Persistencia de datos en Kubernetes

KEEPCODING
Tech School

Persistencia de Datos en Kubernetes

- En esta sección:
  - Introducción / Tipos de storage
  - Objetos relacionados con la persistencia:
    - PodSpec → volumes + mounts (no son objetos como tales)
    - PersistentVolumes (PV)
    - PersistentVolumeClaim (PVC)
    - Storage classes.

Introducción / tipos de storage

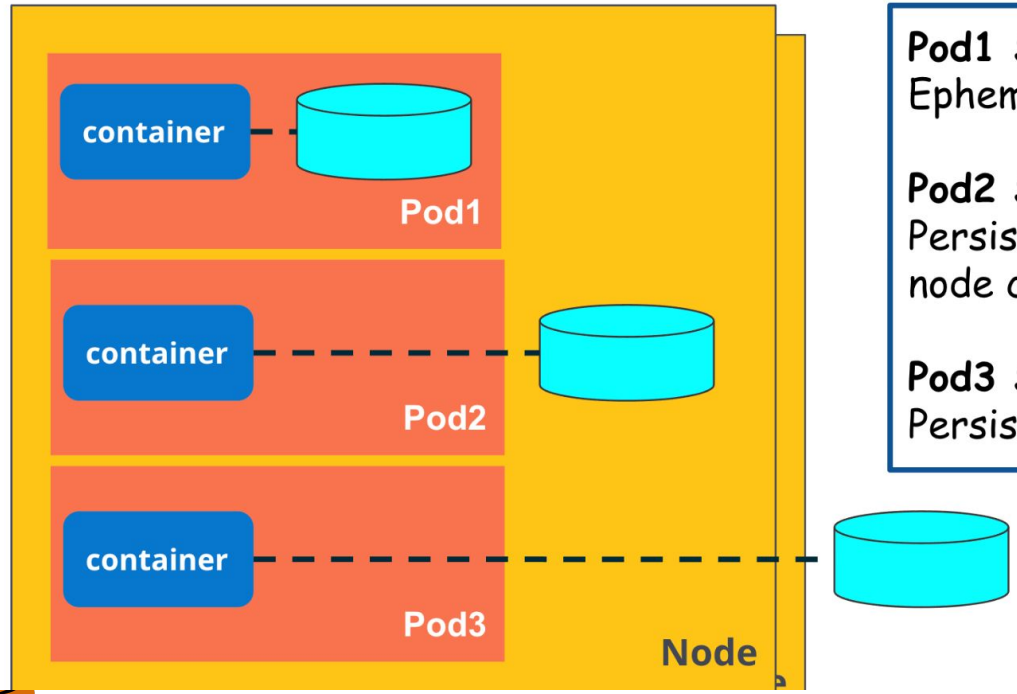
KEEPCODING
Tech School

Persistencia de Datos en Kubernetes

- Storage en Kubernetes:
  - Kubernetes no proporciona una solución de storage distribuído por defecto, sino que es compatible con muchos tipos.
  - Existe multitud de tipos de volúmenes o soluciones de storage que podremos utilizar, cuya clasificación podría resumirse en:
    - **Efímero / Persistente**
    - **Local / Remoto**
  - Algunos tipos de storage permiten **provisionamiento dinámico**, otros no.
  - Distintos tipos de storage permitirán diferentes **tipos de acceso** (ReadWriteOnce, ReadOnlyMany, ReadWriteMany, …)

Persistencia de Datos en Kubernetes

Storage



Pod1 Storage
Ephemeral

Pod2 Storage
Persistent but hard to handle when node changes

Pod3 Storage
Persistent & easy to reuse

https://kubernetes.io/docs/concepts/storage/volumes

Volúmenes (volume)

KEEPCODING
Tech School

Volúmenes (volume)

- Los utilizamos para declarar **datos externos al filesystem del contenedor.** Tendrán que ser **montados** en los contenedores. Tanto para inyectar datos en forma de ficheros ó directorios (configMaps / secrets) como para declarar volúmenes externos persistentes (al estilo docker volume).
- Los definimos directamente en el POD, no son objetos de Kubernetes con su propio ciclo de vida. Se montan a nivel de container.
- Dependiendo del tipo, tendrán que estar creados previamente o no.
- Obviedades / consideraciones:
  - Si elegimos un volúmen efímero (**emptyDir**) y destruye y crea un POD los datos se perderán.
  - Si elegimos un volúmen local (**hostPath**) y al destruir y recrear un POD éste se levanta en otro nodo perderemos la información.

https://kubernetes.io/docs/concepts/storage/volumes

Volúmenes - tipos

Tipos de volúmenes. Cada uno tiene sus parámetros.

- **awsElasticBlockStorage**
- **azureDisk**
- cephfs
- **configMap**
- **emptyDir**
- **gcePersistentDisk**
- **hostPath**
- nfs
- **secret**
- vsphereVolume
- **persistentVolumeClaim → Ref. a PVC.**

Specs: https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.31#volume-v1-core

https://kubernetes.io/docs/concepts/storage/volumes

Volúmenes - montaje

Los volúmenes se han de montar en los contenedores de los pods.

- **`podSpec→Containers→volumeMounts`**

Conclusión:

- volumes (nivel de POD): declara los volumenes
- volumeMounts (nivel de Contaier): monta esos volúmenes en directorios

volumeMount spec:
https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.31/#volumemount-v1-core

https://kubernetes.io/docs/concepts/storage/volumes

emptyDir (storage local)

- Es el tipo de volúmen más básico, efímero.
- Un contenedor que se reinicia no se cambia de nodo. Un emptyDir nos permite persistir la información entre reinicios de los contenedores. No persiste el borrado y recreación del Pod.
- Nos **permite compartir información entre contenedores de un mismo Pod**

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

https://kubernetes.io/docs/concepts/storage/volumes

hostPath (storage local, como un bind-mount)

- Nos permite montar un volúmen del host (o crear un directorio específico dentro)
- Útil si queremos acceder a paths específicos del Host, por ejemplo /var/lib/docker
- Muy útiles para **DaemonSets**
- Al estilo de los bind-mounts de docker a nivel de host. Cuidado con lo que montamos!

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    hostPath:
      path: /data
```

https://kubernetes.io/docs/concepts/storage/volumes

gcePersistentDisk, aws ELBs, etc

- Storage remoto (network based storage).
- Nos permite montar un disco de Google Compute Engine directamente en un POD.
- **Tiene que estar creado a priori → gestión manual e independiente de Kubernetes (no los usaremos así).**
- Kubernetes se encarga de formatearlo si no lo está.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: test-pd
spec:
  containers:
  - image: k8s.gcr.io/test-webserver
    name: test-container
    volumeMounts:
    - mountPath: /test-pd
      name: test-volume
  volumes:
  - name: test-volume
    # This GCE PD must already exist.
    gcePersistentDisk:
      pdName: my-data-disk
      fsType: ext4
```

```
~$ gcloud compute disks create --size=10GB --zone=europe-west1-b my-data-disk
```

https://kubernetes.io/docs/concepts/storage/volumes

Volúmenes

- Demo!

https://kubernetes.io/docs/concepts/storage/volumes

PersistentVolumes, Claims y StorageClasses

KEEPCODING
Tech School

StorageClasses

- Permite definir clases / tipos de almacenamiento..
- Pueden ser por velocidad de escritura, por topología, por políticas de backup, etc.
- Generalmente se usan con provisionamiento dinámico.
- Parámetros importantes:
  - provisioner
  - reclaimPolicy
  - parameters

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: slow
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  replication-type: none
```

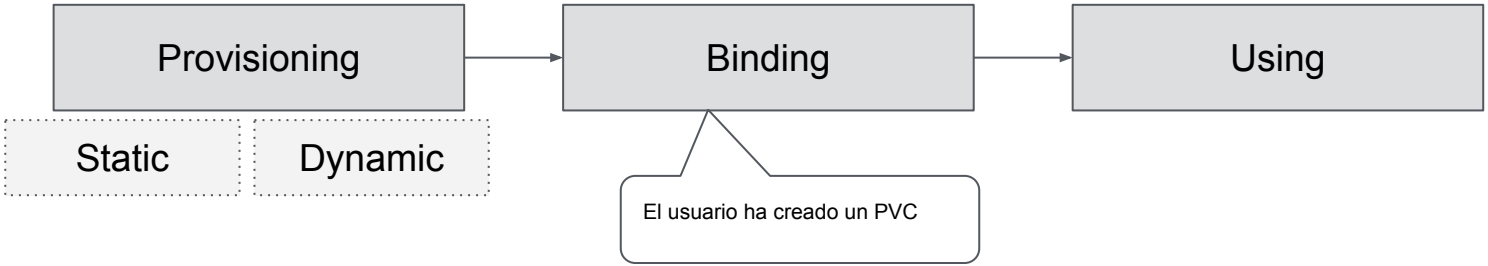
```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  replication-type: none
```

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast-regional
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-ssd
  replication-type: regional-pd
```

https://kubernetes.io/docs/concepts/storage/storage-classes/

PersistentVolume (PV / objeto)

- Nos permite gestionar volúmenes persistentes como objetos de Kubernetes (con su ciclo de vida).
- Representan volúmenes de almacenamiento que han sido **provisionados manualmente por un administrador o dinámicamente mediante storage classes + PVCs**.
- Los pods los consumen a través de PersistenVolumeClaims (NO enlazan con PVs directamente).
- Ejemplo de creación de PVs manualmente aquí



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

PersistentVolumeClaims (PVC)

- Representan **solicitudes de storage**. El PVC consumirá PVs al igual que un POD consume recursos de los nodos (CPU y memoria).
- Los PersistentVolumeClaims nos permiten **abstraer completamente del tipo de almacenamiento** que haya por debajo.
- Utiliza las **clases de storage**, y a través de ellas, se enlazará a los PVs.
- Si no hay PVs disponibles (**provisión manual**) el sistema intentará auto-provisionar PVs (**provisión dinámica** basada en storage classes).
- Definidos normalmente en volumeClaimTemplates y usados en statefulsets.
- Un deployment (o pod) también puede usar un PVC, pero con las limitaciones que ya conocemos en cuanto a réplicas.
- Al borrar un PVC se borrarán o no los PVs asociados dependiendo de la reclaimPolicy (más aquí)

https://kubernetes.io/docs/concepts/storage/persistent-volumes/#persistentvolumeclaims

PersistentVolumeClaims

- Especificación aquí
- Ejemplo simple:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  storageClassName: standard
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
```

https://kubernetes.io/docs/concepts/storage/storage-classes/

PersistentVolumeClaims

- Ejemplo obtenido de un StatefulSet (`volumeClaimTemplates`)

```
volumeClaimTemplates:
- metadata:
    name: www
  spec:
    accessModes: [ "ReadWriteOnce" ]
    storageClassName: "my-storage-class"
    resources:
      requests:
        storage: 1Gi
```

https://kubernetes.io/docs/concepts/storage/storage-classes/

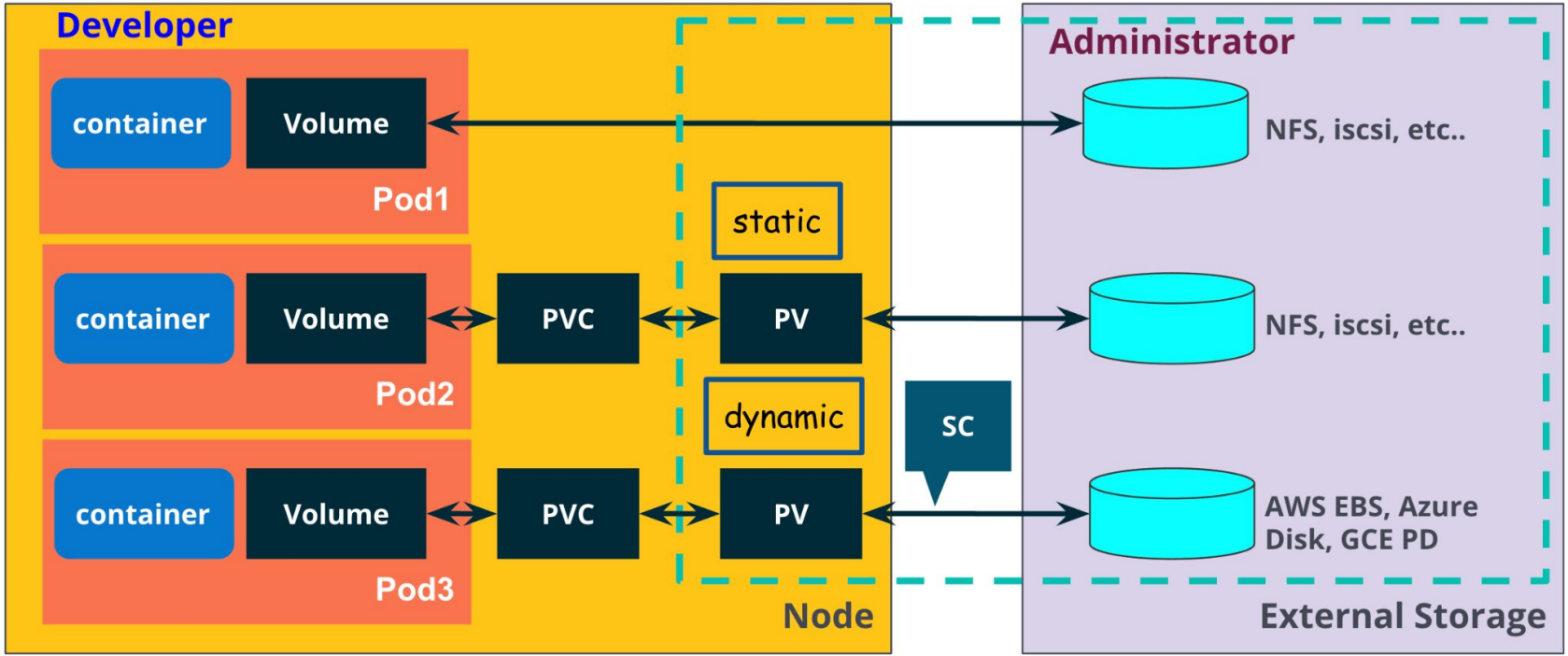
Storage - Demo

- Demo!
- Veremos los PVC y PVs con provisionamiento automático en la demo de Statefulsets
- Para PVCs y PVs con provisionamiento manual:
  - https://kubernetes.io/docs/tasks/configure-pod-container/configure-persistent-volume-storage/

https://kubernetes.io/docs/concepts/storage/storage-classes/

Storage - Resumen

Volume, Persistent Volume (PV), Persistent Volume Claim (PVC), Storage Class (SC)



Ejercicios persistencia de datos

KEEPCODING
Tech School

2.8.1

- Crea un POD con 2 contenedores que tengan visibilidad entre ellos a través de un directorio (no ha de ser persistente).

- Crea un Deployment con 2 réplicas (pero sólo 1 contenedor por POD) que tengan visibilidad entre ellas a través de un directorio persistente (puedes utilizar PVC).

- Analiza las implicaciones y diferencias de los dos casos anteriores.
  - ¿Pueden ser los accesos Read/Write en todos los casos?
  - ¿Qué solución hace falta en cada caso?

2.8.2

Crea un Deployment que monte un disco persistente de tipo “gcePersistentDisk” en GCP en un directorio /data.

Se puede hacer de dos formas diferentes:

- Usando un "volume" directamente de tipo "gcePersistentDisk"
- A través de un PVC

¿Qué forma te resulta más cómoda?

Haciendolo vía PVC podemos controlar la creación y destrucción del disco con kubectl. De la otra forma el disco ha de manejarse de forma separada.

2.8.3

Lo mismo que en 2.8.2, pero que el disco se provisione de forma automática con un tamaño de 20GB.

¿Para qué nos vale la retain policy?

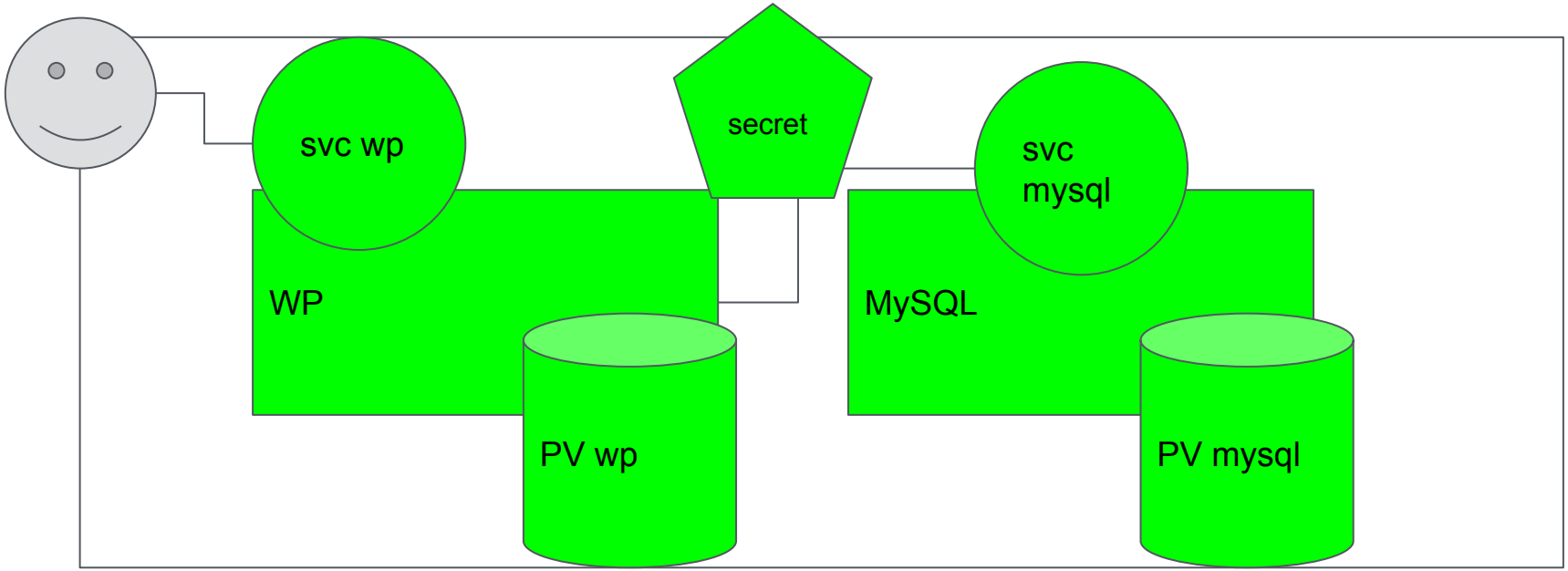
2.8.4

- Crea un stack de Wordpress con MySQL con persistencia ambos

https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/

2.8.4

- Crea un stack de Wordpress con MySQL con persistencia ambos



https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/

KEEPCODING

Tech School

Madrid | Barcelona | Bogotá