Introducción a Docker  
UD 08. Caso práctico 02 - Wordpress con Kubernetes

short line

Autor: Sergi García Barea

Actualizado Abril 2021

Licencia

**Reconocimiento – NoComercial - CompartirIgual (BY-NC-SA)**: No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

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

📖 **Importante**

❕ **Atención**

💬 **Interesante**

**1.** [**Introducción**](#_dbh0n1vac4c8) **3**

**2.** [**Paso 0: iniciar MiniKube**](#_3b05p2f899fh) **3**

**3.** [**Paso 1 (Fichero YAML):Desplegando MySQL mediante fichero YAML**](#_v5ot8tcfc7uh) **3**

**4.** [**Paso 2 (Fichero YAML):Desplegando Wordpress mediante fichero YAML**](#_ollh1s527my7) **5**

**5.** [**Paso 3: Accediendo al servicio**](#_rgj10zjv46q5) **6**

**6.** [**Paso 4(Fichero YAML): Autoescalado de Wordpress con fichero YAML**](#_m1jtbkgv3d98) **8**

**7.** [**Paso 5: eliminando lo creado**](#_9p0xnot11ut4) **8**

**8.** [**Bibliografía**](#_72yv1gty1dke) **9**

UD08. Caso práctico 02

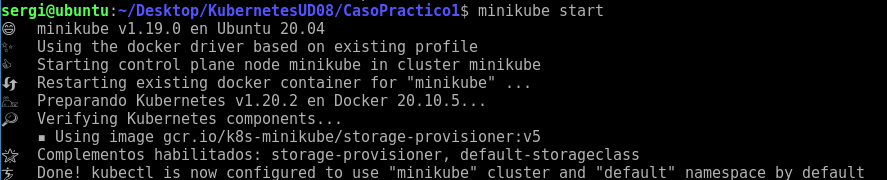
# Introducción

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.

# Paso 0: iniciar MiniKube

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

|  |
| --- |
| minikube start |



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

# Paso 1 (Fichero YAML):Desplegando MySQL mediante fichero YAML

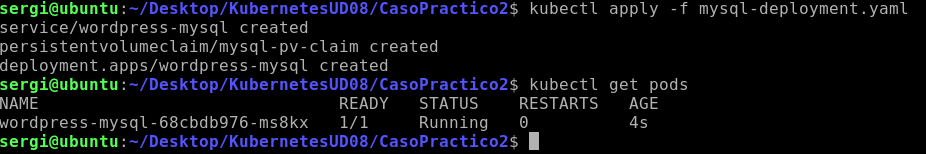
Vamos a definir la configuración en un único fichero tanto la configuración del despliegue, servicio y volumen persistente de **MySQL**. Está disponible en un fichero **YAML** comentado “***mysql-deployment.yaml***” con el siguiente contenido:

|  |
| --- |
| *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:



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

# Paso 2 (Fichero YAML):Desplegando Wordpress mediante fichero YAML

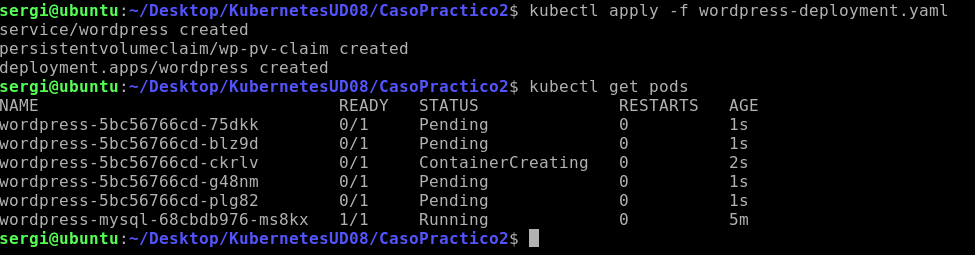
Vamos a definir la configuración en un único fichero tanto la configuración del despliegue, servicio y volumen persistente de Wordpress. Este Wordpress beberá del servicio MySQL configurado en el paso anterior. Está disponible en un fichero **YAML** comentado “***wordpress-deployment.yaml***” con el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

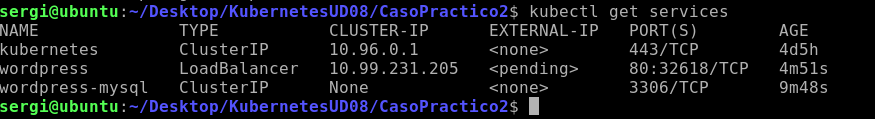


# Paso 3: Accediendo al servicio

Una vez hecho esto si examinamos los servicios:

|  |
| --- |
| kubectl get services |

Obteniendo algo similar a:

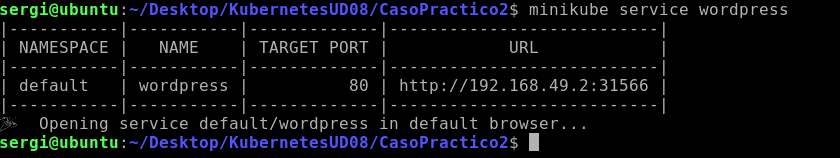


Si observamos detenidamente, la IP Externa de nuestro servicio está “***Pending***” y no lo tenemos expuesto directamente. Esto es debido a que todo el cluster “***Kubernetes***” está dentro de “***MiniKube***”. Para acceder al contenido, tenemos dos formas:

**Forma 1:** accederemos a la IP de “***MiniKube***” y nos expondrá el servicio en un puerto random.

|  |
| --- |
| minikube service wordpress |

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:



Tras ello podemos observar que nuestra aplicación está siendo servida:

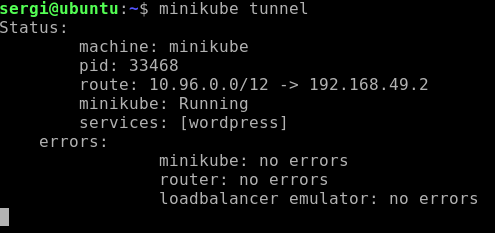


**Forma 2:** expondremos el servicio con la IP de “***MiniKube***” y accederemos a él.

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

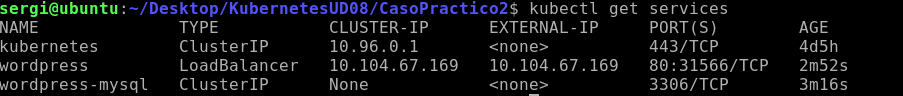
|  |
| --- |
| minikube tunnel |

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:



Si mientras este comando está en ejecución hacemos

|  |
| --- |
| kubectl get services |



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

# Paso 4(Fichero YAML): Autoescalado de Wordpress con fichero YAML

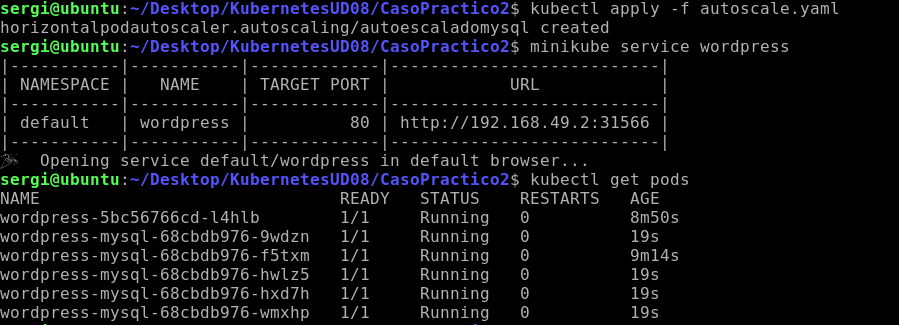
Vamos a definir el autoescalado del servicio Wordpress. Este está disponible en un fichero **YAML** comentado “***autoescale.yaml***” con el siguiente contenido:

|  |
| --- |
| 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 "autoescale.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:



# 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 |

# Bibliografía

[1] Kubernetes <https://kubernetes.io/>

[2] Kubernetes docs <https://kubernetes.io/docs/home/>

[3] MiniKube <https://minikube.sigs.k8s.io/docs/>