UNIDAD 5
ALMACENAMIENTO

Volúmenes en Kubernetes

Los volúmenes se utilizan en k8s para proporcionar almacenamiento:

- Adicional o secundario al disco que define la imagen
- Persistente e independiente de la vida del pod
- Compartido entre dos contenedores en el mismo pod

Hay muchos tipos de volúmenes en k8s, podemos destacar los siguientes:

- Proporcionados por el proveedor de cloud: AWS, Azure, GCE, OpenStack, etc
- Propios de k8s:
  - **configMap**: Para usar un configMap como un directorio desde el pod
  - **emptyDir**: Volumen efímero con la misma vida que el pod. Usado como almacenamiento secundario o para compartir entre contenedores del mismo pod
  - **hostPath**: Monta un directorio del host en el pod (usado excepcionalmente)
  - **secret:** Para pasar información sensible
  - **local:** Volumen persistente, pero asociado a un nodo. Requiere definir la afinidad correctamente. Si falla el nodo, el almacenamiento no estará disponible
  - **projected:** Varios tipos en una sola definición (secret, configMap, etc.)
  - **PersistentVolumeClaim**: Para montar un volumen persistente
- Habituales en despliegues “on premises”: glusterfs, cephfs, iscsi, nfs, etc.

Volúmenes en Kubernetes

- El uso de volúmenes comprende dos funciones claramente diferenciadas:
  - La gestión del almacenamiento en el clúster de k8s para proporcionar almacenamiento a las aplicaciones, entrando a detalle en configurar los diferentes mecanismo, bien proporcionados por el proveedor de cloud o configurados directamente. Esta tarea compete normalmente al administrador de k8s. Casos típicos:
    - Configurar Azure Disk para que pueda usarlo k8s
    - Configurar Cephfs o RBD en la red local para usarlo en k8s

  - La utilización de volúmenes desde las diferentes aplicaciones que se ejecutan en el clúster. A las aplicaciones les interesa más la disponibilidad y características del almacenamiento, que los detalles sobre el mecanismo de almacenamiento. Casos típicos:
    - 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

StorageClasses

- Los administradores pueden definir diferentes clases de almacenamiento que pueden usar las aplicaciones
- Permite definir volúmenes persistentes creados de forma dinámica
- Los volúmenes no están asociados a un espacio de nombres. Son recursos del cluster
- Se puede definir un StorageClass por defecto (`isDefaultClass: Yes`)

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```

`provisioner`: Plugin a usar (AWS, Azure Disk, GlusterFS, local, etc)

`parameters`: Propios de cada plugin

`reclaimPolicy`: Delete (por defecto) o Retain

PersistentVolumeClaim

- Los desarrolladores usan volúmenes a través de la API, pidiendo características a los diferentes “StorageClasses” disponibles.
- Si es posible, se asignará el volumen con las características solicitadas. Es análogo a crear un recurso como un pod.
- No se preocupan por la configuración del “backend” de almacenamiento

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-claim
spec:
  storageClassName: standard
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
```

Sólo un pod

`storageClassName`: Si no se especifica se utiliza default

`AccessModes`: ReadWriteOnce, ReadWriteMany, ReadOnlyMany

`reclaimPolicy`: Delete (por defecto) o Retain

PersistentVolume

- Los pvc pueden utilizar volúmenes persistentes previamente definidos de forma estática por los administradores
- No todos los tipos de volúmenes soportan aprovisionamiento dinámico
- En general es menos cómodo el sistema de usar pv, pero está más controlado

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv1
spec:
  storageClassName: manual
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  hostPath:
    path: /data/pv1
```

hostPath: ← Tipo de volumen

`persistentVolumereclaimPolicy`: Recycle, Delete o Retain

Ciclo de vida

1. Aprovisionamiento
   1. Estático: Se define el `persistentVolume`
   2. Dinámico: Se define el `storageClass`
2. Conexión (*binding*). Se aplica un `persistentVolumeClaim` que se crea si es posible de forma estática o dinámica
3. Uso. Se utiliza el volumen desde un pod, definiendo el pvc como un volumen
4. Liberación. Cuando el pod termina, el pvc mantiene el volumen reservado (*bound*). Es necesario que se borre explícitamente el pvc para liberarlo.
5. Recuperación. Una vez liberado el volumen, la `claimPolicy` determinará si es:
   1. Reciclar: En aprovisionamiento estático. Se borra el contenido, pero se puede usar el mismo volumen de nuevo, sin tener que definir un nuevo pv
   2. Borrar: Se borra el contenido y el volumen
   3. Retener: Se mantiene el contenido y el volumen. Se puede crear un nuevo pvc que pueda usar el mismo volumen.

Backend de almacenamiento

https://unofficial-kubernetes.readthedocs.io/en/latest/concepts/storage/persistent-volumes/

Volume Plugin	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWSElasticBlockStore	✓	-	-
AzureFile	✓	✓	✓
AzureDisk	✓	-	-
CephFS	✓	✓	✓
Cinder	✓	-	-
FC	✓	✓	-
FlexVolume	✓	✓	-
Flocker	✓	-	-
GCEPersistentDisk	✓	✓	-
Glusterfs	✓	✓	✓
HostPath	✓	-	-
iSCSI	✓	✓	-
PhotonPersistentDisk	✓	-	-
Quobyte	✓	✓	✓
NFS	✓	✓	✓
RBD	✓	✓	-
VsphereVolume	✓	-	-
PortworxVolume	✓	-	✓
ScaleIO	✓	✓	-

Ejemplo 1: Crear un pod que use un pvc dinámico
Ejemplo 2: Crear un pod que use un pvc estático
Ejemplo 3: Despliegue de Wordpress escalable con almacenamiento en NFS