Práctica Kubernetes

Oscar H. Mondragón

1. Objetivo

Comprender el funcionamiento de Kubernetes, una plataforma de orquestación de contenedores

2. Herramientas a utilizar

- Docker
- Kubernetes (minikube)
- Vagrant
- VirtualBox

3. Desarrollo de la Práctica

NOTA: Si su instalación local de minikube presenta problemas, puede usar uno de los siguientes playgrounds para la práctica

Katacoda

https://www.katacoda.com/courses/kubernetes/playground

Play with Kubernetes

https://labs.play-with-k8s.com/

Tutoriales Interactivos en la pagina de Kubernetes

https://kubernetes.io/docs/tutorials/kubernetes-basics/

3.1. Vagrantfile

Esta práctica la desarrollaremos usando un Box de Ubuntu 22.04 en Vagrant con al menos dos CPUs y 3GB de memoria. El Vagrantfile que usaremos es el siguiente:

```
# -*- mode: ruby -*-
```

```
# vi: set ft=ruby :

Vagrant.configure("2") do |config|

config.vm.define :servidorUbuntu do |servidorUbuntu|
    servidorUbuntu.vm.box = "bento/ubuntu-22.04"
    servidorUbuntu.vm.network :private_network, ip: "192.168.100.3"
    servidorUbuntu.vm.hostname = "servidorUbuntu"
    servidorUbuntu.vm.provider "virtualbox" do |v|
        v.cpus = 2
        v.memory = 3072
    end
  end
end
```

3.2. Instalar y Ejecutar Minikube en Linux

En esta practica utilizaremos Minikube una herramienta para correr Kubernetes localmente. Minikube corre un cluster Kubernetes de un único nodo en una maquina virtual.

Instale minikube

```
curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 \
  && chmod +x minikube
```

Para tener disponible en la consola el comando minikube, puedes añadir el comando al $PATH o moverlo por ejemplo a /usr/local/bin:

```
sudo cp minikube /usr/local/bin && rm minikube
```

Instale kubectl, una interfaz de línea de comandos para ejecutar comandos en clusters Kubernetes.

```
sudo apt-get update && sudo apt-get install -y apt-transport-https
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee -a
/etc/apt/sources.list.d/kubernetes.list
sudo apt-get update
sudo apt-get install -y kubectl
```

3.3. Desplegar un cluster de Kubernetes para desarrollo local usando minikube

Cree el grupo de docker y agregue el usuario vagrant:

1. Cree el grupo docker (es probable que ya este creado)

```
$ sudo groupadd docker
```

2. Agregue su usuario a el grupo docker

```
$ sudo usermod -aG docker $USER
```

3. Cierre sesión en la máquina vagrant y vuelva a ingresar

```
$ exit
$ vagrant ssh servidorUbuntu
```

Verifique que minikube esta instalado correctamente, corriendo el comando para ver su versión:

```
$ minikube version
```

Inicie el cluster

```
$ minikube start
Starting local Kubernetes cluster...
Kubernetes is available at https://192.168.99.100:8443.
Kubectl is now configured to use the cluster.
```

Verifique la versión del cluster

```
kubectl version
```

Visualice detalles del cluster creado

```
kubectl cluster-info
```

Para mirar los nodos creados en el cluster ejecute,

```
$ kubectl get nodes
NAME       STATUS AGE
minikubevm Ready  3d
```

Inspeccione los pods que estan corriendo

```
$ kubectl get pods --all-namespaces
NAMESPACE   NAME                            READY STATUS  RESTARTS AGE
kube-system kube-addon-manager-minikubevm 1/1   Running 3        3d
kube-system kubernetes-dashboard-06y41    1/1   Running 3        3d
```

3.3. Instale el pod hello-minikube

Minikube viene con una imagen Docker pre-constuida llamada hello-minikube. Ejecute la imagen hello-minikube,

```
$ kubectl create deployment hello-minikube --
image=gcr.io/google_containers/echoserver:1.4 --port=8080
deployment.apps/hello-minikube created
```

Podemos ahora verificar los pods y deployments para verificar que han sido actualizados ejecutando los siguientes comandos,

```
$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
hello-minikube-6d4df66d87-ql9fh   1/1     Running   0          119s
```

```
~$ kubectl get deployments
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
hello-minikube   1/1     1            1           2m51s
```

Ver los eventos del clúster:

```
$ kubectl get events
```

Ver la configuración kubectl:

```
$ kubectl config view
```

Crear un service

Para acceder al servicio de hello-minikube, primero debemos exponer el deployment a la red externa vía el comando,

```
$ kubectl expose deployment hello-minikube --type=NodePort
service/hello-minikube exposed
```

El flag `--type=NodePort` expone el Servicio en la IP de cada Nodo en un puerto estático (el NodePort).

Podemos verificar si el servicio fue expuesto apropiadamente listando los servicios,

```
$ kubectl get services
NAME             TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
hello-minikube   NodePort    10.104.149.73   <none>        8080:32479/TCP   39s
kubernetes       ClusterIP   10.96.0.1       <none>        443/TCP          24h
```

Podemos ahora usar el web browser o el comando `curl` para acceder al servicio.

Para chequear la IP externa asignada y el puerto, podemos usar el comando,

```
$ minikube service hello-minikube --url
http://172.17.0.2:32479
```

```
$ curl $(minikube service hello-minikube --url)
CLIENT VALUES:
client_address=172.18.0.1
command=GET
real path=/
query=nil
request_version=1.1
request_uri=http://172.17.0.2:8080/

SERVER VALUES:
server_version=nginx: 1.10.0 - lua: 10001

HEADERS RECEIVED:
accept=*/*
host=172.17.0.2:32479
user-agent=curl/7.68.0
BODY:
-no body in request-
```

Note que puede encontrar la dirección ip usando el comando `minikube ip`

Puede verificar los logs del pod con el comando `kubectl logs -f <nombre del pod>` . El nombre del pod lo puede verificar con `kubectl get pods.` En los logs se puede verificar la petición get que se hizo con la prueba:

```
vagrant@servidorUbuntu:~$ kubectl logs -f hello-minikube-6d4df66d87-ql9fh
172.18.0.1 - - [07/Oct/2020:14:09:31 +0000] "GET / HTTP/1.1" 200 388 "-"
"curl/7.68.0"
```

Para salir presione CTRL+C

Dashboard Kubernetes

Para abrir el tablero de Kubernetes dashboard en un navegador, puede seguir los siguientes pasos, en dos terminales:

1. Abrir un proxy (por defecto abre un proxy en '0.0.0.0:8001')

```
$ kubectl proxy --address='0.0.0.0' --disable-filter=true
W0925 16:00:34.496367   71864 proxy.go:167] Request filter disabled, your proxy
is vulnerable to XSRF attacks, please be cautious
Starting to serve on [::]:8001
```

2. Ejecute el dashboard **en otro terminal dentro del servidor**

```
$ minikube dashboard
    Verifying dashboard health ...
    Launching proxy ...
    Verifying proxy health ...
    Opening http://127.0.0.1:39711/api/v1/namespaces/kubernetes-
dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
    http://127.0.0.1:39711/api/v1/namespaces/kubernetes-
dashboard/services/http:kubernetes-dashboard:/proxy/
```

3. Reemplace 127.0.0.1 por 192.168.100.3 y visite el dashboard en su navegador usando

```
http://192.168.100.3:8001/api/v1/namespaces/kubernetes-
dashboard/services/http:kubernetes-dashboard:/proxy/
```

Eliminar deployment y servicios

Vuelva al terminal original. Una vez terminemos con el servicio de hello-minikube, podemos eliminar sus deployment y servicios asociados para liberar recursos. Ejecute los siguientes comandos,

```
$ kubectl delete service,deployment hello-minikube
service "hello-minikube" deleted
deployment.apps "hello-minikube" deleted
```

```
$ kubectl get pods
No resources found in default namespace.
```

```
$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   18h
```

3.4. Construya e instale un nodo de servicio con Docker

En esta parte crearemos un pequeño nodo servidor en una imagen de Docker y lo correremos en Kubernetes,

Creemos proyecto llamado hello-node,

```
$ mkdir hello-node && cd hello-node && touch Dockerfile server.js
```

```
$ tree
.
├── Dockerfile
└── server.js
```

Ahora, creemos un servidor http que retorne una cadena como respuesta.

```
$ vim server.js
```

```
var http = require('http');
var requests=0;
var podname= process.env.HOSTNAME;
var startTime;
var host;
var handleRequest = function(request, response) {
```

```
  response.setHeader('Content-Type', 'text/plain');
  response.writeHead(200);
  response.write("Hello Kubernetes! | Running on: ");
  response.write(host);
  response.end(" | v=1\n");
  console.log("Running On:" ,host, "| Total Requests:", ++requests,"| App
Uptime:", (new Date() - startTime)/1000 , "seconds", "| Log Time:",new Date());
}
var www = http.createServer(handleRequest);
www.listen(8080,function () {
    startTime = new Date();;
    host = process.env.HOSTNAME;
    console.log ("Kubernetes App Started At:",startTime, "| Running On: " ,host,
"\n" );
});
```

NOTA: Tómese un tiempo para verificar que quedo bien copiado el contenido anterior.

Editemos el Dockerfile para declarar que esta imagen usará node 4.4, y el container correrá el servicio ejecutando el archivo server.js

```
$ vim Dockerfile
```

```
FROM node:4.4
EXPOSE 8080
COPY server.js .
CMD node server.js
```

Configuremos el environment Docker,

```
$ eval $(minikube docker-env)
```

El comando `minikube docker-env` devuelve un conjunto de exportaciones de variables de entorno Bash para configurar su entorno local para reutilizar el demonio Docker dentro de la instancia de Minikube.

Pasar esta salida a través de `eval` hace que bash evalúe estas exportaciones y las ponga en práctica.

Puede revisar los comandos específicos que se ejecutarán en su shell omitiendo el paso de evaluación y ejecutando `minikube docker-env` directamente. Sin embargo, esto no realizará la configuración; la salida debe evaluarse para eso.

Construyamos la imagen (note el '.' Al final del comando)

```
$ docker build -t hello-node:v1 .
Sending build context to Docker daemon 6.144 kB
Step 1 : FROM node:4.4
 ---> abb6383ef5fe
Step 2 : EXPOSE 8080
 ---> Running in a4330e4790a7
 ---> a1f021471d1c
Removing intermediate container a4330e4790a7
Step 3 : COPY server.js .
 ---> 1b394ed6fbaf
Removing intermediate container 31c5e5e505bf
Step 4 : CMD node server.js
 ---> Running in 6e62fbd3174f
 ---> ff428ba5193c
Removing intermediate container 6e62fbd3174f
Successfully built ff428ba5193c
```

```
$ docker images
REPOSITORY                                          TAG                 IMAGE
ID            CREATED             SIZE
hello-node                                          v1
ff428ba5193c        20 seconds ago      656.9 MB
…
```

Ahora, podemos desplegar el pod hello-node en nuestro cluster local usando kubectl

```
$ kubectl create deployment hello-node --image=hello-node:v1 --port=8080
deployment.apps/hello-node created
```

```
$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
hello-node-6689b477fc-p6x98   1/1     Running   0          83s
```

```
$ kubectl get deployments
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
hello-node   1/1     1            1           3m39s
```

Al igual que antes, debemos exponer el deployment para que use una dirección IP externa. En este caso usaremos el tipo NodePort.

```
$ kubectl expose deployment hello-node --type=NodePort
service "hello-node" exposed
```

En caso de error en el despliegue, verifique los logs del pod así (para obtener el nombre del pod use kubectl get pods):

```
Kubectl logs <nombre del pod>
```

Verifique el servicio

```
$ kubectl get services
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
hello-node   NodePort    10.108.88.211   <none>        8080:30468/TCP   33s
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP          23h
```

```
$ curl $(minikube service hello-node --url)
Hello Kubernetes! | Running on: hello-node-6689b477fc-p6x98 | v=1
```

3.4. Explore la aplicación desplegada

Para ver los pods existentes ejecute,

```
$ kubectl get pods
```

Para ver que containers existen dentro del pod y que imágenes se usaron para construir dicho pod ejecute,

```
$ kubectl describe pods
```

En otro terminal ejecute

```
kubectl port-forward --address 0.0.0.0 service/hello-node 8090:8080
```

Pruebe desde el browser:

http://192.168.100.3:8090

Deje el proxy corriendo y regrese a la ventana original. Obtenga el nombre del pod y almacénelo en una variable POD_NAME

```
$ export POD_NAME=$(kubectl get pods -o go-template --template '{{range
.items}}{{.metadata.name}}{{"\n"}}{{end}}')
```

Verifique el nombre del pod

```
$ echo Name of the Pod: $POD_NAME
```

Ver los logs del contenedor,

```
$ kubectl logs $POD_NAME
```

Ejecute comandos en el contenedor,

Liste las variables de entorno en el contenedor

```
$ kubectl exec $POD_NAME -- env
```

Abra una sesión bash

```
$ kubectl exec -ti $POD_NAME -- bash
```

Una vez en la consola del contenedor, ejecute:

`cat server.js`

Verifique que la aplicación esta corriendo,

`curl localhost:8080`

Para cerrar la conexión con el contenedor, escriba `exit`.

3.5. Escalar la aplicación

Escalar la aplicación a 4 replicas,

```
$ kubectl scale deployments/hello-node --replicas=4
```

Para listar los deployments, usar `get deployments`:

```
$ kubectl get deployments
```

Chequear si el numero de pods ha cambiado. Ahora hay 4 pods con IPs diferentes. Verifiquelo ejecutando,

```
$ kubectl get pods -o wide
```

Verifique los deployments,

```
$ kubectl describe deployments/hello-node
```

3.5. Load Balancing

Verifiquemos que el servicio esta haciendo load balancing del tráfico. Para verificar la IP y puertos expuestos, usar,

```
$ kubectl describe services/hello-node
```

Crear una nueva variable de entorno llamada NODE_PORT que tiene como valor Node Port

```
$ export NODE_PORT=$(kubectl get services/hello-node -o go-template='{{(index
.spec.ports 0).nodePort}}')
```

```
$ echo NODE_PORT=$NODE_PORT
```

Luego ejecutemos un `curl` hacia la IP o puerto expuestos. Ejecute el siguiente comando múltiples veces,

```
$ curl $(minikube service hello-node --url)
```

o

```
$ curl <IP>:$NODE_PORT
```

Donde la <IP> puede ser verificada usando `kubectl describe pods`

Ejemplo:

```
Node:         minikube/172.17.0.3
```

En cada request un pod diferente deberá atender la petición. Esto demuestra que el load balancing esta trabajando.

3.5. Scale Down

Para reducir el numero de replicas a 2 usar,

```
$ kubectl scale deployments/hello-node --replicas=2
```

Liste los deployments

```
$ kubectl get deployments
```

El numero de replicas se redujo a 2, liste el numero de pods,

```
$ kubectl get pods -o wide
```

Esto debe confirmer que 2 pods fueron terminados.

3.5. Limpieza

Eliminemos el servicio y el deployment para hello-node y apaguemos minikube.

```
$ kubectl delete service,deployment hello-node
service "hello-node" deleted
deployment.apps "hello-node" deleted
```

```
$ minikube stop
✋  Stopping node "minikube"  ...
🛑  Powering off "minikube" via SSH ...
🛑  1 nodes stopped.
```

4. Ejercicio

Realice uno de los siguientes ejercicios:

- Investigue en que consisten los Ingress Controller de Kubernetes y muéstrelo con un escenario de prueba. Se sugiere reproducir la guía en:

  https://kubernetes.io/docs/tasks/access-application-cluster/ingress-minikube/

5. Desafío [Hasta 0.5 Puntos en una nota de Practicas]

Realice una de las siguientes actividades:

- Investigue como se realiza orquestación de contenedores usando **Docker Swarm** y configure un escenario para demostrar su uso.

- Investigue como se realizan rolling updates en Kubernetes y configure un escenario para demostrarlo.

  Se sugiere basarse en el tutorial interactivo en:

  https://kubernetes.io/docs/tutorials/kubernetes-basics/update/update-intro/

6. Entregables y Evaluación

- Sustentación de la práctica

7. Referencias

- Kubectl https://kubernetes.io/docs/user-guide/kubectl-overview/
- Instalar minikube: https://kubernetes.io/es/docs/tasks/tools/install-minikube/
- Instalar kubectl: https://kubernetes.io/es/docs/tasks/tools/install-kubectl/
- Hello Minikube: https://kubernetes.io/es/docs/tutorials/hello-minikube/