Curso de Docker y Kubernetes:

Trabajo Practico Integrador Kubernetes

Profesor: Marcos Tonina

Estudiante: Oscar Anibal Martinez

Año: 2024 – UTN.BA Centro de e-learning

Desarrollo:

1- Configuramos la conexión al cluster con:
$ kubectl config get-contexts

```
oscar@DESKTOP-8FEH7A0:/mnt/c/Users/Oscar$ kubectl config get-contexts
CURRENT   NAME                                                   CLUSTER
          gke_numeric-melody-442300-q5_southamerica-east1-a_cluster-1   gke_numeric-melody-44230
*         minikube                                               minikube
```

2- Se va a utilizar el contexto de Google Kubernetes Engine (GKE)
$ kubectl config use-context gke_numeric-melody-442300-q5_southamerica-east1-a_cluster-1

```
oscar@DESKTOP-8FEH7A0:/mnt/c/Users/Oscar$ kubectl config use-context gke_numeric-melody-442300-q5_southamerica-east1-a_cluster-1
Switched to context "gke_numeric-melody-442300-q5_southamerica-east1-a_cluster-1".
oscar@DESKTOP-8FEH7A0:/mnt/c/Users/Oscar$ kubectl config get-contexts
CURRENT   NAME                                                   CLUSTER
*         gke_numeric-melody-442300-q5_southamerica-east1-a_cluster-1   gke_numeric-melody-442300-q5_southamerica-east1-a_cluste
          minikube                                               minikube
```

3- Se obtiene el código fuente desde git
$ git clone https://github.com/oskartinez/node-docker.git

Este es un servidor nodejs que publica los siguientes endpoints.
Por ejemplo:
http://localhost:8080/api/?abc=123
http://localhost:8080/hola/oscar
http://localhost:8080/hostname
http://localhost:8080/logs

Cada vez que se invoca /api o /hola, se genera una respuesta por navegador y escribe la misma respuesta en un archivo dentro de la carpeta logs del servidor.

4- Para que nuestro Pod tenga un volumen persistente donde guardar los logs debemos crear el volumen claim con los requisitos de almacenamiento:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: mi-pvc
spec:
 accessModes:
  - ReadWriteOnce
 resources:
  requests:
   storage: 200Mi
```

5- Generamos el deployment de para un contenedor que se encuentra en el repositorio en https://hub.docker.com/r/oskartinez/node-docker (servidor nodejs)

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nodeapp-deployment
spec:
 selector:
  matchLabels:
   app: node-app
 replicas: 2
 template:
  metadata:
   labels:
    app: node-app
  spec:
   containers:
   - name: nodeapp
    image: oskartinez/node-docker:v2
    volumeMounts:
    - mountPath: /logs
      name: my-pvc-volume
    ports:
    - containerPort: 8080
   volumes:
   - name: my-pvc-volume
    persistentVolumeClaim:
     claimName: mi-pvc
```

*Téngase en cuenta que el servicio de pvc que ofrece GKE, no es concurrente es de tipo ReadWriteOnce, un solo un pod puede estar utilizando dicho recurso y generará un error de “Multi-Attach error for volume”, que puede verse con el comando kubectl get events. Si se cambia a ReadWriteMany el driver no lo soporta y no se produce el binding con el PV, quedando en estado pending.

3 – Ingresamos con kubectl al cluster para aplicar los archivos declarativos.

$ kubectl apply -f pvc.yaml

```
oscar@DESKTOP-8FEH7A0:/mnt/c/users/oscar/Proyectos/node-docker$ kubectl apply -f pvc.yaml
persistentvolumeclaim/mi-pvc created
```

$ kubectl apply -f deployment.yaml

```
oscar@DESKTOP-8FEH7A0:/mnt/c/users/oscar/Proyectos/node-docker$ kubectl apply -f deployment.yaml
deployment.apps/nodeapp-deployment created
```

4- Observamos la creación de los objetos: pods, replicaset, deployment.

$ kubectl get all

```
oscar@DESKTOP-8FEH7A0:/mnt/c/Users/Oscar/Proyectos/node-docker$ kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/nodeapp-deployment-5cc45ddc9f-9j6lc   1/1     Running   0          68s
pod/nodeapp-deployment-5cc45ddc9f-v4zsq   1/1     Running   0          68s

NAME                 TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)   AGE
service/kubernetes   ClusterIP   34.118.224.1   <none>        443/TCP   21d

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/nodeapp-deployment   2/2     2            2           69s

NAME                                          DESIRED   CURRENT   READY   AGE
replicaset.apps/nodeapp-deployment-5cc45ddc9f   2         2         2       69s
```

6- Creamos un objeto service (tipo NodePort) para acceder al cluster.
$ kube apply -f svc.yaml

```
oscar@DESKTOP-8FEH7A0:/mnt/c/Users/Oscar/Proyectos/node-docker$ kubectl apply -f svc.yaml
service/nodeapp-svc created
oscar@DESKTOP-8FEH7A0:/mnt/c/Users/Oscar/Proyectos/node-docker$ kubectl get svc
NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes    ClusterIP   34.118.224.1     <none>        443/TCP          21d
nodeapp-svc   NodePort    34.118.230.122   <none>        8080:30000/TCP   7s
```

7- Para acceder al servicio mediante NodePort, debemos conocer las IP que fueron asignadas a los nodos workers.
$ kubectl get nodes -o wide

```
oscar@DESKTOP-8FEH7A0:/mnt/c/Users/Oscar/Proyectos/node-docker$ kubectl get nodes -o wide
NAME                                   STATUS   ROLES    AGE   VERSION               INTERNAL-IP    EXTERNAL-IP
gke-cluster-1-pool-1-9e67b1b9-mtb5   Ready    <none>   105m   v1.30.5-gke.1699000   10.158.0.14   34.95.186.201
gke-cluster-1-pool-1-9e67b1b9-qvd8   Ready    <none>   105m   v1.30.5-gke.1699000   10.158.0.15   35.199.117.0
```

En la última columna veremos las IP asignadas.

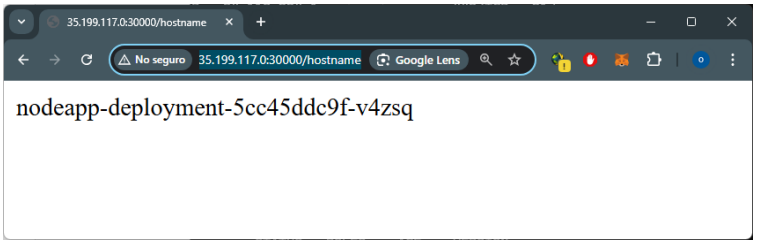
Para poder acceder desde el exterior a la nube de Google debemos habilitar el acceso a los nodos en puerto 30000.

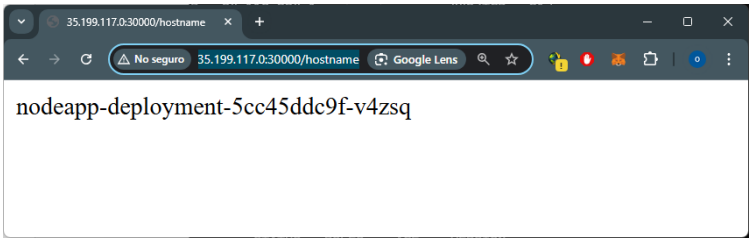
$ gcloud compute firewall -rules create test-node-port-ks –allow tcp:30000

```
oscar@DESKTOP-8FEH7A0:/mnt/c/users/oscar/Proyectos/node-docker$ gcloud compute firewall
-rules create test-node-port-ks --allow tcp:30000
Creating firewall...⠹Created [https://www.googleapis.com/compute/v1/projects/numeric-me
lody-442300-q5/global/firewalls/test-node-port-ks].
Creating firewall...done.
NAME              NETWORK  DIRECTION  PRIORITY  ALLOW      DENY  DISABLED
test-node-port-ks  default  INGRESS    1000      tcp:30000        False
```

8- Abrimos un navegador y colocamos la dirección del nodo

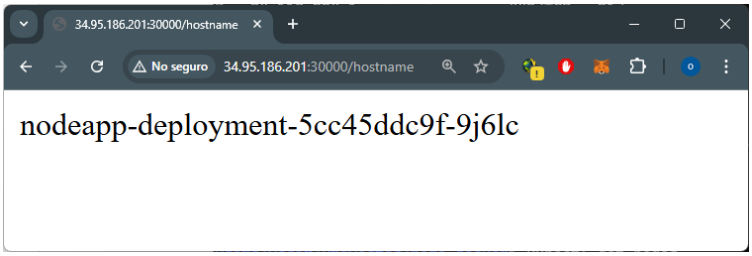
http://35.199.117.0:30000/hostname

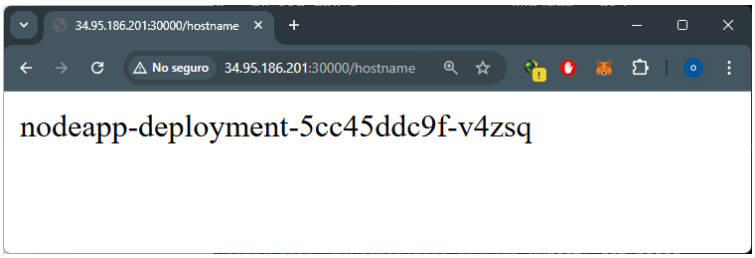




Vemos que la respuesta varía según el nodo que recibe la respuesta, ya que funciona un balanceador de carga interno.

Probando con el otro host, ocurre lo mismo.





La desventaja del uso de NodePort es que debemos de conocer las IP de los nodos, y éstos puede variar con el tiempo, resultando poco práctico. Para ello se puede utilizar otro servicio llamado LoadBalancer, pero que solo funciona en una plataforma cloud que admita este tipo de servicio.

Ahora, cuando crea un servicio LoadBalancer, Kubernetes detecta en qué plataforma de computación en la nube se ejecuta su clúster y crea un balanceador de carga en la infraestructura del proveedor de la nube. El

balanceador de carga tendrá su propia dirección IP única y de acceso público que los clientes pueden usar para conectarse a su aplicación.

9- Creamos el servicio tipo LoadBalancer.

```
apiVersion: v1
kind: Service
metadata:
 name: nodeapp-svc
spec:
 type: LoadBalancer
 ports:
 - port: 8080
  targetPort: 8080
 selector:
  app: node-app
```

Luego aplicamos esa configuración:

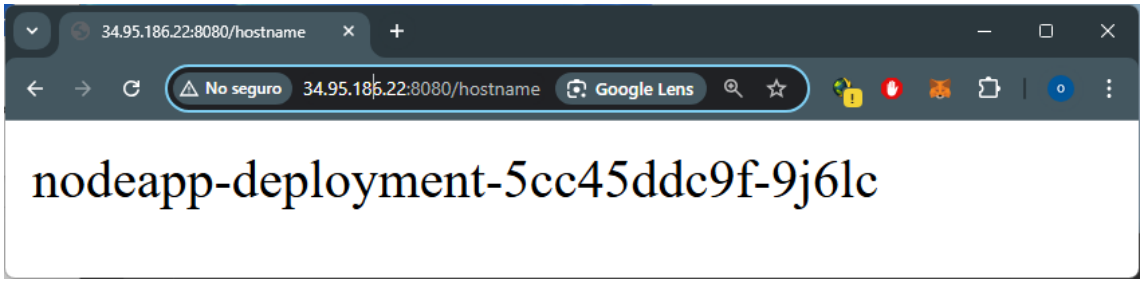
$ kubectl apply -f svc-lb.yaml

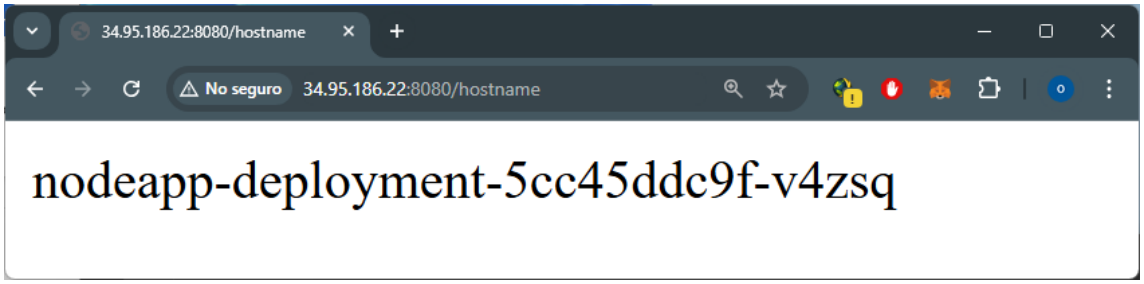
```
oscar@DESKTOP-8FEH7A0:/mnt/c/Users/Oscar/Proyectos/node-docker$ kubectl apply -f svc-lb.yaml
service/nodeapp-svc configured
```

Luego comprobamos en unos instantes la IP pública que le fue otorgada.

```
oscar@DESKTOP-8FEH7A0:/mnt/c/Users/Oscar/Proyectos/node-docker$ kubectl get svc
NAME          TYPE           CLUSTER-IP       EXTERNAL-IP    PORT(S)          AGE
kubernetes    ClusterIP      34.118.224.1     <none>         443/TCP          21d
nodeapp-svc   LoadBalancer   34.118.230.122   34.95.186.22   8080:30000/TCP   22m
```

Abrimos el navegador en http://34.95.186.22:8080/hostname





Vemos que funciona el balanceador de carga, apuntando a esa IP pública fija.