Introducción a Docker

UD 08. Caso práctico 02 - Wordpress con Kubernetes

Autor: Sergi García Barea

Actualizado Abril 2022

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

📖 Importante

❕ Atención

💬 Interesante

UD08. CASO PRÁCTICO 02

1. INTRODUCCIÓN

En este caso práctico vamos a poner en marcha un despliegue que implementará un sistema completo **Wordpress** usando "***Kubernetes***" y "***MiniKube***". Para ellos desplegamos una aplicación **MySQL** y otra **Wordpress**. Además, realizaremos un autoescalado con este último despliegue.

2. PASO 0: INICIAR MINIKUBE

Antes de empezar el caso práctico, debemos poner en marcha nuestro clúster con:

```
minikube start
```

```
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico1$ minikube start
😄  minikube v1.19.0 en Ubuntu 20.04
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🔄  Restarting existing docker container for "minikube" ...
🐳  Preparando Kubernetes v1.20.2 en Docker 20.10.5...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Complementos habilitados: storage-provisioner, default-storageclass
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

Una vez puesto en marcha, podemos proseguir con el caso práctico.

3. PASO 1 (FICHERO YAML):DESPLEGANDO MYSQL MEDIANTE FICHERO YAML

Vamos a definir la configuración en un único fichero tanto la configuración del despliegue, servicio y volumen persistente de **MySQL**. Está disponible en un fichero **YAML** comentado "***mysql-deployment.yaml***" con el siguiente contenido:

```
apiVersion: v1
#Definimos el servicio en el que expondremos MySQL
kind: Service
metadata:
  name: wordpress-mysql
  labels:
        app: wordpress
#Características del servicio MySQL
spec:
  ports:
        #MySQL se expone en el puerto 3306
        - port: 3306
  selector:
        app: wordpress
        tier: mysql
  clusterIP: None
---
#Declaramos un volumen de persistencia para almacenar el contenido de la base de datos
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  labels:
        app: wordpress
spec:
  accessModes:
        - ReadWriteOnce
  resources:
```

```
        requests:
        storage: 20Gi
---
#Definimos como será el despliegue del MySQL
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
        app: wordpress
spec:
  selector:
        matchLabels:
        app: wordpress
        tier: mysql
  strategy:
        type: Recreate
  template:
        metadata:
        labels:
        app: wordpress
        tier: mysql
        spec:
        #Imagen para el contenedor
        containers:
        - image: mysql:5.6
        name: mysql
        #Variables de entorno definidas
        env:
        - name: MYSQL_ROOT_PASSWORD
        value: CEFIREdocker
        #Puerto a exponer de estos contenedores
        ports:
        - containerPort: 3306
        name: mysql
        #Montaje dentro del contenedor del volumen persistente
        volumeMounts:
        - name: mysql-persistent-storage
        mountPath: /var/lib/mysql
        #Indicamos que estos contenedores utilizaran el volumen persistente
        volumes:
        - name: mysql-persistent-storage
        persistentVolumeClaim:
        claimName: mysql-pv-claim
```

Una vez listo, podemos lanzar nuestro despliegue usando el comando:

```
kubectl apply -f "mysql-deployment.yaml"
```

Con esto habremos creado nuestro despliegue. Observaremos algo similar a:

```
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$ kubectl apply -f mysql-deployment.yaml
service/wordpress-mysql created
persistentvolumeclaim/mysql-pv-claim created
deployment.apps/wordpress-mysql created
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$ kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
wordpress-mysql-68cbdb976-ms8kx    1/1     Running   0          4s
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$
```

Con esto ya tenemos listo el despliegue y servicio para **MySQL** y podemos proceder a desplegar la aplicación **Wordpress** que utilizará **MySQL**.

4. PASO 2 (FICHERO YAML):DESPLEGANDO WORDPRESS MEDIANTE FICHERO YAML

Vamos a definir la configuración en un único fichero tanto la configuración del despliegue, servicio y volumen persistente de Wordpress. Este Wordpress beberá del servicio MySQL configurado en el paso anterior. Está disponible en un fichero **YAML** comentado "***wordpress-deployment.yaml***" con el siguiente contenido:

```
#Definimos la información del servicio
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
        app: wordpress
spec:
  ports:
        #El servicio se expone en el puerto 80
        - port: 80
  selector:
        app: wordpress
        tier: frontend
  #Aplicamos balanceo de carga para facilitar su escalado horizontal
  type: LoadBalancer
---
#Definimos un volumen persistente
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
        app: wordpress
spec:
  #Indica que solo puede ser montado para lectura/escritura por un nodo. Para el resto lectura.
  #En este caso, se usa para modificar un fichero de configuración.
  accessModes:
        - ReadWriteOnce
  resources:
        requests:
        storage: 20Gi
---
#definimos el despliegue
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
  labels:
        app: wordpress
spec:
  selector:
        matchLabels:
        app: wordpress
        tier: frontend
  strategy:
        type: Recreate
  template:
        metadata:
        labels:
        app: wordpress
        tier: frontend
        spec:
        #Imagen
```

```
        containers:
        - image: wordpress:4.8-apache
        name: wordpress
        #Indicamos variables de entorno
        env:
        - name: WORDPRESS_DB_HOST
        value: wordpress-mysql
        - name: WORDPRESS_DB_PASSWORD
        value: CEFIREdocker
        ports:
        - containerPort: 80
        name: wordpress
        volumeMounts:
        - name: wordpress-persistent-storage
        mountPath: /var/www/html
        volumes:
        - name: wordpress-persistent-storage
        persistentVolumeClaim:
        claimName: wp-pv-claim
```

Una vez listo, podemos lanzar nuestro despliegue usando el comando:

```
kubectl apply -f "wordpress-deployment.yaml"
```

Con esto habremos creado nuestro despliegue. Observaremos algo similar a:

```
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$ kubectl apply -f wordpress-deployment.yaml
service/wordpress created
persistentvolumeclaim/wp-pv-claim created
deployment.apps/wordpress created
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$ kubectl get pods
NAME                              READY   STATUS              RESTARTS   AGE
wordpress-5bc56766cd-75dkk        0/1     Pending             0          1s
wordpress-5bc56766cd-blz9d        0/1     Pending             0          1s
wordpress-5bc56766cd-ckrlv        0/1     ContainerCreating   0          2s
wordpress-5bc56766cd-g48nm        0/1     Pending             0          1s
wordpress-5bc56766cd-plg82        0/1     Pending             0          1s
wordpress-mysql-68cbdb976-ms8kx   1/1     Running             0          5m
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$
```

5. PASO 3: ACCEDIENDO AL SERVICIO

Una vez hecho esto si examinamos los servicios:

```
kubectl get services
```

Obteniendo algo similar a:

```
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$ kubectl get services
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes        ClusterIP      10.96.0.1       <none>        443/TCP        4d5h
wordpress         LoadBalancer   10.99.231.205   <pending>     80:32618/TCP   4m51s
wordpress-mysql   ClusterIP      None            <none>        3306/TCP       9m48s
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$
```

Si observamos detenidamente, la IP Externa de nuestro servicio está "***Pending***" y no lo tenemos expuesto directamente. Esto es debido a que todo el clúster "***Kubernetes***" está dentro de "***MiniKube***". Para acceder al contenido, tenemos dos formas:

Forma 1: accederemos a la IP de "***MiniKube***" y nos expondrá el servicio en un puerto aleatorio.

```
minikube service wordpress
```

A veces, el comando anterior da problemas. Si es así, podemos usar alternativamente, para exponer todos los servicios usandoen lugar de "wordpress" el parámetro "--all".

```
minikube service --all
```

Tras lanzar este comando, se nos abrirá un navegador accediendo al servicio en uno de los puertos que expone "***MiniKube***" y aparecerá un texto similar a:

```
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$ minikube service wordpress
|-----------|-----------|-------------|-------------------------|
| NAMESPACE |   NAME    | TARGET PORT |           URL           |
|-----------|-----------|-------------|-------------------------|
| default   | wordpress |          80 | http://192.168.49.2:31566 |
|-----------|-----------|-------------|-------------------------|
  Opening service default/wordpress in default browser...
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$
```

Tras ello podemos observar que nuestra aplicación está siendo servida:

Forma 2: expondremos el servicio con la IP de "***MiniKube***" y accederemos a él.

Para hacer esto, en una terminal aparte lanzaremos el siguiente comando

```
minikube tunnel
```

Este comando se ejecutará en la terminal "de forma indefinida" y mientras esté en funcionamiento, establecerá un túnel para acceder al servicio. Veremos algo similar a:

```
sergi@ubuntu:~$ minikube tunnel
Status:
        machine: minikube
        pid: 33468
        route: 10.96.0.0/12 -> 192.168.49.2
        minikube: Running
        services: [wordpress]
    errors:
                minikube: no errors
                router: no errors
                loadbalancer emulator: no errors
```

Si mientras este comando está en ejecución hacemos

```
kubectl get services
```

```
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$ kubectl get services
NAME              TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)        AGE
kubernetes        ClusterIP      10.96.0.1       <none>          443/TCP        4d5h
wordpress         LoadBalancer   10.104.67.169   10.104.67.169   80:31566/TCP   2m52s
wordpress-mysql   ClusterIP      None            <none>          3306/TCP       3m16s
```

Ya observaremos una IP. En este ejemplo, accediendo a http://10.104.67.169 accederemos a la aplicación.

6. PASO 4(FICHERO YAML): AUTOESCALADO DE WORDPRESS CON FICHERO YAML

Vamos a definir el autoescalado del servicio Wordpress. Este está disponible en un fichero **YAML** comentado "***autoscale.yaml***" con el siguiente contenido:

```
apiVersion: autoscaling/v1
#Tipo autoescalado horizontal
kind: HorizontalPodAutoscaler
metadata:
 name: autoescaladowordpress
spec:
 #Indicamos a quien se aplica el auto-escalado
 scaleTargetRef:
   apiVersion: apps/v1
   kind: Deployment
   name: wordpress
 #Mínimo y máximo de réplicas
 minReplicas: 5
 maxReplicas: 10
 #Máximo de CPU a usar durante el auto-escalado
 targetCPUUtilizationPercentage: 50
```

Una vez listo, podemos lanzar nuestro despliegue usando el comando:

```
kubectl apply -f "autoscale.yaml"
```

Con esto habremos creado nuestro autoescalado para estos dos despliegues. Una vez hecho el escalado, realizaremos una petición al servicio para que se haga efectivo. Observaremos algo similar a:

```
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$ kubectl apply -f autoscale.yaml
horizontalpodautoscaler.autoscaling/autoescaladomysql created
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$ minikube service wordpress
|-----------|-----------|-------------|--------------------------|
| NAMESPACE |   NAME    | TARGET PORT |            URL           |
|-----------|-----------|-------------|--------------------------|
| default   | wordpress |          80 | http://192.168.49.2:31566 |
|-----------|-----------|-------------|--------------------------|
  Opening service default/wordpress in default browser...
sergi@ubuntu:~/Desktop/KubernetesUD08/CasoPractico2$ kubectl get pods
NAME                              READY   STATUS    RESTARTS   AGE
wordpress-5bc56766cd-l4hlb        1/1     Running   0          8m50s
wordpress-mysql-68cbdb976-9wdzn   1/1     Running   0          19s
wordpress-mysql-68cbdb976-f5txm   1/1     Running   0          9m14s
wordpress-mysql-68cbdb976-hwlz5   1/1     Running   0          19s
wordpress-mysql-68cbdb976-hxd7h   1/1     Running   0          19s
wordpress-mysql-68cbdb976-wmxhp   1/1     Running   0          19s
```

7. PASO 5: ELIMINANDO LO CREADO

Si queremos eliminar todos los elementos creados, podemos hacerlo con los siguientes comandos:

```
kubectl delete deployment wordpress
kubectl delete deployment wordpress-mysql
kubectl delete service wordpress-mysql
kubectl delete service wordpress
kubectl delete service wordpress
kubectl delete persistentvolumeclaim mysql-pv-claim
kubectl delete persistentvolumeclaim wp-pv-claim
kubectl delete HorizontalPodAutoscaler autoescaladowordpress
```

8. Bibliografía

[1] Kubernetes https://kubernetes.io/
[2] Kubernetes docs https://kubernetes.io/docs/home/
[3] MiniKube https://minikube.sigs.k8s.io/docs/