

K8s

Kubernetes

Antonio Espín Herranz

Contenidos

- Orquestación de microservicios con **Kubernetes**:
 - Creación de **Pods**, **Deployments**, y **Services** en Kubernetes.
 - Estrategias de despliegue y escalabilidad en Kubernetes.

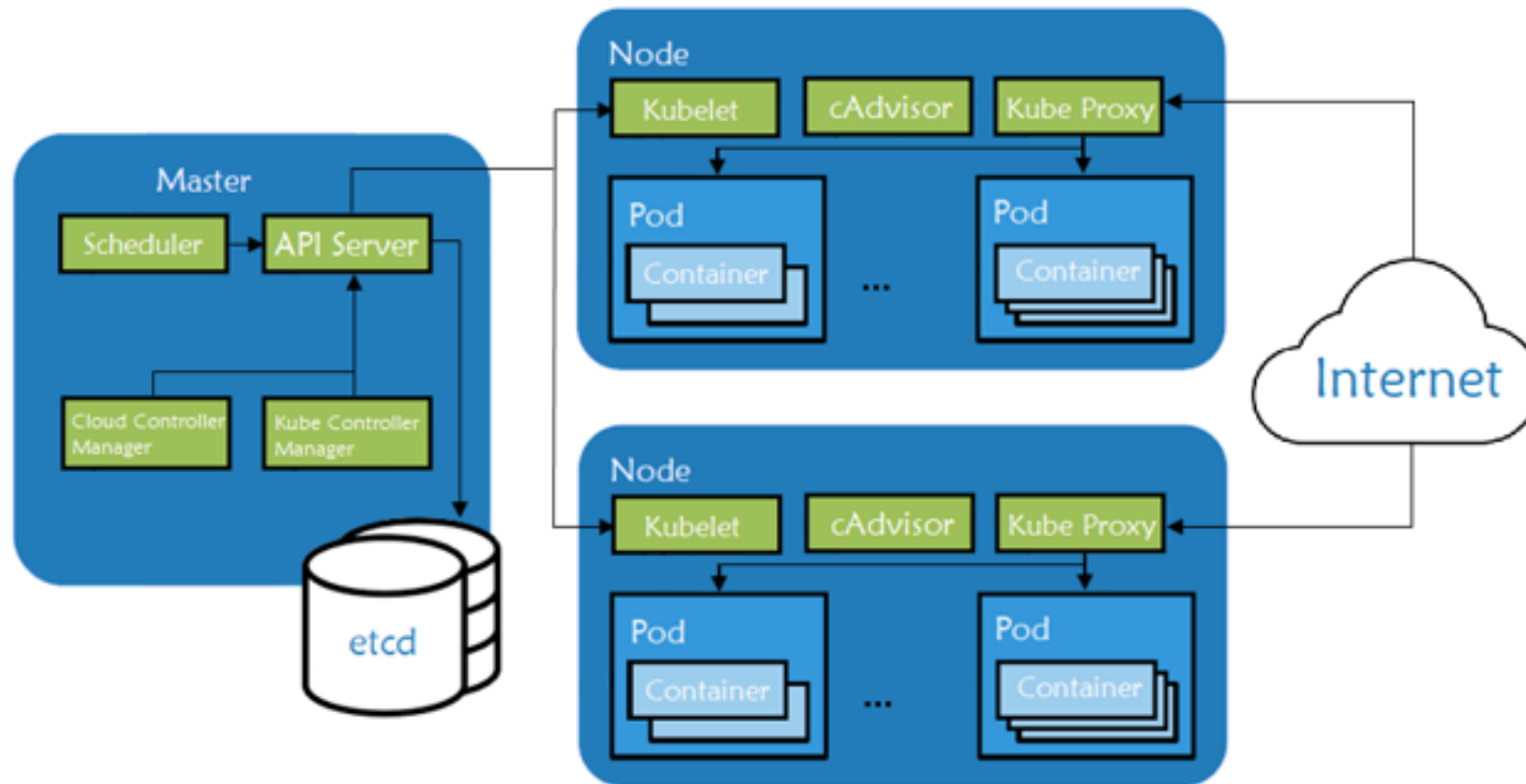
Introducción

- Definición: Kubernetes (**K8s**) es una plataforma de **código abierto** para la **orquestación** de **contenedores** que automatiza la implementación, escalado y gestión de aplicaciones en contenedores.
- Origen: Fue desarrollado por Google y donado a la Cloud Native Computing Foundation (**CNCF**) en 2015.
- Propósito: Simplifica la gestión de aplicaciones en contenedores, permitiendo que sean altamente escalables y resilientes.

Limitaciones de Docker

- **Docker** (docker engine) *gestiona completamente la ejecución de un contenedor en un determinado nodo a partir de una imagen*, pero no proporciona toda la funcionalidad que necesitamos para ejecutar aplicaciones en entornos en producción.
- Existen diferentes preguntas que nos podemos hacer acerca de esto :
 - ¿Qué hacemos con los cambios entre versiones?
 - ¿Cómo hacemos los cambios en producción?
 - ¿Cómo se balancea la carga entre múltiples contenedores iguales?
 - ¿Cómo se conectan contenedores que se ejecuten en diferentes demonios de docker?
 - ¿Se puede hacer una actualización de una aplicación sin interrupción?
 - ¿Se puede variar a demanda el número de réplicas de un determinado contenedor?
 - ¿Es posible mover la carga entre diferentes nodos?
 - ¿Qué pasa si cae un contenedor?

Arquitectura



- Un cluster de kubernetes está formado por un nodo Master y de 2 a n nodos Worker.

Arquitectura

- **Nodo maestro (Master Node):** Es el cerebro de Kubernetes y coordina el clúster. Incluye:
 - **Kube API Server:** Maneja las solicitudes externas y sirve como el punto de entrada.
 - **Scheduler:** Decide en qué nodos deben ejecutarse los contenedores.
 - **Controller Manager:** Asegura que el estado deseado del clúster se mantenga.
 - **etcd:** Una base de datos distribuida para guardar toda la configuración del clúster.
- **Nodos de trabajo (Worker Nodes):** Son los responsables de ejecutar las aplicaciones en contenedores. Cada nodo tiene:
 - **Kubelet:** Asegura que los contenedores estén ejecutándose correctamente.
 - **Kube-proxy:** Gestiona la red para la comunicación entre contenedores y servicios.
 - **Contenedor Runtime:** Es el software encargado de ejecutar los contenedores (como Docker o containerd).
 - **Container Advisor,** es una herramienta de código abierto diseñada para analizar y exponer datos sobre el uso de recursos y el rendimiento de contenedores en ejecución. Funciona como un proceso en segundo plano (daemon) que recopila, procesa y exporta información sobre contenedores.

Cluster

- Es el **conjunto de todos los nodos** (tanto el plano de control como los nodos de trabajo) que trabajan juntos para orquestar aplicaciones.
- **Networking en Kubernetes:**
 - Kubernetes utiliza un modelo de red plano donde cada Pod obtiene su propia dirección IP.
 - Las políticas de red permiten controlar la comunicación entre Pods y servicios.
- **Objetos de Kubernetes:**
 - Son entidades persistentes que describen el estado deseado del clúster.
 - **Pods:** Unidades básicas de implementación.
 - **Services:** Permiten exponer Pods a otras partes del clúster o al exterior.
 - **Deployments:** Gestionan la creación y el escalado de Pods.
 - **ConfigMaps** y **Secrets:** Almacenan datos de configuración para los Pods.

Funcionalidades principales

- **Escalado automático:** Ajusta **dinámicamente** la **cantidad de Pods** según la demanda.
- **Autocuración:** **Reemplaza Pods fallidos** automáticamente.
- **Despliegues declarativos:** **Define** el **estado deseado** de la aplicación y Kubernetes lo mantiene.
- **Balanceo de carga:** **Distribuye el tráfico** entre los Pods disponibles.
- **Actualizaciones continuas:** Permite **realizar despliegues** y actualizaciones **sin interrumpir** la disponibilidad.

Casos de uso

- **Microservicios:** Gestión de aplicaciones desglosadas en componentes pequeños y autónomos.
- **Cargas de trabajo híbridas:** Ejecuta aplicaciones tanto en la nube como en entornos locales.
- **Procesamiento en lote:** Ideal para tareas que requieren muchos recursos momentáneamente.
- **Entornos de desarrollo y prueba:** Simplifica la configuración y creación de entornos aislados.

Ventajas

- **Portabilidad: Compatible** con múltiples proveedores de nube y entornos locales.
- **Escalabilidad:** Gestiona fácilmente el crecimiento de aplicaciones y recursos.
- **Resiliencia: Mantiene las aplicaciones funcionando a pesar de fallos** en los contenedores o nodos.
- **Eficiencia operativa:** Optimiza el uso de los recursos de infraestructura.

Los controladores en Kubernetes

- En **Kubernetes**, los **controladores** (controllers) son componentes clave que **se encargan de garantizar que el estado real del clúster coincida con el estado deseado definido por el usuario**.
 - Actúan como un sistema de "bucle de control", detectando discrepancias y realizando las acciones necesarias para corregirlas.
- Cómo funcionan los controladores:
 - **Definición del estado deseado:** El usuario describe el estado deseado en un manifiesto (archivo YAML/JSON), como el número de réplicas de un servicio o la configuración de un objeto.
 - **Supervisión constante:** El controlador supervisa el estado actual del clúster a través del API Server.
 - **Corrección:** Si detecta una discrepancia entre el estado real y el deseado, toma medidas para corregirla automáticamente.

minikube

- **Minikube** es una herramienta que permite ejecutar un **clúster** de k8s en **local**.
 - Crea un clúster de un solo nodo, pero es un cluster real.
 - ***Nos sirve para hacer desarrollo, pruebas.***
 - Trabajar con minikube
 - Acceso al Dashboard
 - Crear nuevos clusters (locales, en la propia máquina)
- Para **entornos de producción** se puede utilizar:
 - Kubeadm
 - K3s
 - Cloud-managed Kubernetes

minikube

- El comando sin opciones responde con la ayuda del comando.
- **minikube status**
 - Comprobar si minikube funciona correctamente.
 - Muestra información del host, kubelet, apiserver y kubeconfig
 - Si lo muestra parados, podemos reiniciar con:
 - **minikube stop**
 - **minikube start**
- **host:** Indica si la máquina virtual (VM) o el contenedor que ejecuta el clúster está en funcionamiento.
- **kubelet:** Muestra el estado del proceso de Kubelet, encargado de la comunicación entre los nodos y el clúster.
- **apiserver:** Indica si el servidor de la API de Kubernetes está disponible y en funcionamiento.
- **kubeconfig:** Verifica si tu configuración está correctamente vinculada al clúster.

```
C:\Users\Anton>minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

minikube

- **minikube logs**

- Genera la información de lo que va haciendo minikube.

- **minikube ip**

- Nos muestra la IP por si nos tenemos que conectar.

- Arranque y parada del cluster

- **minikube start**
- **minibuke stop**

minikube

- Borrar el cluster (ojo, se pierde todo)
 - **minikube delete**
- El dashboard (herramienta de monitorización)
 - **minikube dashboard**
 - Lanza una aplicación Web en un navegador para monitorizar el clúster.
- Pausar / Continuar el cluster
 - **minikube pause**
 - **minikube unpause**

minikube

- Login en el entorno de minikube (entramos dentro del contenedor de minikube)
 - **minikube ssh**
- **Directorios** que crea **minikube**, en la carpeta del usuario de **Windows**:
 - **.kube**: dentro tiene un fichero **config** que almacena información sobre el cluster.
 - **.minikube**
 - Ficheros de minikube, no hace falta modificarlo.
 - Hay un directorio con las máquinas

minikube

- Para **recrear** el cluster de kubernetes:
 - **minikube delete**
 - **minikube start**
- Si es otro clúster distinto para borrarlo, indicar el nombre:
 - **minikube delete --profile <nombre_del_cluster>**
 - **El clúster se conoce por profile**

Ejemplo

- Crear otro clúster dentro de minikube.
 - El clúster queda representado por un nombre de perfil (**profile**)
- Ver los cluster que tengo dentro de minikube:
 - **minikube profile list**

Profile	VM Driver	Runtime	IP	Port	Version	Status	Nodes	Active Profile	Active Kubecontext
cluster2	docker	docker	192.168.58.2	8443	v1.32.0		2		
minikube	docker	docker	192.168.49.2	8443	v1.32.0	OK	1	*	*

- Se pueden crear mas clúster y se pueden añadir varios nodos
 - **minikube start --driver=docker -p cluster2 --nodes=2**
 - -p El nombre del clúster, driver → Docker y el número de nodos 2
 - Al crearlo puede tardar un poco.
 - Después consultarlo con el comando: **minikube profile list**
 - **Ojo, los nodos consumen memoria y CPU**

minikube

- minikube **profile**
 - Nos informa con que clúster estamos trabajando
- Otra forma de saber con que clúster estamos trabajando:
 - Dentro del directorio **usuario/.kube** es donde se indica la configuración de los clusters de kubernetes
 - Este directorio está presente en todos los ordenadores que se conectan con kubernetes (no es de la herramienta minikube)
 - Dentro del directorio tenemos un fichero **config** que es donde se almacenan los clusters que tenemos.
 - Cada vez que se añade un cluster viene a este fichero y lo añade.

Fichero config

```
clusters:
- cluster:
  certificate-authority: C:\Users\Anton\.minikube\ca.crt
  extensions:
  - extension:
    last-update: Fri, 28 Mar 2025 12:19:33 CET
    provider: minikube.sigs.k8s.io
    version: v1.35.0
    name: cluster_info
    server: https://127.0.0.1:63221
    name: cluster2
- cluster:
  certificate-authority-data: LS0tLS1CRUdJTiBDRVJUSUZJQ0FURSB
YVdOQm9ZcW5rQnhmL25hckpVVTh0enBQQ25KQWw5dG12akFTUkt3QXZXZTY1TC
  server: https://kubernetes.docker.internal:6443
  name: docker-desktop
- cluster:
  certificate-authority: C:\Users\Anton\.minikube\ca.crt
  extensions:
  - extension:
    last-update: Fri, 28 Mar 2025 11:42:47 CET
    provider: minikube.sigs.k8s.io
    version: v1.35.0
    name: cluster_info
    server: https://127.0.0.1:61253
    name: minikube
```

Vemos también las URLs
De conexión

Cambiar de profile / cluster

- **minikube profile otro_cluster**
 - Cambiar el cluster por defecto al indicado
 - En el listado de clúster veremos activado el otro clúster.
 - minikube profile list
 - Si editamos el fichero **config** la propiedad **current-context** habrá cambiando → **comprobarlo**
 - Si cambiamos la línea del fichero config tendríamos el mismo efecto y a partir del cambio, cualquier comando que lancemos será sobre el nuevo clúster

Prueba

- **kubectl get nodes**

- Este comando se lanzará siempre sobre el clúster actual

```
C:\Users\Anton>minikube profile
* cluster2

C:\Users\Anton>kubectl get nodes
NAME           STATUS    ROLES          AGE      VERSION
cluster2       Ready     control-plane   4h35m    v1.32.0
cluster2-m02   Ready     <none>          4h32m    v1.32.0

C:\Users\Anton>minikube profile minikube
* minikube profile was successfully set to minikube

C:\Users\Anton>kubectl get nodes
NAME           STATUS    ROLES          AGE      VERSION
minikube       Ready     control-plane   27h      v1.32.0
```

minikube dashboard

- Herramienta gráfica (aplicación Web)
- Podemos gestionar los clústers y asignarles recursos.
- Tampoco es la panacea, pero podemos crear objetos.
- Es otra opción a parte del modo comando.



kubectl

Kubectl

- Es la herramienta que vamos a utilizar para interactuar con los clústers de kubernetes sin importar si son locales o están en la nube.
- Se utiliza en modo comando (igual que Docker)
- Ya viene instalada con Docker desktop
- Sin parámetros responde con las opciones del comando.
- Permite realizar tareas como desplegar aplicaciones, inspeccionar recursos, modificar configuraciones, y gestionar los distintos objetos de kubenertes que se almacenan en un clúster.

Resumen de comandos

- Listar recursos:
 - Kubectl get pods
 - Kubectl get nodes
 - Kubectl get services
 - Kubectl get deployments
 - Kubectl get all (ver todos)
 - Permite utilizar abreviaturas: kubectl get po, kubectl get pod
 - O varios tipos a la vez: kubectl get pod, service
- Describir recursos (como inspect de Docker):
 - Kubectl describe pod nombre_del_pod
 - Kubectl describe node nombre_del_nodo

Resumen de comandos

- Crear un recurso con un archivo YAML
- Si no existe, para crearlo (si existe → error)
 - `Kubectl create -f archivo.yaml`
- Si no existe, lo crea, si existe lo actualiza
 - `Kubectl apply -f archivo.yaml`
- Borrar recursos (hay que indicar el tipo de recurso):
 - `Kubectl delete pod nombre_pod`
 - `Kubectl delete service nombre_service`
 - Etc.

Resumen de comandos

- Ejecutar un comando en un POD:
 - `Kubectl exec -it nombre_pod`
- Los logs de un POD
 - `Kubectl logs nombre_pod`
 - `Kubectl logs -f nombre_pod`
- Escalar un deployment:
 - `kubectl scale deployment nombre-del-deployment --replicas=5`

Resumen de comandos

- Actualizar la imagen de un contenedor en el deployment
 - `kubectl set image deployment/nombre-del-deployment nombre-del-contenedor=nueva-imagen:etiqueta`
- Exponer un deployment como servicio:
 - `kubectl expose deployment nombre-del-deployment --type=LoadBalancer --port=80 --target-port=8080`

Resumen de comandos

- Información del clúster
 - Kubectl get nodes
- Ver información del clúster
 - Kubectl cluster-info
- Mostrar todos los objetos en un namespace específico.
 - kubectl get all -n nombre-del-namespace
 - Los namespace permite agrupar recursos dentro de un mismo clúster, pueden utilizarse para dividir entornos: desarrollo, producción, etc.
- Borrar todos los recursos de un archivo YAML
 - kubectl delete -f archivo.yaml

Enlaces

- Documentación:
 - <https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#-strong-getting-started-strong->

Pods
Services
Deployments

Recursos del clúster

- **Pod**
 - Es la unidad más pequeña en Kubernetes.
 - Contiene uno o más contenedores que comparten red y almacenamiento.
 - **Las buenas prácticas 1 POD → 1 CONTENEDOR**
 - Se usa para ejecutar una aplicación o servicio.
 - Los pods son efímeros: si se caen, Kubernetes puede reemplazarlos, pero no los repara.
 - Por ejemplo: un POD puede tener un contenedor con un microservicio en C++
- **Service**
 - Es una abstracción de red que **expone uno o varios pods**.
 - Permite que otros componentes (o usuarios) accedan a los pods de forma estable, aunque los pods cambien.
 - Puede tener distintos tipos:
 - **ClusterIP**: acceso interno.
 - **NodePort**: acceso externo por un puerto del nodo.
 - **LoadBalancer**: acceso externo con balanceo de carga.
 - Por ejemplo: exponer un servicio para que sea accesible desde fuera del clúster.
- **Deployment**
 - Es un controlador que gestiona el ciclo de vida de los pods.
 - Permite definir cuántas réplicas quieres, actualizaciones automáticas, y rollback si algo falla.
 - Es la forma más común de desplegar aplicaciones en Kubernetes.
 - Por ejemplo: un deployment puede mantener siempre 3 réplicas de tu app corriendo

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"]
```

Todos los recursos en el fichero YAML tienen:

- **apiVersion**: versión de la API que usa.
- **kind**: tipo de recurso (Pod, Service, etc.).
- **metadata**: nombre, etiquetas, namespace, etc.
- **spec**: definición deseada.
- **status**: estado actual (lo gestiona Kubernetes).

• 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?

- **Si se puede, 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/>

Apéndice YAML

Introducción a YAML

#Las cadenas no requieren comillas:

Título: Introducción a YAML

Pero se pueden usar:

title-w-quotes: 'Introducción a YAML'

Las cadenas multilínea comienzan con | (no usar los tabs.)

ejecutar: |

npm ci

npm build

prueba npm

Introducción a YAML

#Secuencias

#Las secuencias nos permiten definir listas en YAML:

Una lista de números usando guiones:

números:

- uno
- dos
- Tres

La versión en línea:

números: [uno, dos, tres]

Introducción a YAML

#Valores anidados

#Podemos usar todos los tipos anteriores para crear un objeto con valores anidados, así:

Lista de libros

- 1984:

autor: George Orwell

publicado en: 1949-06-08

recuento de páginas: 328

descripción: |

Una novela, a menudo publicada como 1984, es una novela distópica del novelista inglés George Orwell.

Fue publicado en junio de 1949 por Secker & Warburg como noveno y último b de Orwell.

- el Hobbit:

autor: J. R. R. Tolkien

publicado en: 1937-09-21

recuento de páginas: 310

descripción: |

The Hobbit, o There and Back Again es una novela de fantasía para niños del autor inglés J. R. R.

Archivos YAML de kubernetes

Estructura

- **apiVersion:** Define la versión de la API de Kubernetes que se usará para el objeto.
- **kind:** Especifica el tipo de objeto que estás creando (por ejemplo: Pod, Service, Deployment, etc.).
- **metadata:** Contiene información adicional como el nombre del objeto y etiquetas (labels).
- **spec:** Describe las especificaciones y configuraciones del objeto. El contenido dentro de spec depende del tipo de objeto que estás definiendo.

Ejemplo de un POD

- **apiVersion:** v1: Utiliza la versión 1 de la API.
- **kind:** Pod: El objeto que se va a crear es un Pod.
- **metadata:**
 - name: Se asigna el nombre "ejemplo-pod" al Pod.
 - labels: Se añade una etiqueta con clave app y valor ejemplo-app.
- **spec:**
 - containers: Lista de contenedores dentro del Pod. En este caso, solo hay uno llamado ejemplo-contenedor.
 - image: El contenedor utiliza la imagen de Docker nginx:latest.
 - ports: Expone el puerto 80 del contenedor.

```
apiVersion: v1
kind: Pod
metadata:
  name: ejemplo-pod
  labels:
    app: ejemplo-app
spec:
  containers:
  - name: ejemplo-contenedor
    image: nginx:latest
    ports:
    - containerPort: 80
```

labels

- Se usan para agruparlos y hacer consultas o referenciar más fácilmente diferentes tipos de objetos. Ejemplos:
 - “frontend” o “backend” asociados a la clave “capa”
 - “pruebas” o “produccion” asociado a la clave “tipo”
- Es posible etiquetar varios pods de la infraestructura y hacer una operación sobremuchos nodos
 - Con consultas o referencias que usen las etiquetas de los pods que los contienen: capa=frontend AND tipo=pruebas
 - Se facilita aplicar determinadas operaciones sobre conjuntos de contenedores

Ejemplo de un service

- apiVersion: v1
- kind: Service
- metadata:
 - name: ejemplo-service
- spec:
 - selector:
 - app: ejemplo
 - ports:
 - protocol: TCP
 - port: 80
 - targetPort: 80

Ejemplo de un deployment

- apiVersion: apps/v1
- kind: Deployment
- metadata:
 - name: ejemplo-deployment
- spec:
 - replicas: 3
 - selector:
 - matchLabels:
 - app: ejemplo
 - template:
 - metadata:
 - labels:
 - app: ejemplo
 - spec:
 - containers:
 - name: ejemplo-container
 - image: nginx
 - ports:
 - containerPort: 80