Red y Persistencia en Kubernetes

Administración de Sistemas

Unai Lopez Novoa

unai.lopez@ehu.eus

eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

Contenido

1. Configuración de red
2. Persistencia
3. Variables de entorno y objetos *Secret*.

Red

- Los objetos *Service* se utilizan para configurar elementos de la red en k8s.

- Su uso toma más sentido al utilizar objetos *Deployment* en lugar de *Pod*.
  - Cada Pod recibe una IP diferente dentro del cluster
    - Y esa IP puede cambiar durante su ciclo de vida
  - El número de réplicas (Pods idénticos) es un parámetro en el fichero de configuración.
  - Los objetos Service nos abstraen de la configuración de red.

Red

- La configuración de las IPs es visible utilizando:

```
kubectl get pods -o wide
```

  - Ejemplo:
    - Utilizando un *Deployment* con 4 réplicas de un Pod con Redis:

```
unai@unai-server:~/AS$ kubectl get pods -o wide
NAME                                     READY   STATUS    RESTARTS   AGE     IP              NODE
deployment-red-redis-5fcb5bdcb4-2dm6b    1/1     Running   0          6m54s   10.100.1.194    gk3-k8s-auto…
deployment-red-redis-5fcb5bdcb4-44hp6    1/1     Running   0          6m54s   10.100.1.66     gk3-k8s-auto…
deployment-red-redis-5fcb5bdcb4-h9w8k    1/1     Running   0          7h29m   10.100.1.3      gk3-k8s-auto…
deployment-red-redis-5fcb5bdcb4-mnw5h    1/1     Running   0          6m54s   10.100.1.130    gk3-k8s-auto…
```

La IP es diferente para cada Pod

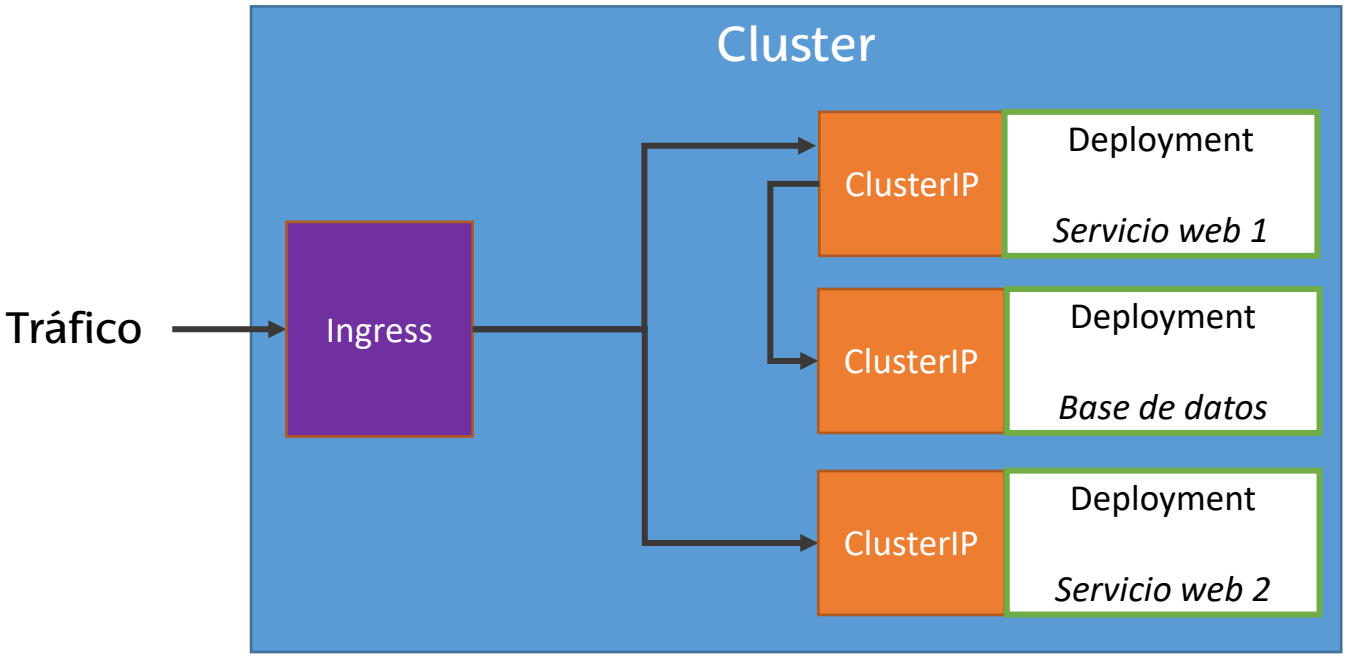
Red

- La configuración de red se hace con objetos *Service*.

- Hay 4 tipos:
  - LoadBalancer
    - Expone un puerto de un Pod al exterior del cluster.
    - Requiere un objeto LoadBalancer por cada Pod (o conjunto) a exponer.
  - NodePort
    - Expone un puerto de un Pod al exterior del cluster.
    - El puerto expuesto al exterior sólo puede estar en el rángo 30000-32767.
  - ClusterIP
    - Permite conectar diferentes objetos a nivel de cluster.
  - Ingress
    - Expone rutas HTTP y HTTPS al exterior del cluster.

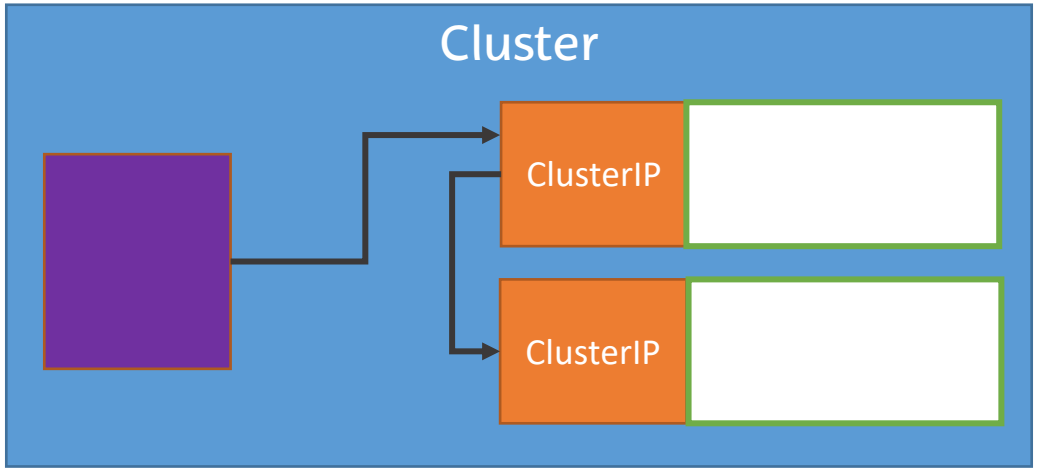
Red

- Un posible escenario:
  - Diagrama con objetos *Deployment* y objetos *Service* de red.



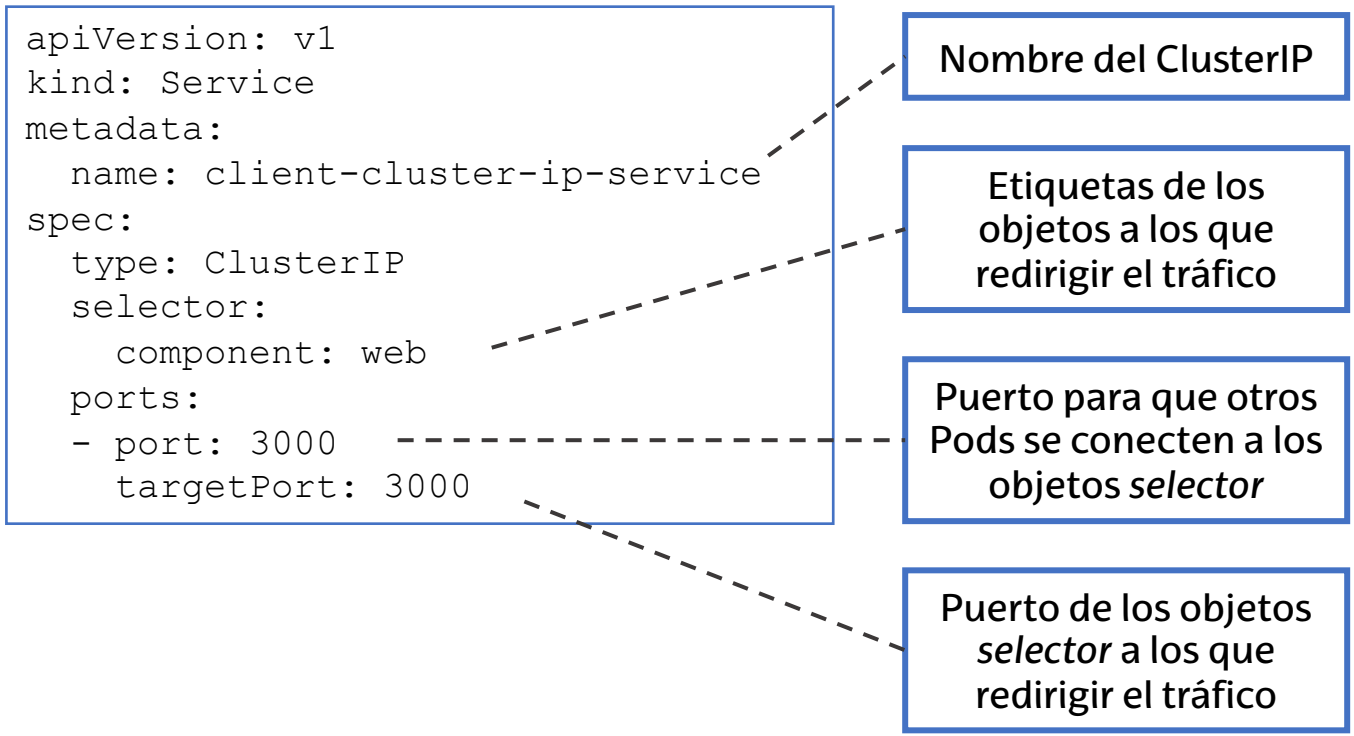
Objeto ClusterIP

- ClusterIP sirve para exponer un conjunto de Pods a otros objetos en el cluster.
  - Se utiliza para conectar objetos del cluster entre sí
  - No expone los Pods al exterior del cluster.



Objeto ClusterIP

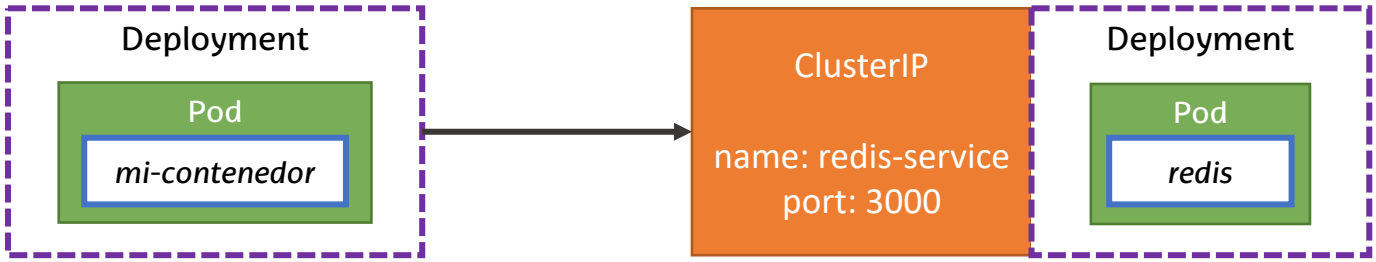
- Fichero de configuración
  - client-cluster-ip-service.yml



Objeto ClusterIP

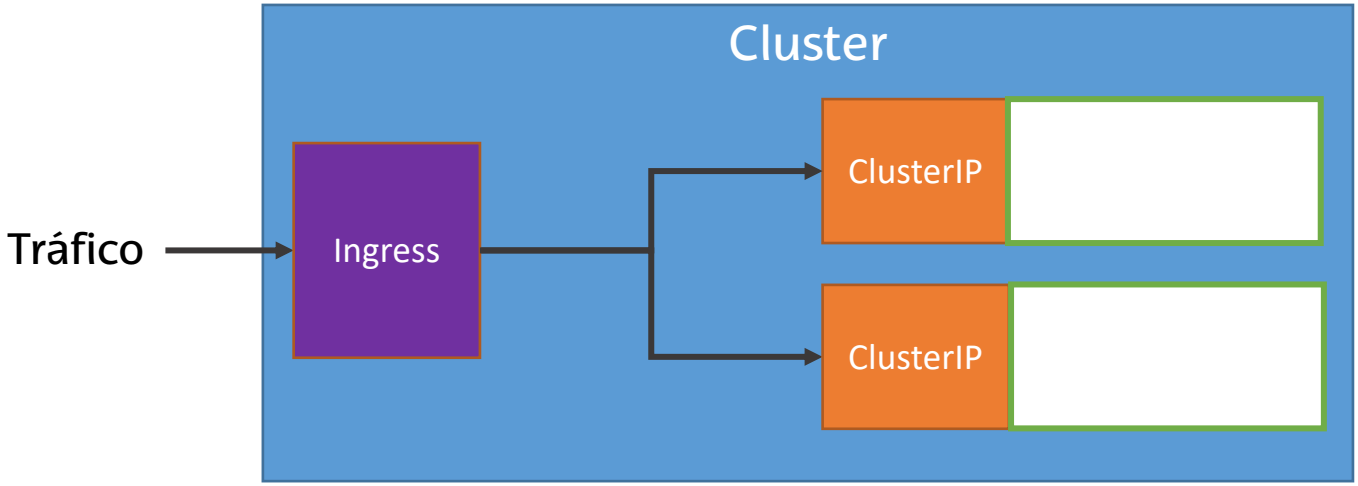
- El nombre del ClusterIP se utiliza como URL de conexión.

- Ejemplo:
  - Dos Pods en ejecución dentro del mismo cluster.
  - Para que *mi-contenedor* se conecte a *redis*, debe utilizar como dirección "redis-service" y como puerto 3000.



Objeto Ingress

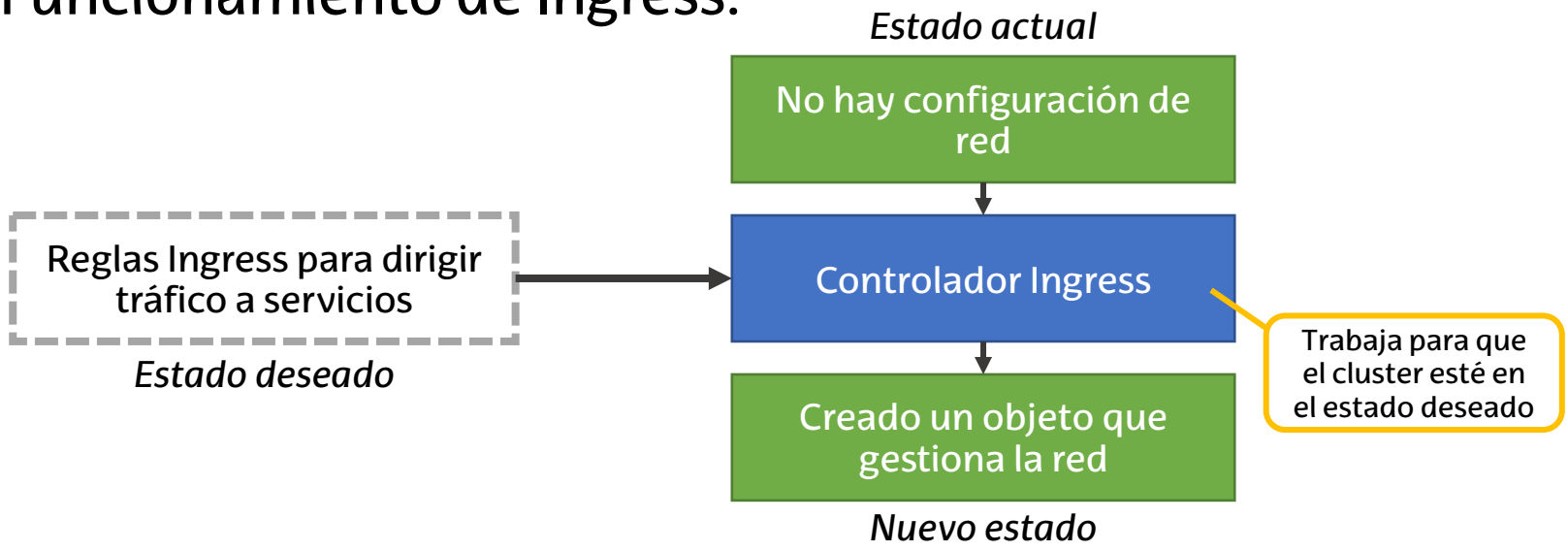
- Sirve para dirigir tráfico del exterior a objetos del cluster.
- Se configura utilizando reglas.
  - Cada regla define a qué objeto dirigir cada petición.



Objeto Ingress

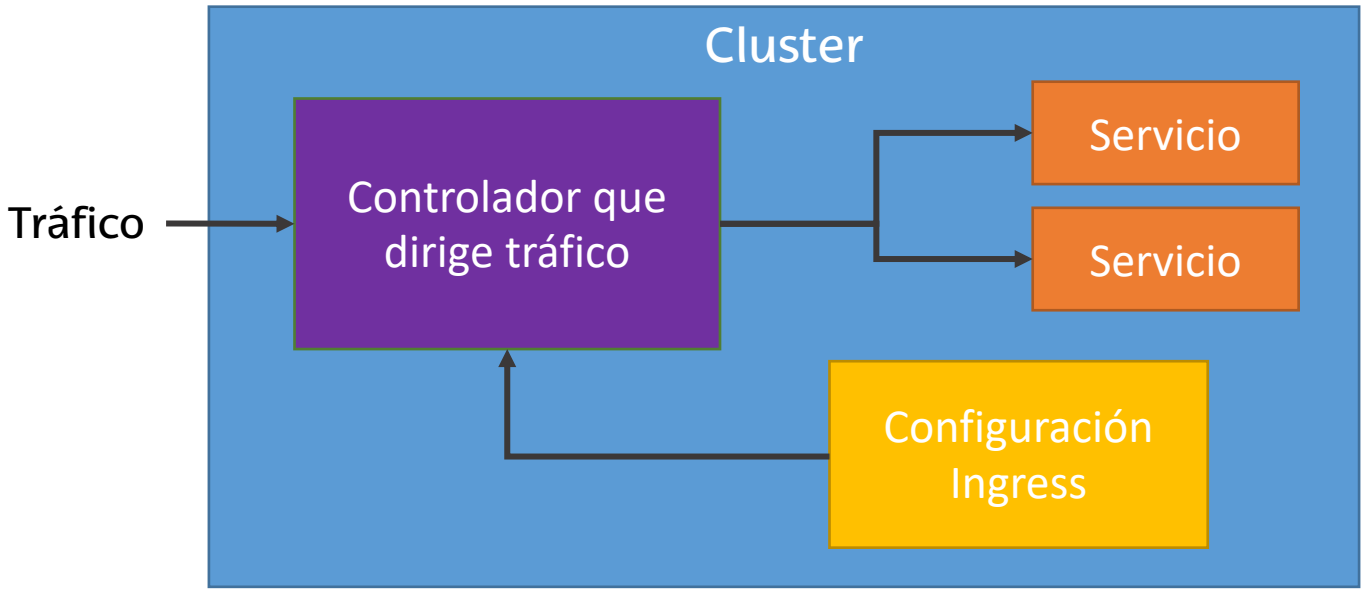
- Ingress es un tipo de objeto "controlador"
  - Los objetos controlador mantienen y gestionan un aspecto del cluster.
    - Los objetos *Deployment* también son controladores.

- Funcionamiento de Ingress:



Objeto Ingress

- Funcionamiento de Ingress:
  - Ingress redirige el tráfico externo en función de la configuración.



Objeto Ingress

- Configuración
  - Fichero mi-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
  - http:
      paths:
      - path: /*
        pathType: ImplementationSpecific
        backend:
          service:
            name: servicio-raiz
            port:
              number: 80
      - path: /config/*
        pathType: ImplementationSpecific
        backend:
          service:
            name: servicio-config
            port:
              number: 3000
```

Objeto Ingress

- Configuración
  - Fichero mi-ingress.yaml
    - *Nota:* Ingress en GKE necesita que *pathType* sea *ImplementationSpecific*.

```
…
 paths:
 - path: /*
   pathType: ImplementationSpecific
   backend:
     service:
       name: servicio-raiz
       port:
         number: 80
 - path: /config/*
   pathType: ImplementationSpecific
   backend:
     service:
       name: servicio-config
       port:
         number: 3000
```

Regla 1:
- Cuando se busque / (la raíz)
  - (p.e. http://11.22.33.44)
- Redirigir a:
  - Servicio ClusterIP "servicio-raiz"
  - Utilizar el puerto 80

Regla 2:
- Cuando se busque /config
  - (p.e. http://11.22.33.44/config)
- Redirigir a:
  - Servicio ClusterIP "servicio-config"
  - Utilizar el puerto 3000

Objeto Ingress

- Aplicar la configuración de Ingress
  - Utilizar:

```
kubectl apply -f <fichero-config-ingress>
```

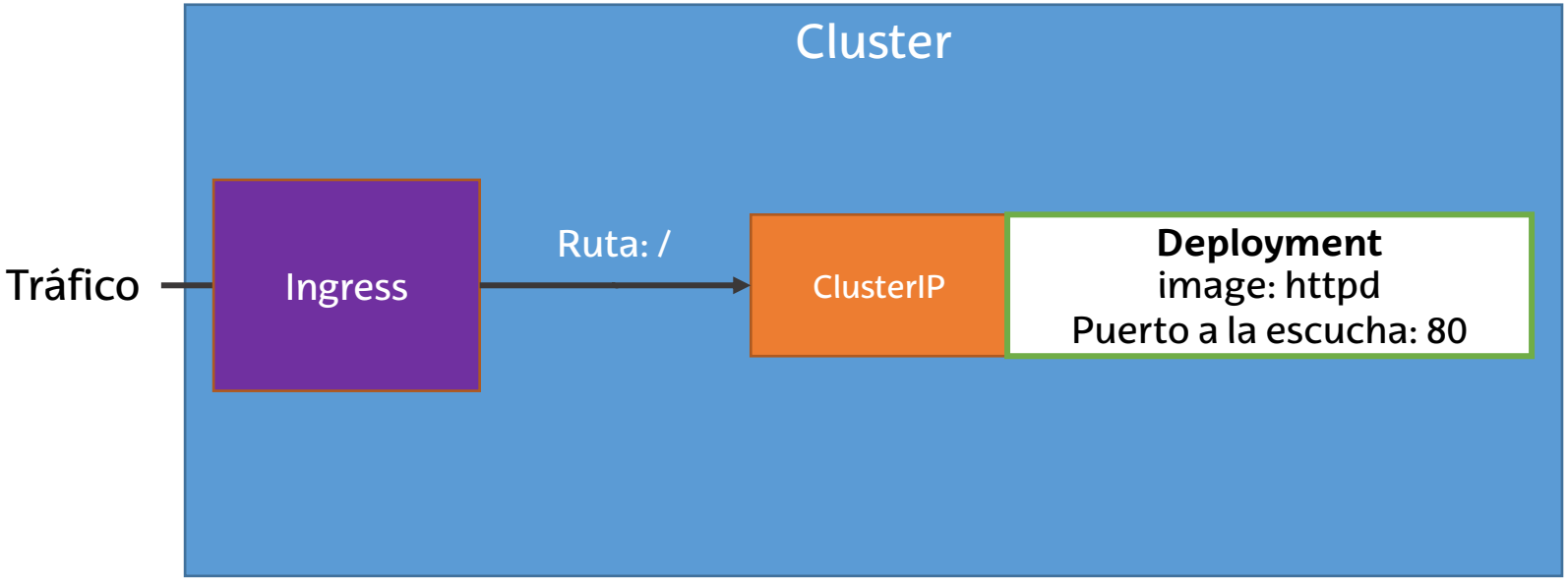
  - Verificar su estado:

```
kubectl describe ingress
```

    - Muestra:
      - IP del cluster
      - Resumen de las reglas
      - Datos de configuración
      - Registro de últimos eventos

Ejercicio 1

- Configurar Kubernetes para que ejecute un servidor web Apache usando los siguientes objetos:

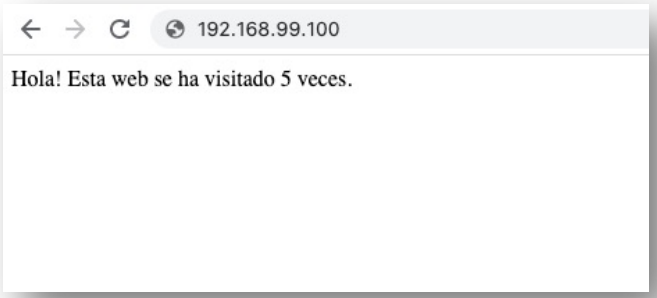


Ejercicio 1

- Crear los objetos necesarios:
  - 1 objeto Deployment
    - Réplicas: 1
    - Imagen: httpd
  - 1 objeto ClusterIP.
  - 1 objeto Ingress.
    - Definir 1 regla de Ingress para redirigir el tráfico.
    - La creación en GKE puede llevar varios minutos.
- Verificar con un navegador que el despliegue funciona.

Ejercicio 2

- Configurar un despliegue que sirva las 2 siguientes páginas web:

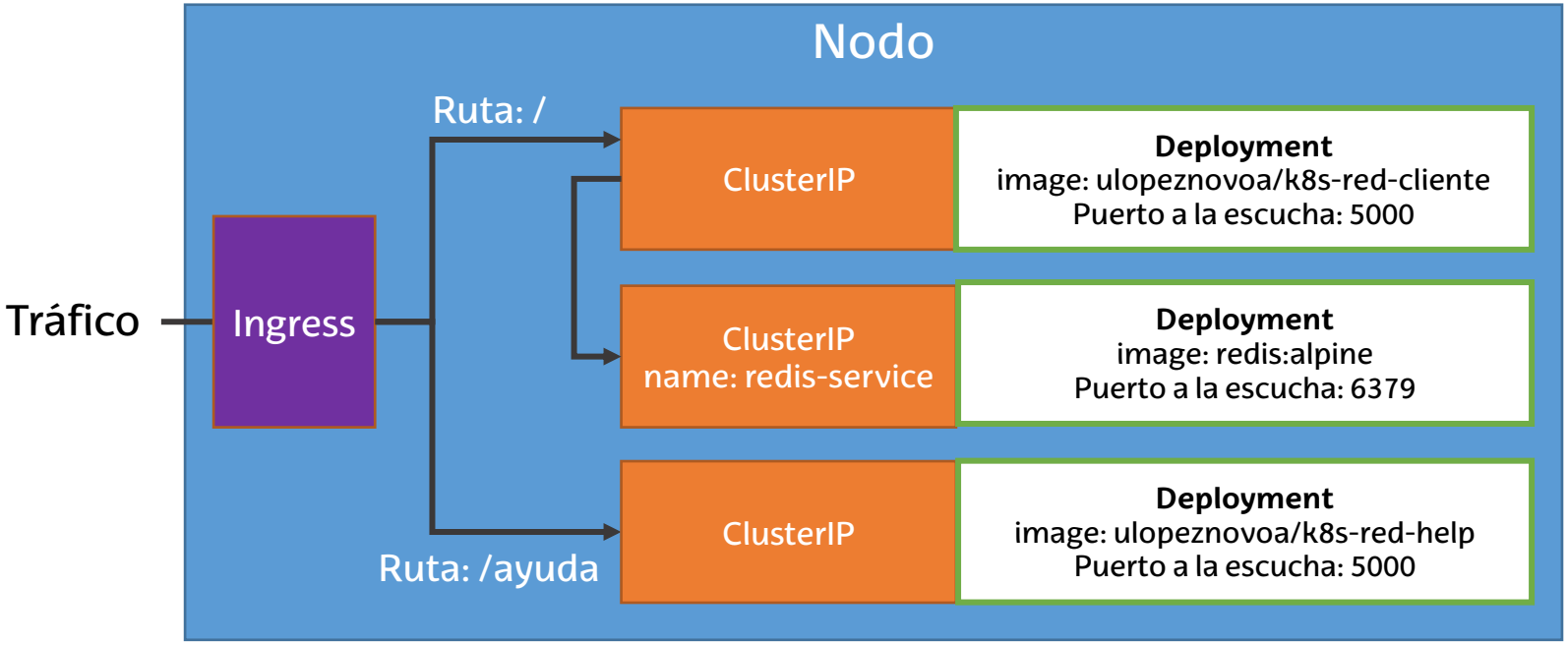


Ruta “/” (raíz)
Contador de visitas

Ruta “/ayuda”
Mensaje de ayuda

Ejercicio 2

- La estructura del despliegue es la siguiente:
  - Se deben respetar los nombres y números de puerto escritos en el diagrama.



Ejercicio 2

- Crear los objetos necesarios
  - 3 objetos Deployment
    - Replicas: 1 (en cada Deployment)
    - Ver las imágenes en el diagrama anterior.
  - 3 objetos ClusterIP.
  - 1 objeto Ingress.
    - Definir reglas de Ingress para redirigir:
      - La ruta "/" al Deployment con "ulopeznovoa/k8s-red-cliente".
      - La ruta "/ayuda" al Deployment con "ulopeznovoa/k8s-red-help".
    - La creación en GKE puede llevar varios minutos.
- Verificar con un navegador que el despliegue funciona.

Persistencia

- Los contenedores en ejecución son procesos volátiles.
  - No almacenan información o estado de manera persistente.
  - Toda la información se pierde tras un reinicio.
- Algunos contenedores necesitan una capa de persistencia.
  - Por ejemplo, una base de datos.
- Para esto se utilizan los volúmenes de Kubernetes.

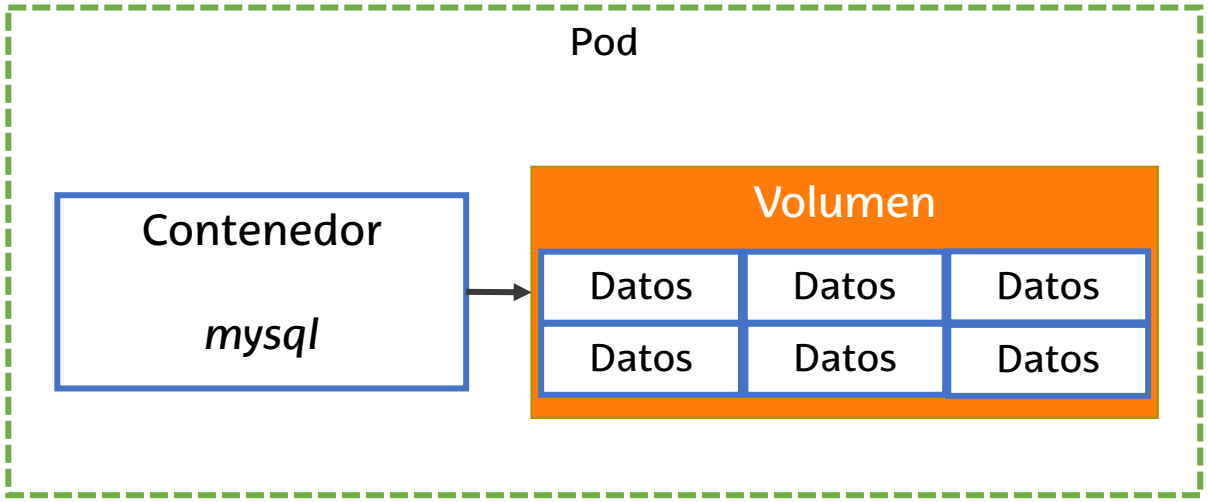
Persistencia

- En Kubernetes, hay 3 conceptos relacionados:

1) Volúmenes
   - No son apropiados para datos persistentes.
2) Volúmenes persistentes
3) Reclamación de volúmenes persistentes
   - Adaptación del inglés: Persistent Volume Claim (PVC)

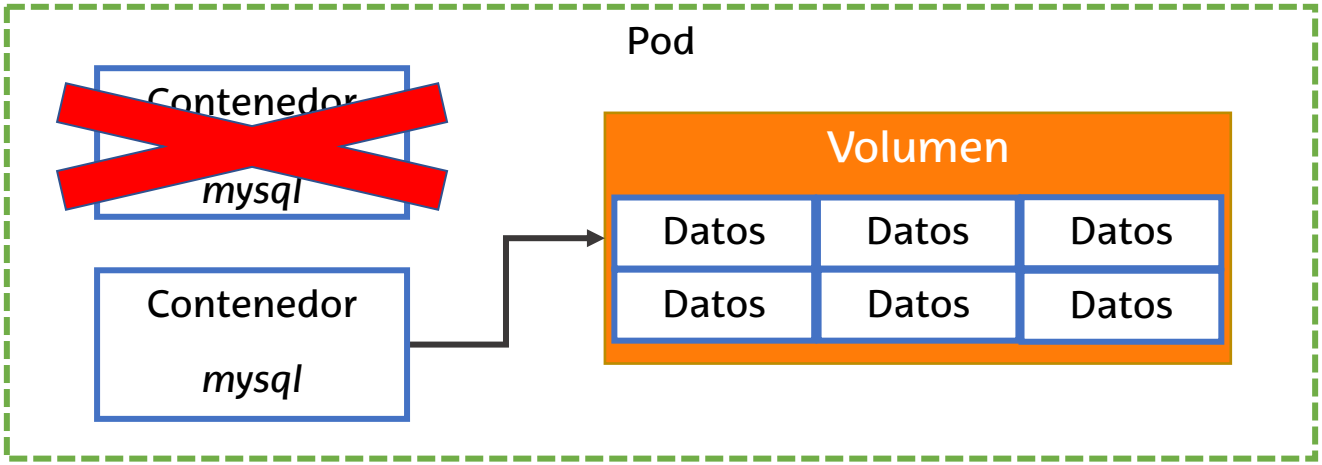
Volúmenes

- Son objetos que permiten que un contenedor almacene información a nivel de Pod.
  - Espacio de almacenamiento vinculado a un Pod
  - No son equivalentes a los volúmenes Docker.



Volúmenes

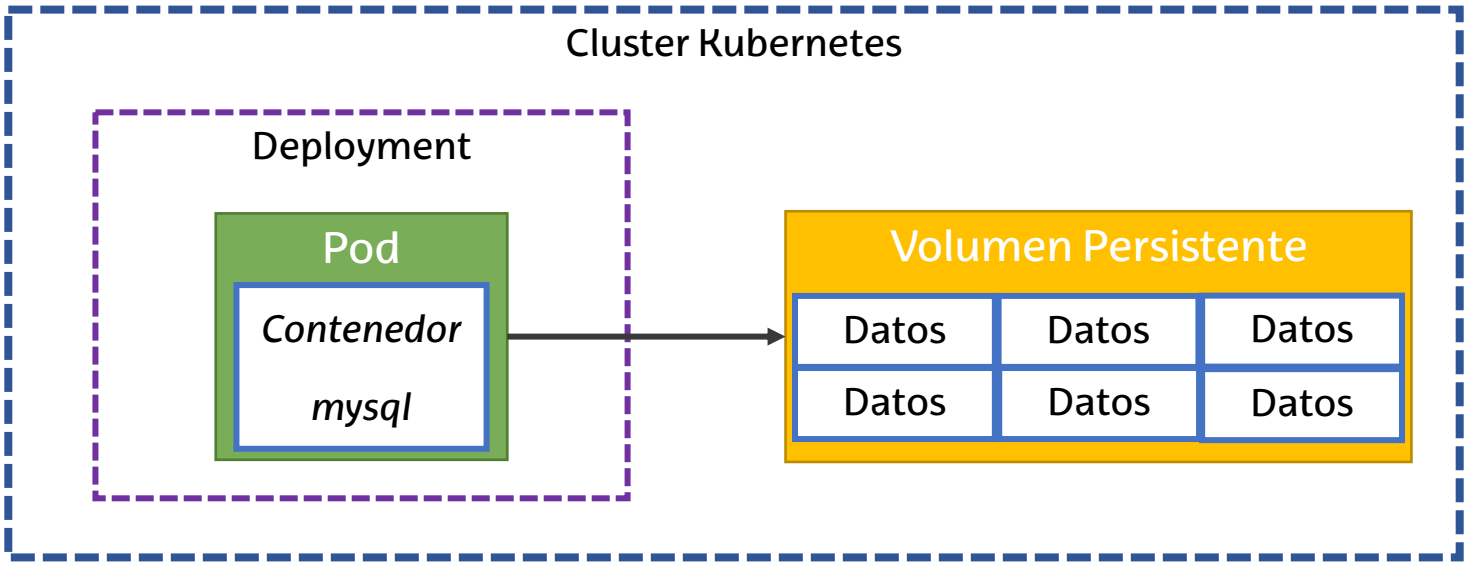
- Si hubiera un reinicio del contenedor, los datos seguirían estando accesibles.



- Si el Pod se elimina o se reinicia, los datos se pierden.
  - No utilizar volúmenes para almacenar datos persistentes.

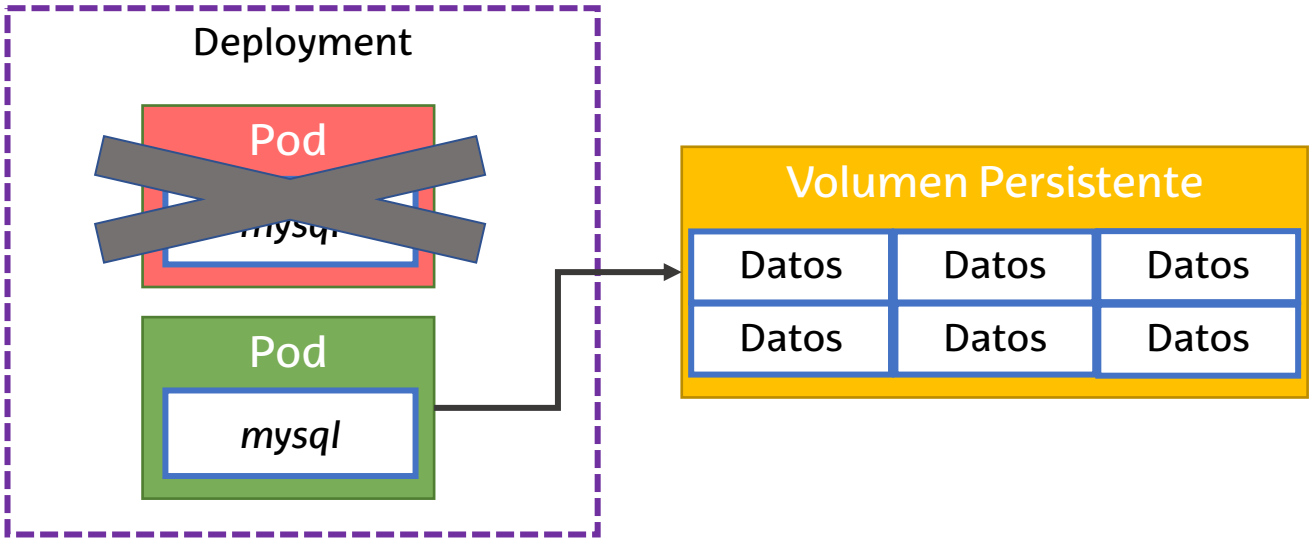
Volúmenes Persistentes

- Es un tipo de almacenamiento persistente que se almacena fuera de un Pod.
  - No está vinculado a ningún contenedor o Pod



Volúmenes Persistentes

- Si un Pod se reinicia, los nuevos contenedores pueden seguir accediendo a los datos de un volumen persistente.
  - El volumen persistente existe hasta que un usuario/administrador del cluster lo elimina.



Volúmenes Persistentes

- Creación de un volumen persistente
  - Fichero mi-volumen-persistente.yml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mi-volumen-persistente
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
```

Configuración del volumen persistente:

- 10 GB de capacidad.
- Acceso exclusivo para un nodo.

  - Modos de acceso posibles:
    - ReadWriteOnce — Un único nodo podrá leer y escribir.
    - ReadOnlyMany — Múltiples nodos pueden leer de él.
    - ReadWriteMany — Múltiples nodos pueden leer y escribir de él.

Volúmenes Persistentes

- Aplicar la creación del volumen:

```
kubectl apply -f mi-volumen-persistente.yml
```

- Verificar la creación del volumen persistente:
  - Listar volúmenes persistentes:

```
kubectl get pv
```

- Más información:
  - https://kubernetes.io/docs/concepts/storage/persistent-volumes/

Reclamación de Volumen Persistente

- Un Pod accede a un Volumen Persistente, a través de una "Reclamación de Volumen Persistente".
  - La Reclamación de VP es un objeto que solicita la creación de un VP al cluster.

- Los volúmenes persistentes pueden ser de 2 tipos:
  - Reservados estáticamente
    - A mano por el administrador del cluster
  - Reservados dinámicamente
    - A través de la API Storage Class
  - *La forma de reserva del volumen es transparente para el Pod*

Reclamación de Volumen Persistente

- Fichero de configuración
  - mi-reclamacion-vp.yml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mi-reclamacion-vp
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 2Gi
```

Esta configuración solicitará un volumen:

- Accesible por un único nodo en modo lectura y escritura.
- Con 2 GB de capacidad.

Reclamación de Volumen Persistente

- Añadir Reclamación de VP a la configuración de un Pod:
  - Como parte de un fichero para configurar 1 *Deployment*:

```
…
  template:
    metadata:
      labels:
        component: mysql
    spec:
      volumes:
        - name: mysql-storage
          persistentVolumeClaim:
            claimName: persistent-volume-claim
      containers:
        - name: mysql
          image: mysql
          ports:
            - containerPort: 5432
          volumeMounts:
            - name: mysql-storage
              mountPath: /var/lib/mysql/data
              subPath: mysql
```

Reclamación de Volumen Persistente

- Añadir Reclamación de VP a la configuración de un Pod:
  - Como parte de un fichero para configurar 1 *Deployment*:

```
…
spec:
  volumes:
    - name: mysql-storage
      persistentVolumeClaim:
        claimName: persistent-volume-claim
  containers:
    …
      volumeMounts:
        - name: mysql-storage
          mountPath: /var/lib/mysql/data
          subPath: mysql
```

Debe coincidir con el nombre asignado en el fichero de configuración de la solicitud

Configuración de la solicitud de volumen persistente:
- name: Nombre del volumen*.
- claimName: nombre de la solicitud.

Configuración del volumen persistente:
- name: Nombre del volumen*.
- mountPath: punto de montaje en el contenedor.
- subPath: ruta donde guardar los datos en el volumen (opcional).

* Ambos deben coincidir.

Reclamación de Volumen Persistente

- Añadir Reclamación de VP a la configuración de un Pod:
  - Añadir la configuración al cluster:

```
kubectl apply -f <fichero-de-configuración>
```

  - Verificar la creación del volumen persistente:
    - Listar volúmenes persistentes:

```
kubectl get pv
```

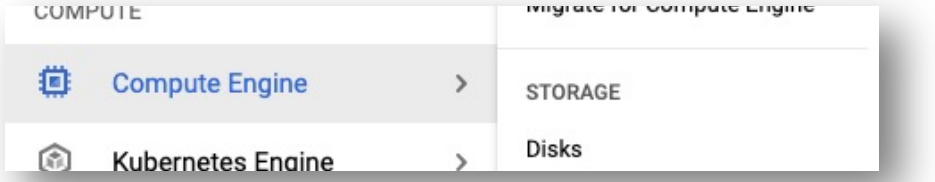
    - Listar solicitudes de volúmenes persistentes:

```
kubectl get pvc
```

Persistencia en GCP/GKE

- En el modo Autopilot de GKE no es necesario crear volúmenes persistentes a mano.
  - Se crean automáticamente al crear una Reclamación de VP.

- En GCP, por defecto, los volúmenes persistentes se crean como unidades de Google Persistent Disk
  - Servicio de almacenamiento en bloques
    - Web: https://cloud.google.com/persistent-disk
  - Los volúmenes creados así se pueden consultar en la consola de GCP
    - Sección “Compute Engine”, subsección “Discos”



Ejercicio 3

- Crear una *Reclamación de VP* para un volumen de 1 GB.
  - Nombre: "mi-rvp"
  - Modo de acceso: ReadWriteOnce
- Crear un *Deployment* con:
  - Imagen: ulopeznovoa/hola-k8s
  - Número de replicas: 1
  - Montar el volumen de "mi-rvp" en /datos1
- Acceder al *Pod* del *Deployment* en modo interactivo y crear un fichero "mi-texto.txt" con texto aleatorio en /datos1.
  - Utilizar *kubectl exec*.

(Continúa en la siguiente diapositva)

Ejercicio 3

- Eliminar el *Deployment* recién creado.
- Crear un 2º *Deployment* con:
  - Imagen: httpd:alpine
  - Número de replicas: 1
  - Montar el volumen de "mi-rvp" en /datos2
- Acceder al *Pod* del 2º *Deployment* en modo interactivo y verificar que el fichero con texto aleatorio es accesible.

Variables de entorno

- Los datos de conexión (direcciones y puertos) y autenticación (usuarios y contraseñas) entre objetos:
  - No deberían estar escritas en el código.
  - No deberían estar en ficheros estáticos dentro de un contenedor.
- La forma estándar de definirlos es utilizar variables de entorno.
  - Se definen como parte del despliegue.
  - Las aplicaciones en los contenedores las leen en tiempo de ejecución.

Variables de entorno

- Añadir variables de entorno a un objeto Deployment.
  - En el fichero de configuración, sección “env:” en “containers:”.
  - Añadir una entrada con “name” y “value” con la clave y valor de cada variable.
  - Ejemplo:

```
…
  spec:
    containers:
      - name: client
        image: ulopeznovoa/new-client
        env:
          - name: REDIS_HOST
            value: redis-cluster-ip-service
          - name: REDIS_PORT
            value: 6379
```

Define las siguientes variables:
- “REDIS_HOST” con valor “redis-cluster-ip-service”.
- “REDIS_PORT” con valor “6379”.

Objeto *Secret*

- Los *Secret* son objetos que permiten guardar y gestionar información de autenticación en Kubernetes.
  - P.e. contraseñas, tokens OAuth, claves SSH.

- Caso de uso frecuente:
  - Imágenes Docker almacenadas en repositorios privados.

- Más información:
  - https://kubernetes.io/docs/concepts/configuration/secret/

Objeto *Secret*

- Para configurar una imagen de un repositorio Docker Hub protegido con usuario y contraseña:

1) Crear un secreto que almacene las credenciales de Docker Hub:

```
kubectl create secret docker-registry <nombre-secreto> --docker-
server=<dirección-registro> --docker-username=<usuario> --docker-
password=<contraseña> --docker-email=<e-mail-usuario>
```

- Ejemplo:

```
kubectl create secret docker-registry regcred --docker-username=ulopeznovoa
--docker-password=1234 --docker-email=unai.lopez@ehu.eus
```

Objeto *Secret*

- Para configurar una imagen de un repositorio Docker Hub protegido con usuario y contraseña:

2) (Opcional) Verificar que el secreto se ha creado correctamente:

```
kubectl get secret regcred --output=yaml
```

Objeto *Secret*

- Para configurar una imagen de un repositorio Docker Hub protegido con usuario y contraseña:

3) Definir el secreto en la sección "spec:" del fichero de despliegue.
   - Añadir "imagePullSecrets".

```
…
spec:
  containers:
  - name: mi-linux
    image: ulopeznovoa/mi-linux
  imagePullSecrets:
  - name: regcred
```

Objeto *Secret*

- Si la imagen no se carga correctamente en el contenedor:
  - Buscar el identificador del Pod:

```
kubectl get pods
```

  - Leer el registro de eventos del Pod para encontrar el problema:

```
kubectl describe pod <identificador-pod>
```

- Para eliminar el objeto *Secret*:

```
kubectl delete secret <nombre-secreto>
```

Ejercicio 4

- Parte 1:
  - Obtener la imagen del servidor web Apache (“httpd”) y renombrarla como *‹vuestro-Docker-ID›*/mi-apache
  - Crear un repositorio “mi-apache” en Docker Hub y subir la imagen.
  - Crear un despliegue con:
    - Un *Deployment* que sirva la imagen.
    - Un *LoadBalancer* que permita acceder al Pod desde un navegador.
- Parte 2:
  - Convertir el repositorio DockerHub de la imagen en privado.
  - Eliminar el Deployment en uso.
  - Configurar un secreto para que Kubernetes pueda recuperar la imagen.
  - Activar el Deployment de nuevo.

Bibliografía

- Stephen Grider. “Docker and Kubernetes: The complete guide”, Udemy, 2020[1].
  - https://www.udemy.com/course/docker-and-kubernetes-the-complete-guide
- Nana Janashia, “Kubernetes Volumes explained | Persistent Volume, Persistent Volume Claim & Storage Class”, 2020[1].
  - https://youtu.be/0swOh5C3OVM
- Kubernetes Docs, 2021[2].
  - https://kubernetes.io/docs
- GCP Docs, “Volúmenes persistentes y aprovisionamiento dinámico”, 2021[2].
  - https://cloud.google.com/kubernetes-engine/docs/concepts/persistent-volumes

- Consultados en noviembre 2020[1] y noviembre 2021[2].