

Universidad de Guadalajara.

Computación Tolerante a fallas.



Diego Ivan Becerra Gonzalez.

Sección D06.

Objetivo.

Responder a las preguntas y generar un ejemplo utilizando kubernetes.

Desarrollo.

¿Qué es Kubernetes?

Kubernetes es un sistema de código abierto para implementar, escalar y administrar aplicaciones alojadas en contenedores en cualquier lugar. Kubernetes automatiza las tareas operativas de la administración de contenedores e incluye comandos integrados para implementar aplicaciones, actualizarlas, escalarlas y mucho más.

¿Qué es Ingress?

Es una forma de acceder a tus pods desde fuera del cluster. Este recurso nos permite acceder a servicios a través de HTTP(S) y el tráfico se controla utilizando un conjunto de reglas que tú defines. Además de dar a tus aplicaciones una URL externa que permita el acceso, también se puede configurar para el balanceo de carga o terminación SSL.

¿Qué es un LoadBalancer?

Un «load balancer» o balanceador de carga permite repartir la carga de trabajo entre los diferentes servidores o aplicaciones y puede instalarse en una infraestructura tanto física como virtual. Así pues, los programas de load balancing adoptan la forma de un controlador de entrega de aplicaciones o ADC (del inglés «aplicación delivery controller»), permitiendo al usuario escalar la carga automáticamente en función de las previsiones de tráfico.

Ahora seguimos con la ejecución del tutorial.

Lo primer que hacemos es crear una carpeta llamada node-hello-app. Después, abrimos la línea de comandos, nos vamos a la carpeta y ejecutamos “npm init -y” para que nos cree un proyecto de npm. Después ejecutamos “npm install express” para que nos instale la librería.

En la misma carpeta creamos un archivo index.js, el cual servirá para simular nuestra api usando node.js.

```
const express = require('express')
const os = require('os')

const app = express()
app.get('/', (req, res) => {
  res.send(`HELLO from ${os.hostname()}!`)
})
const port = 3000
app.listen(port, () => console.log(`listening on port ${port}`))
```

Para probar que funcione bien usamos el comando “node index.js” y si nos vamos al navegador y buscamos “localhost:3000”, nos debería aparecer el mensaje que indicamos en index.

Ahora crearemos la imagen de Docker, para eso creamos un Dockerfile en la misma carpeta y colocamos el siguiente código.

```

FROM node:13-alpine

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install --production

COPY . .

EXPOSE 3000





CMD node index.js

```

Esto le indica a Docker que instalara el sistema operativo alpine, y que arrancara ejecutando el archivo index.

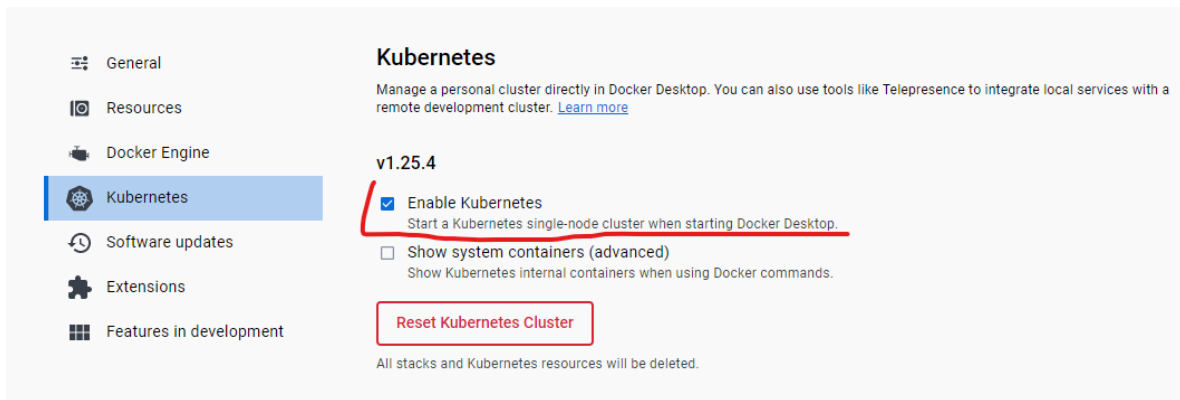
Ahora ejecutamos en la terminal el comando “docker build -t username/node-hello-app.” para que Docker construya nuestra imagen de acuerdo al Dockerfile.

Y ejecutamos el contenedor con “docker run --rm -d -p 3000:3000 username/node-hello-app”. Los parámetros indican que se deberá borrar la imagen cuando el contenedor se cierre y que correrá en segundo plano.

<input type="checkbox"/>	Name	Image	Status	Port(s)	Last started	Actions
<input type="checkbox"/>	 modest_ramanujan 0193dee3aa9c	diegoivan1987/node-hello-ap	Running	3000:3000	10 minutes ago	  

También lo subimos a DockerHub con “docker push diegoivan1987/node-hello-app”.

Ahora pasamos a la parte de kubernetes. Es importante mencionar que para poder utilizarlo necesitamos un cluster y la herramienta kubectl, los cuales vienen incluidos al instalar Docker desktop, sin embargo, debemos activar el cluster manualmente en la configuración de Docker.



Una vez activada la opción, podemos ver nuestro cluster.

```
C:\Users\diego>kubectl get node
NAME                STATUS    ROLES    AGE   VERSION
docker-desktop      Ready     control-plane  15h   v1.25.4
```

Ahora creamos un recurso de tipo deployment en nuestro cluster. Este recurso lo que hace es administrar el estado de nuestros pods, así que cuando un pod falla, lo reemplazara con alguna copia.

```
C:\Users\diego>kubectl create deployment --image diegoivan1987/node-hello-app node-app
deployment.apps/node-app created
```

Al ejecutar el comando anterior, creamos el recurso deployment, y al crearse el recurso, nos crea un pod y otro recurso llamado “replicaset”. Esto lo podemos ver con el comando “kubectl get all”.

```
C:\Users\diego>kubectl get all
NAME                                READY    STATUS    RESTARTS   AGE
pod/node-app-76b6cbdbd5-m84jz      1/1      Running   0           20m

NAME                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)    AGE
service/kubernetes  ClusterIP   10.96.0.1     <none>         443/TCP    16h

NAME                                READY    UP-TO-DATE   AVAILABLE   AGE
deployment.apps/node-app  1/1      1             1           20m

NAME                                DESIRED   CURRENT   READY   AGE
replicaset.apps/node-app-76b6cbdbd5  1         1         1       20m
```

Por ejemplo, aquí nuestro pod está corriendo.

```
C:\Users\diego>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
node-app-76b6cbdbd5-m84jz          1/1     Running   0           22m
```

Pero si elimino uno de los pods y vuelvo a revisar los pods activos, podemos ver que se eliminó el anterior, pero se creó uno nuevo, esto gracias al deployment.

```
C:\Users\diego>kubectl delete pod node-app-76b6cbdbd5-m84jz
pod "node-app-76b6cbdbd5-m84jz" deleted
```

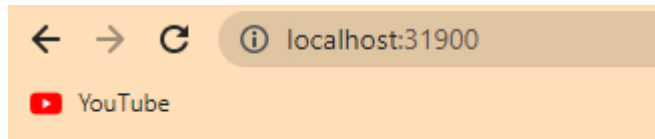
```
C:\Users\diego>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
node-app-76b6cbdbd5-kzfdg          1/1     Running   0           19s
node-app-76b6cbdbd5-m84jz          1/1     Terminating   0           23m
```

Con el comando “kubectl scale deployment node-app --replicas 3” cambiamos el numero de replicas del pod a 3, por lo que si vemos todos los pods nos aparecen 3 distintos.

```
C:\Users\diego>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
node-app-76b6cbdbd5-kzfdg          1/1     Running   0           89m
node-app-76b6cbdbd5-pmospv          1/1     Running   0           10s
node-app-76b6cbdbd5-tll42          1/1     Running   0           10s
```

Ya tenemos nuestra aplicación corriendo en el cluster, sin embargo, aun no podemos acceder a ella desde fuera del mismo cluster, por lo que tenemos que exponerla, para eso usamos el comando “kubectl expose deployment node-app --type NodePort --port 3000”, que expone el puerto 3000 del pod a un puerto aleatorio de nuestro pc.

```
C:\Users\diego>kubectl get service
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes    ClusterIP   10.96.0.1     <none>         443/TCP          17h
node-app      NodePort    10.106.63.19  <none>         3000:31900/TCP   72s
```



HELLO from node-app-76b6cbdbd5-kzfdg!

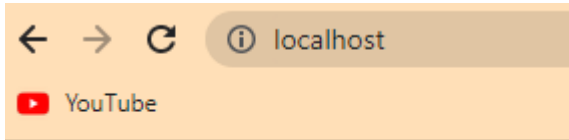
Usamos “kubectl edit service node-app” para abrir el archivo .yaml de nuestro servicio y editamos las siguientes líneas, de:

```
port: 3000
protocol: TCP
targetPort: 3000
selector:
  app: node-app
sessionAffinity: None
type: NodePort
```

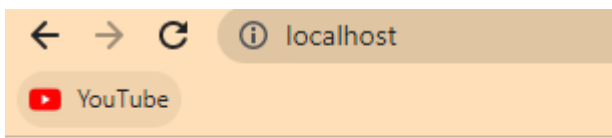
a

```
port: 80
protocol: TCP
targetPort: 3000
selector:
  app: node-app
sessionAffinity: None
type: LoadBalancer
```

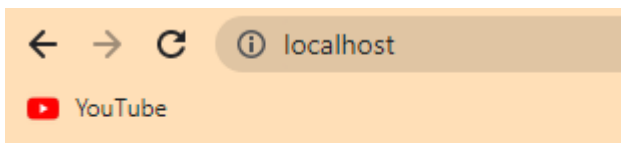
Aquí lo que hacemos es crear un LoadBalancer, el cual ayuda a dirigir las peticiones a los diferentes pods para distribuir de una mejor manera el trabajo. Como podemos observar, el nombre del equipo cambia al hacer varias peticiones.



HELLO from node-app-76b6cbdbd5-kzfdg!



HELLO from node-app-76b6cbdbd5-tll42!



HELLO from node-app-76b6cbdbd5-pmspv!

Que son los id de los pods.

```
C:\Users\diego>kubectl get pod -o wide
NAME                                READY   STATUS    RESTARTS
node-app-76b6cbdbd5-kzfdg          1/1     Running   1 (2m38s ago)
node-app-76b6cbdbd5-pmspv          1/1     Running   1
node-app-76b6cbdbd5-tll42          1/1     Running   1 (2m43s ago)
```

Conclusión.

La herramienta de kubernetes es muy útil en cuanto al ámbito de la tolerancia a fallas ya que el servicio de deployment permite restaurar y reemplazar pods que hayan fallado. También permite que haya una redundancia al hacer copias de un

mismo servicio. Por último, el LoadBalancer distribuye la carga de trabajo entre los pods, lo que evita que haya cuellos de botella.

Bibliografía.

Load balancer con Kubernetes. (s. f.). OVHcloud.

<https://www.ovhcloud.com/es/public-cloud/kubernetes/kubernetes-load-balancer/#:~:text=Un%20%C2%ABload%20balancer%C2%BB%20o%20balanceador,infraestructura%20tanto%20f%C3%ADsica%20como%20virtual.>

¿Qué es Kubernetes? | Google Cloud | Google Cloud. (s. f.). Google Cloud.

<https://cloud.google.com/learn/what-is-kubernetes?hl=es-419>

Torres, G. (2021). Acceder a tus aplicaciones en Kubernetes a través de Ingress.

return(GIS); <https://www.returngis.net/2019/04/acceder-a-tus-aplicaciones-en-kubernetes-a-traves-de-ingress/>

Amazon Web Services. (2017, 31 marzo). *Spire Labs: Fault-tolerance with Kubernetes on AWS* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=xvXy2BiaWrQ>

DigitalOcean. (2020, 27 mayo). *Production-ready Node.js on Kubernetes* [Vídeo].

YouTube. <https://www.youtube.com/watch?v=T4lp6wtS--4>

KodeKloud. (2018, 23 noviembre). *Kubernetes For Beginners: Taints & Tolerations*

[Vídeo]. YouTube. <https://www.youtube.com/watch?v=mo2UrkjA7FE>

TechWorld with Nana. (2020, 25 enero). *Benefits of Kubernetes | Scalability, High Availability, Disaster Recovery | Kubernetes Tutorial 16* [Vídeo]. YouTube.

<https://www.youtube.com/watch?v=g8Sf-6EsgZM>