docker container kubernetes tp01

TP01 | K8s - Installation

Rappel Kubernetes

C'est quoi Kubernetes ?

Kubernetes (également connu sous le nom de k8s ou "kube") est une plateforme d'orchestration de conteneurs open source qui automatise une grande partie des processus manuels impliqués dans le déploiement, la gestion et la mise à l'échelle des applications conteneurisées[1].

Le nom Kubernetes est d'origine grecque et signifie timonier ou pilote. L'abréviation K8s résulte du comptage des huit lettres entre le "K" et le "s". Google a ouvert le projet Kubernetes en 2014. Kubernetes combine plus de 15 ans d'expérience de Google dans l'exécution de charges de travail de production à grande échelle avec les meilleures idées et pratiques de la communauté.

Pourquoi utiliser K8S ?

Dans le monde du "cloud", les développeurs/DevOps sont de plus en plus utilisés pour automatiser les déploiements de différentes applications. Le but premier de Kubernetes est de pouvoir déployer en quelques clics une infrastructure complète d'une application, en parallèle ou en remplacement d'un déploiement précédent. De plus, Kubernetes permet de gérer la mise à l'échelle de cette infrastructure et donc, d'optimiser le rendu pour l'utilisateur final à la visite de l'application.

Déroulement du TP

L'objectif du TP est, grâce à vos connaissances sur les containers, de déployer un cluster Kubernetes sous docker via l'outil kind[2]. Et aussi d'interagir avec celui-ci. Par la suite nous verrons le déploiement d'applications dans kubernetes. Le TP va se dérouler par une phase d'installation d'outils, et de découverte de kubernetes. Il est essentiel de finir ce TP pour la suite. Si vous n'y arrivez pas n'hésiter pas à me solliciter et de solliciter vos camarades les plus rapides. Le travail de groupe est essentiel !

Prérequis

- Connaissances du module R4.08 sur les containers
- Accès au proxy IUT
- VM Ubuntu Server 22.04 LTS avec comme spécifications :
  - *8 GB DE RAM*
  - *35 GB de disque*
  - *2 Coeurs*
- Votre cerveau 😄

Installation des logiciels

Note

C'est une étape primordiale afin de mener a bien votre TP. Quelques commandes Linux sont utiles quand on ne sait pas : `man` `--help`. ⚠️ Attention à bien spécifier votre proxy ! ⚠️

Installation de docker

Attention à bien exporter votre proxy ! ⚠️

Tip

```
export {https_proxy,http_proxy,HTTP_PROXY,HTTPS_PROXY}="http://
user:mdp@host:port"
export {no_proxy,NO_PROXY}="localhost,127.0.0.1/8"
```

Pour faire fonctionner apt créer un fichier `/etc/apt/apt.conf.d/80proxy`

```
Acquire::http::proxy "http://user:mdp@IP:port";
Acquire::https::proxy "http://user:mdp@IP:port";
```

Il y a plusieurs modes d'installations de docker, personnellement je préfère de gérer l'installation via apt. Vous pouvez aussi suivre la documentation officielle ici

Préparation du répertoire Docker APT

```
# Add Docker's official GPG key:
sudo apt-get update
```

```
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
-o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# Add the repository to Apt sources:
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
sudo apt-get update
```

Installation des packages docker

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
```

Vérification de l'installation

```
docker --version
```

Tip

Ajouter votre utilisateur dans le groupe docker

Installation de kubectl

Attention à bien exporter votre proxy ! ⚠️ Comme pour docker, la documentation officielle est ici. Pour cette installation je préconise, pour bien gérer les versions, une installation via `curl`.

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.

Note

Vous devez utiliser une version de kubectl qui diffère seulement d'une version mineure de la version de votre cluster. Par exemple, un client v1.2 doit fonctionner avec un master v1.1, v1.2 et v1.3. Pour des raisons pratiques nous allons installer la dernière version.

Installer le binaire de kubectl avec curl

```
curl -LO https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
```

Rendez le binaire executable

```
chmod +x ./kubectl
```

Déplacer le binaire dans votre PATH

```
sudo mv kubectl /usr/local/bin/kubectl
```

Testez la version de kubectl

```
kubectl version --client
```

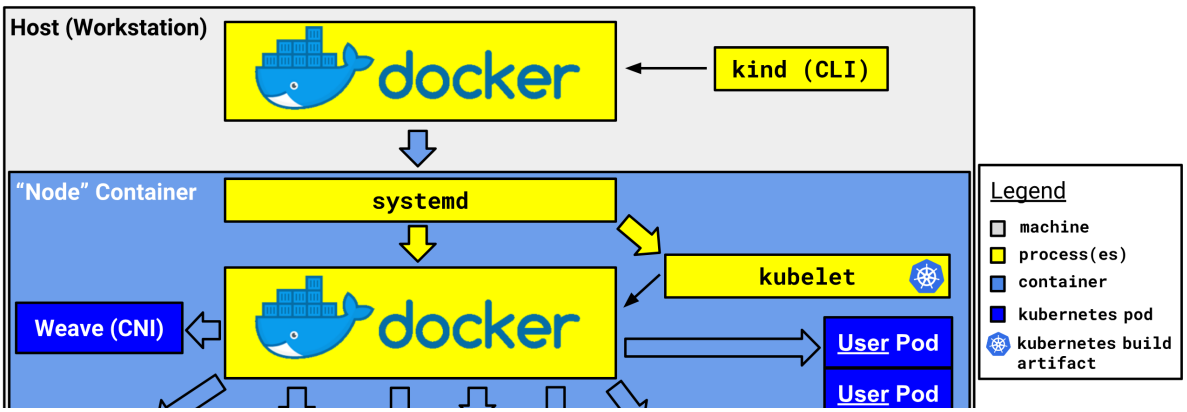
Note

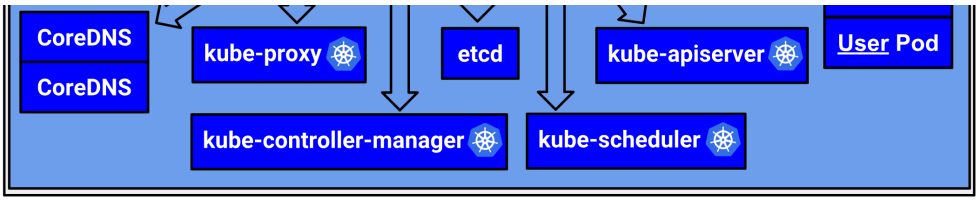
Maintenant que nous avons installé les prérequis ; **docker** et **kubectl**, nous pouvons installer l'outil **kind** qui nous permettra de générer un/des cluster(s) kubernetes. L'objectif est de vous familiarisé avec l'outil afin de créer votre cluster pour la suite du TP.

Principes de kind

Le principe de kind est de déployer des clusters locaux à des fins de développement/testing. De faire fonctionner des clusters à faible coût que n'importe quel développeur peut répliquer localement. De s'intégrer dans n'importe quel type d'environnement.

kind - Kubernetes IN Docker





Installation de kind

Pour cette installation je recommande une installation via curl, mais la documentation officielle est disponible ici

```
[ $(uname -m) = x86_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/
v0.20.0/kind-linux-amd64
chmod +x ./kind
sudo mv ./kind /usr/local/bin/kind
```

Initialiser son premier cluster via la CLI

⚠️ Attention à bien spécifier votre proxy pour docker afin de pull les images de kind et n'oubliez pas le paramètres `NO_PROXY` !⚠️
Exemple :

```
[Service]
Environment="HTTP_PROXY=http://user:mdp@ip:port"
Environment="HTTPS_PROXY=http://user:mdp@ip:port"
Environment="NO_PROXY=localhost,127.0.0.1"
```

> **Tip**
>
> Je vous conseille de lire intégralement cette partie afin de bien commencer

La création d'un cluster Kubernetes est très simple : `kind create cluster`

Cela va démarrer un cluster Kubernetes en utilisant une image de nœud préconstruite. Les images préconstruites sont hébergées sur dockerhub à l'adresse kindest/node, mais pour la suite du TP nous allons utiliser des images avec une version spécifique.

Pour spécifier une autre image, utilisez l'option `--image`. Exemple : `kind create cluster --image=...`

L'utilisation d'une image différente vous permet de changer la version Kubernetes du cluster créé.

Kind offre la possibilité de créer nos propres images. Nous n'allons pas l'aborder dans ce TP.

Par défaut, le cluster reçoit le nom kind. Utilisez l'option `--name` pour attribuer au cluster un nom de contexte différent.

Pour en savoir plus sur l'utilisation de kind, consultez la page suivante : `kind create cluster --help`

Interagir avec son cluster

Après avoir créé votre cluster, vous pouvez interagir avec lui avec `kubectl`. Par défaut la configuration des accès pour se connecter au cluster ce trouve dans le fichier `${HOME}/.kube/config`

Pour voir la liste des clusters créée, voici la commande `get clusters`

Mise en situation, j'initialise 2 clusters :

```
kind create cluster # Context par défaut kind
kind create cluster --name info
```

Je regarde ma configuration :

```
kind get clusters
>kind
>info
```

Pour interagir avec un cluster spécifique, il suffit de spécifier le nom du cluster en tant que contexte dans kubectl

```
kubectl cluster-info --context kind-kind
kubectl cluster-info --context kind-info
```

Maintenant à vous de créer 2 clusters, et d'afficher la version des nodes de CHAQUE cluster kubernetes via la commande `kubectl`

> **Tip**
>
> `kubectl --help`

Suppression de vos clusters

Grâce à la commande `kind delete cluster` vous pouvez supprimer vos clusters. Si le paramètre

`--name` n'est pas spécifié par défaut kind supprimera le context `kind`

Configuration via un fichier yaml

> **Note**
>
> Lisez tout ce chapitre avant de passer à la création de votre cluster.

Maintenant que nous avons vu kind via la ligne de commande, nous pouvons aller plus loin et de spécifier un fichier de configuration en yaml. Pour configurer la création d'un cluster kind, vous devrez créer un fichier de configuration en YAML. Ce fichier suit les conventions de Kubernetes pour les versions, etc..

```
1  kind: Cluster
2  apiVersion: kind.x-k8s.io/v1alpha4
```

Cette configuration spécifie simplement que nous configurons un cluster KIND (kind : Cluster) et que la version de la configuration de KIND que nous utilisons est v1alpha4 (apiVersion : kind.x-k8s.io/v1alpha4).

Une version donnée de kind peut supporter différentes versions qui auront des options et des comportements différents. C'est pourquoi nous devons toujours spécifier la version.

Ce mécanisme est inspiré de la configuration des ressources et des composants de Kubernetes.

Pour utiliser cette configuration, placez le contenu dans un fichier config.yaml et exécutez ensuite `kind create cluster --config=config.yaml` depuis le même répertoire.

Donner un nom au cluster

Vous pouvez spécifier un nom à votre cluster

```
1  kind: Cluster
2  apiVersion: kind.x-k8s.io/v1alpha4
3  name: app-1-cluster
```

Cluster multi-noeuds

```
1  kind: Cluster
2  apiVersion: kind.x-k8s.io/v1alpha4 # Version d'API utilisé
3  # patch le déploiement de kube en passant des options (permet de customiser
4  l'installation du cluster)
```

```
kubeadmConfigPatches:
- |
  apiVersion: kubelet.config.k8s.io/v1beta1
  kind: KubeletConfiguration
  evictionHard:
    nodefs.available: "0%"
# 1 control plane node et 3 workers
nodes:
# le control plane node
- role: control-plane
# les 3 workers
- role: worker
- role: worker
- role: worker
```

Ici on créé un cluster de 4 noeuds, avec un controle-plane et 3 workers. Le controle-plane a le rôle de gérer la "logique" du cluster.

Cluster Haute-Disponibilité

Dans certains cas, il est utile de simuler un cluster HA (Haute disponibilité) afin d'avoir de la résilience quand un noeud tombe.

```
# a cluster with 3 control-plane nodes and 3 workers
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: control-plane
- role: control-plane
- role: worker
- role: worker
- role: worker
```

Spécifier une version de kubernetes

Vous pouvez spécifier une version de kubernetes, la liste des versions est disponible ici, en définissant l'image du conteneur du nœud. Veuillez utiliser le sha256 pour la version de Kubernetes souhaitée, comme indiqué dans cet exemple :

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  image: kindest/
node:v1.27.3@sha256:3966ac761ae0136263ffdb6cfd4db23ef8a83cba8a463690e98317add2c9
- role: worker
  image: kindest/
node:v1.27.3@sha256:3966ac761ae0136263ffdb6cfd4db23ef8a83cba8a463690e98317add2c9
```

Extra port et volumes

Des mappages de ports supplémentaires peuvent être utilisés pour rediriger les ports vers les nœuds de kind. Il s'agit d'une option multiplateforme qui permet d'acheminer le trafic vers votre cluster kind :

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  # port forward 80 sur l`hote vers le port 80 du container
  # Ajoute un montage de /path/to/my/files du host vers /files sur le noeud
  extraMounts:
  - hostPath: /path/to/my/files
    containerPath: /files
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
    # Optionnel: TCP, UDP, SCTP.
    # TCP par défaut
    protocol: TCP
```

Exporter les logs du cluster

Kind a la capacité d'exporter tous les logs relatifs à kind pour que vous puissiez les explorer. Pour exporter tous les journaux du cluster par défaut (nom de contexte kind) :

```
kind export logs
```

Comme pour toutes les autres commandes, si vous souhaitez effectuer l'action sur un cluster avec un nom de contexte différent, utilisez l'option `--name`

Création de votre cluster avec un fichier yaml

- Les images du noeud doivent être en version `1.27.3`
- Forward le port 80 et 443 sur le port 80 et 443 de la machine hôte.

- ✅ Le cluster doit avoir 2 control-plane et 1 worker.
- ✅ Avoir un fichier contenant un certificat partagé entre l'hôte et un noeud du control-plane

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -sha256 -days 3650 -nodes -subj "/C=TP/ST=IFS/L=Paris/O=IUT/OU=CAMPUS3/CN=MonClusterK8s"
```

Pour les plus rapides

C'est quoi un ingress controller ?

Un Ingress controller fait abstraction de la complexité du routage du trafic des applications Kubernetes et fournit un pont entre les services Kubernetes et les services externes. Les Ingress controller de Kubernetes acceptent le trafic provenant de l'extérieur de la plateforme Kubernetes et l'équilibrent en charge vers les pods (conteneurs) fonctionnant à l'intérieur de la plateforme.

Configuration kubeadm ingress

Nous pouvons exploiter l'option de configuration `extraPortMapping` de KIND lors de la création d'un cluster pour transférer les ports de l'hôte vers un contrôleur d'entrée fonctionnant sur un nœud comme vu plus haut.

Nous pouvons également configurer un node-label personnalisé en utilisant `node-labels` dans la configuration initiale de kubeadm, qui sera utilisé par le `nodeSelector` de l'ingress controller avec la syntaxe suivante :

```yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
```

Créer un cluster

- ✅ Supprimer vos/votre ancien(s) cluster(s)
- ✅ Le cluster doit avoir 2 control-plane et 1 worker.
- ✅ Forward le port 80 et 443 sur le port 80 et 443 de la machine hôte.

✅ Mettre un `node-labels: "ingress-ready=true"` sur le noeud ou il y a le port forward.

Déployer un ingress controller

Il existe plusieurs technologies d'Ingress controller, le plus utilisé/documenté dans Kubernetes est Nginx. Pour déployer un ingress controller dans votre cluster il vous faudra appliquer des manifests. (Fichier yaml ou il y a la description des choses à déployer)

Quelques exemples de manifest :
⚠️ À ne pas appliquer dans votre cluster c'est à titre d'exemple ! ⚠️

Ici nous créeons un namespace dans kubernetes

```yaml
---
apiVersion: v1
kind: Namespace
metadata:
  labels:
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
  name: ingress-nginx
```

Ici nous créeons une classe d'ingress

```yaml
---
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/instance: ingress-nginx
    app.kubernetes.io/name: ingress-nginx
    app.kubernetes.io/part-of: ingress-nginx
    app.kubernetes.io/version: 1.9.3
  name: nginx
spec:
  controller: k8s.io/ingress-nginx
```

Installation de nginx

Ici nous avons tous les manifests pour déployer un ingress controller grâce au label du noeud spécifié plus tôt.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml
```

Maintenant, l'Ingress est prêt. Attendez qu'il soit prêt à traiter les demandes en cours d'exécution :

```
kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=90s
```

Tester l'ingress avec un service web

Les manifests suivant crée des services http-echo simples et un objet Ingress pour acheminer les données vers ces services. Appliquer ce manifest grâce à la commande `kubectl apply`. Il faudra enregistrer ce yaml dans un fichier.

```
---
kind: Pod
apiVersion: v1
metadata:
  name: foo-app
  labels:
    app: foo
spec:
  containers:
  - command:
    - /agnhost
    - netexec
    - --http-port
    - "8080"
    image: registry.k8s.io/e2e-test-images/agnhost:2.39
    name: foo-app
---
kind: Service
apiVersion: v1
metadata:
  name: foo-service
spec:
  selector:
    app: foo
  ports:
  # Default port used by the image
  - port: 8080
---
kind: Pod
apiVersion: v1
metadata:
  name: bar-app
  labels:
    app: bar
spec:
  containers:
  - command:
```

```
      - /agnhost
      - netexec
      - --http-port
      - "8080"
      image: registry.k8s.io/e2e-test-images/agnhost:2.39
      name: bar-app
  ---
  kind: Service
  apiVersion: v1
  metadata:
    name: bar-service
  spec:
    selector:
      app: bar
    ports:
    # Default port used by the image
    - port: 8080
  ---
  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    name: example-ingress
    annotations:
      nginx.ingress.kubernetes.io/rewrite-target: /$2
  spec:
    rules:
    - http:
        paths:
        - pathType: Prefix
          path: /foo(/|$)(.*)
          backend:
            service:
              name: foo-service
              port:
                number: 8080
        - pathType: Prefix
          path: /bar(/|$)(.*)
          backend:
            service:
              name: bar-service
              port:
                number: 8080
```

Vérification de l'ingress

Maintenant vérifier que l'Ingress controller fonctionne

```
curl localhost/foo/hostname
curl localhost/bar/hostname
```

Merci de votre attention

1. Voir module R4.08. ↩
2. kind est un outil permettant d'exécuter des clusters Kubernetes locaux en utilisant des conteneurs Docker "nodes" ↩

Last update: November 6, 2023 19:09:35