`
 - IP publique du load balancer
 2. Ajoute `kubelet.conf` dans `~/.kube/config` pour debug local
-

Playbook `configure_crictl.yml`

 **Cible** : masters & workers

 **Objectif** : Installer et configurer `crictl` (outil pour interagir avec containerd).

Étapes :

1. Installe `cri-tools`
2. Crée `/etc/crictl.` avec les bons chemins socket pour containerd

Playbook `deploy_network.yml`

 **Cible** : `k8s-master1`

 **Objectif** : Installer le plugin réseau **Weave Net** (CNI)

Étape unique :

Applique le manifest officiel Weave Net :

```
kubectl apply -f https://github.com/weaveworks/weave/releases/download/v2.8.1/weave-dae
```

Extrait de la documentation

 Kubernetes HA Cluster avec Ansible

Ce projet permet de déployer automatiquement un cluster Kubernetes haute disponibilité (multi-master) avec un Load Balancer, `containerd` comme runtime, et Weave Net comme plugin réseau.

 Technologies utilisées :

- Ansible
- Kubernetes v1.28
- containerd
- HAProxy (load balancing)
- Weave Net (CNI)

```
## 📦 Cloner ce dépôt
```

```
...
```

```
git clone https://github.com/motrabelsi10/k8s-ansible-cluster.git
cd k8s-ansible-cluster
```

Structure du projet

- `inventory.yml` : inventaire des nœuds
- `group_vars/all.yml` : variables globales (versions, dépôts)
- `playbooks/` : scripts Ansible organisés par étape
- `roles/` : tâches organisées par composant (common, containerd, kubernetes, etc.)

Déploiement complet

Pour exécuter tout le déploiement automatiquement :

```
ansible-playbook playbooks/deploy_cluster.yml
```

⚠ Assure-toi que :

- Les clés SSH dans `keys/` sont correctes
- Les VMs sont accessibles depuis la machine d'exécution
- Ansible est installé sur ta machine



Ansible Kubernetes Helm Deployment — MySQL & Joget

Ce projet permet d'automatiser le déploiement de **MySQL** puis **Joget** sur un cluster Kubernetes grâce à **Helm** et **Ansible**, en respectant les bonnes pratiques (rôles, variables, tags, inventaire, etc.).



Arborescence du projet

```
ansible-joget-deploy/
├── ansible.cfg          # Configuration Ansible
├── inventory.yml        # Inventaire des hôtes
├── group_vars/
│   └── all.yml          # Variables globales
├── charts/              # (Optionnel) Chart local
│   └── mysql/           # Chart Helm MySQL
├── playbooks/
│   ├── deploy_mysql.yml # Déploie uniquement MySQL
│   ├── deploy_joget.yml # Déploie uniquement Joget
│   └── deploy_all.yml   # Déploie MySQL puis Joget
└── roles/
    ├── check_helm/      # Vérifie que Helm est installé
    │   └── tasks/main.yml
    ├── mysql/           # Déploiement Helm de MySQL
    │   └── tasks/main.yml
    └── joget/           # Déploiement Helm de Joget
        └── tasks/main.yml
```



Pré-requis

- Cluster Kubernetes opérationnel (avec un nœud `k8s-worker1`)

- Helm installé **uniquement sur** `k8s-master1`
- SSH sans mot de passe fonctionnel avec Ansible
- Les charts Helm `mysql` et `joget` disponibles dans :

```
/home/vagrant/joget-helm/mysql  
/home/vagrant/joget-helm/joget
```



Fichier `group_vars/all.yml`

```
namespace: joget  
mysql_chart_path: /home/vagrant/joget-helm/mysql  
joget_chart_path: /home/vagrant/joget-helm/joget  
release_name_mysql: mysql  
release_name_joget: joget
```



Fichier `inventory.yml`

```
all:  
  vars:  
    ansible_user: vagrant  
    ansible_ssh_private_key_file: ~/.vagrant.d/insecure_private_key  
  
  children:  
    masters:  
      hosts:  
        k8s-master1:  
          ansible_host: 192.168.56.11  
          ansible_user: vagrant  
          ansible_ssh_private_key_file: /home/vagrant/k8s-cluster-ansible/keys/m  
aster1
```



Fichier `ansible.cfg`

```
[defaults]
inventory = inventory.yml
remote_user = root
host_key_checking = False
roles_path = ./roles
```



Rôle `check_helm/tasks/main.yml`

Vérifie si Helm est installé, sinon échoue.

- name: Check if helm is installed
command: helm version
register: helm_check
ignore_errors: true
- name: Fail if helm is not installed
fail:
msg: "Helm n'est pas installé. Veuillez l'installer."
when: helm_check.rc != 0



Rôle `mysql/tasks/main.yml`

Déploie MySQL dans le namespace `joget` :

- name: Deploy MySQL with Helm
command: >
helm upgrade --install {{ release_name_mysql }}
{{ mysql_chart_path }}
-n {{ namespace }} --create-namespace
tags: mysql

Le chart Helm cible spécifiquement k8s-worker1 via un nodeSelector dans values..



Rôle `joget/tasks/main.yml`

Déploie Joget dans le même namespace :

```
- name: Deploy Joget with Helm
  command: >
    helm upgrade --install {{ release_name_joget }}
    {{ joget_chart_path }}
    -n {{ namespace }}
  tags: joget
```



Playbook `deploy_mysql.yml`

```
- name: Check Helm and Deploy MySQL
  hosts: k8s-master1
  become: true
  roles:
    - check_helm
    - mysql
```



Playbook `deploy_joget.yml`

```
- name: Deploy Joget Application
  hosts: k8s-master1
  become: true
  roles:
    - check_helm
    - joget
```

Playbook `deploy_all.yml`

Déploie MySQL puis Joget dans le bon ordre :

```
- name: Deploy MySQL then Joget
  hosts: k8s-master1
  become: true
  roles:
    - check_helm
    - mysql
    - joget
```

Exécution des playbooks

Déployer uniquement MySQL :

```
ansible-playbook playbooks/deploy_mysql.yml
```

Déployer uniquement Joget :

```
ansible-playbook playbooks/deploy_joget.yml
```

Déployer tout (MySQL puis Joget) :

```
ansible-playbook playbooks/deploy_all.yml
```

Vérification

Après exécution, tu peux vérifier les pods :

```
kubectl get pods -n joget -o wide
```

Les pods `mysql` et `joget` doivent être en statut `Running`, et tourner sur `k8s-worker1` si ton `nodeSelector` est bien configuré.

Rôle du VIP (Virtual IP)

Le **VIP** agit comme **un point d'accès unique** vers le **cluster Kubernetes**, en particulier vers les **API Servers** des `master` nodes.

Pourquoi c'est important ?

1. Abstraction du backend

- Tu n'as pas besoin de connaître l'IP réelle des masters (`192.168.56.11`, `.12`, `.13`).
- Tu utilises **une seule IP (VIP)**, par exemple `192.168.56.30`.

2. Équilibrage de charge

- Le **VIP est associé à HAProxy** sur les load balancers (`k8s-lb1`, `k8s-lb2`).
- Le trafic est réparti automatiquement entre les différents masters.

3. Haute disponibilité

- Si `k8s-lb1` tombe, **Keepalived** bascule le VIP vers `k8s-lb2`.
- Le cluster continue de répondre via la **même IP**, sans interruption.

Exemple concret

Dans ta commande `kubeadm init` :

```
kubeadm init \  
  --control-plane-endpoint=192.168.56.30:6443 \  
  ...
```

- `192.168.56.30` est le **VIP**
- Il pointe vers les 2 LBs (grâce à Keepalived)
- Et les LBs redirigent vers les masters (grâce à HAProxy)

✓ 1. Objectif

Mettre en place deux load balancers avec :

- HAProxy : pour répartir la charge entre les 3 masters.
- Keepalived : pour fournir une IP virtuelle (VIP) en cas de panne de l'un des LB.

📁 2. Architecture

```
VIP (ex: 192.168.56.30)
|
|-- k8s-lb1 (192.168.56.14) -- HAProxy
|-- k8s-lb2 (192.168.56.15) -- HAProxy
|_ Cluster Masters: 192.168.56.11, .12, .13
```

⚙️ 3. Étapes pour chaque LB

🧩 3.1. Installer HAProxy et Keepalived

Créer un rôle Ansible `loadbalancer` (ou configurer manuellement) :

```
sudo apt update && sudo apt install -y haproxy keepalived
```

📝 3.2. Configurer HAProxy `/etc/haproxy/haproxy.cfg` :

```
global
log /dev/log local0
maxconn 2000
daemon
```

```

defaults
    log global
    mode tcp
    option tcplog
    timeout connect 10s
    timeout client 1m
    timeout server 1m

frontend kubernetes
    bind *:6443
    default_backend kubernetes-backend

backend kubernetes-backend
    option httpchk GET /healthz
    http-check expect status 200
    server master1 192.168.56.11:6443 check
    server master2 192.168.56.12:6443 check
    server master3 192.168.56.13:6443 check

```

Redémarrer HAProxy :

```
sudo systemctl restart haproxy
```



3.3. Configurer Keepalived `/etc/keepalived/keepalived.conf`

Sur `k8s-lb1` (MASTER) :

```

vrrp_instance VI_1 {
    state MASTER
    interface enp0s8
    virtual_router_id 51
    priority 101
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass secret
    }
}

```

```
}  
virtual_ipaddress {  
    192.168.56.30  
}  
}
```

Sur `k8s-lb2` (BACKUP) :

```
vrrp_instance VI_1 {  
    state BACKUP  
    interface enp0s8  
    virtual_router_id 51  
    priority 100  
    advert_int 1  
    authentication {  
        auth_type PASS  
        auth_pass secret  
    }  
    virtual_ipaddress {  
        192.168.56.30  
    }  
}
```

Redémarrer Keepalived sur les deux :

```
sudo systemctl restart keepalived
```

4. Tester

- Pinger le VIP :

```
ping 192.168.56.30
```

- Vérifier que le port 6443 est ouvert :

```
telnet 192.168.56.30 6443
```

- Pour simuler un failover :





```
sudo systemctl stop keepalived # sur k8s-lb1
```

Documentation des Charts Helm – Joget & MySQL

Qu'est-ce que Helm ?

Helm est un **gestionnaire de packages pour Kubernetes**. Il fonctionne comme `apt` pour Ubuntu ou `yum` pour CentOS, mais pour déployer et gérer des applications dans Kubernetes.

Avantages de Helm :

-  Regroupe tous les fichiers de déploiement dans un seul package appelé **chart**
-  Permet des **mise à jour versionnées** via `helm upgrade`
-  Personnalisation facile via un fichier `values.`
-  Réutilisable, portable, et facile à partager

Structure générale d'un chart Helm

```
my-chart/  
├── Chart.      # Métadonnées du chart  
└── values.    # Paramètres configurables
```

```
└─ templates/      # Fichiers manifestes Kubernetes ( avec Go templates)
  └─ deployment.
  └─ service.
  └─ pvc.
  └─ pv.
  └─ configmap.
  └─ _helpers.tpl   # Fonctions utilitaires (optionnel)
```

Chart Helm – Joget



Chart.

```
apiVersion: v2
name: joget
description: A Helm chart for Joget
version: 0.1.0
appVersion: "8.0"
```

- Définit le nom du chart, sa version et la version de l'application Joget.



values.

```
replicaCount: 1

image:
  repository: motrabelsi10/joget-v7
  tag: latest
  pullPolicy: Always

mysql:
  host: mysql.joget.svc.cluster.local
  database: jwdb
  username: tomcat
  password: tomcat
```



```
service:
  type: NodePort
  portHttp: 8080
  portHttps: 9080
  nodePortHttp: 32325

nodeSelector:
  kubernetes.io/hostname: k8s-worker1
```

- Définit :
 - L'image Docker de Joget
 - La configuration de connexion MySQL
 - Le type de service exposé (`NodePort`)
 - La planification du pod sur `k8s-worker1`

`templates/deployment.`

Déploie Joget avec :

- Un `replica`
- Des variables d'environnement pour la DB
- Un volume monté depuis une PVC

`templates/service.`

Expose Joget avec :

- Deux ports : HTTP (8080) & HTTPS (9080)
- `NodePort` fixé à `32325` (accès depuis l'extérieur)
- Affinité de session (`ClientIP`)

`templates/pv.`

Crée un volume persistant (PV) sur le `worker` :

```
hostPath:
  path: "/mnt/data/joget"
```

Avec :

- 2Gi de stockage
- ReadWriteMany
- Node affinity sur `k8s-worker1`



templates/pvc.

Demande un volume persistant de 2Gi lié au PV précédent.



templates/serviceaccount.

Un `ClusterRoleBinding` pour permettre à Joget d'accéder à certaines ressources Kubernetes.



Commandes Helm Essentielles



1. Installer un chart

```
helm install <release-name> <chart-path> -n <namespace> --create-namesp  
ace
```



Exemple :

```
helm install joget ./joget -n joget --create-namespace  
helm install mysql ./mysql -n joget --create-namespace
```

2. Mettre à jour un chart déjà installé (upgrade)

```
helm upgrade <release-name> <chart-path> -n <namespace>
```

 Exemple :

```
helm upgrade joget ./joget -n joget  
helm upgrade mysql ./mysql -n joget
```

3. Supprimer un déploiement Helm



```
helm uninstall <release-name> -n <namespace>
```

 Exemple :

```
helm uninstall joget -n joget  
helm uninstall mysql -n joget
```

4. Lister les releases installées

```
helm list -A
```

  montre toutes les releases dans tous les namespaces.

5. Afficher les ressources installées par une release

```
helm get manifest <release-name> -n <namespace>
```

 Exemple :

```
helm get manifest joget -n joget
```

6. Tester un chart localement (dry-run)

```
helm install <release-name> <chart-path> --dry-run --debug -n <namespace>
```

 Exemple :

```
helm install joget ./joget --dry-run --debug -n joget
```

7. Afficher les valeurs par défaut d'un chart

```
helm show values <chart-path>
```

8. Personnaliser les valeurs avec un fichier

```
helm upgrade --install <release-name> <chart-path> -f my-values.yaml -n <namespace>
```