

# **PODs – Services - Deployments**

Antonio Espín Herranz

# Tener en cuenta

- 1 POD (envoltorio de un contenedor de Docker)
- 1 Contenedor → 1 POD
- **¿Se puede tener más de un contenedor en un POD?**
  - **Si**, pero no es lo habitual
  - En el fichero de manifest se indican los contenedores en una lista de yaml.
  - Se rompe el principio de individualidad.

# Ejemplo: YAML con dos containers

```
apiVersion: v1
kind: Pod
metadata:
  name: multi
spec:
  containers:
  - name: web
    image: nginx
    ports:
    - containerPort: 80
  - name: frontal
    image: alpine
    command: ["watch", "-n5", "ping", "localhost"]
```

- **2 containers**

- Web imagen → nginx
- Frontal → alpine (hace referencia a un Linux ligero, ver en Docker Hub).

- El comando **watch**

- Cada 5 segundos ejecuta el comando pasado por argumento:
- Lanzar un ping localhost cada 5"
- Comprueba si está funcionando el servidor nginx

# Servicios

- Se utilizan para comunicar los PODs entre sí o para exponer un POD hacia el exterior.

# Deployments

- En un **deployment** se puede lanzar **varios PODs**, pero serán **réplicas del mismo POD**.
- **OJO, no debemos** lanzar 2 PODs distintos en el mismo deployment:
  - Si metemos dos PODs en el mismo deployment, **no se pueden hacer tareas de escalado individuales**, tampoco podemos actualizar uno de los PODs con independencia.

# Funcionamiento

- **ReplicaSets:** Cuando defines un Deployment, especificas el **número de réplicas** que deseas (por ejemplo, replicas: 3 en el archivo YAML).
  - *Kubernetes se encargará de crear y mantener esos pods, utilizando un ReplicaSet.*
- **Escalabilidad:** Puedes **aumentar** o **disminuir** el número de **pods** simplemente modificando el número de réplicas en tu Deployment (manualmente o con un autoescaler).
- **Gestión centralizada:** El Deployment gestiona todos los pods bajo su configuración, asegurándose de que estén funcionando correctamente y reemplazando automáticamente los pods que fallan.

# Ejemplo

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mi-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mi-aplicacion
  template:
    metadata:
      labels:
        app: mi-aplicacion
    spec:
      containers:
        - name: mi-contenedor
          image: mi-imagen:latest
          ports:
            - containerPort: 8080
```

- 3 réplicas del mismo contenedor → 3 PODs con la especificación indicada.
- Por ejemplo, si queremos tener **MySQL y PHP con 3 y 2 réplicas** respectivamente.
- Necesitamos **DOS DEPLOYMENTS** uno para cada tipo.
  - En total tendremos **5 PODs**

# ¿Qué pasa si los ponemos juntos?

- **Se puede si, pero NO ES UNA BUENA PRÁCTICA**
  - **No podrás escalar los contenedores de forma independiente**, ya que al escalar el Deployment se escalarán todos los contenedores del pod juntos.
- **En casos muy puntuales:**
  - Puede ser útil en casos en los que los contenedores necesiten compartir recursos o comunicarse estrechamente, como si formaran parte de una misma unidad lógica.



# Ejemplo: Mysql (3) y PHP (2)

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:latest
        ports:
        - containerPort: 3306
        env:
        - name: MYSQL_ROOT_PASSWORD
          value: "rootpassword"
        - name: MYSQL_DATABASE
          value: "mi_base_datos"
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: php
  template:
    metadata:
      labels:
        app: php
    spec:
      containers:
      - name: php
        image: php:apache
        ports:
        - containerPort: 80
        volumeMounts:
        - name: html-data
          mountPath: /var/www/html
      volumes:
      - name: html-data
        emptyDir: {}
```

# Conectar PHP y MySQL con un Service

**Deployment de MySQL**



**Deployment de PHP**



# Conectar PHP y MySQL con un Service

## SERVICE

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  selector:
    app: mysql
  ports:
    - protocol: TCP
      port: 3306 # Puerto por el que accede PHP
      targetPort: 3306 # Puerto del contenedor MySQL
```

para conectar el service

- El servicio se coloca en otro YAML.
- El nombre del servicio es el que utilizará el POD de PHP.

# Deployment PHP

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: php
  template:
    metadata:
      labels:
        app: php
    spec:
      containers:
        - name: php
          image: php:apache
          ports:
            - containerPort: 80
          env:
            - name: DB_HOST
              value: "mysql-service" # Conecta al Service
            - name: DB_PORT
              value: "3306" # Puerto del servicio MySQL
          volumeMounts:
            - name: html-data
              mountPath: /var/www/html
      volumes:
        - name: html-data
          emptyDir: {}
```

- **Flujo de Comunicación:**

- El pod de **PHP** conecta con **mysql-service** a través de las variables de entorno **DB\_HOST** y **DB\_PORT**.
- *Kubernetes dirige las solicitudes al Service, que redirige las conexiones a uno de los pods de MySQL gestionados por el Deployment de MySQL*

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
        - name: mysql
          image: mysql:latest
          ports:
            - containerPort: 3306
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: "rootpassword" # Cambiar según tus credenciales
            - name: MYSQL_DATABASE
              value: "mi_base_datos"
          volumeMounts:
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql # Directorio donde MySQL almacena datos
      volumes:
        - name: mysql-persistent-storage
          persistentVolumeClaim:
            claimName: mysql-pvc

```

# Deployment MySQL

- El POD de MySQL tiene **3 réplicas** y se **asocia** con el **Service** con las etiquetas del campo **selector**.
- En el **Service** tenemos:

```

spec:
  selector:
    app: mysql

```

# Opciones para lanzar varios ficheros YAML

- Cuando tenemos que lanzar varios ficheros YAML. Por ejemplo: 3 deployments, 2 services, etc.

## 1) Kubectl apply -f <directorio>

- Colocamos todos los ficheros en un directorio, se puede aplicar a todos los archivos, en vez de lanzar uno a uno.

## 2) Combinar todos los archivos en uno solo:

- Se van copian uno después de otro en el mismo fichero.
- **OJO, cada recurso se separa por 3 guiones: - - -**

```
# Deployment 1
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment1
...
---
# Service 1
apiVersion: v1
kind: Service
metadata:
  name: service1
...
```

# Opciones para lanzar varios ficheros YAML 2

- **3) Un script de Linux con los comandos kubectl (comando.sh):**
  - kubectl apply -f service1.yaml
  - kubectl apply -f service2.yaml
  - kubectl apply -f deployment1.yaml
  - Kubectl apply -f deployment2.yaml
  - kubectl apply -f deployment3.yaml
- **4) Con la herramienta: Helm chart**
  - Gestor de paquetes para kubernetes
  - <https://helm.sh/>