Kubernetes - Minikube

Dans ce Lab vous verrez comment installer Minikube, qui est un outil qui fait tourner un cluster Kubernetes à un nœud unique en bare-metal ou en VM sur votre machine.

Pré-requis

Installer Docker

Si vous n'avez pas déjà Docker installé, installez-le maintenant pour votre système d'exploitation:

```
[root@master ~]# yum -y install docker
[root@master ~]# systemctl enable docker --now
```

Installer kubectl

L'outil en ligne de commande de kubernetes, kubectl, vous permet d'exécuter des commandes dans les clusters Kubernetes. Vous pouvez utiliser kubectl pour déployer des applications, inspecter et gérer les ressources du cluster et consulter les logs.

Installez kubectl en suivant les instructions suivantes :

1. Téléchargez la dernière release avec la commande :

```
[root@master ~]# curl -LO https://storage.googleapis.com/kubernetes-
release/release/$(curl -s https://storage.googleapis.com/kubernetes-
release/release/stable.txt)/bin/linux/amd64/kubectl
```

2. Rendez le binaire kubectl exécutable.

```
[root@master ~]# chmod +x ./kubectl
```

3. Déplacez le binaire dans votre PATH.

```
[root@master ~]# mv ./kubectl /usr/local/bin/kubectl
```

4. Testez pour vous assurer que la version que vous avez installée est à jour:

```
[root@master ~]# kubectl version --client -o json
{
  "clientVersion": {
    "major": "1",
    "minor": "18",
    "gitVersion": "v1.18.0",
    "gitCommit": "9e991415386e4cf155a24b1da15becaa390438d8",
    "gitTreeState": "clean",
    "buildDate": "2020-03-25T14:58:59Z",
    "goVersion": "go1.13.8",
    "compiler": "gc",
    "platform": "linux/amd64"
  }
}
```

Installer Minikube

Vous pouvez installer Minikube sur Linux en téléchargeant un package:

```
[root@master01 ~]# curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-1.9.0-
0.x86_64.rpm && rpm -ivh minikube-1.9.0-0.x86_64.rpm

[root@master01 ~]# yum -y install conntrack
```

Création du cluster Kubernetes avec Minikube

À présent, vous allez démarrer Minikube afin de créer votre premier cluster Kubernetes.

Minikube prend en charge les pilotes suivants:

- virtualbox
- vmwarefusion
- kvm2 (installation du pilote)
- hyperkit (installation du pilote)
- hyperv (installation du pilote) Notez que l'adresse IP ci-dessous est dynamique et peut changer. Il peut être récupéré avec minikube ip.
- vmware (installation du pilote) (VMware unified driver)
- none (Exécute les composants Kubernetes sur l'hôte et non sur une machine virtuelle. Il n'est pas recommandé d'exécuter le pilote none sur des postes de travail personnels. L'utilisation de ce pilote nécessite Docker et un environnement Linux)

Si vous souhaitez utiliser votre machine physique en tant que nœud, alors vous devez utiliser l'option « --driver » de la commande *minikube start* avec la valeur ***none***. Cette option exécutera les composants Kubernetes sur votre machine hôte et non sur une machine virtuelle à condition de posséder le moteur Docker sur votre machine hôte Linux. Ce qui nous donne la commande suivante : *minikube start --vm-driver=none*

La commande que nous allons exécuter va créer et exécuter un cluster Kubernetes à un seul nœud sur votre machine locale, elle configurera également notre installation de kubectl de manière à communiquer avec notre cluster.

```
[root@master01 ~]# minikube start --vm-driver=none --apiserver-ips
172.20.10.100 --apiserver-name master01
😄  minikube v1.9.0 on Centos 7.7.1908
✨  Using the none driver based on user configuration
🤹  Running on localhost (CPUs=4, Memory=3931MB, Disk=17394MB) ...
ℹ️  OS release is CentOS Linux 7 (Core)
🐳  Preparing Kubernetes v1.18.0 on Docker 1.13.1 ...
    > kubectl.sha256: 65 B / 65 B [----------] 100.00% ? p/s 0s
    > kubelet.sha256: 65 B / 65 B [----------] 100.00% ? p/s 0s
    > kubeadm.sha256: 65 B / 65 B [----------] 100.00% ? p/s 0s
    > kubeadm: 37.96 MiB / 37.96 MiB [-------] 100.00% 370.62 KiB p/s 1m45s
    > kubectl: 41.98 MiB / 41.98 MiB [-------] 100.00% 237.12 KiB p/s 3m1s
    > kubelet: 108.01 MiB / 108.01 MiB [-----] 100.00% 389.42 KiB p/s 4m44s

🌟  Enabling addons: default-storageclass, storage-provisioner
🤹  Configuring local host environment ...
```

```
❗ The 'none' driver provides limited isolation and may reduce system
security and reliability.
❗ For more information, see:
👉 https://minikube.sigs.k8s.io/docs/reference/drivers/none/

❗ kubectl and minikube configuration will be stored in /root
❗ To use kubectl or minikube commands as your own user, you may need to
relocate them. For example, to overwrite your own settings, run:

    ▪ sudo mv /root/.kube /root/.minikube $HOME
    ▪ sudo chown -R $USER $HOME/.kube $HOME/.minikube

💡 This can also be done automatically by setting the env var
CHANGE_MINIKUBE_NONE_USER=true
E0329 18:45:20.122962    3512 kubeadm.go:346] Overriding stale ClientConfig
host https://master01:8443 with https://172.20.10.100:8443
🏄 Done! kubectl is now configured to use "minikube"
```

Vous pouvez retrouver la configuration complète par défaut de Minikube dans le fichier suivant :

```
[root@master01 ~]# cat ~/.minikube/machines/minikube/config.json
```

Ou avec les commandes suivantes :

```
[root@master01 ~]# minikube config view memory && minikube config view cpu
```

Même si les commandes précédentes n'indiquent rien, le CPU du cluster Minikube par défaut est 2 et la mémoire par défaut du cluster Minikube est 2048M

Vous pouvez créer un cluster Kubernetes contenant un seul nœud avec deux fois plus puissance que la configuration par défaut. Ainsi la commande serait :

minikube start --memory=4096 --cpus=4 disk-size=40g

Pour obtenir les noms des champs configurables de notre nœud Minikube, nous exécuterons alors la commande suivante :

```
[root@master01 ~]# minikube config -h
Configurable fields:

 * driver
 * vm-driver
 * container-runtime
 * feature-gates
 * v
 * cpus
 * disk-size
 * host-only-cidr
 * memory
...
 * cache
 * embed-certs
* native-ssh
```

Manipulation du cluster Kubernetes avec Minikube

Commençons par vérifier l'état de notre cluster Kubernetes, comme suit :

```
[root@master01 ~]# minikube status

host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

On va utiliser l'outil kubectl afin de récupérer la liste des nœuds de notre cluster Kubernetes :

```
[root@master01 ~]# kubectl get nodes
NAME       STATUS   ROLES    AGE   VERSION
master01   Ready    master   25m   v1.18.0
```

Ainsi que les Pods en cours d'exécution sur Kubernetes :

```
[root@master01 ~]# kubectl get pods -A
NAMESPACE     NAME                                READY   STATUS    RESTARTS   AGE
kube-system   coredns-66bff467f8-fpkxn            1/1     Running   0          7m13s
kube-system   coredns-66bff467f8-z5lhg            1/1     Running   0          7m13s
kube-system   etcd-master01                       1/1     Running   0          7m19s
kube-system   kube-apiserver-master01             1/1     Running   0          7m19s
kube-system   kube-controller-manager-master01    1/1     Running   0          7m19s
kube-system   kube-proxy-lvcbb                    1/1     Running   0          7m13s
kube-system   kube-scheduler-master01             1/1     Running   0          7m19s
kube-system   storage-provisioner                 1/1     Running   0          7m18s
```

Nous pouvons aussi récupérer des informations sur notre cluster Kubernetes :

```
[root@master01 ~]# kubectl cluster-info
Kubernetes master is running at https://172.20.10.100:8443
KubeDNS is running at https://172.20.10.100:8443/api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info
dump'.
```

Si jamais, vous rencontrez quelques soucis avec votre cluster Kubernetes, n'hésitez alors pas à fouiller dans les logs de Minikube afin de connaître la source du problème :

```
[root@master01 ~]# minikube logs
```

Cette commande vient avec trois options qui peuvent vous être bien utiles :

-f ou --follow : suivre en permanence les logs de Minikube (idem tail -f sous Linux)

-n ou --length : nombre de lignes à afficher (50 par défaut)

--problems : afficher uniquement les logs qui pointent vers des problèmes connus

De la même façon, vous pouvez aussi vérifier si votre minikube est à jour de la façon suivante :

```
[root@master01 ~]# minikube update-check
CurrentVersion: v1.9.0
```

```
LatestVersion: v1.9.0
```

Tableau de bord (Dashboard)

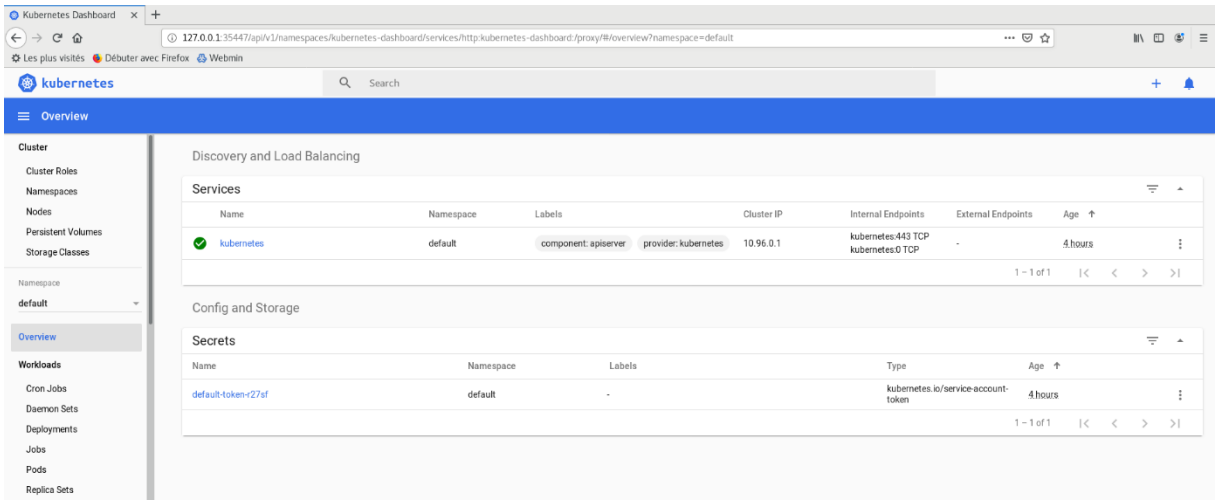
Le tableau de bord (Dashboard) est une interface web pour Kubernetes. Vous pouvez utiliser ce tableau de bord pour déployer des applications conteneurisées dans un cluster Kubernetes, dépanner votre application conteneurisée et gérer les ressources du cluster. Vous pouvez utiliser le tableau de bord pour obtenir une vue d'ensemble des applications en cours d'exécution dans votre cluster, ainsi que pour créer ou modifier des ressources Kubernetes individuelles. (comme des Deployments, Jobs, DaemonSets, etc). Par exemple, vous pouvez redimensionner un Deployment, lancer une mise à jour progressive, recréer un pod ou déployez de nouvelles applications à l'aide d'un assistant de déploiement.

Le tableau de bord fournit également des informations sur l'état des ressources Kubernetes de votre cluster et sur les erreurs éventuelles.

L'interface utilisateur du tableau de bord n'est pas déployée par défaut. Pour le déployer, exécutez la commande suivante **dans un autre terminal** :

```
[root@master01 ~]# minikube dashboard
🔌  Enabling dashboard ...
🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
http://127.0.0.1:35447/api/v1/namespaces/kubernetes-
dashboard/services/http:kubernetes-dashboard:/proxy/
^C
```

Ouvrez l'URL affichée dans votre navigateur :



Vous pouvez facilement supprimer votre cluster Minikube, à l'aide de la commande suivante :

```
[root@master01 ~]# minikube delete
```

Creation d'un Pod

Déployons une application NGINX sur Minikube dont voici les étapes:

1. Créez un pod:

```
[root@master01 ~]# kubectl run ej-nginx --image=nginx:latest --port=80
pod/ej-nginx created

[root@master01 ~]# kubectl get pods -o wide
NAME                  READY   STATUS    RESTARTS   AGE    IP           NODE       NOMINATED
ej-nginx              1/1     Running   0          4m8s   172.17.0.7   master01   <none>
```

En arrière plan l'image niginx est en cours de téléchargement

2. Vous pouvez déjà vérifier que le Pod fonctionne et que donc Nginx présente sa page par défaut :

```
[root@master01 ~]# curl 172.17.0.7:80
<!DOCTYPE html>
<html>
…
<h1>Welcome to nginx!</h1>
…
</html>
```

3. Créez un service:

Sauf que cette adresse (172.17.0.7, celle du pod) ne sera accessible que depuis votre nœud *master01*, il faut donc exposer le Pod sur l'adresse externe de l'hôte (--type=NodePort).

```
[root@master01 ~]# kubectl expose pod ej-nginx --type=NodePort
service/ej-nginx exposed
```

4. Vérifiez l'état de votre service:

```
[root@master01 ~]# kubectl get services -o wide
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP  PORT(S)         AGE   SELECTOR
ej-nginx     NodePort    10.105.159.112   <none>       80:30045/TCP    3m9s  run=ej-nginx
kubernetes   ClusterIP   10.96.0.1        <none>       443/TCP         4h6m  <none>
```

5. Voyons sur quelle adresse et quel port Minikube exécute ej-nginx:

```
[root@master01 ~]# minikube service ej-nginx --url
http://172.20.10.100:30045
```

Avec cette adresse vous pouvez accéder à Nginx depuis l'extérieur de votre master01.

Exposition des pods au cluster

Nous l'avons fait dans l'exemple précédent avec un POD, mais faisons-le encore une fois avec un *Deployment* dans un fichier de spécifications yml et concentrons-nous sur la perspective du réseautage.

Reprenons la commande de création du pod précédent et réexécutons-la (sans vraiment l'exécuter) avec les options --dry-run et --output yaml :

```
[root@master01 ~]# kubectl run ej-nginx --image=nginx:latest --port=80 \
--dry-run --output yaml
W0330 18:02:39.154712    5615 helpers.go:535] --dry-run is deprecated and
can be replaced with --dry-run=client.
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: ej-nginx
  name: ej-nginx
spec:
  containers:
  - image: nginx:latest
    name: ej-nginx
    ports:
    - containerPort: 80
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Notez qu'il s'agit bien d'un pod avec un conteneur nginx associé au port 80.

Créez maintenant un Deployment nginx et notez qu'il n'y a aucune spécification de port de conteneur donc ce sera 80 par défaut :

```
[root@master01 ~]# kubectl create deployment nginx --image=nginx --dry-run
--output yaml
W0330 18:06:32.555846    5697 helpers.go:535] --dry-run is deprecated and
can be replaced with --dry-run=client.
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: nginx
  name: nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx
        resources: {}
status: {}
```

Essayons maintenant de spécifier ce port lors de la création du Deployment directement dans le fichier yaml.

```
[root@master01 ~]# vim run-my-nginx.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: xataz/nginx:mainline
        ports:
        - containerPort: 8080
```

Cela le rend accessible à partir de n'importe quel nœud de votre cluster. Vérifiez les nœuds sur lesquels le pod s'exécute:

```
[root@master01 ~]# kubectl apply -f ./run-my-nginx.yaml
deployment.apps/my-nginx created

[root@master01 ~]# kubectl get pods -l run=my-nginx -o wide
NAME                        READY   STATUS    RESTARTS   AGE   IP           NODE
my-nginx-6676fc446-b4zll    1/1     Running   0          74m   172.17.0.4   master01
my-nginx-6676fc446-r75lr    1/1     Running   0          74m   172.17.0.2   master01
```

Création d'un service

Nous avons donc des pods exécutant nginx dans un espace d'adressage plat à l'échelle du cluster. En théorie, vous pourriez parler directement à ces modules, mais que se passe-t-il lorsqu'un nœud meurt? Les pods meurent avec, et le déploiement en créera de nouveaux, avec différentes IP. C'est le problème qu'un service résout.

Vous pouvez créer un service pour vos 2 répliques nginx avec kubectl expose:

```
[root@master01 ~]# kubectl expose deployment/my-nginx --port=80 --target-
port=8080
service/my-nginx exposed
```

Cette commande créera un service qui cible le port TCP 80 sur n'importe quel pod avec l'étiquette *my-nginx*, et l'exposera sur un port de service abstrait (***target-port***: est le port sur lequel le conteneur accepte le trafic, ***port*** : est le port de service abstrait, qui peut être n'importe quel autre port que les autres pods utilisent pour accéder au Service). Affichez l'objet API de service pour afficher la liste des champs pris en charge dans la définition de service. Vérifiez votre service:

```
[root@master01 ~]# kubectl get svc my-nginx
NAME       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
```

```
my-nginx    ClusterIP    10.105.160.18    <none>        80/TCP     71m
```

```
[root@master01 ~]# curl 10.105.160.18

    <!DOCTYPE html>
    <head><title>Alpine Docker and Nginx</title></head>
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-b4zll</h1>
    <p>
        Nginx running inside a 16MB Alpine Docker Container.<br>
    </p>
```

Remarque:

Lors de la création du service ej-nginx de l'exercice précédent on avait ajouté une option pour mapper l'adresse du cluster à celle de l'hôte (--type=NodePort) chose qui n'a pas été faite cette fois ci :

```
[root@master01 ~]# minikube service my-nginx --url
😿  service default/my-nginx has no node port
```

Deployment en action

Nous avons lancé deux pods nginx dans replicaset, si on termine un des pods, kubernetes tentera de ré-équilibrer le nombre de réplicaset à 2.

```
[root@master01 ~]# kubectl delete pod my-nginx-6676fc446-r75lr && kubectl
get pods
pod "my-nginx-6676fc446-r75lr" deleted
NAME                        READY   STATUS    RESTARTS   AGE
my-nginx-6676fc446-b4zll    1/1     Running   0          79m
my-nginx-6676fc446-hjpmn    1/1     Running   0          2s
```

Vous voyez un deuxième pod lancé il y a à peine 2s.

Et comme vous allez le voir les deux reçoivent et traitent les requêtes :

```
[root@master01 ~]# i=0; while [ $i -lt 10 ]; do curl http://10.105.160.18
2> /dev/null | grep h1; sleep 1; i=$(($i+1)); done
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-hjpmn</h1>
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-hjpmn</h1>
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-hjpmn</h1>
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-b4zll</h1>
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-b4zll</h1>
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-hjpmn</h1>
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-hjpmn</h1>
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-hjpmn</h1>
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-hjpmn</h1>
    <h1>Nginx 1.13.8 on my-nginx-6676fc446-b4zll</h1>
```

Vous pouvez changer le nombre de pods en exécution dans le réplicaset :

```
[root@master01 ~]# kubectl scale deployment my-nginx --replicas=5
deployment.apps/my-nginx scaled
[root@master01 ~]# kubectl get pods -l run=my-nginx -o wide
NAME                        READY   STATUS    RESTARTS   AGE    IP           NODE
```

```
my-nginx-6676fc446-25m89  1/1    Running   0          10s    172.17.0.8    master01
my-nginx-6676fc446-b4zll  1/1    Running   0          101m   172.17.0.4    master01
my-nginx-6676fc446-hjpmn  1/1    Running   0          22m    172.17.0.2    master01
my-nginx-6676fc446-jck7c  1/1    Running   0          10s    172.17.0.10   master01
my-nginx-6676fc446-qs47k  1/1    Running   0          10s    172.17.0.9    master01
```

```
[root@master01 ~]# kubectl get replicasets
NAME                 DESIRED   CURRENT   READY   AGE
my-nginx-6676fc446   5         5         5       104m
```

Ps: à explorer --type=loadbalancer et ingress