Installer et configurer Kubernetes sur Arch Linux

Table of Contents

- Installer et configurer Kubernetes sur Arch Linux
  - Prérequis
  - Installation de Docker
  - Installation des packages Kubernetes
  - Configuration de l'équilibreur de charge
  - Amorcer le cluster Kubernetes
  - Ajouter plus de nœuds au cluster
  - Accéder au cluster Kubernetes en dehors des nœuds du plan de contrôle
  - Affichage et gestion des jetons
  - Installer le tableau de bord Kubernetes

Dans cet article, on verra comment installer et configurer un cluster **Kubernetes** sur les nœuds Arch Linux.

La configuration étudiée dans le cadre de cet article sera un cluster **Kubernetes** hautement disponible. On aura trois nœuds pour le plan de contrôle, qui seront également décontaminés, afin qu'on puisse planifier des pods sur eux. Chaque nœud du cluster fonctionnera avec 2 processeurs et 4 Go de mémoire. Si nécessaire, on peut étendre le cluster par la suite en y ajoutant plus de nœuds de travail.

Prérequis

Tout d'abord, Arch Linux doit être installé.

Chaque nœud du cluster Kubernetes aura 2 processeurs et 4 Go de mémoire et pour amorcer le cluster, on utilisera l'outil **kubeadm**. Tout d'abord, il faut s'assurer que les nœuds Arch Linux ont l'ensemble de configuration de base appliqué, par ex. configurer des noms d'hôtes, définir des adresses IP statiques, configurer des serveurs DNS, des serveurs NTP, etc.

On aura besoin de **devtools** et de packages de développement de base pour construire les outils de ligne de commande **kubelet**, **kubeadm** et **kubectl**. Si on utilise un outil pour installer des packages à partir d'AUR, On peut l'utiliser à la place pour installer les outils **kubelet**, **kubeadm** et **kubectl**. Dans cet article, on va construire ces packages et utiliser pacman pour les installer, afin de garder les choses simples et directes.

```
sudo pacman -S devtools base-devel
```

Lors de l'amorçage du cluster Kubernetes, l'un des contrôles en amont effectués par Kubernetes

consiste à voir si on a un swap actif. S'il trouve une partition de swap active, il échouera et s'arrêtera là. Afin de réussir les contrôles en amont, il faut désactiver l'échange sur chaque nœud du cluster Kubernetes.

```
sudo swapoff -a
```

Il faut également de mettre à jour /etc/fstab, afin que le swap ne soit pas monté pendant le démarrage.

Si on n'a vraiment pas le choix et qu'on doit utiliser swap, bien que non recommandé, On peut ajouter l'indicateur suivant au fichier /etc/default/kubernetes.

`--fail-swap-on=false`

Installation de Docker

Installer et configurer Docker en tant que Container Runtime pour Kubernetes. Si on a des exigences particulières pour les images et conteneurs Docker, par ex. on veut qu'ils résident sur un volume LVM, il est maintenant temps de créer ce volume et de le monter sous /var/lib/docker.

Installer Docker :

```
sudo pacman -S docker
```

Si on souhaite que des utilisateurs puissent interagir avec Docker, On peut également les ajouter au groupe docker, par ex.

```
sudo usermod -a -G docker <nom d'utilisateur>
```

Configurer Docker pour utiliser le pilote de stockage overlay2 et systemd comme pilote de groupe de contrôle. Modifier le fichier `/etc/docker/daemon.json`.

```
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
```

Créer une unité d'accueil systemd et définir l'option `TasksMax=infinity.`

```
sudo systemctl edit docker
```

Ajouter le contenu suivant.

```
[Service]
TasksMax=infinity
```

Activer et démarrer le service Docker.

```
sudo systemctl daemon-reload
sudo systemctl enable docker
sudo systemctl start docker
```

Vérifier que Docker a été installé et fonctionne correctement.

```
sudo docker info
```

Installation des packages Kubernetes

On peut maintenant installer les packages **kubectl**, **kubelet** et **kubeadm**, en utilisant le script suivant pour récupérer les packages d'AUR et les installer.

```
#!/usr/bin/env sh

OLD_PWD="${OLDPWD}"
PACKAGE_DIR="${HOME}/kube-packages/$( date +%Y-%m-%d )"

mkdir -p "${PACKAGE_DIR}"

for p in kubectl-bin kubelet-bin kubeadm-bin cni-plugins-bin; do
    cd "${PACKAGE_DIR}"

    echo "> Fetching ${p} ..."
    curl -O https://aur.archlinux.org/cgit/aur.git/snapshot/${p}.tar.gz
    tar zxvf ${p}.tar.gz

    cd "${PACKAGE_DIR}/${p}"
    echo "> Building ${p} ..."
    makepkg

    echo "> Installing ${p} ..."
    sudo pacman --noconfirm -U *.tar.zst
done

cd "${OLD_PWD}"
```

```
echo "> Done"
```

Enregistrer le script ci-dessus quelque part (par exemple ~/install-kube-packages.sh) et l'exécuter.

```
chmod +x install-kube-packages.sh
./install-kube-packages.sh
```

Vérifier que **kubectl**, **kubelet** et **kubeadm** ont été correctement installés. Il est important de vérifier que **kubectl**, **kubelet** et **kubeadm** aient tous la même version.

```
$ kubelet --version
Kubernetes v1.17.4
```

```
$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.4",
GitCommit:"8d8aa39598534325ad77120c120a22b3a990b5ea", GitTreeState:"clean",
BuildDate:"2020-03-12T21:01:11Z", GoVersion:"go1.13.8", Compiler:"gc",
Platform:"linux/amd64"}
```

```
$ kubectl version
Client Version: version.Info{Major:"1", Minor:"17", GitVersion:"v1.17.4",
GitCommit:"8d8aa39598534325ad77120c120a22b3a990b5ea", GitTreeState:"clean",
BuildDate:"2020-03-12T21:03:42Z", GoVersion:"go1.13.8", Compiler:"gc",
Platform:"linux/amd64"}
The connection to the server localhost:8080 was refused - did you specify
the right host or port?
```

Il est normal pour l'instant de voir la connexion refusée comme le montre la dernière ligne de la sortie de la version **kubectl**, car on n'a pas encore les services opérationnels. Ce qu'il est important de noter ici, c'est que tous les outils sont à la même version que la sortie ci-dessus.

Enfin, installer quelques packages supplémentaires requis par **kubeadm**, qui sont utilisés dans le cadre du processus d'amorçage.

```
sudo pacman -S ethtool ebtables socat conntrack-tools
```

Configuration de l'équilibreur de charge

Étant donné qu'on créer un plan de contrôle hautement disponible pour le cluster Kubernetes, on a besoin d'équilibrer la charge des requêtes vers les nœuds du plan de contrôle du cluster. Pour cela, on utilisera un équilibreur de charge HAProxy.

Le service HAProxy qu'on mettra en place dans cette partie de la documentation s'exécute en fait sur un système distinct, en dehors du cluster Kubernetes.

```
sudo pacman -S haproxy
```

Ouvrir /etc/haproxy/haproxy.cfg dans un éditeur.

```
#---------------------------------------------------------------------
# Example configuration.  See the full configuration manual online.
#
#   http://www.haproxy.org/download/1.7/doc/configuration.txt
#
#---------------------------------------------------------------------

global
    maxconn     20000
    log         127.0.0.1 local0
    user        haproxy
    chroot      /usr/share/haproxy
    pidfile     /run/haproxy.pid
    daemon

frontend k8s
    bind            :6443
    mode            tcp
    option          tcplog
    log             global
    option          dontlognull
    timeout client  30s
    default_backend k8s-control-plane

backend k8s-control-plane
    mode            tcp
    balance         roundrobin
    timeout         connect 30s
    timeout         server  30s
    server          k8s-node01 192.168.88.230:6443 check
    server          k8s-node02 192.168.88.231:6443 check
    server          k8s-node03 192.168.88.232:6443 check
```

Spécifier les paramètres corrects pour les nœuds Kubernetes dans le fichier de configuration ci-dessus. Enfin, activer et démarrer le service HAProxy.

```
sudo systemctl enable haproxy
sudo systemctl start haproxy
```

Si on vérifie les journaux du service, on verra des avertissements indiquant que les nœuds principaux du plan de contrôle Kubernetes ne sont pas disponibles, et c'est correct pour le moment, car on ne les a pas encore amorcés.

Amorcer le cluster Kubernetes

Afin d'amorcer le cluster, on utilisera l'outil **kubeadm**. On utilisera **Calico** comme interface réseau de conteneurs (CNI), qui utilise par défaut le CIDR 192.168.0.0/16 lors de la configuration du cluster à l'aide de **kubeadm**.

Si on préfére un autre CNI, il faut spécifier le CIDR correct pour le réseau de pods.

S'assurer que le service kubelet est activé.

```
sudo systemctl enable kubelet
```

Il est temps d'amorcer le cluster Kubernetes.

```
sudo kubeadm init --pod-network-cidr=192.168.0.0/16 --control-plane-endpoint
<k8s>:6443 --upload-certs
```

Il faut remplacer <k8s> dans la commande ci-dessus par l'adresse IP ou le nom DNS de l'équilibreur de charge qu'on a configuré à l'étape précédente. Une fois le cluster initialisé, on doit voir une sortie similaire.

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular
user:

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

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
  https://kubernetes.io/docs/concepts/cluster-administration/addons/

You can now join any number of the control-plane node running the following
command on each as root:

  kubeadm join <k8s>:6443 --token ccgnsn.1kwq7gnvm1xbiwvd \
    --discovery-token-ca-cert-hash
sha256:4da745ded3fb48f8894c247c3cdb123e4230c2f5d01bf37c1d1e3a4838c95cf5 \
    --control-plane --certificate-key
57ed53d8968473217fcb6254caba1b5ecc63794f9ae6887a2c8032c8a6a5cb03

Please note that the certificate-key gives access to cluster sensitive data,
keep it secret!
As a safeguard, uploaded-certs will be deleted in two hours; If necessary,
you can use
```

```
"kubeadm init phase upload-certs --upload-certs" to reload certs afterward.

Then you can join any number of worker nodes by running the following on
each as root:

kubeadm join <k8s>:6443 --token ccgnsn.1kwq7gnvm1xbiwvd \
    --discovery-token-ca-cert-hash
sha256:4da745ded3fb48f8894c247c3cdb123e4230c2f5d01bf37c1d1e3a4838c95cf5
```

Exécuter les commandes telles qu'elles sont imprimées dans la sortie ci-dessus.

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

On peut maintenant déployer un réseau de pods sur la grappe.

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

Une fois terminé, on doit voir une sortie similaire à celle ci-dessous.

```
configmap/calico-config created
customresourcedefinition.apiextensions.k8s.io/felixconfigurations.crd.projec
tcalico.org created
customresourcedefinition.apiextensions.k8s.io/ipamblocks.crd.projectcalico.o
rg created
customresourcedefinition.apiextensions.k8s.io/blockaffinities.crd.projectcal
ico.org created
customresourcedefinition.apiextensions.k8s.io/ipamhandles.crd.projectcalico.
org created
customresourcedefinition.apiextensions.k8s.io/ipamconfigs.crd.projectcalico.
org created
customresourcedefinition.apiextensions.k8s.io/bgppeers.crd.projectcalico.org
created
customresourcedefinition.apiextensions.k8s.io/bgpconfigurations.crd.projectc
alico.org created
customresourcedefinition.apiextensions.k8s.io/ippools.crd.projectcalico.org
created
customresourcedefinition.apiextensions.k8s.io/hostendpoints.crd.projectcalic
o.org created
customresourcedefinition.apiextensions.k8s.io/clusterinformations.crd.projec
tcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworkpolicies.crd.proj
ectcalico.org created
customresourcedefinition.apiextensions.k8s.io/globalnetworksets.crd.projectc
alico.org created
customresourcedefinition.apiextensions.k8s.io/networkpolicies.crd.projectcal
ico.org created
```

```
customresourcedefinition.apiextensions.k8s.io/networksets.crd.projectcalico.
org created
clusterrole.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrolebinding.rbac.authorization.k8s.io/calico-kube-controllers created
clusterrole.rbac.authorization.k8s.io/calico-node created
clusterrolebinding.rbac.authorization.k8s.io/calico-node created
daemonset.apps/calico-node created
serviceaccount/calico-node created
deployment.apps/calico-kube-controllers created
serviceaccount/calico-kube-controllers created
```

Supprimer les taches sur le nœud, afin de pouvoir planifier des pods dessus. Pour tout nœud de plan de contrôle nouvellement ajouté pour lequel on veut pouvoir planifier des pods dessus, exécuter simplement la commande là aussi. Si on a beaucoup de nœuds de travail et qu'on souhaite laisser les nœuds du plan de contrôle seuls, ignorer la commande suivante.

```
kubectl taint nodes --all node-role.kubernetes.io/master-
```

Si on vérifie l'état de votre cluster Kubernetes, on doit voir un seul nœud maître.

```
$ kubectl get nodes -o wide
NAME         STATUS   ROLES    AGE   VERSION   INTERNAL-IP      EXTERNAL-IP
OS-IMAGE     KERNEL-VERSION   CONTAINER-RUNTIME
k8s-node01   Ready    master   16m   v1.17.4   192.168.88.230   <none>
Arch Linux   5.5.10-arch1-1   docker://19.3.8
```

Et ce sont les pods qu'on a jusqu'à présent dans le cluster.

```
$ kubectl get pods --all-namespaces=true
NAMESPACE     NAME                                       READY   STATUS
RESTARTS   AGE
kube-system   calico-kube-controllers-5b644bc49c-dph5v   1/1     Running   0
15m
kube-system   calico-node-h2qdx                          1/1     Running   0
15m
kube-system   coredns-6955765f44-2zvgw                   1/1     Running   0
17m
kube-system   coredns-6955765f44-5p6ck                   1/1     Running   0
17m
kube-system   etcd-k8s-node01                            1/1     Running   0
17m
kube-system   kube-apiserver-k8s-node01                  1/1     Running   0
17m
kube-system   kube-controller-manager-k8s-node01         1/1     Running   0
17m
kube-system   kube-proxy-kmm5k                           1/1     Running   0
17m
kube-system   kube-scheduler-k8s-node01                  1/1     Running   0
```

```
17m
```

Dans la section suivante de ce document, on verra comment ajouter plus de nœuds au cluster Kubernetes.

Ajouter plus de nœuds au cluster

Une fois le cluster Kubernetes initialisé, en utilisant la sortie fournie par la commande kubeadm init, on peut utiliser cette commande afin d'ajouter un nouveau nœud de plan de contrôle.

```
sudo kubeadm join <k8s>:6443 --token ccgnsn.1kwq7gnvm1xbiwvd \
        --discovery-token-ca-cert-hash
sha256:4da745ded3fb48f8894c247c3cdb123e4230c2f5d01bf37c1d1e3a4838c95cf5 \
        --control-plane --certificate-key
57ed53d8968473217fcb6254caba1b5ecc63794f9ae6887a2c8032c8a6a5cb03
```

Si on veut pouvoir planifier des pods sur les nœuds du plan de contrôle, on doit les décontaminer, par ex.

```
kubectl taint nodes --all node-role.kubernetes.io/master-
```

Afin d'ajouter des nœuds de travail au cluster, il faut utiliser cette commande à la place (à nouveau à partir de la sortie de la commande kubeadm init à l'étape précédente).

```
sudo kubeadm join <k8s>:6443 --token ccgnsn.1kwq7gnvm1xbiwvd \
        --discovery-token-ca-cert-hash
sha256:4da745ded3fb48f8894c247c3cdb123e4230c2f5d01bf37c1d1e3a4838c95cf5
```

Accéder au cluster Kubernetes en dehors des nœuds du plan de contrôle

Afin d'accéder au cluster Kubernetes depuis l'extérieur des nœuds du plan de contrôle, On peut copier le fichier de configuration admin sur le système local, par ex.

```
scp k8s-node01.example.org:/etc/kubernetes/admin.conf ~/.kube/config-lab-
admin.yml
```

Utiliser l'option `--kubeconfig` pour se connecter au cluster Kubernetes, par ex.

```
kubectl --kubeconfig ~/.kube/config-lab-admin.yml get nodes -o wide
```

Affichage et gestion des jetons

Afin d'obtenir les jetons qu'on peut utiliser pour rejoindre de nouveaux nœuds dans le cluster, on peut utiliser la commande `kubeadm token list`, par ex.

```
$ kubeadm token list
TOKEN                     TTL         EXPIRES                     USAGES
DESCRIPTION                                                EXTRA GROUPS
ccgnsn.1kwq7gnvm1xbiwvd   22h         2020-03-11T13:59:48+02:00
authentication,signing   The default bootstrap token generated by 'kubeadm
init'.   system:bootstrappers:kubeadm:default-node-token
```

Si on n'a pas la valeur de l'indicateur `--discovery-token-ca-cert-hash` lorsqu'on rejoin un nœud, on peut la récupérer en exécutant la commande suivante sur le nœud du plan de contrôle.

```
openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -
outform der 2>/dev/null | \
   openssl dgst -sha256 -hex | sed 's/^.* //'
```

Lorsque le jeton expire, On peut en créer un nouveau en utilisant la commande suivante, qui imprimera également la commande de jointure qu'on peut utiliser.

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

Installer le tableau de bord Kubernetes

On peut éventuellement installer le tableau de bord, qui est une interface utilisateur Web pour un cluster Kubernetes.

Déployer le tableau de bord en appliquant le manifeste suivant.

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0-beta8/aio/depl
oy/recommended.yaml
```

Créer le manifeste pour le compte de service admin-user. Voici à quoi ressemble le manifeste `dashboard-adminuser.yml`:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
```

Créer le manifeste pour la liaison de rôle de cluster. Voici à quoi ressemble le manifeste `dashboard-cluster-rolebinding.yaml`:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
  - kind: ServiceAccount
    name: admin-user
    namespace: kubernetes-dashboard
```

Appliquer les manifestes.

```
kubectl apply -f dashboard-adminuser.yml
kubectl apply -f dashboard-cluster-rolebinding.yml
```

Obtenir le jeton qu'on peut utiliser pour s'authentifier auprès du service de tableau de bord. Tout d'abord, répertorier les secrets dans l'espace de noms `kubernetes-dashboard` à l'aide de la commande suivante.

```
$ kubectl -n kubernetes-dashboard get secret
NAME                               TYPE
DATA   AGE
admin-user-token-zw6sr             kubernetes.io/service-account-token   3
10m
default-token-6g4tt                kubernetes.io/service-account-token   3
14m
kubernetes-dashboard-certs         Opaque                                0
14m
kubernetes-dashboard-csrf          Opaque                                1
14m
kubernetes-dashboard-key-holder    Opaque                                2
14m
kubernetes-dashboard-token-nbxl7   kubernetes.io/service-account-token   3
14m
```

Afficher les détails du jeton.

```
$ kubectl -n kubernetes-dashboard describe secret admin-user-token-zw6sr
Name:         admin-user-token-zw6sr
Namespace:    kubernetes-dashboard
Labels:       <none>
```

```
Annotations:  kubernetes.io/service-account.name: admin-user
              kubernetes.io/service-account.uid: d8ec3893-7f64-47a2-978a-
c06d0d876b7e

Type:  kubernetes.io/service-account-token

Data
====
token:
eyJhbGciOiJSUzI1NiIsImtpZCI6IjF0eTFoM3dkaVJTcUxNQXM1Z21DWllDY0EzeE54ajVFejA0
LWdVVmVKYzQifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlc
y5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1Y
mVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi11c2VyLXRva2VuL
Xp3NnNyIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZ
SI6ImFkbWluLXVzZXIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb
3VudC51aWQiOiJkOGVjMzg5My03ZjY0LTQ3YTItOTc4YS1jMDZkMGQ4NzZiN2UiLCJzdWIiOiJze
XN0ZW06c2VydmljZWFjY291bnQ6a3ViZXJuZXRlcy1kYXNoYm9hcmQ6YWRtaW4tdXNlciJ9.SaWj
CqjkPqvB-
bPWcGO0Q3DbG8YAYYvCoNjr7E48vcLoY5IS4cp_mQAu3Nf1g1rzbxfK66chgHnzB5IqhAIvK0JlD
L-6hn1Yuy5IGAa4qL0-OHbkU4IXQQSfjsNJcSh7MnY9Z8j2-
p2ejvm9us83u1y_ZUWkFhe28ej0eNFrLJ_dtaEP-4jua5SrGi3E3_MRGRIdGmAJqY5ZyLmlIfHB9
qEefToq8EepZTWmcF7MitcC6PZLyd1AgblYuDCCFc7a3rqiZMlzOqtzMH1hPVDMJkuxZmSmG_T_Q
T5KqEDWhC3LBR_LktK9oR_akS2UHeAyklA0RoJK-GPBCHyQh4T0Ug
ca.crt:     1025 bytes
namespace:  20 bytes
```

On a besoin du jeton qui est affiché dans la section Data. Il s'agit du jeton Web JSON, qu'on utilisera pour s'authentifier auprès du service Dashboard.

Et enfin utiliser la commande suivante, qui permettra d'accéder au service de tableau de bord. On doit utiliser la commande `proxy kubectl`, car le service de tableau de bord n'est pas exposé en dehors du cluster Kubernetes (il est configuré par défaut en tant que service ClusterIP).

Depuis le système local.

```
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

Ouvrir un navigateur à l'URL suivante et s'authentifier à l'aide du jeton JWT qu'on a récupéré précédemment.

```
http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:
kubernetes-dashboard:/proxy/#/login
```