Práctica: Kubernetes

Grado en Ingeniería Informática

kind
https://kind.sigs.k8s.io/

- **kind** es una herramienta que define un entorno local para la ejecución de Kubernetes.
- **kind** fue diseñado principalmente para probar Kubernetes, pero puede usarse para desarrollo local.
- La alternativa a **kind** es **Minikube**, son herramientas muy similares, pero en el caso de **kind**, los clusters están dentro de un contenedor Docker, mientras en **Minikube** están en vez de una máquina virtual
- **kind** es más ligero que Minikube. Instalación de kind:

```
$ 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
$ kind version # Comprobamos la versión de kind
```

kind

- Creación de un cluster básico. (Docker debe estar funcionando. Instalarlo antes si no está instalado)

```
$ kind create cluster
$ kind create cluster --name kind-2 # Crear un cluster con un nombre específico
$ kind get clusters # Obtener un listado de clusters
$ kind delete cluster # Borrar cluster por defecto
$ kind delete cluster --name kind-2 # Borrar cluster kind-2
```

- El cluster creado es con un imagen predefinida en **kind**. Esta imagen por defecto se puede cambiar

- Crear cluster con una configuración determinada

```
$ kind create cluster --config kind-example-config.yaml
```

- Fichero `kind-example-config.yaml`:

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
name: kind3
nodes:
- role: control-plane
- role: worker
- role: worker
```

Kubectl

- **kubectl** es una herramienta que permite la configuración y administración de Kubernetes
- Los primeros pasos van a consistir en instalar kubectl
- La orden `cat` introduce el contenido posterior (sin EOF) en /etc/yum.repos.d/kubernetes.repo

```
$ curl -LO https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl
(la anterior orden va en la misma línea)

$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

kubectl

- **kubectl** es una herramienta que permite la configuración y administración de kubernetes
- Inicialmente se comprobará que todo está en funcionamiento, tanto **kubectl**, como el cluster creado

```
$ kubectl version
$ kubectl cluster-info
$ kubectl cluster-info --context kind-kind3 #Info de un cluster determinado
$ kubectl get nodes
```

Creando Deployments

- Vamos a crear un Deployment de una aplicación concretamente de un contenedor con ***Nginx***.
- Una alternativa es hacerlo de forma imperativa

```
$ kubectl create deployment nginx-deployment --image=nginx --port=8800
```

- Se usa una imagen existente en Docker Hub
- Comprobar que está funcionando

```
$ kubectl get deployments
```

Creando Deployments

- A partir de ficheros (p.ej: `nginx-deployment.yaml`)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

- En este fichero se especifica que vamos a desplegar la imagen `nginx` (alojada en Docker Hub)
- Simplemente con cambiar la imagen por otra de Docker Hub estaremos cambiando la image que se despliega

```
image: arivera/nginx:latest
```

Creando Deployments

- Puesta en marcha del Deployment a partir del fichero anterior.
- Después se monitoriza el Deployments, Replicas, Pods, …

```
$ kubectl apply -f nginx-deployment.yaml
$ kubectl get deployment nginx-deployment
$ kubectl describe deployment nginx-deployment
$ kubectl get replicasets # Obtener los Replicaset (similar a kubectl get rs)
$ kubectl describe replicasets -l app=nginx # Especificando el Deployment
$ kubectl scale deployment nginx-deployment --replicas=2 # Reescalar réplicas
$ kubectl get pods -l app=nginx --output=wide # Ver los Pods
```

Creando/exponiendo servicios

- Una vez creado un Deployment habrá que asignarle una serie de recursos (direcciones IP, puertos, …) para hacer la aplicación accesible:
- Hay tres tipos de servicios:
  - ClusterIP: es el predeterminado y crea solamente un servicio interno. No se puede acceder al servicio desde fuera. Se recomienda para comunicaciones intra-cluster
  - NodePort: Se expone un servicio normalmente dentro de una red privada interna. Se usa para pruebas. Permite visitar la aplicación visitando conectándote específicamente a cualquiera de los nodos del cluster node2-ip:port (por ejemplo).
  - LoadBalancer: Define un acceso único público a la aplicación, se define principalmente en proveedores de servicios en la nube (que asignan recursos de forma automática)

Creando/exponiendo servicios

- Con “kubectl expose deployment …” se crea un servicio de tipo NodePort, con asignaciones de puertos por defecto.
- En este caso no se han utilizado las opciones --port (para definir el puerto del servicio interno) o --target-port (para definir el puerto del contenedor, que nosotros definimos en el Deployment en el puerto 80)
- El tráfico se enrutará desde nuestro nodo en el puerto 32178 al servicio interno (10.96.168.80:8080), y luego desde el servicio a nuestro contenedor (puerto 80)

```
$ kubectl expose deployment nginx-deployment --type=NodePort --name my-nginx-service
$ kubectl get services # Chequeando el servicio
NAME               TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
my-nginx-service   NodePort    10.96.168.80   <none>        80:32178/TCP   40s
```

Creando/exponiendo servicios

- A continuación chequemos la dirección IP de nuestros nodos.
- Con esta dirección IP y el puerto asignado anteriormente (32178) podemos acceder a nuestra aplicación.

```
$ kubectl get nodes -o wide # Volvemos a chequear los nodos
.. VERSION   INTERNAL-IP   ..
.. v1.25.0   172.18.0.2   ..
$ curl 172.18.0.2:32178
Este mi nginx
```