

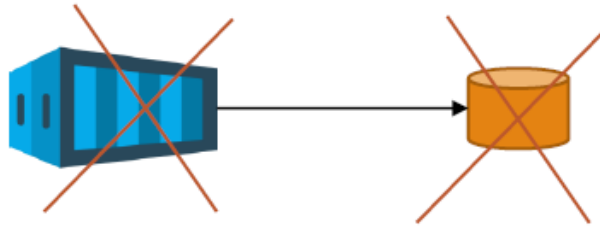
Volúmenes

Antonio Espín Herranz

Almacenamiento en K8s

❑ ALMACENAMIENTO EFÍMERO-TEMPORAL

- ❑ Por defecto el almacenamiento dentro de un contenedor de kubernetes es efímero ,temporal.
- ❑ Se crea de forma temporal en un directorio
- ❑ Es decir que cuando el pod se destruye el almacenamiento creado también es eliminado



- ❑ Esto se debe sobre todo al concepto de inmutabilidad que tiene Kubernetes.
- ❑ Recordemos el concepto de “Estado deseado” de un POD, y por lo tanto eliminarlo supone que ese estado es la destrucción completa de todos los recursos asociados a ese POD: memoria, CPU y almacenamiento .

Almacenamiento en K8s

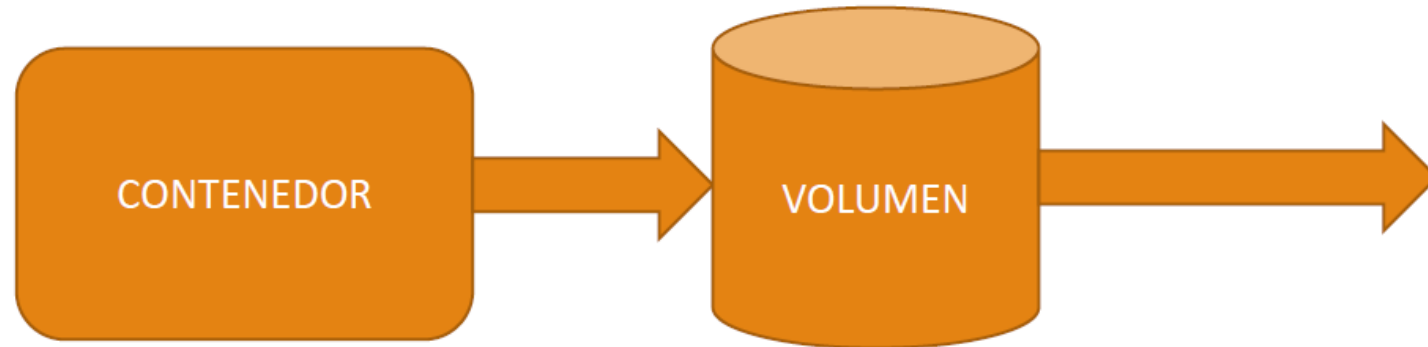
- Para almacenamiento persistente tenemos los volúmenes, no podemos utilizar el almacenamiento efímero de los PODs, por que al borrar el POD se borra todo el contenido.
 - ❑ Evidentemente esto no es siempre factible porque existen múltiples aplicaciones y PODS que necesitarán persistir este almacenamiento.
 - ❑ Para solucionar este problema se utilizar VOLUMENES, similares a los usados con Docker.
 - ❑ Kubernetes soporta volúmenes de distintos tipos (y persistente):
 - ❑ Locales
 - ❑ Externos, como cabinas y dispositivos de almacenamiento similar
 - ❑ Almacenamiento en cloud de distinto tipo, ficheros u objetos
 - ❑

Almacenamiento en K8s

- K8s soporta múltiples drivers para trabajar con volúmenes.
- **Los drivers de los entornos cloud están obsoletos y se utiliza una tecnología CSI:**
 - La tecnología CSI (**Container Storage Interface**) en Kubernetes **es un estándar** que permite la integración de sistemas de almacenamiento con plataformas de orquestación de contenedores como Kubernetes2. Antes de CSI, los proveedores de almacenamiento tenían que desarrollar controladores específicos para Kubernetes, lo que complicaba la implementación y actualización de nuevas funciones.
 - Con CSI, los proveedores pueden crear complementos que conectan sistemas de almacenamiento con Kubernetes sin modificar el código central de Kubernetes2. Esto facilita la creación de volúmenes persistentes, que son esenciales para aplicaciones que necesitan almacenar datos de manera confiable. Además, CSI permite la provisión dinámica de volúmenes, lo que mejora la flexibilidad y escalabilidad en entornos de contenedores2

Almacenamiento en K8s

DRIVERS SOPORTADOS PARA LOS VOLÚMENES



awsElasticBlockStore (deprecated)
azureDisk (deprecated)
azureFile (deprecated)
cephfs
Cinder (deprecated)
configMap
csi
downwardAPI
emptyDir
fc (fibre channel)
flexVolume
flocker
gcePersistentDisk (deprecated)
gitRepo (deprecated)
Glusterfs (deprecated)
hostPath
iscsi
local
nfs
persistentVolumeClaim
projected
portworxVolume (deprecated)
quobyte
rbd
scaleIO
secret
storageos
vsphereVolume (deprecated)

❑ CSI-Container Storage Interface

- ❑ CSI es una extension de Kubernetes que simplifica la gestión del almacenamiento
- ❑ Es un estándar que permite exponer almacenamiento de todo tipo a los workloads de kubernetes
- ❑ Si no tuviéramos este interface se necesitaría integrar el driver de almacenamiento dentro de kubernetes de forma manual y además con posibles problemas de seguridad y rendimiento ya que cada fabricante tendría su propio plugin.
- ❑ CSI implementa una arquitectura de plugin extensible, de forma que podemos añadir de forma sencilla dispositivos de almacenamiento.
- ❑ Podemos ver la lista de drivers CSI soportados en:

Drivers vs Estándar CSI

- Es un estándar , los fabricantes no tienen que crear plugins nuevos para cada cambio de versión en Kubernetes.
- Nos evitamos muchos problemas, al basarnos en un estándar se sabe de antemano que K8S trabaja con ese estándar y va a poder acceder al nuevo plugin que se desarrolle.

Añadir volúmenes a los PODs

- ❑ Utilizamos dos cláusulas para implementar el almacenamiento dentro de un POD
- ❑ Una para indicarle dónde montarlos dentro de el POD
- ❑ Una para indicar los volúmenes que vamos a utilizar

Si utilizamos un tipo de volumen de Amazon
La configuración cambiará y habrá que
Poner las propiedades que indica Amazon
Por ejemplo: **awsElasticBlockStore**


```
apiVersion: v1
kind: Pod
metadata:
  name: volumen
spec:
  containers:
  - name: nginx
    image: nginx
  volumeMounts:
  - mountPath: /home
    name: home
  volumes:
  - name: home
    hostPath:
      path: /home/kubernetes/datos
```

- **hostPath:** es de tipo local
- <https://kubernetes.io/docs/concepts/storage/volumes/>

Añadir volúmenes a los PODs

- Tener en cuenta por el anidamiento que:
- **volumes:** pertenece al **POD**
- **volumeMounts** al **contenedor**.
- En caso (raro) de tener dos contenedores en el mismo POD podrían compartir el mismo volumen y en cada contenedor se montaría una carpeta.

```
apiVersion: v1
kind: Pod
metadata:
  name: volumen
spec:
  containers:
    - name: nginx
      image: nginx
      volumeMounts:
        - mountPath: /home
          name: home
  volumes:
    - name: home
      hostPath:
        path: /home/kubernetes/datos
```



Ejemplo: Volumen en entorno local

- Un **POD** con una serie de Volúmenes.
- Indicamos la cláusula **volumeMount** para indicar la asociación entre volumen del contenedor y los espacios físicos de fuera.
- Los deben estar mapeados **3** **volumeMounts** os dentro de la clausula **volumes**
 - De **hostPath** se indica el **path real del nodo de kubernetes. (máquina física)**
 - **Aquí es válido si solo tenemos un nodo.**

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-volumen1
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - mountPath: /home
      name: home
    - mountPath: /temp
      name: temp
  volumes:
  - name: home
    hostPath:
      path: /home/kubernetes/datos
  - name: temp
    emptyDir: {}
```

OJO, **gitRepo**
Se retira por ser vulnerable

Mejor conectar al
Contenedor y clonar dentro
El repositorio

El
directorio
Tiene que
existir

Ejemplo: Volumen en entorno local

- Hacer **un describe del POD** y ver los **Mounts** dentro del apartado de Containers:
- Pueden ser **ro** (read only) **rw** (read-write)
- Luego en la sección de **Volumes** podemos ver más detalles de los volúmenes.
- Si nos movemos dentro de del directorio:
/home/kubernetes/datos veríamos lo que teníamos dentro del directorio /home del contenedor del POD

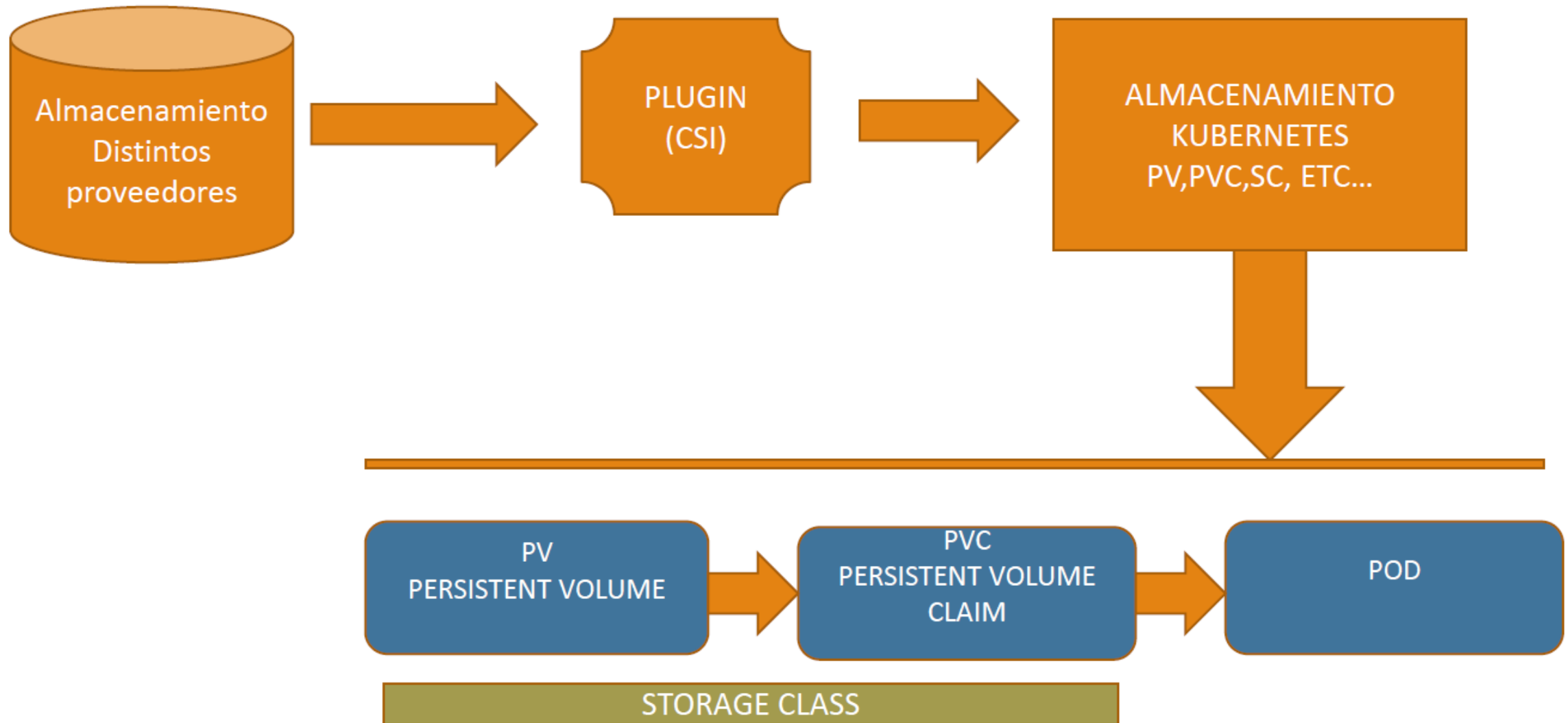
Prueba

- Crear un POD con un volumen
- Revisarlo con describe
- Borrar el POD
- Comprobar si tenemos los datos en el directorio del nodo
- Volver a lanzar el POD
- Aquí podemos tener un problema y es si k8s crear el POD en otro nodo del clúster → solución “volúmenes persistentes”

Volúmenes Persistentes (pv)

- K8S oculta al desarrollador o usuario final la complejidad real de lo que hay por detrás → esto lo puede hacer por el estándar CSI
- Un PV apunta al almacenamiento real (por ejemplo: un disco en Amazon)
- Es como un disco virtual que apunta al disco real que hay por detrás.
- Los desarrolladores para utilizar los PV crean un PVC (Persistent Volume Claim, es como una petición de volumen).
 - El PVC se liga con un PV y luego se utilizará en el POD
 - En vez de tener PV se crean clases (Storage Class) para agruparlos en clases.
 - Luego solicitamos desde el PVC, buscará un PV, lo suele hacer por un tamaño de almacenamiento.

Arquitectura



Ciclo de vida de un volumen y una reclamación

- Los PV son recursos del clúster.
- Los PVC son solicitudes de dichos recursos y también actúan como comprobaciones de reclamaciones para el recurso.
- La interacción entre los PV y los PVC sigue este ciclo de vida
- Aprovisionamiento
 - Estático o Dinámico

Ciclo de vida de un volumen y una reclamación

- **Estático**

- Un administrador de clúster crea varios PV.
- Estos contienen la información del almacenamiento real, disponible para los usuarios del clúster.
- Se encuentran en la API de Kubernetes y están disponibles para su uso.

- **Dinámica**

- Cuando ninguno de los PV estáticos creados por el administrador coincide con el PersistentVolumeClaim de un usuario, el clúster podría intentar aprovisionar dinámicamente un volumen específico para el PVC

Tipos de Volúmenes I

- **emptyDir**: Un volumen temporal que se crea cuando un pod es inicializado. Se utiliza principalmente para datos intermedios y desaparece cuando el pod se elimina.
- **hostPath**: Permite que un pod acceda a un directorio en el sistema de archivos del nodo anfitrión. Es útil para depuración, pero debe usarse con cuidado debido a posibles problemas de seguridad.
- **PersistentVolumeClaim (PVC)**: Se utiliza para solicitar almacenamiento persistente de un PersistentVolume (PV). Esto permite que los pods almacenen datos que permanecen incluso después de reiniciarse.
- **configMap** y **secret**: Diseñados para almacenar datos de configuración y secretos, como credenciales o configuraciones de aplicaciones, y proveerlos a los pods de manera segura.
- **csi**: Permite usar volúmenes proporcionados por controladores de almacenamiento CSI. Esto es altamente flexible y soporta múltiples sistemas de almacenamiento.

Tipos de Volúmenes II

- **nfs**: Facilita la conexión con sistemas de archivos de red (NFS), ideal para el acceso compartido entre varios pods.
- **awsElasticBlockStore, azureDisk, gcePersistentDisk**, etc.: Integración con servicios de almacenamiento de nubes públicas como AWS, Azure y Google Cloud Platform.
- **local**: Usado para acceder a discos locales de los nodos, adecuado para aplicaciones que requieren baja latencia.
- **ephemeral**: Introducido más recientemente, permite volúmenes efímeros para datos que solo necesitan existir mientras un pod está funcionando.

Tipos de Política de Reclamación

- En Kubernetes, las políticas de reclamación (**Reclaim Policy**) determinan qué sucede con un PersistentVolume (PV) cuando se elimina su correspondiente PersistentVolumeClaim (PVC). Hay dos tipos principales:
- **Retain** (Conservar): Esta política mantiene los datos en el volumen incluso después de que el PVC sea eliminado.
 - Se utiliza cuando necesitas garantizar que los datos no se pierdan y prefieres gestionarlos manualmente. Es ideal para casos en los que los datos son críticos y deben ser preservados.
- **Delete** (Eliminar): Con esta política, el volumen y los datos asociados se eliminan automáticamente cuando se elimina el PVC.
 - Es útil para entornos donde el almacenamiento se usa de manera temporal y no requiere persistencia una vez que el recurso es eliminado.
- ***Estas políticas las define el administrador del clúster al crear el PersistentVolume***

Ejemplo

- apiVersion: storage.k8s.io/v1
- kind: StorageClass
- metadata:
 - name: mi-storage-class
- provisioner: kubernetes.io/aws-ebs
- parameters:
 - type: gp2
- reclaimPolicy: Retain
- volumeBindingMode: Immediate

Tipos de Acceso a los Volúmenes

- En Kubernetes, los volúmenes pueden ser accedidos por los contenedores en un pod a través de diferentes modos de acceso. Estos modos determinan cómo los pods pueden interactuar con el volumen.
- Los principales modos son:
 - **ReadWriteOnce (RWO)**: Este modo permite que un único nodo monte el volumen en modo de lectura y escritura. Es ideal cuando el volumen necesita ser utilizado solo por un pod en un nodo.
 - **ReadOnlyMany (ROX)**: En este caso, múltiples nodos pueden montar el volumen al mismo tiempo, pero únicamente en modo de solo lectura.
 - **ReadWriteMany (RWX)**: Este modo permite que múltiples nodos monten el volumen simultáneamente en modo de lectura y escritura. Es útil para aplicaciones que necesitan acceso concurrente al almacenamiento.
 - **ReadWriteOncePod (RWOP)** (Introducido en Kubernetes 1.22): Similar a ReadWriteOnce, pero restringe el acceso a un único pod en lugar de un solo nodo.
- ***La elección del modo de acceso dependerá de las necesidades de tu aplicación y del tipo de Persistent Volume (PV) que estés utilizando, ya que no todos los proveedores de almacenamiento soportan todos los modos de acceso.***

Trabajando con volúmenes

- Desde el punto de vista del desarrollador de aplicaciones que van a ser ejecutadas en el cluster de Kubernetes:
- A los desarrolladores de aplicaciones les interesa más la disponibilidad y las características del almacenamiento que los detalles sobre el mecanismo de almacenamiento. Para solicitar almacenamiento se va a utilizar el recurso del cluster **PersistentVolumeClaim**. Ejemplos:
 - Quiero **20 GiB de almacenamiento permanente** que pueda compartir entre varios Pods de varios nodos **en modo lectura**.
 - Quiero **10 GiB de almacenamiento provisional** para usar desde un Pod en modo **lectura y escritura**.

NFS (Network File System)

- En K8S hay que configurar un servidor NFS que actúa como el almacenamiento central.
- Después crear recursos como **PersistentVolumes (PV)** y **PersistentVolumeClaims (PVC)** para que los pods puedan acceder al almacenamiento compartido.

NFS (Network File System)

Configurar el servidor NFS en Windows:

- En **Windows Server**, puedes instalar el rol de **servidor NFS** desde el Administrador del Servidor. Esto te permitirá compartir carpetas a través del protocolo NFS.
- Configura las carpetas compartidas y ajusta los permisos para que los clientes puedan acceder a ellas. Asegúrate de habilitar el acceso para usuarios no asignados si es necesario

NFS (Network File System)

Configurar Minikube para usar NFS:

- **Minikube puede actuar como cliente NFS.**
 - Se necesita instalar y configurar un servidor NFS en tu máquina host (Windows) para que Minikube pueda acceder al servidor NFS.
- Una vez configurado, puedes crear un PersistentVolume (PV) en Kubernetes que apunte al servidor NFS y luego usar PersistentVolumeClaims (PVC) para que tus pods accedan al almacenamiento.

NFS (Network File System)

- **metadata.name:** El nombre del PV, en este caso nfs-pv.
- **capacity.storage:** El tamaño del almacenamiento (10Gi).
- **accessModes:** Define los modos de acceso. ReadWriteMany permite que múltiples pods escriban y lean.
- **nfs.path:** La ruta al recurso compartido en el servidor NFS.
- **nfs.server:** La dirección IP o el nombre del servidor NFS.

apiVersion: v1

kind: **PersistentVolume**

metadata:

name: nfs-pv

spec:

capacity:

storage: 10Gi

accessModes:

- ReadWriteMany

nfs:

path: /ruta/compartida/en/nfs

server: 192.168.1.100

NFS (Network File System)

- **metadata.name:** El nombre del PVC, en este caso **nfs-pvc**.
- **accessModes:** Debe coincidir con los modos de acceso del PV, como ReadWriteMany.
- **resources.requests.storage:** Solicita 10Gi de almacenamiento, lo que debe coincidir con la capacidad del PV disponible.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nfs-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

NFS (Network File System)

- El **POD** hace referencia a PVC a través de un volumen.
- **volumeMounts** hace referencia al volumen y tiene que indicar el punto de montaje dentro del contenedor

apiVersion: v1

kind: **Pod**

metadata:

name: nfs-test-pod

spec:

containers:

- name: test-container

image: nginx

volumeMounts:

- **mountPath: "/mnt/storage"**

name: nfs-volumen

volumes:

- **name: nfs-volume**

persistentVolumeClaim:

claimName: nfs-pvc

NFS (Network File System)

- **spec.selector.matchLabels:** Se utiliza para seleccionar los pods gestionados por este Deployment, basado en las etiquetas.
- **template:** Define cómo deben ser los **PODs** creados por el Deployment:
- **spec.containers.image:** Especifica la imagen del contenedor, en este caso nginx.
- **volumeMounts.mountPath:** Indica dónde se montará el **almacenamiento dentro del contenedor**.
- **volumes.persistentVolumeClaim.claimName:** Enlaza el Deployment con el PVC **nfs-pvc**.

apiVersion: apps/v1

kind: **Deployment**

metadata:

name: nfs-deployment

spec:

replicas: 2

selector:

matchLabels:

app: nfs-app

template:

metadata:

labels:

app: nfs-app

spec:

containers:

- name: nfs-container

image: nginx

volumeMounts:

- mountPath: "/mnt/storage"

name: **nfs-volumen**

volumes:

- name: **nfs-volume**

persistentVolumeClaim:

claimName: **nfs-pvc**