Certified Tech Developer
The Ultimate Degree

Infraestructura III

Desplegamos servicios internos

Objetivo

El objetivo del ejercicio es desplegar un servicio de tipo ClusterIP que balancee la carga entre tres pods.

Consigna

Vamos a deplegar 3 pods con un manifiesto de Deployment que contendrá una aplicación de tipo Hello World que devuelve el nombre de host. Luego intentaremos configurar un service ClusterIP que balancee la carga entre los tres pods, para verificarlo vamos a utilizar la herramienta de linux 'curl' enviando requests al service ClusterIP para ver si redirige el tráfico hacia los pods y devuelve el mensaje Hello World.

¡Éxitos!

1. Vamos a crear archivos .yaml donde pondremos las definiciones de cada objeto. Crear una carpeta de nombre clase14 y desde la terminal posicionarse dentro de la misma.
2. Crear un archivo de nombre 'namespace.yaml'. Este manifiesto servirá para crear de forma declarativa un namespace de nombre 'hello-namespace'. Debe tener estos datos:

```yaml
apiVersion: v1

kind: Namespace

metadata:

 name: hello-namespace
```

3. Crear un archivo de nombre ‘deployment.yaml’. Este manifiesto servirá para desplegar de forma declarativa y a prueba de fallos 3 pods de nombre ’hello-deployment-xxxxxxxxxxx’ (cada pod tendrá al final del nombre un hash en lugar de xxxxxxxxxx que agrega automáticamente el deployment). El archivo debe contener estos datos:

```yaml
apiVersion: apps/v1

kind: Deployment

metadata:

 name: hello-deployment

 namespace: hello-namespace

spec:

 selector:

   matchLabels:

     app: hello

 replicas: 3

 template:

   metadata:

     labels:

       app: hello
```

```
  spec:
    containers:
    - name: hello
      image:
"us-docker.pkg.dev/google-samples/containers/gke/hello-app:2.0"
```

4. Crear un archivo de nombre ‘clusterip_service.yaml’. Este manifiesto servirá para desplegar de forma declarativa un servicio de tipo ClusterIP que servirá para balancear la carga de requests entre los 3 pods. El archivo debe contener estos datos:

```
apiVersion: v1
kind: Service
metadata:
 name: hello-service
 namespace: hello-namespace
spec:
 type: ClusterIP
 selector:
   app: hello
 ports:
 - protocol: TCP
   port: 80
   targetPort: 8080
```

5. Una vez creados los 3 archivos desde la terminal y dentro de la carpeta clase14, vamos a desplegar todo usando los siguientes comandos:

kubectl apply -f namespace.yaml

kubectl apply -f .

Nota: En el primer comando le pasamos el manifiesto del namespace y en el segundo con el punto le pasamos todos los archivos de la ubicación actual. Podríamos pasar los archivos uno por uno también, pero si el namespace no está creado los demás manifiestos van a mostrar un error.

6. Vamos a validar con los siguientes comandos que toda la configuración esté correcta, para ello podemos usar el siguiente comando para ver todos los objetos desplegados dentro del namespace 'hello-namespace':

   kubectl get all -n hello-namespace

La salida del comando nos debe mostrar los tres pods, luego un service clusterIP y al final el deployment y el replicaset que se genera con el deployment. Observen que en los services se muestran las IPs bajo CLUSTER-IP:

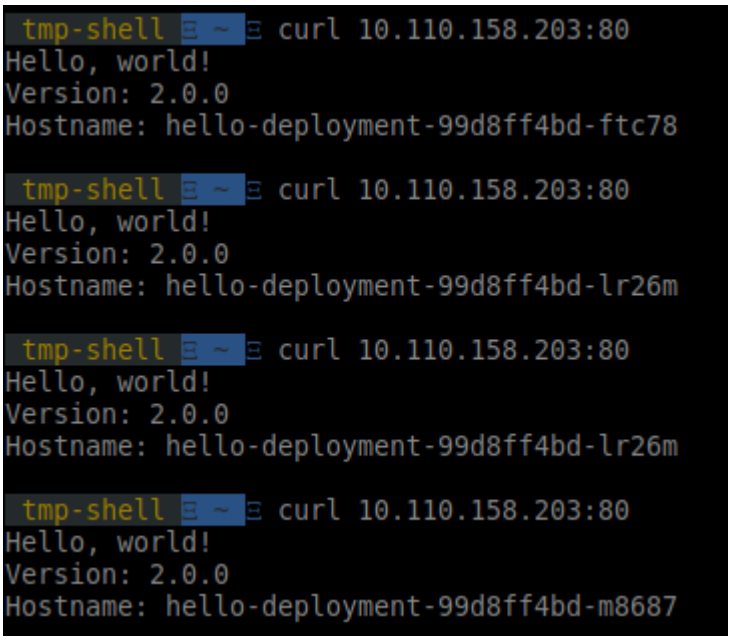
```
maury@MD:~/clase14$ kubectl get all -n hello-namespace
NAME                                     READY   STATUS    RESTARTS   AGE
pod/hello-deployment-99d8ff4bd-826ns     1/1     Running   0          10s
pod/hello-deployment-99d8ff4bd-bqbl7     1/1     Running   0          10s
pod/hello-deployment-99d8ff4bd-wb57f     1/1     Running   0          10s

NAME                    TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/hello-service   ClusterIP   10.99.203.50   <none>        80/TCP    10s

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-deployment    3/3     3            3           10s

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-deployment-99d8ff4bd     3         3         3       10s
```

```
maury@MD:~/clase14$ kubectl describe service hello-service -n hello-namespace
Name:              hello-service
Namespace:         hello-namespace
Labels:            <none>
Annotations:       <none>
Selector:          app=hello
Type:              ClusterIP
IP Family Policy:  SingleStack
IP Families:       IPv4
IP:                10.110.158.203
IPs:               10.110.158.203
Port:              <unset>  80/TCP
TargetPort:        8080/TCP
Endpoints:         172.17.0.3:8080,172.17.0.4:8080,172.17.0.5:8080
Session Affinity:  None
Events:            <none>
```

8. A continuación, vamos a validar que el service ‘hello-service’ de tipo ClusterIP responda a los requests. Como sabemos, los pods y el service ClusterIP están aislados dentro del cluster. Para poder hacer un curl y llegar al service vamos a desplegar con el siguiente comando un pod que se utiliza para hacer pruebas de conectividad en kubernetes:

   kubectl run tmp-shell -n hello-namespace --rm -i --tty --image nicolaka/netshoot

Veremos que el prompt cambia, esto es porque el comando nos mete directamente dentro del pod de prueba. Ahora, si podemos hacer un curl a la ip del service, hacemos curl a la ip del servicio y al puerto 80. Pueden observar que si repetimos el comando el Hostname va cambiando de forma aleatoria:



Luego con el comando ‘exit’ salimos de pod de prueba. Con esta prueba validamos que la configuración del service y los pods funcionen correctamente.

9. Por último vamos a eliminar el namespace junto con todo su contenido con este comando:

   kubectl delete -f .