k8s基础篇-持久化存储

Volumes

介绍

官方：

http://docs.kubernetes.org.cn/429.html

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

默认情况下容器中的磁盘文件是非持久化的，对于运行在容器中的应用来说面临两个问题，第一：当容器挂掉kubelet将重启启动它时，文件将会丢失；第二：当Pod中同时运行多个容器，容器之间需要共享文件时。Kubernetes的Volume解决了这两个问题。

背景

在Docker中也有一个docker Volume的概念，Docker的Volume只是磁盘中的一个目录，生命周期不受管理。当然Docker现在也提供Volume将数据持久化存储，但支持功能比较少（例如，对于Docker 1.7，每个容器只允许挂载一个Volume，并且不能将参数传递给Volume）。

另一方面，Kubernetes Volume具有明确的生命周期 - 与pod相同。因此，Volume的生命周期比Pod中运行的任何容器要持久，在容器重新启动时能可以保留数据，当然，当Pod被删除不存在时，Volume也将消失。注意，Kubernetes支持许多类型的Volume，Pod可以同时使用任意类型/数量的Volume。

内部实现中，一个Volume只是一个目录，目录中可能有一些数据，pod的容器可以访问这些数据。至于这个目录是如何产生的、支持它的介质、其中的数据内容是什么，这些都由使用的特定Volume类型来决定。

要使用Volume，pod需要指定Volume的类型和内容（`spec.volumes`字段），和映射到容器的位置（`spec.containers.volumeMounts`字段）。

卷的类型

Kubernetes支持Volume类型有：

```
- emptyDir
- hostPath
- gcePersistentDisk
- awsElasticBlockStore
- nfs
- iscsi
- fc (fibre channel)
- flocker
- glusterfs
```

```
- rbd
- cephfs
- gitRepo
- secret
- persistentVolumeClaim
- downwardAPI
- projected
- azureFileVolume
- azureDisk
- vsphereVolume
- Quobyte
- PortworxVolume
- ScaleIO
- StorageOS
- local
```

EmptyDir实现数据共享

使用emptyDir，当Pod分配到Node上时，将会创建emptyDir，并且只要Node上的Pod一直运行，Volume就会一直存。当Pod（不管任何原因）从Node上被删除时，emptyDir也同时会删除，存储的数据也将永久删除。注：删除容器不影响emptyDir。

示例：

```
[root@k8s-master emptydir]# pwd
/root/k8s_practice/emptydir
[root@k8s-master emptydir]# cat pod_emptydir.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-emptydir
  namespace: default
spec:
  containers:
  - name: myapp-pod
    image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
    imagePullPolicy: IfNotPresent
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  - name: busybox-pod
    image: registry.cn-beijing.aliyuncs.com/google_registry/busybox:1.24
    imagePullPolicy: IfNotPresent
    command: ["/bin/sh", "-c", "sleep 3600"]
    volumeMounts:
    - mountPath: /test/cache
      name: cache-volume
```

```
  volumes:
  - name: cache-volume
    emptyDir: {}
```

启动pod，并查看状态

```
[root@k8s-master emptydir]# kubectl apply -f pod_emptydir.yaml
pod/pod-emptydir created
[root@k8s-master emptydir]#
[root@k8s-master emptydir]# kubectl get pod -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP             NODE         NOMINATED
NODE   READINESS GATES
pod-emptydir   2/2     Running   0          10s   10.244.2.166   k8s-node02   <none>
        <none>
[root@k8s-master emptydir]#
[root@k8s-master emptydir]# kubectl describe pod pod-emptydir
Name:         pod-emptydir
Namespace:    default
Priority:     0
Node:         k8s-node02/172.16.1.112
Start Time:   Fri, 12 Jun 2020 22:49:11 +0800
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":
{},"name":"pod-emptydir","namespace":"default"},"spec":{"containers":[{"image":"...
Status:       Running
IP:           10.244.2.166
IPs:
  IP:  10.244.2.166
Containers:
  myapp-pod:
    Container ID:
docker://d45663776b40a24e7cfc3cf46cb08cf3ed6b98b023a5d2cb5f42bee2234c7338
    Image:          registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
    Image ID:       docker-pullable://10.0.0.110:5000/k8s-
secret/myapp@sha256:9eeca44ba2d410e54fccc54cbe9c021802aa8b9836a0bcf3d3229354e4c8870e
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Fri, 12 Jun 2020 22:49:12 +0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /cache from cache-volume (rw)  ##### 挂载信息
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-v48g4 (ro)
  busybox-pod:
    Container ID:
docker://c2917ba30c3322fb0caead5d97476b341e691f9fb1990091264364b8cd340512
```

```
    Image:          registry.cn-beijing.aliyuncs.com/google_registry/busybox:1.24
    Image ID:       docker-pullable://registry.cn-
beijing.aliyuncs.com/ducafe/busybox@sha256:f73ae051fae52945d92ee20d62c315306c593c59a429
ccbbdcba4a488ee12269
    Port:           <none>
    Host Port:      <none>
    Command:
      /bin/sh
      -c
      sleep 3600
    State:          Running
      Started:      Fri, 12 Jun 2020 22:49:12 +0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /test/cache from cache-volume (rw)  ##### 挂载信息
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-v48g4 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  cache-volume:
    Type:       EmptyDir (a temporary directory that shares a pod's lifetime)
    Medium:
    SizeLimit:  <unset>
  default-token-v48g4:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-v48g4
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age   From                Message
  ----    ------     ----  ----                -------
  Normal  Scheduled  3s    default-scheduler   Successfully assigned default/pod-
emptydir to k8s-node02
  Normal  Pulled     2s    kubelet, k8s-node02  Container image "registry.cn-
beijing.aliyuncs.com/google_registry/myapp:v1" already present on machine
  Normal  Created    2s    kubelet, k8s-node02  Created container myapp-pod
  Normal  Started    2s    kubelet, k8s-node02  Started container myapp-pod
  Normal  Pulled     2s    kubelet, k8s-node02  Container image "registry.cn-
beijing.aliyuncs.com/google_registry/busybox:1.24" already present on machine
  Normal  Created    2s    kubelet, k8s-node02  Created container busybox-pod
```

```
  Normal  Started    2s    kubelet, k8s-node02  Started container busybox-pod
```

emptyDir验证

在pod中的myapp-pod容器内操作

```
[root@k8s-master emptydir]# kubectl exec -it pod-emptydir -c myapp-pod -- sh
/ # cd /cache
/cache #
/cache # pwd
/cache
/cache #
/cache # date >> data.info
/cache # ls -l
total 4
-rw-r--r--    1 root     root            29 Jun 12 14:53 data.info
/cache # cat data.info
Fri Jun 12 14:53:27 UTC 2020
```

在pod中的busybox-pod容器内操作

```
[root@k8s-master emptydir]# kubectl exec -it pod-emptydir -c busybox-pod -- sh
/ # cd /test/cache
/test/cache # ls -l
total 4
-rw-r--r--    1 root     root            29 Jun 12 14:53 data.info
/test/cache # cat data.info
Fri Jun 12 14:53:27 UTC 2020
/test/cache #
/test/cache # echo "===" >> data.info
/test/cache # date >> data.info
/test/cache # cat data.info
Fri Jun 12 14:53:27 UTC 2020
===
Fri Jun 12 14:56:05 UTC 2020
```

由上可见，一个Pod中多个容器可共享同一个emptyDir卷。

HostPath挂载宿主机路径

hostPath允许挂载Node上的文件系统到Pod里面去。如果Pod需要使用Node上的文件，可以使用hostPath。

示例

hostPath 的一些用法有

- 运行一个需要访问 Docker 引擎内部机制的容器；请使用 hostPath 挂载 /var/lib/docker 路径。
- 在容器中运行 cAdvisor 时，以 hostPath 方式挂载 /sys。
- 允许 Pod 指定给定的 hostPath 在运行 Pod 之前是否应该存在，是否应该创建以及应该以什么方式存在。

支持类型

除了必需的 path 属性之外，用户可以选择性地为 hostPath 卷指定 type。支持的 type 值如下：

取值	行为
	空字符串（默认）用于向后兼容，这意味着在安装 hostPath 卷之前不会执行任何检查
DirectoryOrCreate	如果指定的路径不存在，那么将根据需要创建空目录，权限设置为 0755，具有与 Kubelet 相同的组和所有权
Directory	给定的路径必须存在
FileOrCreate	如果给定路径的文件不存在，那么将在那里根据需要创建空文件，权限设置为 0644，具有与 Kubelet 相同的组和所有权【前提：文件所在目录必须存在；目录不存在则不能创建文件】
File	给定路径上的文件必须存在
Socket	在给定路径上必须存在的 UNIX 套接字
CharDevice	在给定路径上必须存在的字符设备
BlockDevice	在给定路径上必须存在的块设备

```
[root@k8s-master hostpath]# pwd
/root/k8s_practice/hostpath
[root@k8s-master hostpath]# cat pod_hostpath.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-hostpath
  namespace: default
spec:
  containers:
  - name: myapp-pod
    image: registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
    imagePullPolicy: IfNotPresent
    volumeMounts:
    - name: hostpath-dir-volume
      mountPath: /test-k8s/hostpath-dir
    - name: hostpath-file-volume
      mountPath: /test/hostpath-file/test.conf
  volumes:
```

```
  - name: hostpath-dir-volume
    hostPath:
      # 宿主机目录
      path: /k8s/hostpath-dir
      # hostPath 卷指定 type，如果目录不存在则创建(可创建多层目录)
      type: DirectoryOrCreate
  - name: hostpath-file-volume
    hostPath:
      path: /k8s2/hostpath-file/test.conf
      # 如果文件不存在则创建。 前提：文件所在目录必须存在  目录不存在则不能创建文件
      type: FileOrCreate
```

启动pod，并查看状态

```
[root@k8s-master hostpath]# kubectl apply -f pod_hostpath.yaml
pod/pod-hostpath created
[root@k8s-master hostpath]#
[root@k8s-master hostpath]# kubectl get pod -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP             NODE         NOMINATED
NODE   READINESS GATES
pod-hostpath   1/1     Running   0          17s   10.244.4.133   k8s-node01   <none>
        <none>
[root@k8s-master hostpath]#
[root@k8s-master hostpath]# kubectl describe pod pod-hostpath
Name:         pod-hostpath
Namespace:    default
Priority:     0
Node:         k8s-node01/172.16.1.111
Start Time:   Sat, 13 Jun 2020 16:12:15 +0800
Labels:       <none>
Annotations:  kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"v1","kind":"Pod","metadata":{"annotations":
{},"name":"pod-hostpath","namespace":"default"},"spec":{"containers":[{"image":"...
Status:       Running
IP:           10.244.4.133
IPs:
  IP:  10.244.4.133
Containers:
  myapp-pod:
    Container ID:
docker://8cc87217fb483288067fb6d227c46aa890d02f75cae85c6d110646839435ab96
    Image:          registry.cn-beijing.aliyuncs.com/google_registry/myapp:v1
    Image ID:       docker-pullable://registry.cn-
beijing.aliyuncs.com/google_registry/myapp@sha256:9eeca44ba2d410e54fccc54cbe9c021802aa8
b9836a0bcf3d3229354e4c8870e
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Sat, 13 Jun 2020 16:12:17 +0800
```

```
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /test-k8s/hostpath-dir from hostpath-dir-volume (rw)
      /test/hostpath-file/test.conf from hostpath-file-volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-v48g4 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  hostpath-dir-volume:
    Type:          HostPath (bare host directory volume)
    Path:          /k8s/hostpath-dir
    HostPathType:  DirectoryOrCreate
  hostpath-file-volume:
    Type:          HostPath (bare host directory volume)
    Path:          /k8s2/hostpath-file/test.conf
    HostPathType:  FileOrCreate
  default-token-v48g4:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-v48g4
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
  Type    Reason     Age        From                 Message
  ----    ------     ----       ----                 -------
  Normal  Scheduled  <unknown>  default-scheduler    Successfully assigned default/pod-
hostpath to k8s-node01
  Normal  Pulled     12m        kubelet, k8s-node01  Container image "registry.cn-
beijing.aliyuncs.com/google_registry/myapp:v1" already present on machine
  Normal  Created    12m        kubelet, k8s-node01  Created container myapp-pod
  Normal  Started    12m        kubelet, k8s-node01  Started container myapp-pod
```

宿主机操作

根据pod，在k8s-node01节点宿主机操作【因为Pod分配到了该节点】

```
# 对挂载的目录操作
[root@k8s-node01 hostpath-dir]# pwd
/k8s/hostpath-dir
[root@k8s-node01 hostpath-dir]# echo "dir" >> info
[root@k8s-node01 hostpath-dir]# date >> info
```

```
[root@k8s-node01 hostpath-dir]# cat info
dir
Sat Jun 13 16:22:37 CST 2020
# 对挂载的文件操作
[root@k8s-node01 hostpath-file]# pwd
/k8s2/hostpath-file
[root@k8s-node01 hostpath-file]# echo "file" >> test.conf
[root@k8s-node01 hostpath-file]# date >> test.conf
[root@k8s-node01 hostpath-file]#
[root@k8s-node01 hostpath-file]# cat test.conf
file
Sat Jun 13 16:23:05 CST 2020
```

在Pod 容器中操作

```
# 进入pod 中的指定容器【如果只有一个容器，那么可以不指定容器】
[root@k8s-master hostpath]# kubectl exec -it pod-hostpath -c myapp-pod -- /bin/sh
##### 对挂载的目录操作
/ # cd /test-k8s/hostpath-dir
/test-k8s/hostpath-dir # ls -l
total 4
-rw-r--r--    1 root     root            33 Jun 13 08:22 info
/test-k8s/hostpath-dir # cat info
dir
Sat Jun 13 16:22:37 CST 2020
/test-k8s/hostpath-dir #
/test-k8s/hostpath-dir # date >> info
/test-k8s/hostpath-dir # cat info
dir
Sat Jun 13 16:22:37 CST 2020
Sat Jun 13 08:26:10 UTC 2020
##### 对挂载的文件操作
# cd /test/hostpath-file/
/test/hostpath-file # cat test.conf
file
Sat Jun 13 16:23:05 CST 2020
/test/hostpath-file # echo "file====" >> test.conf
/test/hostpath-file # cat test.conf
file
Sat Jun 13 16:23:05 CST 2020
file====
```

挂载NFS至容器

NFS 是Network File System的缩写，即网络文件系统。Kubernetes中通过简单地配置就可以挂载NFS到Pod中，而NFS中的数据是可以永久保存的，同时NFS支持同时写操作。Pod被删除时，Volume被卸载，内容被保留。这就意味着NFS能够允许我们提前对数据进行处理，而且这些数据可以在Pod之间相互传递。

所有节点

```
yum install nfs-utils -y
```

安装nfs服务, node01

```
systemctl start nfs
sudo systemctl enable nfs

mkdir -p /data/nfs

vim /etc/exports
/data/nfs 10.4.0.0/24(rw,sync,no_subtree_check,no_root_squash)

exportfs -r

systemctl reload nfs-server
```

master01

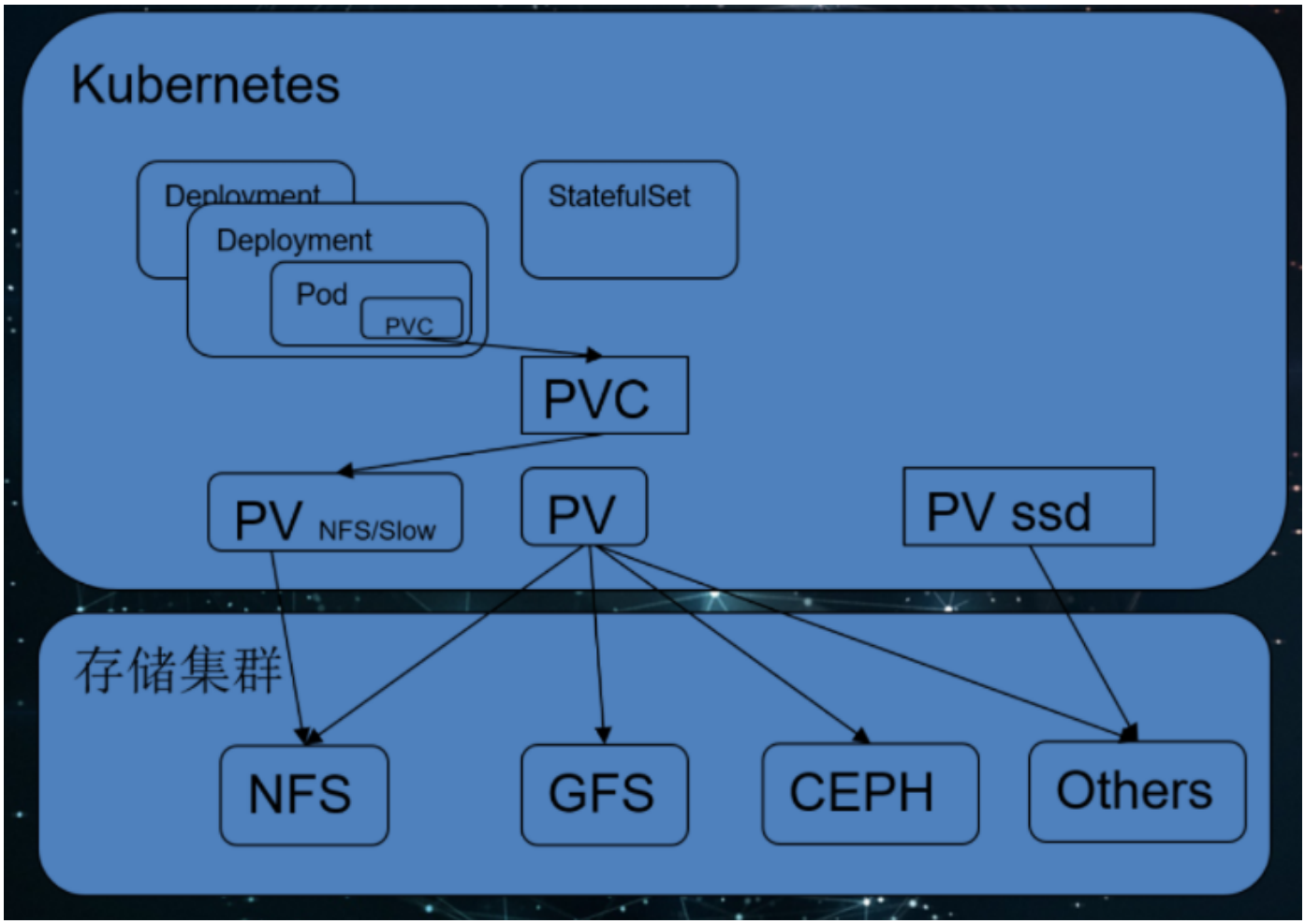
```
mount -t nfs 192.168.0.204:/data/nfs /mnt/
cd /mnt/
touch 123

unmount /mnt
```

```
# volumes挂载
# 案例https://www.cnblogs.com/xiajq/p/11395211.html
# 生产环境不用nfs，也不直接用NFS，用PV来连接
# 可以用云服务商的来存储
```

PV&PVC

PV、PVC图解



一些概念：

```
Volume：NFS、CEPH、GFS
PersistentVolume：NFS、CEPG/GFS

PV、PVC
PV：由k8s配置的存储，PV同样是集群的一类资源，可以用yaml定义。

除了Volume之外，kubernetes还提供了Persistent Volume的方法。Volume主要是为了存储一些有必要保存的数据,而Persistent Volume主要是为了管理集群的存储。

PVC：对PV的申请，
PersistentVolumeClaim



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

Nfs类型的PV：
apiVersion: v1
```

```
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2

persistentVolumeReclaimPolicy：#类型
  Recycle：回收，rm -rf
    Deployment -> PVC  PV，Recycle。
  Retain：保留。
  Delete：PVC --> PV,删除PVC后PV也会被删掉，这一类的PV，需要支持删除的功能，动态存储默认方式。

Capacity：PV的容量。
volumeMode：挂载的类型，Filesystem，block

accessModes：这个的PV访问模式：
  ReadWriteOnce：RWO，可以被单节点以读写的模式挂载。
  ReadWriteMany：RWX，可以被多节点以读写的形式挂载。
  ReadOnlyMany：ROX，可以被多个节点以只读的形式挂载。
storageClassName：PV的类，可以说是一个类名，PVC和PV的这个名字一样，才能被绑定。

Pv的状态：
  Available：空闲的PV，没有被任何PVC绑定。
  Bound：已经被PVC绑定
  Released：PVC被删除，但是资源未被重新使用
  Failed：自动回收失败。
```

创建一个NFS的PV

```
[root@k8s-master01 app]# cat nfs-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
```

```
    storage: 5Gi    # 定义内存大小，PVC的要比这个小
  volumeMode: Filesystem  # 文件系统类型
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle  # 策略上面有介绍
  storageClassName: nfs-slow   # 这个名字是PVC创建的时候要对应的名字
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:   #对应上nfs服务器的 ip 共享的文件夹
    path: /data/nfs
    server: 192.168.1.104

# CREATE pv
[root@k8s-master01 app]# kubectl create -f nfs-pv.yaml
persistentvolume/pv0001 created

#  查看PV
[root@k8s-master01 app]# kubectl get pv
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS
REASON   AGE
pv0001   5Gi        RWO            Recycle          Available           nfs-slow
         2m3s
```

创建一个PVC

```
# 绑定到指定类型的PV
[root@k8s-master01 app]# cat test-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: myclaim   # PVC的名字，可自取
spec:
  accessModes:
    - ReadWriteMany
  volumeMode: Filesystem
  resources:
    requests:
      storage: 2Gi
  storageClassName: nfs-slow  # 名字要对应上想绑定的PV上

# create PVC
[root@k8s-master01 app]# kubectl create -f  test-pvc.yaml
persistentvolumeclaim/myclaim created

# 查看PVC
[root@k8s-master01 app]# kubectl get pvc
NAME      STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
```

```
myclaim    Bound     pv0001    5Gi        RWX            nfs-slow       34s

# 再次查看PV，可看到状态已经发生改变
[root@k8s-master01 app]# kubectl get pv
NAME      CAPACITY    ACCESS MODES    RECLAIM POLICY    STATUS    CLAIM
STORAGECLASS    REASON    AGE
pv0001    5Gi         RWX             Recycle           Bound     default/myclaim    nfs-slow
                13m
```

更改deployment使用PVC类型的volume

```
# 更改后的yaml，在对应位置加上以下参数调用PVC
volumeMounts:
- mountPath: /opt/pvc
  name: mypd

volumes:
   - name: mypd
     persistentVolumeClaim:
       claimName: myclaim

# 然后进入容器查看是否挂载成功
[root@k8s-master01 app]# kubectl exec -it nginx-5bb6d88dfb-w78k8 -c nginx2 -- bash
root@nginx-5bb6d88dfb-w78k8:/# df -h
Filesystem               Size  Used Avail Use% Mounted on
overlay                   37G  4.3G   33G  12% /
tmpfs                     64M     0   64M   0% /dev
tmpfs                    985M     0  985M   0% /sys/fs/cgroup
/dev/mapper/centos-root   37G  4.3G   33G  12% /mnt
192.168.1.104:/data/nfs   37G  3.0G   35G   9% /opt/pvc    # 这就是刚刚挂载的PVC

# 文件能共享
root@nginx-5bb6d88dfb-w78k8:/opt/pvc# ls
pvc  qqq  test
root@nginx-5bb6d88dfb-w78k8:/opt/pvc# echo 11 > test
root@nginx-5bb6d88dfb-w78k8:/opt/pvc# cat test
11
```

很多情况下：

创建PVC之后，一直绑定不上PV（Pending）：

1. PVC的空间申请大小大于PV的大小
2. PVC的StorageClassName没有和PV的一致
3. PVC的accessModes和PV的不一致

创建挂载了PVC的Pod之后，一直处于Pending状态：

1. PVC没有被创建成功，或者被创建
2. PVC和Pod不在同一个Namespace

删除PVC后□k8s会创建一个用于回收的Pod□根据PV的回收策略进行pv的回收□回收完以后PV的状态就会变成可被绑定的状态也就是空闲状态□其他的Pending状态的PVC如果匹配到了这个PV，他就能和这个PV进行绑定。