Kubernetes容器持久化存储

卷

- emptyDir
  - emptyDir是一种简单的、用于临时数据存储的空目录，其生命周期与Pod生命周期同步，即当Pod被删除时，emptyDir卷中的内容就会丢失，所以常常用于存储应用系统产生的中间临时文件
- hostPath
  - hostPath卷指向工作节点文件系统上特定的目录或者是文件，类似于Docker中的mount path。hostPath卷中的内容不会因为Pod的删除而丢失，因为数据是保存在工作节点中的文件系统中
  - 但是，由于使用的是节点的文件系统，而Pod的节点调度通常是不固定的，那么一旦Pod被调度到其它节点时，数据同样会丢失。所以，hostPath卷在实际中并不常使用
- NAS
  - 使用网络存储，例如GCE持久磁盘，NFS存储集群

使用持久存储

- 当用户想要在集群中使用数据库或者其它需要将数据持久化，并且Pod重启时数据仍然可用时，就必须建立持久化存储方案
- emptyDir与hostPath能满足需求吗?
  - emptyDir类型Volume的生命周期与Pod生命周期相同，当Pod被删除时，该Volume也会被删除，无法做到持久化存储
  - hostPath使用节点的目录作为挂载目录，当Pod删除时，节点的数据仍然存在。但是此时就需要将Pod与当前节点进行强绑定，并不是个好主意
- 使用NAS作为k8s持久化存储
  - 既然无法使用常规卷作为持久化存储，但是我们可以用NAS(Network Attached Storage，网络附加存储)作为持久化存储卷
  - 当前k8s支持众多的网络附加存储方案，如GCE持久磁盘，AWS弹性块存储以及NFS共享存储等
  - 以NFS卷为例

```yaml
volumes:
  - name: mysql-data
  nfs:
    server: 1.2.3.4
    path: /opt/mysql
```

  - 使用 kubectl describe pod.sepc 获取更多的NAS支持

在pod配置中直接定义NAS是个好主意吗?

- 在上面的例子中，在pod.spec.volumes中直接定义了NFS服务器路径以及共享路径，这就需要开发人员知道详细的NFS共享服务器信息，而NFS这类基础架构，通常是由其它专业人员维护的。如此一来，就在pod中添加了本不属于开发人员职责范围的配置信息
- k8s的一个重要理念就是为应用程序以及开发人员隐藏真实的基础设施，使他们不必担心基础设施的具体状态。而直接将NFS配置信息写入至pod配置中，显然违反了这一理念:开发人员需要了解NFS的细节

使用PV与PVC解耦pod

PV

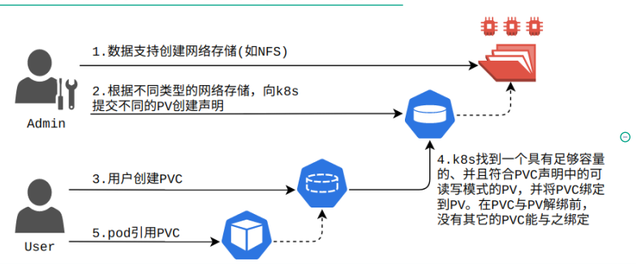
- PersistentVolume，持久卷，是k8s提供的一种对NAS等网络存储的一种封装资源，其内部保存着k8s所支持的实际存储类型(NFS、GCE持久磁盘等)、位置(如IP地址，共享目录路径)和其它属性(容量、读取策略等)

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv
spec:
  capacity:
    storage: 200Gi
  accessModes:
    - ReadWriteOnce
    - ReadOnlyMany
  persistentVolumeReclaimPolicy: Retain
  storageClassName: mario
  nfs:
    server: 1.2.3.4
    path: /opt/mysql
```

  - capacity
    - 定义PersistentVolume的大小
  - accessModes
    - 定义客户端的读写模式
    - ReadWriteOnce
      - 只能被单个客户端挂载为读写模式
    - ReadOnlyMany
      - 可以被多个客户端挂载为只读模式
    - ReadWriteMany
      - 可以被多个客户端挂载为读写模式
  - persistentVolumeReclaimPolicy
    - 当持久卷声明的绑定被删除时当前持久卷的保留策略
    - Retain
      - 保留数据与持久卷
    - Delete
      - 删除该持久卷
  - storageClassName
    - 当前PV所属的StorageClass名称
    - 该字段缺省值为empty string
- PV不属于任何命名空间，属于节点层面的资源。可以认为PV是对网络存储空间的一层封装，将外部存储空间划分成一个又一个的PV

PVC

- 当集群用户需要在其pod中使用持久化存储时，首先需要创建持久卷声明(Persistent Volume Claim, PVC)，指定所需要的最低容量要求和访问模式，然后用户将持久卷声明提交给k8s，k8s将找到可匹配的持久卷并将其绑定到持久卷声明
- 简单来说，PVC描述了pod所希望使用的持久化存储的属性，例如容量大小，可读写权限等等
- 用户创建的PVC若想要真正被容器使用，则必须满足两个条件
  - pv.spec与pvc.spec必须匹配，如容量，可读写权限等
  - PV和PVC的storageClassName的字段必须一致
- PVC与PV的绑定过程
  - 用户创建PVC，指定持久化存储的容量、可读写模式等信息，提交给k8s，k8s将在可用的PV中选择符合用户需求的PV，将该PV与PVC进行绑定
  - PVC表明了用户"想要什么"，k8s会自动地完成选择并绑定的操作



PV与PVC的设计

- 可以看到，从Pod到最底层的网络存储，k8s额外提供了两层抽象: PV与PVC。其目的就在于屏蔽硬件细节，以及更灵活的替换
- PV是底层网络存储的抽象，而PVC则是用户需求的抽象