

AUTRE_Installation complètes

Ce chapitre comprend des exemples d'installation de A à Z. Ce sont en gros des copier collés des différents blocs que vous pouvez consulter dans le livre "kubernetes"

- Initialisation de mon tout premier cluster HA
- Cluster HA - Loghorn - Cilium - KubeVip

Initialisation de mon tout premier cluster HA

Architecture

Hostname	RAM	CPU	DISQUE	IP	FONCTION
k8s-master-1	4 go	2	60 SSD + 100 HDD (/opt)	192.168.1.101	etcd + control-plane
k8s-master-2	4 go	2	60 SSD + 100 HDD (/opt)	192.168.1.102	etcd + control-plane
k8s-master-3	4 go	2	60 SSD + 100 HDD (/opt)	192.168.1.103	etcd + control-plane
k8s-worker-1	16 go	4	60 SSD + 100 HDD (/opt)	192.168.1.104	
k8s-worker-2	16 go	4	60 SSD + 100 HDD (/opt)	192.168.1.105	
k8s-worker-3	16 go	4	60 SSD + 100 HDD (/opt)	192.168.1.106	

Tous les noeuds sont sur debian 13.

Plan de déploiement du cluster :

- Préparation du système
- installation des prérequis
 - conainer.io
 - kubelet
 - kubeadm
 - kubectl
- Préparation du cluster HA (KubeVip)
- Initialisation du cluster

- Configuration du cluster
- Ajout des différents noeud au cluster

Préparation du système

Commandes à réaliser sur tous les noeuds

1) Désactivation de la swap

```
swapoff -a
```

Désactivation de la swap de manière pérenne :

```
sed -i ' / swap / s/^(.*)$/#\1/g' /etc/fstab
```

2) Configuration du kernel

Chargement au démarrage des modules pour k8s :

```
cat <<EOF | tee /etc/modules-load.d/k8s.conf
overlay
br_netfilter
EOF
```

Chargement immédiat des modules pour k8s :

```
modprobe overlay
modprobe br_netfilter
```

Le module overlay permet le fonctionnement du système de fichier OverlayFS.
Le module br_netfilter permet d'appliquer les iptables à un bridge.

Configuration des paramètres réseaux :

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

Permet aux bridges de faire appel au FW iptables
Permet d'activer le routage

Application des paramètres :

```
sysctl --system
```

installation des prérequis

Commandes à réaliser sur tous les nœuds

Installation des paquets courants :

```
apt-get update  
apt-get install -y ca-certificates curl gnupg lsb-release htop jq tcpdump ethtool
```

Installation de containerd.io

```
install -m 0755 -d /etc/apt/keyrings  
  
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg  
  
chmod a+r /etc/apt/keyrings/docker.gpg  
  
echo \  
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \  
https://download.docker.com/linux/debian bookworm stable" \  
| tee /etc/apt/sources.list.d/docker.list > /dev/null  
  
apt update  
  
apt install -y containerd.io
```

Configuration de containerd :

```
containerd config default | sudo tee /etc/containerd/config.toml
```

Activation de containerd :

```
systemctl enable containerd
```

Activation de l'emploi de systemd comme gestionnaire de Cgroup :

```
sudo sed -i 's/SystemdCgroup = false/SystemdCgroup = true/g' /etc/containerd/config.toml
```

Redémarrage de Containerd :

```
systemctl restart containerd
```

Installation de kubernetes

Installation de kubelet kubeadm et kubectl :

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.32/deb/Release.key | gpg --dearmor | sudo tee
/etc/apt/keyrings/kubernetes-apt-keyring.gpg > /dev/null
sudo chmod 644 /etc/apt/keyrings/kubernetes-apt-keyring.gpg

echo "deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://pkgs.k8s.io/core:/stable:/v1.32/deb/
/" | sudo tee /etc/apt/sources.list.d/kubernetes.list

sudo apt update

sudo apt install -y kubelet kubeadm kubectl

sudo apt-mark hold kubelet kubeadm kubectl
```

Préparation du cluster HA (KubeVip)

Sur tous les noeuds master UNIQUEMENT

```
export VIP="192.168.1.100"
export INTERFACE="ens18"
```

Ici ens18 correspond à mon interface de prod et 192.168.1.100 correspondra à l'ip qui se "baladera" sur mes noeuds master selon leur disponibilité -> elle n'est pas fixée à un seul endroit.

```
KVVERSION=$(curl -sL https://api.github.com/repos/kube-vip/kube-vip/releases/latest | jq -r ".name")
echo "Version kube-vip : $KVVERSION"
```

```
sudo mkdir -p /etc/kubernetes/manifests/
# Pull de l'image kube-vip
sudo ctr image pull ghcr.io/kube-vip/kube-vip:$KVVERSION
# Génération du manifeste Static Pod
sudo ctr run --rm --net-host ghcr.io/kube-vip/kube-vip:$KVVERSION vip \
/kube-vip manifest pod \
--interface $INTERFACE \
--address $VIP \
--controlplane \
--services \
--arp \
--leaderElection | sudo tee /etc/kubernetes/manifests/kube-vip.yaml
```

Initialisation du cluster

Sur le noeuds k8s-master-1 UNIQUEMENT

Application "à la main" de l'ip vip le temps qu'elle soit gérée par K8S :

```
sudo ip addr add 192.168.1.100/32 dev ens18
```

Initialisation du cluster :

```
sudo kubeadm init \
--control-plane-endpoint "192.168.1.100:6443" \
--upload-certs \
--pod-network-cidr=10.10.0.0/16
```

```
sudo kubeadm init phase upload-certs --upload-certs
```

Configurer kubectl pour l'utilisateur courant :

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

Vérification du bon déploiement du cluster "basique" :

```
kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
etcd-k8s-master-1	1/1	Running	0	4m57s
kube-apiserver-k8s-master-1	1/1	Running	0	4m57s
kube-controller-manager-k8s-master-1	1/1	Running	0	4m57s
kube-scheduler-k8s-master-1	1/1	Running	0	4m57s
kube-vip-k8s-master-1	1/1	Running	0	4m57s

Déploiement du réseau Calico

```
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manifests/tigera-operator.yaml
```

Attendre que l'opérateur ait fini :

```
kubectl rollout status deployment tigera-operator -n tigera-operator
```

Déployer les ressources Calico :

```
kubectl create -f https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manifests/custom-resources.yaml
```

Définir le sous réseau de K8S :

```
kubectl patch installation default --type=merge -p '{"spec": {"calicoNetwork": {"ipPools": [{"cidr": "10.10.0.0/16", "encapsulation": "VXLANCrossSubnet", "natOutgoing": "Enabled", "nodeSelector": "all()"}]}}'
```

Suivi des états :

```
watch kubectl get pods -n calico-system
```

Configuration du cluster

Désactivation de l'ip VIP "manuelle" :

```
sudo ip addr del 192.168.1.100/32 dev ens18
```

Ajout de l'ip vp dans le configmap de kubeadmin :

```
kubectl edit configmap kubeadm-config -n kube-system
```

```
apiVersion: v1
data:
```

```
ClusterConfiguration: |
  apiServer: {}
  apiVersion: kubeadm.k8s.io/v1beta4
  caCertificateValidityPeriod: 87600h0m0s
  certificateValidityPeriod: 8760h0m0s
  certificatesDir: /etc/kubernetes/pki
  clusterName: kubernetes
  controllerManager: {}
--> controlPlaneEndpoint: "192.168.1.100:6443"
```

Ici on ajoute uniquement "controlPlaneEndpoint."

```
kubeadm init phase upload-certs --upload-certs
```

Application du label control-plane à k8s-master-1

```
kubectl label node k8s-master-1 node-role.kubernetes.io/control-plane=
```

Génération de la commande permettant d'ajouter des noeuds masters :

```
sudo kubeadm token create --print-join-command --certificate-key $(sudo kubeadm init phase upload-certs --
upload-certs | tail -1)
```

Génération de la commande permettant d'ajouter des noeuds worker :

```
kubeadm token create --print-join-command
```

Ajout des différents noeuds master au cluster

Commandes à exécuter sur les noeuds k8s-master-2 et k8s-master-3

```
kubeadm join 192.168.1.100:6443 --token ld3z42.3n2zlgdtoy2to2 --discovery-token-ca-cert-hash
sha256:03d97c70e97b300bb1088fa63f1005d52dcf45d813e6ec535897f2151c8a61c2 --control-plane --
certificate-key c9413180b64c612ee58713da91a4d9b77e95bee06b424aa500ebb4b16fb7769d
```

Ajout des différents noeuds worker au cluster

Commandes à exécuter sur les noeuds k8s-worker-1, k8s-worker-2 et k8s-worker-3

```
// Envoyer la commande générée par kubeadm token create --print-join-command
```

Résultat :

```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
k8s-master-1	Ready	control-plane	81m	v1.32.12
k8s-master-2	Ready	control-plane	6m13s	v1.32.12
k8s-master-3	Ready	control-plane	4m8s	v1.32.12
k8s-worker-1	Ready	<none>	2m3s	v1.32.12
k8s-worker-2	Ready	<none>	115s	v1.32.12
k8s-worker-3	Ready	<none>	105s	v1.32.12

Félicitation ! Vous venez de déployer votre tout premier cluster Kubernettes !

Déployer une solution de stockage

Installation des prérequis sur TOUS LES NOEUDS :

```
sudo apt-get update  
sudo apt-get install -y open-iscsi nfs-common  
sudo systemctl enable --now iscsid
```

Installation de Helm sur le master 1 :

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3  
bash get_helm.sh
```

Préparation et déploiement de Longhorn :

```
helm repo add longhorn https://charts.longhorn.io
helm repo update
```

Dans le fichier : /root/longhorn-values.yaml :

```
defaultSettings:
  defaultDataPath: "/opt/longhorn"
```

Déploiement de longhorn :

```
helm install longhorn longhorn/longhorn \
  --namespace longhorn-system \
  --create-namespace \
  -f /root/longhorn-values.yaml
```

Résultat :

LAST DEPLOYED: Wed Apr 15 22:49:57 2026

NAMESPACE: longhorn-system

STATUS: deployed

REVISION: 1

TEST SUITE: None

NOTES:

Longhorn is now installed on the cluster!

Please wait a few minutes for other Longhorn components such as CSI deployments, Engine Images, and Instance Managers to be initialized.

Visit our documentation at <https://longhorn.io/docs/>

Vérifications :

```
kubectl get pods -n longhorn-system
```

NAME	READY	STATUS	RESTARTS	AGE
csi-attacher-5bcb65bf95-44fv7	1/1	Running	1 (66s ago)	2m27s
csi-attacher-5bcb65bf95-jvqxj	1/1	Running	0	2m27s
csi-attacher-5bcb65bf95-pgf8h	1/1	Running	0	2m27s
csi-provisioner-5d498c6944-d65sl	1/1	Running	0	2m27s
csi-provisioner-5d498c6944-dfl96	1/1	Running	0	2m27s
csi-provisioner-5d498c6944-wvmvs	1/1	Running	0	2m27s
csi-resizer-6c6474c996-9k94v	1/1	Running	0	2m26s
csi-resizer-6c6474c996-mjtsj	1/1	Running	0	2m27s
csi-resizer-6c6474c996-t94bs	1/1	Running	0	2m26s
csi-snapshotter-5cc5fb45d9-pxz5r	1/1	Running	0	2m26s
csi-snapshotter-5cc5fb45d9-swr6x	1/1	Running	0	2m26s
csi-snapshotter-5cc5fb45d9-wsdh6	1/1	Running	0	2m26s
engine-image-ei-75a03ec3-8r6p4	1/1	Running	0	3m23s
engine-image-ei-75a03ec3-nmbdp	1/1	Running	0	3m23s
engine-image-ei-75a03ec3-tl4x2	1/1	Running	0	3m23s
engine-image-ei-75a03ec3-vcq2q	1/1	Running	0	3m23s
instance-manager-8904e58b7c0d881b70d813ddd4fe86f4	1/1	Running	0	2m54s
instance-manager-9176dd8e42515fe6146f445c7a1bc3ad	1/1	Running	0	2m53s
instance-manager-9805597396c5db2820496e3f125bc6bc	1/1	Running	0	2m53s
instance-manager-eaeed3ca06747e29c0f3aa8b9557c5c8	1/1	Running	0	2m53s
longhorn-csi-plugin-7vtx8	3/3	Running	0	2m26s
longhorn-csi-plugin-brx6d	3/3	Running	0	2m26s
longhorn-csi-plugin-d4fxc	3/3	Running	0	2m26s
longhorn-csi-plugin-tsrrg	3/3	Running	0	2m26s
longhorn-driver-deployer-746f54969f-9gmlb	1/1	Running	0	4m5s
longhorn-manager-b7q7k	2/2	Running	0	4m5s
longhorn-manager-fk6f4	2/2	Running	0	4m5s
longhorn-manager-xghbf	2/2	Running	0	4m5s
longhorn-manager-xt4xq	2/2	Running	0	4m5s
longhorn-ui-5df99fc477-7zt5s	1/1	Running	0	4m5s
longhorn-ui-5df99fc477-tzwdp	1/1	Running	0	4m5s

Accéder à l'interface graphique :

```
kubect patch svc longhorn-frontend -n longhorn-system -p '{"spec": {"type": "NodePort"}}'
```

Afficher le port :

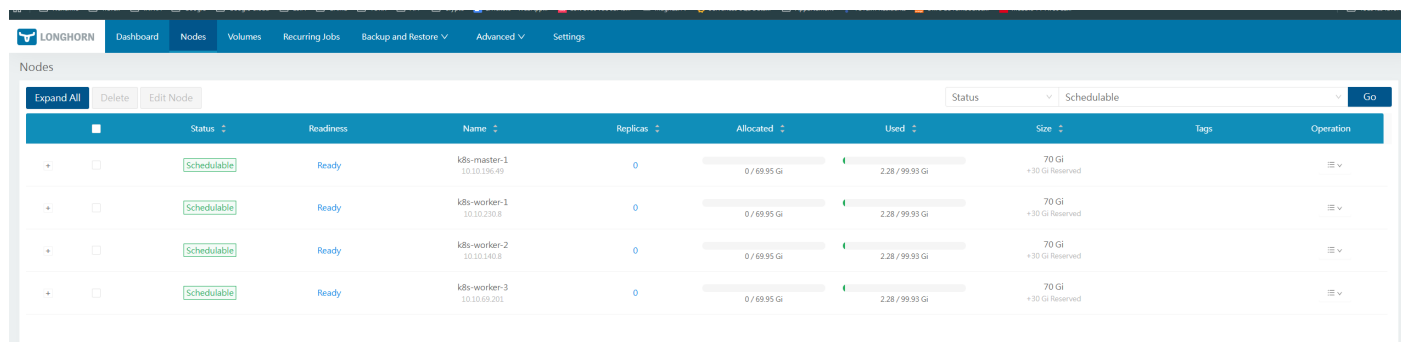
```
kubectl get svc longhorn-frontend -n longhorn-system
```

On va utiliser le port :

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
longhorn-frontend	NodePort	10.102.85.80	<none>	80:32555/TCP	6m22s

Exemple ici : <http://192.168.1.104:32555/#/dashboard>

On peut utiliser n'importe quelle IP du cluster.



	Status	Readiness	Name	Replicas	Allocated	Used	Size	Tags	Operation
+	☐	Schedulable	Ready	k8s-master-1 10.10.156.49	0	0 / 69.95 Gi	2.28 / 99.93 Gi	70 Gi + 30 Gi Reserved	⋮
+	☐	Schedulable	Ready	k8s-worker-1 10.10.230.8	0	0 / 69.95 Gi	2.28 / 99.93 Gi	70 Gi + 30 Gi Reserved	⋮
+	☐	Schedulable	Ready	k8s-worker-2 10.10.140.8	0	0 / 69.95 Gi	2.28 / 99.93 Gi	70 Gi + 30 Gi Reserved	⋮
+	☐	Schedulable	Ready	k8s-worker-3 10.10.69.201	0	0 / 69.95 Gi	2.28 / 99.93 Gi	70 Gi + 30 Gi Reserved	⋮

Tester le stockage :

Déclaration du PVC :

```
hello-pvc.yaml
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: longhorn-hello-pvc
```

```
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: longhorn # On utilise le driver Longhorn
  resources:
    requests:
      storage: 1Gi # 1 Go c'est largement assez pour un test
```

Création d'un conteneur utilisant le PVC :

```
hello-app.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: hello-longhorn
spec:
  containers:
    - name: hello-world
      image: busybox
      # On écrit la date dans /data/coucou.txt en boucle
      command: ["sh", "-c", "while true; do date >> /data/coucou.txt; echo 'Donnée écrite !'; sleep 5; done"]
      volumeMounts:
        - name: storage-volume
          mountPath: /data
  volumes:
    - name: storage-volume
      persistentVolumeClaim:
        claimName: longhorn-hello-pvc
```

Application des changements :

```
kubectl apply -f hello-pvc.yaml
kubectl apply -f hello-app.yaml
```

Côté interface graphique :

LONGHORN

DashboardNodesVolumesRecurring JobsBackup and RestoreAdvancedSettings

Volumes

Custom ColumnCreate Volume

DeleteAttachDetachCreate BackupClone Volume

NameGo

	State	Name	Size	Actual Size	Created	Data Engine	PV/PVC	Namespace	Attached To	Schedule	Last Backup At	Operation
☐	Healthy	pvc-488a0c97-ef99-4764-a775-fa4b724f2681	1 Gi	38.6 Mi	4 minutes ago	v1	Bound	default	hello-longhorn on k8s-worker-1			

Suppression du pod :

```
kubectl delete pod hello-longhorn
```

Suppression du pvc :

```
kubectl delete pvc longhorn-hello-pvc
```

Cluster HA - Loghorn - Cilium - KubeVip

Architecture

Hostname	RAM	CPU	DISQUE	IP	FONCTION
k8s-master-1	4 go	2	60 SSD	192.168.1.101	etcd + control-plane
k8s-master-2	4 go	2	60 SSD	192.168.1.102	etcd + control-plane
k8s-master-3	4 go	2	60 SSD	192.168.1.103	etcd + control-plane
k8s-worker-1	16 go	4	60 SSD + 100 Go HDD (/opt)	192.168.1.104	
k8s-worker-2	16 go	4	60 SSD + 100 Go HDD (/opt)	192.168.1.105	
k8s-worker-3	16 go	4	60 SSD + 100 Go HDD (/opt)	192.168.1.106	
VIP CLUSTER	192.168.1.100				
VIP WORKERS	192.168.1.99				

Tous les noeuds sont sur debian 13.

Résumé :

Type de cluster : **Haute Disponibilité**

Stockage : **Loghorn**

Réseau : **Cilium**

Exposition des workers : **KubeVip**

Préparation du système

CF : <https://docs.nehemiebarkia.fr/books/kubernetes/page/initialisation-de-mon-tout-premier-cluster-ha>

Installation des prérequis

CF : <https://docs.nehemiebarkia.fr/books/kubernetes/page/initialisation-de-mon-tout-premier-cluster-ha>

Préparation du cluster HA (KubeVip)

CF : <https://docs.nehemiebarkia.fr/books/kubernetes/page/initialisation-de-mon-tout-premier-cluster-ha>

Initialisation du cluster

CF : <https://docs.nehemiebarkia.fr/books/kubernetes/page/initialisation-de-mon-tout-premier-cluster-ha>

Installation du réseau Cilium

Installation de cilium-cli :

```
CILIUM_CLI_VERSION=$(curl -s https://raw.githubusercontent.com/cilium/cilium-cli/main/stable.txt)
curl -L --fail --remote-name-all https://github.com/cilium/cilium-
cli/releases/download/${CILIUM_CLI_VERSION}/cilium-linux-amd64.tar.gz
sudo tar xzvf cilium-linux-amd64.tar.gz -C /usr/local/bin
rm cilium-linux-amd64.tar.gz
```

Déploiement :

```
cilium install --set ipam.operator.clusterPoolIPv4PodCIDRList="10.10.0.0/16"
```

Attendre que le statut soit opérationnel :

```
cilium status --wait
```

Vérification de la connectivité :

```
cilium connectivity check
```

Suivi des pods :

```
watch kubectl get pods -n kube-system -l k8s-app=cilium
```

```
cilium upgrade --set l2announcements.enabled=true \  
--set externalIPs.enabled=true \  
--set kubeProxyReplacement=true
```

```
kubectl get pods -n kube-system | grep cili | grep -v opera | grep -v envoy | awk '{print $1}' | xargs  
kubectl delete po -n kube-system
```

Configuration du cluster

CF : <https://docs.nehemiebarkia.fr/books/kubernetes/page/initialisation-de-mon-tout-premier-cluster-ha>

Ajout des différents noeuds master au cluster

CF : <https://docs.nehemiebarkia.fr/books/kubernetes/page/initialisation-de-mon-tout-premier-cluster-ha>

Déployer Loghorn

CF : <https://docs.nehemiebarkia.fr/books/kubernetes/page/initialisation-de-mon-tout-premier-cluster-ha>

Ajout des labels "workers" aux noeuds workers :

```
kubectl label node k8s-worker-1 node-role.kubernetes.io/worker="true"
kubectl label node k8s-worker-2 node-role.kubernetes.io/worker="true"
kubectl label node k8s-worker-3 node-role.kubernetes.io/worker="true"
```

Déployer une VIP pour les noeuds Workers

```
cat <<EOF | kubectl apply -f -
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
  name: worker-vip-pool
spec:
  blocks:
    - start: "192.168.1.2"
      stop: "192.168.1.9"
  serviceSelector:
    matchLabels:
      expose: "true"
EOF
```

```
cat <<EOF | kubectl apply -f -
apiVersion: cilium.io/v2alpha1
kind: CiliumL2AnnouncementPolicy
metadata:
  name: worker-l2-policy
  namespace: kube-system
spec:
```

```
nodeSelector:
  matchLabels:
    node-role.kubernetes.io/worker: "true"
serviceSelector:
  matchLabels:
    expose: "true"
loadBalancerIPs: true
externalIPs: false
interfaces:
  - ens18
EOF
```

```
cat <<EOF | kubectl apply -f -
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
        - name: hello
          image: hashicorp/http-echo:latest
          args:
            - "-text=Hello from Cilium VIP [IP]"
            - "-listen=:8080"
          ports:
            - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: hello-world
  namespace: default
```

```
labels:
  expose: "true"
annotations:
  lbipam.cilium.io/ips: "192.168.1.2" # IP fixe dans le pool
spec:
  type: LoadBalancer
  selector:
    app: hello-world
  ports:
    - port: 80
      targetPort: 8080
EOF
```

