TP Redes

Instalacion

- Instalar Docker
- Instalar Kubernetes
- Instalar kind
- Descargar el repositorio

```
git clone git@github.com:alavarello/tp-redes.git
```

Pasos a seguir

Todos estos comandos se corren desde la carpeta inicial del repositorio

```
cd tp-redes
```

1) Iniciar el cluster usando kind

Kind levanta nodos en docker. Usando el archivo **cluster.yml** podemos configurar como se estructura el cluster. Hay dos tipos de nodos: masters y workers.

```
# Para levantar el cluster
kind create cluster --config cluster.yml
# Para ver los contextos (el cluster deberia estar en el contexto kind-kind)
kubectl config get-contexts
```

2) Levantar el dashboard

El dashboard sirve para administrar y visualizar el cluster de Kubernetes de una mejor manera.

```
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.1/aio/deploy/recommended.yam

# Crear permisos
kubectl create -f dashboard/config.yml
# Generar token y copiarlo para poder hacer login
kubectl -n kubernetes-dashboard describe secret $(kubectl -n kubernetes-dashboard get
secret | grep admin-user | awk '{print $1}')
# Ejecutar en otra terminal o correrlo como un subproceso
kubectl proxy
```

Esperar hasta que todos los pods esten corriendo

```
# Para fijarse el estado de los pods
kubectl get pods -n kubernetes-dashboard
```

Al correr el ultimo comando entrar aqui y pegar el token

3) Levantar una base de datos local

Levantamos una base de datos postgresql en un docker. Este contenedor llamado **database** esta en la misma red (**kind**) que los nodos del cluster. De esta manera el cluster se puede comunicar con el servidor de la base de datos. La base de datos de postgresql que se crea se llama **postgres** con un usuario **postgres** y contraseña **123456**

```
docker container run --network=kind --name database -e POSTGRES_PASSWORD=123456 -d
postgres
```

4) Levantar un nginx

Levantamos un contenedor que tenga una imagen nginx Este escucha en el puerto 80 y lo redirige a los nodos.

```
docker build -t custom-nginx images/nginx/
docker run --network=kind --name nginx -d -p 80:80 custom-nginx
```

5) Deployar la version alpha de la API

La version alpha de la API se puede encontrar como una imagen de docker `alavarello/test-api:alpha`

```
# Primero se crean los secretos
kubectl apply -f deployments/secrets.yml
# Luego se crea el servicio de la base de datos
kubectl apply -f deployments/database.yml
# Despues se deploya la version alpha de la API
kubectl apply -f deployments/alpha.yml
# Deployamos nginx
kubectl apply -f deployments/nginx.yml
```

6) Interactuar con la API

Hay dos formas de interactuar con la API. Se puede hacer desde un docker dentro de la red kind o usando localhost en la computadora.

6.1) Endpoints

```
GET <host>/<api-version>/students/
POST <host>/students/ # Data: username=agus y email=agus@redes.com
```

En esta instancia **api-version** es **v1**. Cuando se exponga la version beta **api-version** es **v2**

6.2) Formas de acceder a la API

6.2.1) USAR LOCALHOST

En este caso se puede usar una aplicacion como postman para hacer los post o mismo curl. El host en este caso es **localhost**

```
# GET para obtener todos los estudiantes
curl localhost/v1/students/
# POST para crear un estudiante
curl --data "username=agus&email=agus@redes.com" localhost/v1/students/
```

6.2.2) Usar un host en la red kind para acceder a las API

Creamos una contenedor de docker dentro de la red **kind** para poder acceder mediante curl a la API. Este contenedor es efimero, cuando se cierre el programa bash se destruye el contenedor. El contenedor esta basado en una imagen que tiene ubuntu y curl

```
docker container run -it --network=kind alavarello/custom-curl
```

```
# GET para obtener todos los estudiantes
curl nginx/<api-version>/students/
# POST para crear un estudiante
curl --data "username=agus&email=agus@redes.com" nginx/v1/students/
```

7) Subir la version beta de la API

Para la version beta de la API usamos la imagen de docker `alavarello/test-api:beta` que tiene unos cambios en como representar un estudiante (el get en el alpha muestra el nombre mientras que en la beta muestra el email).

```
# Subimos algunos Pods de la version beta y bajamos la cantidad de Pods de la version alpha
kubectl apply -f deployments/beta-canary.yml
# Cambiamos el config map de nginx
kubectl apply -f deployments/nginx-canary-map.yml
# Hacemos un rollout de nginx
kubectl rollout restart deployment nginx
```

En este caso se tiene las dos APIs conviviendo al mismo tiempo

8) Sacar la version alpha

```
# Subimos la cantidad total de nodos que queremos de la version beta
kubectl apply -f deployments/beta.yml
# Cambiamos nginx config
kubectl apply -f deployments/nginx-beta-map.yml
kubectl rollout restart deployment nginx
# Borramos el deployment de alpha
kubectl delete -f deployments/alpha.yml
```

9) Otras pruebas

Algunos comandos extra que se corrieron en las pruebas

```
# Para detener el funcionamiento de un nodo
docker stop kind-worker
# Levantar el nodo que detuvimos anteriormente
docker start kind-worker
# Obtener los Pods corriendo
kubectl get pods
# Borrar un pod
kubectl delete pods  <pod> # Replazar <pod> por el nombre del Pod
```

10) Para finalizar

Para terminar y eliminar todo lo que se uso en la prueba

```
# Para borrar el cluster
kind delete cluster
# Eliminar la base de datos
docker rm -f database
```

Nota: El nodo que se uso para hacer los curls se remueve cuando se sale de la terminal.