CANJALE LUCAS

Workshop Kubernetes (bases et stateless)

3 Installation du cluster Kubernetes avec Kind

- **installer kubectl sur votre machine!**

```
test@Lucas-22:~$ [ $(uname -m) = x86_64 ] && curl -Lo ./kind https://kind.sigs.k8s.io/dl/v0.20.0/kind-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    97  100    97    0     0    495      0 --:--:-- --:--:-- --:--:--   497
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 6304k  100 6304k    0     0  79050      0  0:01:21  0:01:21 --:--:--  548k
test@Lucas-22:~$ chmod +x ./kind
test@Lucas-22:~$ sudo mv ./kind /usr/local/bin/kind
test@Lucas-22:~$ kind version
kind v0.20.0 go1.20.4 linux/amd64
test@Lucas-22:~$
```

On voit bien que le Kind est bien installé et sa version.

- **créer votre cluster kind en clonant le repository**

git clone https://github.com/pushou/substrat-labs-k8s.git

```
test@Lucas-22:~$ git clone https://github.com/pushou/substrat-labs-k8s.git
Clonage dans 'substrat-labs-k8s'...
remote: Enumerating objects: 363, done.
remote: Counting objects: 100% (363/363), done.
remote: Compressing objects: 100% (240/240), done.
remote: Total 363 (delta 186), reused 283 (delta 112), pack-reused 0
Réception d'objets: 100% (363/363), 28.60 Mio | 343.00 Kio/s, fait.
Résolution des deltas: 100% (186/186), fait.
test@Lucas-22:~$ cd substrat-labs-k8s/
test@Lucas-22:~/substrat-labs-k8s$ ls
creation-cluster-k3s  creation-cluster-kind  LICENSE  README.md
test@Lucas-22:~/substrat-labs-k8s$
```

Création du cluster en executant le script: ***create-kind-cluster-with-ec.sh***

```
test@Lucas-22:~/substrat-labs-k8s/creation-cluster-kind$ ./creation-kind-cluster-with-ec.sh
fce2b53a421dbc0052e06c37bdbf6cb23c406a446b0e6503bfcc03a97e7edb92
Deleting cluster "tp1k8s" ...
creation du cluster kind avec la feature-gate ephemeral-container
Creating cluster "tp1k8s" ...
 ✓ Ensuring node image (kindest/node:v1.27.3) 🖼
 ✓ Preparing nodes 📦 📦 📦 📦
 ✓ Writing configuration 📜
 ✓ Starting control-plane 🕹️
 ✓ Installing CNI 🔌
 ✓ Installing StorageClass 💾
 ✓ Joining worker nodes 🚜
Set kubectl context to "kind-tp1k8s"
You can now use your cluster with:

kubectl cluster-info --context kind-tp1k8s
```

CANJALE LUCAS

Docker ps:

```
test@Lucas-22:~/substrat-labs-k8s/creation-cluster-kind$ docker ps
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS
                     NAMES
8e65f2bd2598   kindest/node:v1.27.3   "/usr/local/bin/entr…"   9 minutes ago    Up 9 minutes    127.0.0.0:8003->9200/tcp
                     tp1k8s-worker3
95a665d4fae8   kindest/node:v1.27.3   "/usr/local/bin/entr…"   9 minutes ago    Up 9 minutes    127.0.0.0:8001->9200/tcp
                     tp1k8s-worker
9552bde38185   kindest/node:v1.27.3   "/usr/local/bin/entr…"   9 minutes ago    Up 9 minutes    0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp, 0.0.0.0:1024->1024/tcp, 10.202.13.1:6443->6443/tcp,
0.0.0.0:9000->30777/tcp   tp1k8s-control-plane
ac4069a7b12d   kindest/node:v1.27.3   "/usr/local/bin/entr…"   9 minutes ago    Up 9 minutes    127.0.0.0:8002->9200/tcp
                     tp1k8s-worker2
fce2b53a421d   registry:2             "/entrypoint.sh /etc…"   19 minutes ago   Up 19 minutes   127.0.0.1:5000->5000/tcp
                     kind-registry
```

docker exec -it 8e65f2bd2598 bash

```
test@Lucas-22:~/substrat-labs-k8s/creation-cluster-kind$ docker exec -it 8e65f2bd2598 bash
root@tp1k8s-worker3:/#
```

Install kubectl

```
test@Lucas-22:~$ kubectl version --client
Client Version: v1.28.2
Kustomize Version: v5.0.4-0.20230601165947-6ce0bf390ce3
test@Lucas-22:~$
```

Auto-complétion avec Kubectl

```
test@Lucas-22:~$ source <(kubectl completion bash)
test@Lucas-22:~$ echo "source <(kubectl completion bash)" >> ~/.bashrc
test@Lucas-22:~$ alias k=kubectl
test@Lucas-22:~$ complete -o default -F __start_kubectl k
```

```
test@Lucas-22:~/Téléchargements$ k explain pod
KIND:       Pod
VERSION:    v1

DESCRIPTION:
    Pod is a collection of containers that can run on a host. This resource is
    created by clients and scheduled onto hosts.
```

```
test@Lucas-22:~/Téléchargements$ k explain pod.apiVersion
KIND:       Pod
VERSION:    v1

FIELD: apiVersion <string>

DESCRIPTION:
    APIVersion defines the versioned schema of this representation of an object.
    Servers should convert recognized schemas to the latest internal value, and
    may reject unrecognized values. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources
```

```
test@Lucas-22:~/Téléchargements$ k explain pod.apiVersion --recursive
KIND:       Pod
VERSION:    v1

FIELD: apiVersion <string>

DESCRIPTION:
    APIVersion defines the versioned schema of this representation of an object.
    Servers should convert recognized schemas to the latest internal value, and
    may reject unrecognized values. More info:
    https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources
```

```
test@Lucas-22:~/Téléchargements$ kubectl get nodes -o json |
jq ".items[] | {name:.metadata.name} + .status.capacity"
{
  "name": "tp1k8s-control-plane",
  "cpu": "16",
  "ephemeral-storage": "457592136Ki",
  "hugepages-1Gi": "0",
  "hugepages-2Mi": "0",
  "memory": "32613732Ki",
  "pods": "110"
```

Créer cet alias essentiel dans votre .bashrc et activer la complétion pour kubectl:

```
test@Lucas-22:~/Téléchargements$ alias k=kubectl
test@Lucas-22:~/Téléchargements$ echo 'source <(kubectl completion bash)' >>~/.bashrc
test@Lucas-22:~/Téléchargements$ kubectl completion bash >/etc/bash_completion.d/kubectl
bash: /etc/bash_completion.d/kubectl: Permission non accordée
test@Lucas-22:~/Téléchargements$ sudo !!
sudo kubectl completion bash >/etc/bash_completion.d/kubectl
bash: /etc/bash_completion.d/kubectl: Permission non accordée
test@Lucas-22:~/Téléchargements$ echo 'alias k=kubectl' >>~/.bashrc
test@Lucas-22:~/Téléchargements$ echo 'complete -F __start_kubectl k' >>~/.bashrc
test@Lucas-22:~/Téléchargements$ source ~/.bashrc
test@Lucas-22:~/Téléchargements$
```

Image registry.iutbeziers.fr/pythonapp:latest:

```
test@Lucas-22:~$ docker pull registry.iutbeziers.fr/pythonapp:latest
latest: Pulling from pythonapp
756975cb9c7e: Pull complete
d77915b4e630: Pull complete
5f37a0a41b6b: Pull complete
96b2c1e36db5: Pull complete
c495e8de12d2: Pull complete
33382189822a: Pull complete
414ebfa5f45b: Pull complete
dd860911922e: Pull complete
ac1f7f2faf6e: Pull complete
ac38a92df562: Pull complete
4d71fc4753b5: Pull complete
b477a50490a1: Pull complete
4f4fb700ef54: Pull complete
2c5d93e6dff2: Pull complete
Digest: sha256:b23c278eab01274a6f41878fd06da9c05e8902e4b7af5e3b2c6e52cb4b86303c
Status: Downloaded newer image for registry.iutbeziers.fr/pythonapp:latest
registry.iutbeziers.fr/pythonapp:latest
```

upload de l'image sur les containers Docker

```
test@Lucas-22:~$ kind --name tp1k8s load docker-image registry.iutbeziers.fr/pythonapp:latest
Image: "registry.iutbeziers.fr/pythonapp:latest" with ID "sha256:20b638827c2b93109a5aacfe73e0aa1bcf5363402197cf941d3d742e2033b208" found to be already present on all nodes.
```

4 Premiers pas avec Kubernetes

4.1 Conguration des objets en mode déclaratif et impératif

1. Adaptez à votre contexte et passez les commandes suivantes pour vous familiariser avec Kubernetes et son mode impératif:

```
test@Lucas-22:~$ kubectl run nginx-pod --image nginx
pod/nginx-pod created
```

```
test@Lucas-22:~$ kubectl get pods
NAME        READY   STATUS    RESTARTS   AGE
nginx-pod   1/1     Running   0          51s
```

```
test@Lucas-22:~$ kubectl exec -it nginx-pod -- sh
#
```

```
test@Lucas-22:~$ kubectl create deployment hello-nginx --image nginx
deployment.apps/hello-nginx created
test@Lucas-22:~$ kubectl scale deployment hello-nginx --replicas 2
deployment.apps/hello-nginx scaled
test@Lucas-22:~$ kubectl expose deployment hello-nginx --type=LoadBalancer --port 80 --target-port 80
service/hello-nginx exposed
test@Lucas-22:~$
```

2. Visualisez les objets Kubernetes générés avec les commandes suivantes:
commande: k get nodes -o wide --show-labels

```
test@Lucas-22:~$ k get nodes -o wide --show-labels
NAME                 STATUS   ROLES           AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE                         KERNEL-VERSION     CONTAINER-RUNTIME    LABELS
tp1k8s-control-plane   Ready    control-plane   66m   v1.27.3   172.21.0.6    <none>        Debian GNU/Linux 11 (bullseye)   5.10.0-18-amd64   containerd://1.7.1   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,ingress-ready=true,kubernetes.io/arch=amd64,kubernetes.io/hostname=tp1k8s-control-plane,kubernetes.io/os=linux,node-role.kubernetes.io/control-plane=,node.kubernetes.io/exclude-from-external-load-balancers=,run=haproxy-ingress
tp1k8s-worker          Ready    <none>          66m   v1.27.3   172.21.0.4    <none>        Debian GNU/Linux 11 (bullseye)   5.10.0-18-amd64   containerd://1.7.1   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=tp1k8s-worker,kubernetes.io/os=linux
tp1k8s-worker2         Ready    <none>          66m   v1.27.3   172.21.0.5    <none>        Debian GNU/Linux 11 (bullseye)   5.10.0-18-amd64   containerd://1.7.1   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=tp1k8s-worker2,kubernetes.io/os=linux
tp1k8s-worker3         Ready    <none>          66m   v1.27.3   172.21.0.3    <none>        Debian GNU/Linux 11 (bullseye)   5.10.0-18-amd64   containerd://1.7.1   beta.kubernetes.io/arch=amd64,beta.kubernetes.io/os=linux,kubernetes.io/arch=amd64,kubernetes.io/hostname=tp1k8s-worker3,kubernetes.io/os=linux
```

3 & 4. Utilisez la commande '*kubectl create dry-run=client -o yaml>ngnix.yaml* afin de générer les manifests des objets créés en mode impératif précédemment.

kubectl run nginx-pod --image=nginx --dry-run=client -o yaml > nginx-pod.yaml

kubectl create configmap maconfgmap --from-literal=k8s=leprésent --from-literal=virt=legacy

kubectl get configmap maconfgmap -o jsonpath='{.data}'

```
test@Lucas-22:~/kind$ kubectl run nginx-pod --image=nginx --dry-run=client -o yaml > nginx-pod.yaml
test@Lucas-22:~/kind$ ls
nginx-pod.yaml
test@Lucas-22:~/kind$ kubectl create configmap maconfgmap --from-literal=k8s=leprésent --from-literal=virt=legacy
configmap/maconfgmap created
test@Lucas-22:~/kind$ kubectl get configmap maconfgmap -o jsonpath='{.data}'
{"k8s":"leprésent","virt":"legacy"}test@Lucas-22:~/kind$
```

5. Créer de même un secret nommé monsecret avec la valeur mdp=torototo et achez le "décodé" (ilest en base 64).
Commandes passés: **echo -n 'mdp=torototo' | base64**
kubectl create secret generic monsecret --from-literal=mdp=bWRwPXRvcm90b3Rv
kubectl get secret monsecret -o jsonpath='{.data}' -> ***affichage de valeurs***

```
test@Lucas-22:~/kind$ echo -n 'mdp=torototo' | base64
bWRwPXRvcm90b3Rv
test@Lucas-22:~/kind$ kubectl create secret generic monsecret --from-literal=mdp=bWRwPXRvcm90b3Rv
secret/monsecret created
test@Lucas-22:~/kind$ kubectl get secret monsecret -o jsonpath='{.data}'
{"mdp":"YldSd1BYUnZjbTecho -n 'bWRwPXRvcm90b3Rv' | base64 --decodeWRwPXRvcm90b3Rv' | base64 --decode
mdp=torotototest@Lucas-22:~/kind$
test@Lucas-22:~/kind$
```

4.2 Les "PODS" l'unité atomique de Kubernetes

4.2.1 Opérations basiques sur les PODS

1. Générez la conguration d'un pod debian à l'aide de la commande suivante:
k run --dry-run=client debianpod --image=registry.iutbeziers.fr/debianiut:latest -o yaml > monpremierpod.yml

```
test@Lucas-22:~/kind$ k run --dry-run=client debianpod --image=registry.iutbeziers.fr/debianiut:latest -o yaml > monpremierpod.yml
test@Lucas-22:~/kind$ ls
monpremierpod.yml  nginx-pod.yaml
test@Lucas-22:~/kind$
```

POD crée:
kubectl create -f monpremierpod.yml

```
test@Lucas-22:~/kind$ kubectl create -f monpremierpod.yml
pod/debianpod created
test@Lucas-22:~/kind$
```

2. Vérifiez l'état de votre pod:
Commande: **kubectl get pods**

```
test@Lucas-22:~/kind$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
debianpod                     1/1     Running   0          53s
```

3. Sur quel noeud votre pod s'exécute-t-il ?
Commande: **kubectl get pod debianpod -o wide**
Il s'execute sur le node ***tp1k8s-worker3.***

```
test@Lucas-22:~/kind$ kubectl get pod debianpod -o wide
NAME        READY   STATUS    RESTARTS   AGE     IP           NODE           NOMINATED NODE   READINESS GATES
debianpod   1/1     Running   0          2m56s   10.244.2.3   tp1k8s-worker3   <none>           <none>
test@Lucas-22:~/kind$
```

4. Dans quel NameSpace votre Pod s'exécute-t-il ?
Commande: kubectl get pod debianpod -o jsonpath='{.metadata.namespace}'
Il s'exécute dans le ***default.***

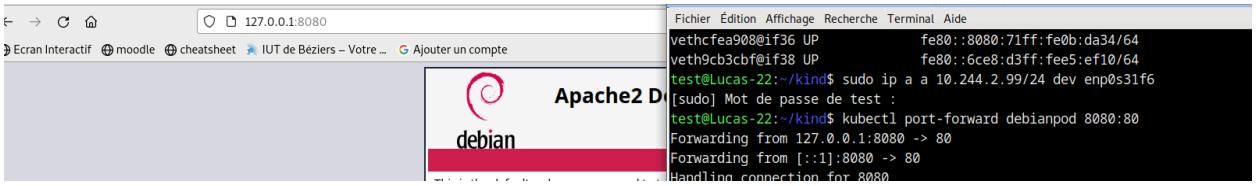
```
test@Lucas-22:~/kind$ kubectl get pod debianpod -o jsonpath='{.metadata.namespace}'
defaulttest@Lucas-22:~/kind$
```

5. Accédez au container en utilisant "kubectl exec". Démarrez un server apache dans le container.
Commande: kubectl exec -it debianpod -- /bin/bash
apt update
apt install -y apache2
service apache2 start

```
test@Lucas-22:~/kind$ kubectl exec -it debianpod -- /bin/bash
root@debianpod:/#
```

```
root@debianpod:/# service apache2 start
Starting Apache httpd web server: apache2AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 10.244.2.3. Set the 'ServerName' directive globally to
 suppress this message
.
root@debianpod:/# service apache2 status
apache2 is running.
root@debianpod:/#
```

6. Pouvez-vous accéder au serveur web Apache qui s'exécute dans ce Pod ? Accéder au serveur Apache depuis votre client Kubernetes en utilisant "kubectl port-forward". Quel est le principe de cette commande ?
Commande: **kubectl port-forward debianpod 8080:80**
Cette commande sert à rediriger le port 80 du pod vers le port 8080 de ma machine locale.



4.2.2 Manipulation des Pods

1. Supprimez le pod et recréez-le cette fois-ci en utilisant un "kubectl apply -votre-manifeste". Quelle est la différence entre "apply" et "create" ?
Command: kubectl delete pod debianpod -> suppression
kubectl apply -f monpremierpod.yml

```
test@Lucas-22:~/kind$ kubectl delete pod debianpod
pod "debianpod" deleted
test@Lucas-22:~/kind$ kubectl apply -f monpremierpod.yml
pod/debianpod created
```

- **kubectl create** :
    - Utilisé pour créer une nouvelle ressource.
    - S'il existe déjà une ressource avec le même nom, cela générera une erreur.

- **kubectl apply** :
    - Utilisé pour créer ou mettre à jour une ressource.
    - Si la ressource n'existe pas, elle sera créée. Si elle existe déjà, elle sera mise à jour avec les changements spécifiés dans le manifeste.

2. Recréer le POD avec en sus un container issu d'une image busybox. Rattachez-vous au pod via "kubectl exec" en précisant avec l'option -c le container nom donné au container busybox.
Fichier yaml: monpremierpod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: debianpod
spec:
  containers:
  - name: debian-container
    image: registry.iutbeziers.fr/debianiut:latest
  - name: busybox-container
    image: busybox:latest
    command:
      - sleep
      - "3600"
```

Command apply: kubectl apply -f monpremierpod.yml
kubectl exec -it debianpod -c busybox-container -- /bin/sh

```
pod/debianpod created
test@Lucas-22:~/kind$
test@Lucas-22:~/kind$ kubectl exec -it debianpod -c busybox-container -- /bin/sh
/ #
/ #
```

3. Modifiez le manifeste du pod an de limiter le cpu et la mémoire consommée par ce pod via des "Limits".
Modifs sur mon fichier yml:

```yaml
        cpu: "0.5" # Limite de CPU à 0.5 unité (1 unité équivaut à 1 vCPU)
        memory: "512Mi" # Limite de mémoire à 512 MiB
```

Après j'ai les applique:

```
test@Lucas-22:~/kind$ kubectl apply -f monpremierpod.yml
pod/debianpod created
```

4. Générer un autre pod sur un autre "node" K8s et effectuez une traceroute entre les deux pods. Quelles différences constatez-vous avec la gestion réseaux des containers Docker ?
Mondeuxiemepod.yml:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: deuxiemepod
spec:
  containers:
  - name: busybox-container
    image: busybox:latest
    command:
      - sleep
      - "3600"
```

Command: kubectl apply -f mondeuxiemepod.yml

```
test@Lucas-22:~/kind$ kubectl apply -f mondeuxiemepod.yml
pod/deuxiemepod created
```

Traceroute sur les deux pods:

Commande: **kubectl exec -it debianpod -- /bin/sh**

Je suis rentré dans le premier pod et installé traceroute:

Commande: **apt install traceroute -y**

```
test@Lucas-22:~/kind$ kubectl exec -it debianpod -- /bin/sh
Defaulted container "debian-container" out of: debian-container, busybox-container
# apt install traceroute -y
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances... Fait
Lecture des informations d'état... Fait
Les paquets suivants seront mis à jour :
  traceroute
1 mis à jour, 0 nouvellement installés, 0 à enlever et 67 non mis à jour.
```

Pour savoir l'adresse du mon deuxieme pod j'ai passé la commande: **kubectl get pod deuxiemepod -o jsonpath='{.status.podIP}'**

```
test@Lucas-22:~/kind$ kubectl get pod deuxiemepod -o jsonpath='{.status.podIP}'
10.244.1.3test@Lucas-22:~/kind$
```

Depuis mon premier pod: **traceroute 10.244.1.3**

```
# traceroute 10.244.1.3
traceroute to 10.244.1.3 (10.244.1.3), 30 hops max, 60 byte packets
 1  10.244.2.1 (10.244.2.1)  0.027 ms  0.008 ms  0.006 ms
 2  tp1k8s-worker.kind (172.21.0.4)  0.089 ms  0.030 ms  0.034 ms
 3  10.244.1.3 (10.244.1.3)  0.051 ms  0.026 ms  0.024 ms
#
```

- Dans Docker, les conteneurs communiquent par défaut via un réseau de pontage (bridge network). Pour communiquer entre des conteneurs sur différents hôtes, vous devez configurer des ponts de réseau, créer des tunnels ou utiliser des solutions comme Docker Swarm ou des réseaux de superposition (overlay networks).
- Kubernetes, quant à lui, gère automatiquement la communication entre les pods sur différents nœuds en utilisant un réseau virtuel créé par le composant de réseau de Kubernetes (CNI, Container Network Interface). Cela facilite grandement la communication entre les pods, quel que soit leur emplacement dans le cluster.

4.2.3 Utilisation des utilitaires des "Container runtime" et de kubectl pour manipuler des containers

1. Lancez un process bash dans le pod créé avec kubectl.

Commande: **kubectl exec -it debianpod -- /bin/bash**

```
test@Lucas-22:~/kind$ kubectl exec -it debianpod -- /bin/bash
Defaulted container "debian-container" out of: debian-container, busybox-container
root@debianpod:/#
```

2. Lister les images des containers utilisées par votre cluster Kubernetes à l'aide de kubectl

kubectl get pods --output=jsonpath='{range.items[*]}{.spec.containers[*].image}{"\n"}{end}' | sort | uniq

```
test@Lucas-22:~/kind$ kubectl get pods --output=jsonpath='{range .items[*]}{.spec.containers[*].image}{"\n"}{end}' | sort | uniq
busybox:latest
nginx
registry.iutbeziers.fr/debianiut:latest busybox:latest
```

3. Utilisez les utilitaires nerdctl et crictl pour récupérer la liste des images sur le noeud "worker" qui exécute le container.

4. Lancez un process bash dans le pod avec crictl. Faite de même avec l'utilitaire nerdctl.

5. Pour information il est possible de lancer un processus dans un container à l'aide de ctr l'utilitaire standard de containerd.

Les étapes sont les suivantes:

4.2.4 Implémentation d'un pattern commun: l'init pod

1. installation d'un init pod:

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: init-demo
spec:
  containers:
  - name: apache
    image: httpd:2.4           # Utilisation de l'image Apache
    ports:
    - containerPort: 80
    volumeMounts:
    - name: workdir
      mountPath: /usr/local/apache2/htdocs   # Le dossier pour les fichiers HTML d'Apache
  initContainers:
  - name: download-html
    image: busybox:1.28
    command:
    - wget
    - "-O"
    - "/work-dir/index.html"
    - http://info.cern.ch
    volumeMounts:
    - name: workdir
      mountPath: "/work-dir"
  dnsPolicy: Default
  volumes:
  - name: workdir
    emptyDir: {}
```

kubectl exec -it init-demo -c apache -- /bin/bash

```
root@init-demo:/usr/local/apache2# curl localhost
<html><head></head><body><header>
<title>http://info.cern.ch</title>
</header>

<h1>http://info.cern.ch - home of the first website</h1>
<p>From here you can:</p>
<ul>
<li><a href="http://info.cern.ch/hypertext/WWW/TheProject.html">Browse the first website</a></li>
<li><a href="http://line-mode.cern.ch/www/hypertext/WWW/TheProject.html">Browse the first website using the line-mode browser simulator</a></li>
<li><a href="http://home.web.cern.ch/topics/birth-web">Learn about the birth of the web</a></li>
<li><a href="http://home.web.cern.ch/about">Learn about CERN, the physics laboratory where the web was born</a></li>
</ul>
</body></html>
```

5. Gestion du cycle de vie des applications dans Kubernetes

5.1 Déploiement de son application Kubernetes

5.1.1 Les "NameSpaces"

1. Créez un namespace "applicatif".

Commande: **kubectl create namespace applicatif**

```
test@Lucas-22:~/kind$ kubectl create namespace applicatif
namespace/applicatif created
test@Lucas-22:~/kind$
```

2. Editez le "manifest" de ce namespace avec kubectl et modier son nom en "applicatifs".

kubectl edit namespace applicatif

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file will be
# reopened with the relevant failures.
#
apiVersion: v1
kind: Namespace
metadata:
  creationTimestamp: "2023-10-17T13:55:03Z"
  labels:
    kubernetes.io/metadata.name: applicatif
  name: applicatifs
  resourceVersion: "29427"
  uid: 48068de9-4683-477c-84a1-80cfd6209975
spec:
  finalizers:
  - kubernetes
status:
  phase: Active
~
```

```
test@Lucas-22:~/kind$ kubectl edit namespace applicatif
A copy of your changes has been stored to "/tmp/kubectl-edit-2868619884.yaml"
error: At least one of apiVersion, kind and name was changed
```

3. Listez l'ensemble des namespaces de ce cluster kubernetes puis l'ensemble des objets kubernetes de votre cluster.

Commande: **kubectl get namespaces**

```
test@Lucas-22:~/kind$ kubectl get namespaces
NAME                 STATUS   AGE
applicatifs          Active   75s
default              Active   5h10m
kube-node-lease      Active   5h10m
kube-public          Active   5h10m
kube-system          Active   5h10m
local-path-storage   Active   5h9m
```

Commande: kubectl get all --all-namespaces

```
test@Lucas-22:~/kind$ kubectl get all --all-namespaces
NAMESPACE           NAME                                                READY   STATUS    RESTARTS        AGE
default             pod/debianpod                                       2/2     Running   1 (33m ago)     93m
default             pod/deuxiemepod                                     1/1     Running   1 (30m ago)     90m
default             pod/hello-nginx-d8bc4c4c6-nzhmg                     1/1     Running   0               4h5m
default             pod/hello-nginx-d8bc4c4c6-whxnl                     1/1     Running   0               4h5m
default             pod/init-demo                                       1/1     Running   0               20m
default             pod/nginx-pod                                       1/1     Running   0               4h8m
kube-system         pod/coredns-5d78c9869d-krp7t                        1/1     Running   0               5h10m
kube-system         pod/coredns-5d78c9869d-lmjw8                        1/1     Running   0               5h10m
kube-system         pod/etcd-tp1k8s-control-plane                       1/1     Running   0               5h10m
kube-system         pod/kindnet-jwkzl                                   1/1     Running   0               5h10m
kube-system         pod/kindnet-kphmw                                   1/1     Running   0               5h10m
kube-system         pod/kindnet-pmc98                                   1/1     Running   0               5h10m
kube-system         pod/kindnet-wfgl5                                   1/1     Running   0               5h10m
kube-system         pod/kube-apiserver-tp1k8s-control-plane             1/1     Running   1 (5h10m ago)   5h10m
kube-system         pod/kube-controller-manager-tp1k8s-control-plane    1/1     Running   3 (4h12m ago)   5h10m
kube-system         pod/kube-proxy-h6kbk                                1/1     Running   0               5h10m
kube-system         pod/kube-proxy-j7pmx                                1/1     Running   0               5h10m
kube-system         pod/kube-proxy-jvvhh                                1/1     Running   0               5h10m
kube-system         pod/kube-proxy-zsk8t                                1/1     Running   0               5h10m
kube-system         pod/kube-scheduler-tp1k8s-control-plane             1/1     Running   3 (4h12m ago)   5h10m
local-path-storage  pod/local-path-provisioner-6bc4bddd6b-hz2pm         1/1     Running   0               5h10m

NAMESPACE     NAME                 TYPE           CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
default       service/hello-nginx  LoadBalancer   10.96.89.6   <pending>     80:32594/TCP             4h5m
default       service/kubernetes   ClusterIP      10.96.0.1    <none>        443/TCP                  5h10m
kube-system   service/kube-dns     ClusterIP      10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   5h10m

NAMESPACE     NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
kube-system   daemonset.apps/kindnet      4         4         4       4            4           kubernetes.io/os=linux   5h10m
kube-system   daemonset.apps/kube-proxy   4         4         4       4            4           kubernetes.io/os=linux   5h10m
```

Commande: kubectl get pods --all-namespaces

```
test@Lucas-22:~/kind$ kubectl get pods --all-namespaces
NAMESPACE           NAME                                                READY   STATUS    RESTARTS        AGE
default             debianpod                                           2/2     Running   1 (34m ago)     94m
default             deuxiemepod                                         1/1     Running   1 (30m ago)     90m
default             hello-nginx-d8bc4c4c6-nzhmg                         1/1     Running   0               4h6m
default             hello-nginx-d8bc4c4c6-whxnl                         1/1     Running   0               4h6m
default             init-demo                                           1/1     Running   0               21m
default             nginx-pod                                           1/1     Running   0               4h9m
kube-system         coredns-5d78c9869d-krp7t                            1/1     Running   0               5h11m
kube-system         coredns-5d78c9869d-lmjw8                            1/1     Running   0               5h11m
kube-system         etcd-tp1k8s-control-plane                           1/1     Running   0               5h11m
kube-system         kindnet-jwkzl                                       1/1     Running   0               5h11m
kube-system         kindnet-kphmw                                       1/1     Running   0               5h11m
kube-system         kindnet-pmc98                                       1/1     Running   0               5h11m
kube-system         kindnet-wfgl5                                       1/1     Running   0               5h11m
kube-system         kube-apiserver-tp1k8s-control-plane                 1/1     Running   1 (5h11m ago)   5h11m
kube-system         kube-controller-manager-tp1k8s-control-plane        1/1     Running   3 (4h13m ago)   5h11m
kube-system         kube-proxy-h6kbk                                    1/1     Running   0               5h11m
kube-system         kube-proxy-j7pmx                                    1/1     Running   0               5h11m
kube-system         kube-proxy-jvvhh                                    1/1     Running   0               5h11m
kube-system         kube-proxy-zsk8t                                    1/1     Running   0               5h11m
kube-system         kube-scheduler-tp1k8s-control-plane                 1/1     Running   3 (4h13m ago)   5h11m
local-path-storage  local-path-provisioner-6bc4bddd6b-hz2pm             1/1     Running   0               5h11m
```

5. A quoi sert le namespace kube-system ? le name-space default ?

- Le **namespace kube-system** est utilisé pour héberger les composants système de Kubernetes qui sont nécessaires pour faire fonctionner le cluster lui-même. Ces composants incluent des choses comme le planificateur (kube-scheduler), le contrôleur de nœuds (kube-controller-manager), et d'autres éléments centraux du système Kubernetes.
- Le **namespace default** est le namespace par défaut où les ressources sont créées si aucun autre namespace n'est spécifié. Cela signifie que si vous créez un objet Kubernetes sans spécifier de namespace, il sera automatiquement placé dans le namespace default.

5.2 Déployer une application Python dans Kubernetes

5.2.1 "Deployment" de l'application

1.Créez un "deployment" avec "kubectl create" utilisant l'image registry.iutbeziers.fr/pythonapp:latest dans le namespace applicatifs.

Création d'environnement:

```
test@Lucas-22:~/kind$ python3 -m venv env
test@Lucas-22:~/kind$ source env/bin/activate
(env) test@Lucas-22:~/kind$
```

Creation du deployment

kubectl create deployment python-app --image=registry.iutbeziers.fr/pythonapp:latest --namespace=applicatifs

```
(env) test@Lucas-22:~/kind$ kubectl create deployment python-app --image=registry.iutbeziers.fr/pythonapp:latest --namespace=applicatifs
deployment.apps/python-app created
(env) test@Lucas-22:~/kind$
```

2. Dumper la configuration de ce "déploiement" dans un fichier yaml. Éditez ce fichier et expliquez les différents objets qui le composent.

Commande: **kubectl get deployment python-app -n applicatifs -o yaml > deployment.yaml**

```
(env) test@Lucas-22:~/kind$ kubectl get deployment python-app -n applicatifs -o yaml > deployment.yaml
(env) test@Lucas-22:~/kind$ ls
apache.yml  deployment.yaml  env  mondeuxiemepod.yml  monpremierpod.yml  nginx-pod.yaml
```

Visualisation du fichier:

```
(env) test@Lucas-22:~/kind$ cat deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "1"
  creationTimestamp: "2023-10-17T14:26:48Z"
  generation: 1
  labels:
    app: python-app
  name: python-app
  namespace: applicatifs
```

3. En modifiant le "deployment" passer à trois "pods" Python.

Commande: **kubectl scale --replicas=3 deployment/python-app -n applicatifs**

kubectl get pods -n applicatifs

```
(env) test@Lucas-22:~/kind$ kubectl scale --replicas=3 deployment/python-app -n applicatifs
deployment.apps/python-app scaled
```

5.2.2 Utilisation des labels

1. Affichez les pods dont le label est "app=pythonapp". Rajoutez un label "env=dev". Vérifiez la prise en tenant compte de votre action. Enlevez le label app. Détruisez les pods labellisés "app=pythonapp". Supprimez au final le déploiement.

Afficher: **kubectl get pods -n applicatifs -l app=python-app**

```
(env) test@Lucas-22:~/kind$ kubectl get pods -n applicatifs -l app=python-app
NAME                          READY   STATUS    RESTARTS   AGE
python-app-6b98bbf48d-gmhbq   1/1     Running   0          17m
python-app-6b98bbf48d-nq577   1/1     Running   0          22m
python-app-6b98bbf48d-z6fgp   1/1     Running   0          17m
```

Ajouter les labels: **kubectl label pods -n applicatifs -l app=python-app env=dev**

```
(env) test@Lucas-22:~/kind$ kubectl label pods -n applicatifs -l app=python-app env=dev
pod/python-app-6b98bbf48d-gmhbq labeled
pod/python-app-6b98bbf48d-nq577 labeled
pod/python-app-6b98bbf48d-z6fgp labeled
(env) test@Lucas-22:~/kind$
```

Vérifier la prise en compte du label env=dev: **kubectl get pods -n applicatifs -l env=dev**

```
(env) test@Lucas-22:~/kind$ kubectl get pods -n applicatifs -l env=dev
NAME                          READY   STATUS    RESTARTS   AGE
python-app-6b98bbf48d-gmhbq   1/1     Running   0          20m
python-app-6b98bbf48d-nq577   1/1     Running   0          25m
python-app-6b98bbf48d-z6fgp   1/1     Running   0          20m
(env) test@Lucas-22:~/kind$
```

Commande: **kubectl delete pods -n applicatifs -l app=python-app**
Destruction de pods labelisez.

```
(env) test@Lucas-22:~/kind$ kubectl delete pods -n applicatifs -l app=python-app
pod "python-app-6b98bbf48d-gmhbq" deleted
pod "python-app-6b98bbf48d-nq577" deleted
pod "python-app-6b98bbf48d-z6fgp" deleted
```

Suppression du déploiement: **kubectl delete deployment python-app -n applicatifs**

```
(env) test@Lucas-22:~/kind$ kubectl delete deployment python-app -n applicatifs
deployment.apps "python-app" deleted
(env) test@Lucas-22:~/kind$
```

2. Labellisez deux nodes "statusnode=maintenance". Drainer les nodes en maintenance en utilisant le label.

kubectl label nodes tp1k8s-worker statusnode=maintenance
kubectl label nodes tp1k8s-worker2 statusnode=maintenance

```
(env) test@Lucas-22:~/kind$ kubectl label nodes tp1k8s-worker statusnode=maintenance
node/tp1k8s-worker labeled
(env) test@Lucas-22:~/kind$ kubectl label nodes tp1k8s-worker2 statusnode=maintenance
node/tp1k8s-worker2 labeled
(env) test@Lucas-22:~/kind$
```

kubectl drain tp1k8s-worker --ignore-daemonsets --force

```
(env) test@Lucas-22:~/kind$ kubectl drain tp1k8s-worker --ignore-daemonsets --force
node/tp1k8s-worker already cordoned
Warning: deleting Pods that declare no controller: default/deuxiemepod; ignoring DaemonSet-managed Pods: kube-system/kindnet-pmc98, kube-system/kube-proxy-j7pmx
evicting pod default/hello-nginx-d8bc4c4c6-whxnl
evicting pod default/deuxiemepod
pod/hello-nginx-d8bc4c4c6-whxnl evicted
pod/deuxiemepod evicted
node/tp1k8s-worker drained
```

kubectl drain tp1k8s-worker2 --ignore-daemonsets --force --delete-emptydir-data

```
(env) test@Lucas-22:~/kind$ kubectl drain tp1k8s-worker2 --ignore-daemonsets --force --delete-emptydir-data
node/tp1k8s-worker2 already cordoned
Warning: ignoring DaemonSet-managed Pods: kube-system/kindnet-kphmw, kube-system/kube-proxy-h6kbk; deleting Pods that declare no controller: default/init-demo, default/nginx-pod
evicting pod default/nginx-pod
evicting pod default/init-demo
evicting pod default/hello-nginx-d8bc4c4c6-czd6b
pod/nginx-pod evicted
pod/hello-nginx-d8bc4c4c6-czd6b evicted
pod/init-demo evicted
node/tp1k8s-worker2 drained
```

3. En utilisant l'adresse obtenue lors de l'affichage du POD essayer d'accédez à l'application en CLI via l'utilitaire curl en dehors du cluster? que constatez-vous ? Essayez depuis un nœud du cluster.

Pour récupérer l'IP du POD: **kubectl get svc -n applicatifs**

```
(env) test@Lucas-22:~/kind$ kubectl get svc -n applicatifs
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
python-app-service  ClusterIP   10.96.187.120   <none>        80/TCP    7s
```

J'ai constaté que pour y accéder depuis un nœud Cluster il me faut une étape supplémentaire à prendre en compte je dois configurer des règles de pare-feu ou des politiques réseau pour permettre la communication entre les nœuds du cluster.

4. Modifiez le déploiement pythonapp afin que le pod s'exécute sur node3.

Sur mon fichier yaml j'ai rajouté:

```
      nodeSelector:  # Ajout du nodeSelector
        nodetype: "node3"
```

Indique à Kubernetes de planifier ce pod uniquement sur des nœuds ayant le label nodetype: "node3".

Ensuite j'ai labellisé le noeud: **kubectl label nodes tp1k8s-worker3 nodetype=node3**

```
(env) test@Lucas-22:~/kind$ kubectl label nodes tp1k8s-worker3 nodetype=node3
node/tp1k8s-worker3 labeled
```

Commande: **k apply -f deployment.yaml**

```
(env) test@Lucas-22:~/kind$ k apply -f deployment.yaml
deployment.apps/python-app created
```

5. Taintez le node "master" an que ne soit plus schedulé à l'avenir de pods dessus.
kubectl taint nodes tp1k8s-control-plane node-role.kubernetes.io/master:NoSchedule

```
(env) test@Lucas-22:~/kind$ kubectl taint nodes tp1k8s-control-plane  node-role.kubernetes.io/master:NoSchedule
node/tp1k8s-control-plane tainted
```

6. Utilisez les fonctionnalités de taint & toleration pour qu'un pod soit schedulé sur node1 - (Taintez diéremment tout vos noeuds et lancez un pod avec une toleration pour node1).Expliquez quel est le principe de "taint & toleration".
R: Le principe de "Taint & Toleration" dans Kubernetes permet de spécifier quelles pods sont autorisées à être planifiées sur quel nœud, en fonction des taints appliqués à ce nœud et des tolérations définies pour les pods.

```
(env) test@Lucas-22:~/kind$ kubectl apply -f taint.yaml
pod/pod-1 created
(env) test@Lucas-22:~/kind$
```

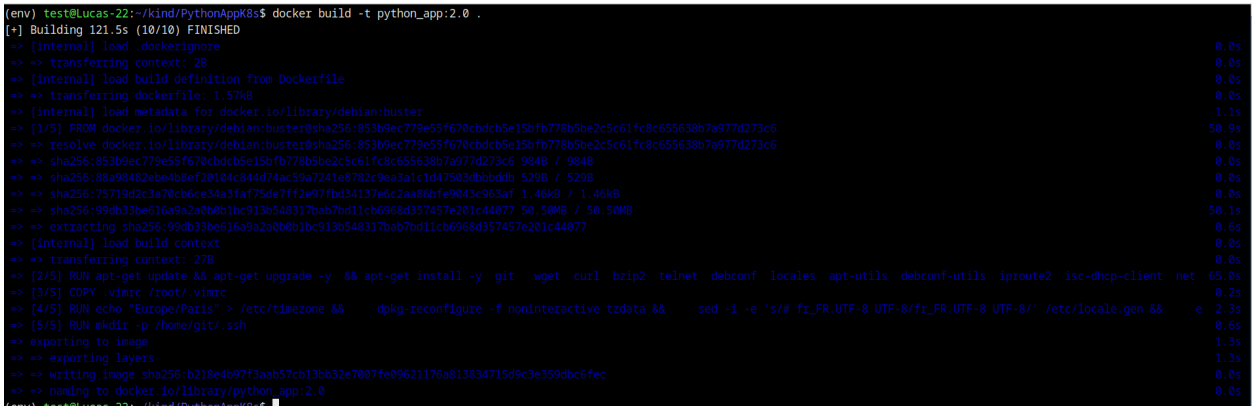
```
(env) test@Lucas-22:~/kind$ cat taint.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-1
  labels:
    security: s1
spec:
  containers:
  - image: registry.iutbeziers.fr/debianiut:latest
    name: debianiut
    command: ["/bin/bash", "-c", "--"]
    args: ["while true; do sleep infinity; done;"]
  tolerations:
  - key: "node"
    operator: "Equal"
    value: "1"
    effect: "NoSchedule"
```

5.3 Réaliser une migration de version de l'image Python

git clone https://registry.iutbeziers.fr:11443/pouchou/PythonAppK8s.git

```
(env) test@Lucas-22:~/kind$ git clone https://registry.iutbeziers.fr:11443/pouchou/PythonAppK8s.git
Clonage dans 'PythonAppK8s'...
remote: Enumerating objects: 38, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 38 (delta 0), reused 0 (delta 0), pack-reused 35
Réception d'objets: 100% (38/38), 5.28 Kio | 5.28 Mio/s, fait.
Résolution des deltas: 100% (16/16), fait.
```

docker build -t python_app:2.0 .



6 Services

6.1 Service de type NodePort

1. On va utiliser la commande impérative suivante afin de générer le manifeste du service de type node port.

```
test@202-13:~/kind/PythonAppK8s$ kubectl create deployment python-app --image=python-app-image
deployment.apps/python-app created
```

k expose deployment python-app --name pythonnp --target-port=5000 --port=9002 --type=NodePort --dry-run=client -o yaml > nodeport.yaml

```
test@202-13:~/kind$ k expose deployment python-app --name pythonnp --target-port=5000 --port=9002 --type=NodePort --dry-run=client -o yaml > nodeport.yaml
test@202-13:~/kind$ ls | grep node
nodeport.yaml
test@202-13:~/kind$
```

Fichier nodeport.yaml

```
test@202-13:~/kind$ cat nodeport.yaml
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: python-app
  name: pythonnp
spec:
  ports:
  - port: 9002
    protocol: TCP
    targetPort: 5000
  selector:
    app: python-app
  type: NodePort
status:
  loadBalancer: {}
```

2. Modifiez le fichier en prenant comme nodeport 31234.

```
apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: python-app
  name: pythonnp
spec:
  ports:
  - port: 31234
    protocol: TCP
    targetPort: 5000
  selector:
    app: python-app
  type: NodePort
status:
  loadBalancer: {}
```

Commande: **k apply -f nodeport.yaml**

```
test@202-13:~/kind$ k apply -f nodeport.yaml
service/pythonnp created
test@202-13:~/kind$
```

3.Quelle est la différence entre port/nodeport/target-port ? testez les nodes et le service.

- **Port :** Le champ port spécifie le port sur lequel le service sera exposé au sein du cluster Kubernetes. C'est le port auquel les autres pods à l'intérieur du cluster peuvent se connecter pour accéder au service.
- **NodePort :** NodePort est le port auquel le service sera exposé au niveau de chaque nœud du cluster. Il permet d'accéder au service à partir de l'extérieur du cluster en utilisant l'adresse IP d'un des nœuds du cluster ainsi que le port NodePort spécifié.
- **TargetPort :** Le champ targetPort spécifie le port sur lequel les pods du service ciblé écoutent. Cela permet au service de rediriger le trafic vers le port correct du pod.

Commande: **kubectl get services**

```
test@202-13:~/kind$ kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP           2d3h
pythonnp     NodePort    10.96.164.169   <none>        31234:30820/TCP   15m
```

4. Quels sont les inconvénients d'un service Node-Port ?Que manque-t-il dans cette architecture ?

- **Exposition sur tous les nœuds :** Un service NodePort expose une application sur tous les nœuds du cluster. Cela signifie que l'application est accessible sur l'ensemble du cluster, ce qui peut être un risque de sécurité si vous ne le configurez pas correctement.
- **Port limité :** Le port NodePort est limité à un certain intervalle de ports (30000-32767 par défaut dans Kubernetes). Cela peut devenir un problème si ces ports sont déjà utilisés ou si vous avez besoin d'exposer de nombreuses applications.
- **Nécessite une gestion des règles de pare-feu :** Vous devez gérer les règles de pare-feu pour contrôler l'accès à l'application exposée via le port NodePort. Sinon, l'application sera accessible publiquement à partir de n'importe quelle adresse IP.
- **Pas de gestion du trafic :** Un service NodePort n'a pas de mécanisme de gestion du trafic avancé, comme la répartition de charge ou le routage basé sur les noms d'hôte. Cela signifie que le trafic est simplement redirigé vers le pod correspondant, sans autre traitement.
- **Pas de support SSL natif :** Le service NodePort ne prend pas en charge SSL nativement. Si vous souhaitez utiliser HTTPS, vous devez mettre en place une solution supplémentaire, comme un Ingress Controller avec un certificat SSL.

5. Récupérez les "endpoints" reliés au service.
kubectl get endpoints pythonnp -n applicatifs

```
test@202-13:~/kind$ kubectl get endpoints pythonnp -n applicatifs
NAME       ENDPOINTS                                            AGE
pythonnp   10.244.1.11:5000,10.244.2.5:5000,10.244.3.7:5000   69m
```

6. Complétez l'architecture avec un équilibreur de charge externe (un haproxy à installer par package) ?
Haproxy.cfg config: **kubectl create configmap haproxy-config --from-file=haproxy.cfg**

```
test@202-13:~/kind$ kubectl create configmap haproxy-config --from-file=haproxy.cfg
configmap/haproxy-config created
```

CANJALE LUCAS

Fichier haproxy.cfg:

```
global
    log /dev/log    local0
    log /dev/log    local1 notice
    chroot /var/lib/haproxy
    stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd listeners
    stats timeout 30s

defaults
    log global
    mode http
    option httplog
    option dontlognull
    timeout connect 5000
    timeout client  50000
    timeout server  50000

frontend myfrontend
    bind *:80
    default_backend mybackend

backend mybackend
    server server1 10.96.0.1:443
    server server2 10.244.1.11:5000
```

Haproxy deployment fichier:

```
test@202-13:~/kind$ cat haproxy-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: haproxy-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: haproxy
  template:
    metadata:
      labels:
        app: haproxy
    spec:
      containers:
      - name: haproxy
        image: haproxy:latest
        ports:
        - containerPort: 80
        volumeMounts:
        - name: haproxy-config
          mountPath: /usr/local/etc/haproxy
      volumes:
      - name: haproxy-config
        configMap:
          name: haproxy-config
```

Commande: **k apply -f haproxy-deployment.yaml**

```
test@202-13:~/kind$ k apply -f haproxy-deployment.yaml
deployment.apps/haproxy-deployment created
```

```
test@202-13:~/kind$ k get services
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
haproxy-service   LoadBalancer   10.96.41.181    <pending>     80:32353/TCP      23m
kubernetes        ClusterIP      10.96.0.1       <none>        443/TCP           2d5h
pythonnp          NodePort       10.96.172.101   <none>        31234:31034/TCP   101m
```

7. Dessinez un schéma de cette architecture sur papier.

6.2 Service de type LoadBalancer

1. Créer un service de type loadbalancer. Pourquoi son EXTERNAL-IP est elle pending ?
Le statut "pending" de l'EXTERNAL-IP signifie que le service LoadBalancer est en attente d'être provisionné. Cela se produit lorsque vous exécutez Kubernetes sur une plateforme qui ne prend pas en charge nativement les services de type LoadBalancer, comme un cluster local ou une installation sur un cloud qui nécessite une configuration supplémentaire.

```
test@202-13:~/kind$ k get services
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)           AGE
haproxy-service   LoadBalancer   10.96.41.181    <pending>     80:32353/TCP      23m
kubernetes        ClusterIP      10.96.0.1       <none>        443/TCP           2d5h
pythonnp          NodePort       10.96.172.101   <none>        31234:31034/TCP   101m
```

2. Installer et utiliser Metallb [1] en mode L2 an de rendre le service loadbalancer accessible depuis l'extérieur.
Vous utiliserez la méthode "d'installation par manifest" et vous configurez une plage IP (voir "De ning the IPs to assign to the Load Balancer services")

```
test@202-13:~$ k get service
NAME                              TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)                                                  AGE
haproxy-service                   LoadBalancer   10.96.41.181    172.21.0.5    80:32353/TCP                                             2d15h
kubernetes                        ClusterIP      10.96.0.1       <none>        443/TCP                                                  4d21h
mycontroller-kubernetes-ingress   NodePort       10.96.59.249    <none>        80:30080/TCP,443:31450/TCP,1024:31315/TCP,6060:30086/TCP  2d14h
pythonnp                          NodePort       10.96.172.101   <none>        31234:31034/TCP                                          2d17h
```

On voit bien que le service haproxy a déjà une adresse externe.
Fichier ipaddr.yaml

```
test@202-13:~/kind$ cat ipaddr.yaml
apiVersion: metallb.io/v1beta1
kind: IPAddressPool
metadata:
  name: first-pool
  namespace: metallb-system
spec:
  addresses:
    - 172.21.0.5-172.21.0.10
```

7.Ingress

1. Installer helm

```
test@202-13:~/kind$ tar -zxvf helm-v3.13.1-linux-amd64.tar.gz linux-amd64/
linux-amd64/
linux-amd64/LICENSE
linux-amd64/helm
linux-amd64/README.md
```

Move to bin:

```
test@202-13:~/kind/linux-amd64$ mv helm /usr/local/bin/helm
mv: impossible de déplacer 'helm' vers '/usr/local/bin/helm': Permission non accordée
test@202-13:~/kind/linux-amd64$ sudo !!
sudo mv helm /usr/local/bin/helm
[sudo] Mot de passe de test :
test@202-13:~/kind/linux-amd64$
```

Helm help

```
test@202-13:~/kind/linux-amd64$ helm help
The Kubernetes package manager

Common actions for Helm:

- helm search:    search for charts
- helm pull:      download a chart to your local directory to view
- helm install:   upload the chart to Kubernetes
- helm list:      list releases of charts
```

2. Déployez le package de l' "ingress controller haproxy" en vous aidant de:

```
test@202-13:~/kind$ helm list
NAME            NAMESPACE       REVISION        UPDATED                                 STATUS          CHART                           APP VERSION
mycontroller    default         3               2023-10-23 08:53:05.904687714 +0200 CEST        deployed        kubernetes-ingress-1.33.1       1.10.8
```

3. Utilisez cet ingress an de rendre accessible l'application Python.

```
test@202-13:~/kind$ helm install moncontrollerhap haproxytech/kubernetes-ingress \
--set controller.kind=DaemonSet \
--set controller.daemonset.useHostPort=true \
--set-string "controller.config.ssl-redirect=false" \
--set controller.service.type=LoadBalancer
NAME: moncontrollerhap
LAST DEPLOYED: Mon Oct 23 09:26:58 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
HAProxy Kubernetes Ingress Controller has been successfully installed.
```

Cette commande installe et configure un contrôleur Ingress qui utilise le type DaemonSet pour s'exécuter sur chaque nœud, expose le service en utilisant le port hôte, désactive la redirection HTTPS automatique et définit le type de service comme LoadBalancer pour obtenir une adresse IP externe.

8. Stockage persistent et non réparti
1. Quel est l'intérêt de faire du stockage local ? du stockage réparti.

Le stockage local se caractérise par sa rapidité, le contrôle total des données et un coût initial moins élevé. Il offre des performances prévisibles, mais peut être limité en termes de capacité et de tolérance aux pannes.

En revanche, le stockage réparti permet de gérer de grandes quantités de données et offre une grande évolutivité. Il garantit la redondance des données, une haute disponibilité et des performances évolutives. Il peut être plus coûteux initialement, mais offre des économies d'échelle à mesure que les besoins augmentent.

2.

```
test@202-13:~/kind$ k get pv -A
No resources found
test@202-13:~/kind$ k get pvc -A
No resources found
test@202-13:~/kind$ k get storageclass -A
NAME                 PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
standard (default)   rancher.io/local-path   Delete          WaitForFirstConsumer   false                  5d
test@202-13:~/kind$
```

3.