Kubernetes 持久化存储

前言：

课程名称：Kubernetes 持久化存储

实验环境：

本章节 Kubernetes 集群环境如下：

角色	IP	主机名	组件	硬件
控制节点	192.168.128.11	k8s-master01	apiserver controller-manager scheduler etcd containerd	CPU：4vCPU 硬盘：100G 内存：4GB 开启虚拟化
工作节点	192.168.128.21	k8s-node01	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化
工作节点	192.168.128.22	k8s-node02	kubelet kube-proxy containerd calico coredns	CPU：6vCPU 硬盘：100G 内存：8GB 开启虚拟化

官方手册：

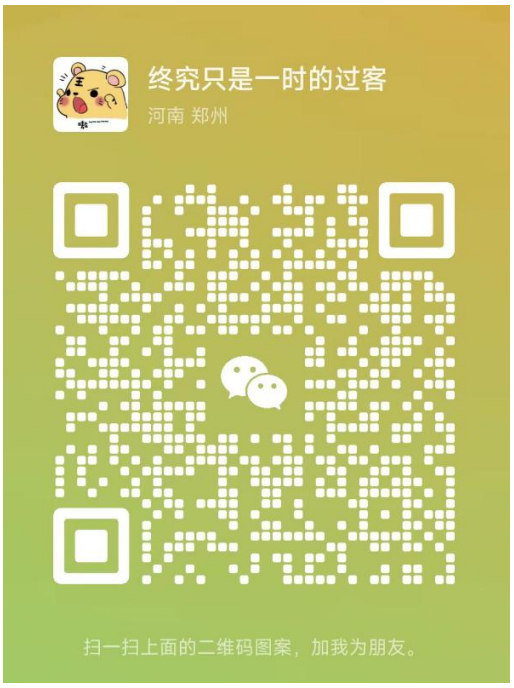
https://kubernetes.io/zh-cn/docs/concepts/storage/

张岩峰老师微信，加我微信，邀请你加入 VIP 交流答疑群：

微信号：ZhangYanFeng0429

二维码：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。



1、在 k8s 中为什么要做持久化存储？

在 k8s 中部署的应用都是以 pod 容器的形式运行的，假如我们部署 MySQL、Redis 等数据库，需要对这些数据库产生的数据做备份。因为 Pod 是有生命周期的，如果 pod 不挂载数据卷，那 pod 被删除或重启后这些数据会随之消失，如果想要长久的保留这些数据就要用到 pod 数据持久化存储。

查看 k8s 支持哪些存储，可以使用如下命令进行查询：

```
[root@k8s-master01 ~]# kubectl explain pods.spec.volumes
KIND:     Pod
VERSION:  v1
RESOURCE: volumes <[]Object>
DESCRIPTION:
     List of volumes that can be mounted by containers belonging to the pod.
     More info: https://kubernetes.io/docs/concepts/storage/volumes
     Volume represents a named volume in a pod that may be accessed by any
     container in the pod.
FIELDS:
   awsElasticBlockStore <Object>（已弃用）
   azureDisk    <Object>（已弃用）
   azureFile    <Object>（已弃用）
   cephfs    <Object>
   cinder    <Object>（已弃用）
   configMap    <Object>
```

```
      csi   <Object>
      downwardAPI   <Object>
      emptyDir <Object>
      ephemeral     <Object>
      fc    <Object>（已弃用）
      flexVolume    <Object>（已弃用）
      flocker   <Object>
      gcePersistentDisk     <Object>（已弃用）
      gitRepo   <Object>（已弃用）
      glusterfs     <Object>（已弃用）
      hostPath <Object>
      iscsi     <Object>
      name <string> -required-
      nfs   <Object>
      persistentVolumeClaim     <Object>
      photonPersistentDisk <Object>
      portworxVolume    <Object>（已弃用）
      projected     <Object>（已弃用）
      quobyte   <Object>
      rbd   <Object>
      scaleIO   <Object>
      secret    <Object>
      storageos     <Object>
      vsphereVolume     <Object>（已弃用）
```

Kubernetes（k8s）支持多种不同类型的存储，包括：

1、空白存储（EmptyDir）：一个临时目录，只在 Pod 的生命周期中存在。

2、主机路径（HostPath）：在宿主机上创建并挂载一个目录，作为 Pod 中的存储卷。

3、基于网络的存储（Network-based storage）：使用网络存储，如 NFS、iSCSI 等。

4、持久化卷（Persistent Volume）：由管理员分配并预设置的存储卷。它们可以在多个 Pod 和节点之间动态地共享和重新分配。

5、存储类（StorageClass）：对于动态分配的持久化卷，允许管理员创建不同的存储类，以满足应用程序不同的存储要求。存储类会根据现有的存储池来创建新的存储。

6、对象存储（Object Storage）：例如 Amazon S3，使用外部存储来存储数据。

7、本地存储（Local Storage）：使用主机的本地存储，适合于需要高效 I/O 的应用程序。

总之，Kubernetes 支持多种存储类型，根据不同的应用场景进行选择和配置。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

2、k8s 持久化存储：emptyDir

EmptyDir 是一种 Kubernetes 中的持久性存储卷，它可以在 Pod 的生命周期内持久保存数据。

当容器使用 EmptyDir 卷时，Kubernetes 会在节点上为其分配一个空目录。在 Pod 的生命周期内，这个目录将一直存在，可以在容器之间共享。如果容器重启或迁移，数据也将保持不变。

需要注意的是，EmptyDir 的数据仅在单个节点上持久，如果该节点发生故障或 Pod 迁移到其他节点，则数据将丢失。

- 实战

目标：创建一个 Pod 挂载临时目录，测试删除 Pod，临时目录的数据是否会丢失。

（1）创建资源清单文件，使用 emptyDir 挂载临时目录

```
[root@k8s-master01 ~]# vi emptydir.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-empty
spec:
  containers:
  - name: container-empty
    image: nginx
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
```

对上面的 yaml 文件说明

这个配置声明了一个名为 pod-empty 的 Pod，在 Pod 中有一个名为 container-empty 的容器，容器会挂载一个名为 cache-volume 的 EmptyDir 卷，并将其挂载到容器的/cache 目录下。

spec.volumes 中的 emptyDir 对象声明了一个空的 EmptyDir 存储卷。此时，Kubernetes 将在当前节点上为此 Pod 创建一个新的临时目录，并将其与该卷绑定。

spec.containers.volumeMounts 中声明了如何将此 EmptyDir 挂载到容器中。在这个示例中，容器的/cache 目录将映射到该卷上，因此容器中的应用程序可以在此目录中创建和访问文件。

需要注意的是，EmptyDir 存储卷的生命周期与 Pod 的生命周期相同，如果 Pod 被删除或终止，则其中包含的所有数据也将被删除。因此，EmptyDir 更适合于临时存储或缓存，

```
而不是长期数据持久化。
```

（2）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f emptydir.yaml
pod/pod-empty created
```

（3）查看 Pod

```
[root@k8s-master01 ~]# kubectl get pods -o wide
```

```
[root@k8s-master01 ~]# kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP              NODE         NOMINATED NODE   READINESS GATES
pod-empty   1/1     Running   0          38s   10.244.85.197   k8s-node01   <none>           <none>
[root@k8s-master01 ~]#
```

可以看到 pod-empty 调度到了 node01 节点

（4）查看 node01 节点临时目录存在的位置，可用如下方法

```
# 查看 pod 的 uid
[root@k8s-master01 ~]# kubectl get pods pod-empty -o yaml | grep uid
  uid: 9a9ef367-9a7b-47b0-bc24-bfcfc39890d8

# 在 k8s-node01 节点查看
[root@k8s-node01 ~]# tree /var/lib/kubelet/pods/9a9ef367-9a7b-47b0-bc24-bfcfc39890d8/
/var/lib/kubelet/pods/9a9ef367-9a7b-47b0-bc24-bfcfc39890d8/
|-- containers
|   `-- container-empty
|       `-- 9b7522e7
|-- etc-hosts
|-- plugins
|   `-- kubernetes.io~empty-dir
|       |-- cache-volume
|       |   `-- ready
|       `-- wrapped_kube-api-access-v4gbg
|           `-- ready
`-- volumes
    |-- kubernetes.io~empty-dir
    |   `-- cache-volume
    `-- kubernetes.io~projected
        `-- kube-api-access-v4gbg
            |-- ca.crt -> ..data/ca.crt
            |-- namespace -> ..data/namespace
            `-- token -> ..data/token

11 directories, 7 files
```

由此可知，临时目录在本地的：

/var/lib/kubelet/pods/9a9ef367-9a7b-47b0-bc24-bfcfc39890d8/volumes/kubernetes.io~empty-dir/cache-volume/

（5）测试

```
1、在临时目录下创建文件
    [root@k8s-node01                                ~]#                                cd /var/lib/kubelet/pods/9a9ef367-9a7b-47b0-bc24-bfcfc39890d8/volumes/kubernetes.io~empty-dir/cache-volume/
    [root@k8s-node01 cache-volume]# echo "I am zyf" > name.txt
    [root@k8s-node01 cache-volume]# ls
    name.txt
    [root@k8s-node01 cache-volume]# cat name.txt
    I am zyf

2、进入 pod-empty 容器内查看文件是否存在
    [root@k8s-master01 ~]# kubectl exec -it pod-empty -- bash
    root@pod-empty:/# cat /cache/name.txt
    I am zyf

3、删除容器检查数据是否还在
    [root@k8s-master01 ~]# kubectl delete -f emptydir.yaml
    pod "pod-empty" deleted

    [root@k8s-node01                                ~]#                                cd /var/lib/kubelet/pods/9a9ef367-9a7b-47b0-bc24-bfcfc39890d8
    -bash: cd: /var/lib/kubelet/pods/9a9ef367-9a7b-47b0-bc24-bfcfc39890d8: No such file or directory
    # 可以看到以容器 ID 命名的目录以及被删除了。
```

3、k8s 持久化存储：hostPath

- hostPath 存储卷介绍

hostPath 是 Kubernetes 中一种简单的持久性存储卷类型，它可将节点的文件系统中的文件或目录直接挂载到 Pod 中。

该卷类型通过在 Pod 中指定主机路径和容器中所需挂载该路径的位置来工作。它适用于需要访问节点中文件系统上的目录或文件的应用程序。

需要注意的是，使用 hostPath 存储卷需要非常小心，因为它会直接暴露节点的文件系统。另外，当 Pod 迁移到其他节点时，这个目录映射不会跟随迁移而改变，因此可能会影响到应用程序的可移植性。

- hostPath 语法说明

hostPath 语法可以通过如下命令进行查询：

```
[root@k8s-master01 ~]# kubectl explain pods.spec.volumes.hostPath
KIND:     Pod
VERSION:  v1

RESOURCE: hostPath <Object>

DESCRIPTION:
     hostPath represents a pre-existing file or directory on the host machine
     that is directly exposed to the container. This is generally used for
     system agents or other privileged things that are allowed to see the host
     machine. Most containers will NOT need this. More info:
     https://kubernetes.io/docs/concepts/storage/volumes#hostpath

     Represents a host path mapped into a pod. Host path volumes do not support

FIELDS:
   path <string> -required-
   type <string>
```

参数解释如下表：

属性名称	取值类型	是否必选	取值说明
path	string	required	主机上目录的路径。如果路径是符号链接，它将链接到真实路径
type	string		主机默认卷的类型。可选值： ● DirectoryOrCreate：如果指定的目录不存在，就自动创建一个空目录，权限设置为 0755，与 kubelet 具有相同的组和所有权。 ● Directory：表示将使用现有主机上的目录。给定的目录必须存在。 ● FileOrCreate：如果在主机上指定的文件不存在，则创建一个该空文件，权限设置为 0644，与 kubelet 具有相同的组和所有权。否则将使用现有文件。 ● File：表示将使用现有主机上的文件。给定的文件必须存在。 ● Socket：表示使用主机上的 Unix 套接字文件。必须存在。 ● CharDevice：表示使用主机上的字符设备文件。必须存在。 ● BlockDevice：表示使用主机上的块设备文件。必须存在。

● 实战

目标：创建一个 Pod 挂载工作节点的/data1 目录，如果目录不存在就创建。测试删除 Pod，目录的数据是否会丢失。

（1）创建资源清单文件

```
[root@k8s-master01 ~]# vi hostpath.yaml
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
apiVersion: v1
kind: Pod
metadata:
  name: test-hostpath
spec:
  containers:
  - name: test-nginx
    image: nginx
    volumeMounts:
    - name: test-volume
      mountPath: /test-nginx
  - name: test-tomcat
    image: tomcat:8.5-jre8-alpine
    volumeMounts:
    - name: test-volume
      mountPath: /test-tomcat
  volumes:
  - name: test-volume
    hostPath:
      path: /data1
      type: DirectoryOrCreate
```

对上面的 yaml 文件说明：

这个 Kubernetes Pod 的 YAML 配置，演示了如何在 Pod 中使用 hostPath 持久化存储卷，并将其挂载到多个容器中。

在这个示例中，spec.volumes 中声明了一个名为 test-volume 的 hostPath 存储卷，它使用的路径为/data1，并且指定了类型为 DirectoryOrCreate。这意味着如果在主机上指定的目录不存在，则会在主机上创建该目录。否则，Kubernetes 将使用现有目录。

然后，在 spec.containers 中声明了两个容器，并在每个容器中都挂载了 test-volume 存储卷。test-nginx 容器将卷挂载到/test-nginx 目录下，而 test-tomcat 容器则将卷挂载到/test-tomcat 目录下。

需要注意的是，在使用 hostPath 持久化存储卷时，需要小心不要不小心地将敏感数据暴露到 Pod 中，因为它直接暴露了节点的文件系统。此外，当 Pod 迁移到其他节点时，这个目录映射不会跟随迁移而改变，因此可能会影响到应用程序的可移植性。

（2）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f hostpath.yaml
pod/test-hostpath created
```

（3）查看 pod 调度到了哪个物理节点

```
[root@k8s-master01 ~]# kubectl get pods -o wide
```

```
[root@k8s-master01 ~]# kubectl get pods -o wide
NAME           READY   STATUS    RESTARTS   AGE     IP              NODE        NOMINATED NODE   READINESS GATES
test-hostpath   2/2     Running   0          7m41s   10.244.58.199   k8s-node02   <none>           <none>
[root@k8s-master01 ~]#
```

（4）创建临时数据，测试容器内是否同步

1、上面可以看到 pod 调度到了 k8s-node02 上，登录到 k8s-node02 机器，查看是否在这台机器创建了存储目录。

```
[root@k8s-node02 ~]# ll -d /data1/
drwxr-xr-x 2 root root 22 Jun  2 21:45 /data1/
[root@k8s-node02 ~]# cd /data1/
[root@k8s-node02 data1]# echo "I am zyf" > name.txt
[root@k8s-node02 data1]# cat name.txt
I am zyf
```

2、登录到 test-hostpath pod 下的 test-nginx 容器，检查创建的文件是否存在

```
[root@k8s-master01 ~]# kubectl exec -it test-hostpath -c test-nginx -- /bin/bash
root@test-hostpath:/# cd /test-nginx/
root@test-hostpath:/test-nginx# ls
name.txt
root@test-hostpath:/test-nginx# cat name.txt
I am zyf
```

3、登录到 test-hostpath pod 下的 test-tomcat 容器，检查创建的文件是否存在

```
[root@k8s-master01 ~]# kubectl exec -it test-hostpath -c test-tomcat -- /bin/bash
bash-4.4# cd /test-tomcat/
bash-4.4# ls
name.txt
bash-4.4# cat name.txt
I am zyf
```

通过上面测试可以看到，同一个 pod 里的 test-nginx 和 test-tomcat 这两个容器是共享存储卷的。

hostpath 存储卷缺点：

单节点，pod 删除之后重新创建必须调度到同一个 node 节点，数据才不会丢失。

（5）删除 Pod，检查数据是否还在

1、删除 Pod

```
[root@k8s-master01 ~]# kubectl delete -f hostpath.yaml
pod "test-hostpath" deleted
```

```
2、检查数据是否还在
    [root@k8s-node02 ~]# ll -d /data1/
    drwxr-xr-x 2 root root 22 Jun  2 21:45 /data1/
    [root@k8s-node02 ~]# ls /data1/ -l
    total 4
    -rw-r--r-- 1 root root 9 Jun  2 21:45 name.txt
    [root@k8s-node02 ~]# cat /data1/name.txt
    I am zyf
```

（6）建议

若生产使用 hostPath 挂载方式，必须指定 nodeName，否则 pod 删除掉之后，下次再创建 pod，不一定能百分百调度到之前的 node 节点上。

```
    [root@k8s-master01 ~]# vi hostpath.yaml
    apiVersion: v1
    kind: Pod
    metadata:
      name: test-hostpath
    spec:
      nodeName: k8s-node02
      containers:
      - name: test-nginx
        image: nginx
        volumeMounts:
        - name: test-volume
          mountPath: /test-nginx
      - name: test-tomcat
        image: tomcat:8.5-jre8-alpine
        volumeMounts:
        - name: test-volume
          mountPath: /test-tomcat
      volumes:
      - name: test-volume
        hostPath:
          path: /data1
          type: DirectoryOrCreate
```

也可以采用网络文件系统或分布式文件存储系统：nfs、cephfs、glusterfs 等。

4、k8s 持久化存储：nfs

- nfs 存储卷介绍

上小节说的 hostPath 存储，存在单点故障，pod 挂载 hostPath 时，只有调度到同一个节点，数据才不会丢失。那可以使用 nfs 作为持久化存储可以避免这些问题。

NFS（Network File System）是一种分布式文件系统协议，允许计算机之间共享文件和目录。在 Kubernetes 之中，可以使用 NFS 作为一种持久性存储卷类型，使得多个 Pod 之间可以共享相同的数据。

在使用 NFS 存储卷时，需要先安装 NFS 服务器，并创建一个共享目录。在创建 Pod 时，可以直接将 NFS 共享的目录挂载到容器内。

其实相对而言，nfs 也是不够安全的，也是存在单点故障，我们可以采用对接分布式文件存储来实现存储高可用。这些我们后面会讲到。

- nfs 语法说明

nfs 语法可以通过如下命令进行查询：

```
[root@k8s-master01 ~]# kubectl explain pods.spec.volumes.nfs
KIND:     Pod
VERSION:  v1

RESOURCE: nfs <Object>

DESCRIPTION:
     nfs represents an NFS mount on the host that shares a pod's lifetime More
     info: https://kubernetes.io/docs/concepts/storage/volumes#nfs

     Represents an NFS mount that lasts the lifetime of a pod. NFS volumes do
     not support ownership management or SELinux relabeling.

FIELDS:
   path <string> -required-
   readOnly <boolean>
   server   <string> -required-
```

参数解释如下表：

属性名称	取值类型	是否必选	取值说明
path	string	required	NFS 服务器共享的目录。
readOnly	boolean		此处 readOnly 如果设置为 true 时，将强制 NFS 以只读权限挂载。默认为 false。
server	string	required	NFS 服务器的主机名或 IP 地址

- 实战

目标：使用 nfs 共享/data/volumes 目录并挂载到 Pod 内。测试删除 Pod，目录的数据是否会丢失。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

（1）搭建 nfs 服务端

以 k8s 的控制节点作为 NFS 服务端（所有 k8s 工作节点都需要安装这个包，否则 pod 无法挂载 nfs）

```
1、安装 nfs
    [root@k8s-master01 ~]# yum install nfs-utils -y

2、创建 NFS 需要的共享目录
    [root@k8s-master01 ~]# mkdir /data/volumes -p

3、配置 nfs 共享服务器，共享/data/volumes 目录
    [root@k8s-master01 ~]# vi /etc/exports
    /data/volumes 192.168.128.0/24(rw,no_root_squash)
    # no_root_squash: 用户具有根目录的完全管理访问权限

4、启动 nfs 服务
    [root@k8s-master01 ~]# systemctl enable nfs-server.service --now
```

（2）在 k8s-node01 节点上手动挂载 nfs 共享目录测试：

```
    [root@k8s-node01 ~]# mkdir /test
    [root@k8s-node01 ~]# mount 192.168.128.11:/data/volumes /test
    [root@k8s-node01 ~]# df -Th | grep test
    192.168.128.11:/data/volumes nfs4       99G  4.9G   95G   5% /test
    [root@k8s-node01 ~]# umount /test/
```

（3）创建资源清单文件

```
    [root@k8s-master01 ~]# vi nfs.yaml
    apiVersion: v1
    kind: Pod
    metadata:
     name: test-nfs-volume
    spec:
     containers:
     - name: test-nfs
       image: nginx
       imagePullPolicy: IfNotPresent
       ports:
       - containerPort: 80
         protocol: TCP
       volumeMounts:
       - name: nfs-volumes
         mountPath: /usr/share/nginx/html
     volumes:
```

```
    - name: nfs-volumes
      nfs:
       path: /data/volumes
       server: 192.168.128.11
```

对上面的 yaml 文件说明：

这是一个使用 NFS 持久化存储卷的 Kubernetes Pod 配置文件示例。具体来说，这个 Pod 会在容器中使用 nginx 镜像作为 Web 服务器，并将 NFS 存储卷挂载到/usr/share/nginx/html 目录下。

其中，spec.containers 中定义了一个名为 test-nfs 的容器，并指定了 nginx 镜像。该容器监听了 80 端口，并将该端口暴露给其他 Pod。

spec.volumes 中定义了一个名为 nfs-volumes 的持久卷，并在其中使用 nfs 指定 NFS 服务器的地址、共享目录的路径。在这个例子中，nfs.server 指向 IP 地址为 192.168.128.11 的 NFS 服务器，而 nfs.path 则为共享目录的路径/data/volumes。

最后，在 spec.containers.volumeMounts 中将该 NFS 存储卷挂载到了容器中的/usr/share/nginx/html 目录下，以提供持久性存储。

（4）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f nfs.yaml
pod/test-nfs-volume created
```

（5）查看创建的 Pod

```
[root@k8s-master01 ~]# kubectl get pods test-nfs-volume -o wide
[root@k8s-master01 ~]# kubectl get pods test-nfs-volume -o wide
NAME              READY   STATUS    RESTARTS   AGE     IP              NODE         NOMINATED NODE   READINESS GATES
test-nfs-volume   1/1     Running   0          2m15s   10.244.85.198   k8s-node01   <none>           <none>
[root@k8s-master01 ~]#
```

（6）登录到 nfs 服务器，在共享目录创建一个 index.html

```
[root@k8s-master01 ~]# cd /data/volumes/
[root@k8s-master01 volumes]# echo "My Name is ZYF" > index.html
[root@k8s-master01 volumes]# cat index.html
My Name is ZYF
```

（7）进入 pod，查看

```
[root@k8s-master01 ~]# kubectl exec -it test-nfs-volume -- /bin/bash
root@test-nfs-volume:/# cd /usr/share/nginx/html/
root@test-nfs-volume:/usr/share/nginx/html# ls
index.html
root@test-nfs-volume:/usr/share/nginx/html# cat index.html
My Name is ZYF
```

（8）请求 Pod，测试

```
[root@k8s-master01 ~]# curl 10.244.85.198
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
My Name is ZYF
```

经过上面测试说明挂载 nfs 存储卷成功了，nfs 支持多个客户端挂载，可以创建多个 pod，挂载同一个 nfs 服务器共享出来的目录；但是 nfs 如果宕机了，数据也就丢失了，所以需要使用分布式存储，常见的分布式存储有 glusterfs 和 cephfs。

（9）清除 Pod

```
[root@k8s-master01 ~]# kubectl delete -f nfs.yaml
pod "test-nfs-volume" deleted
```

（10）查看数据是否被清除

```
[root@k8s-master01 ~]# ls /data/volumes/
index.html

[root@k8s-master01 ~]# cat /data/volumes/index.html
My Name is ZYF
```

5、k8s 持久化存储：PV 与 PVC

5.1、初识 PV 与 PVC

5.1.1、k8s PV 是什么？

在 Kubernetes 中，PV（PersistentVolume）是一种独立于 Pod 而存在，并且可以被多个 Pod 共享的存储资源。PV 的创建由管理员完成，而 Pod 中的容器则通过 PVC（PersistentVolumeClaim）来申请 PV 的使用权限。

PV 可以连接到不同种类的持久化存储后端，比如 NFS、iSCSI、AWS EBS 等，以提供 Kubernetes 中的持久性存储。在 Pod 和 PV 之间需要 PVC 来进行匹配，以保证 Pod 能够获取到需要的 PV 资源。

在 PV 中，管理员可以定义各种存储相关的参数，包括存储类别、访问模式、存储容量等等。PVC 需要提供的信息则包括需要的存储容量、访问模式等，只有满足这些条件的 PV 才能被绑定到 PVC 上面，并能够被 Pod 访问。

由于 PV 是独立于 Pod 的资源对象，在多个 Pod 之间共享数据变得非常简便。使用 PV 还可以使数据得到持久化，这样便可以在 Pod 被迁移或升级时，保留已有的数据文件。

总之，PV 提供了 Kubernetes 集群中持久性存储的完整抽象层次，并为 Pod 提供了访问 PV 的接口，从而实现了数据持久化，并且能够为多个 Pod 提供共享数据的能力。

5.1.2、k8s PVC 是什么？

在 Kubernetes 中，PVC（PersistentVolumeClaim）用来申请使用 PV

（PersistentVolume）的存储资源。PVC 是一个表示 Pod 中容器需要的持久化存储的逻辑卷，它声明了需要的存储资源及访问模式，Kubernetes 集群会为它自动匹配可用的 PV 资源。

PVC的申请过程可以理解为Pod容器与实际存储之间的“中介”机制。在Pod的生命周期中，容器会需要持久化存储来保存数据。PVC 允许应用程序声明自己需要使用一定数量的存储空间，并定义访问存储的模式。

PVC 可以跨命名空间使用，但只能匹配同一命名空间中的 PV。如果希望 PVC 可以访问某个特定的 PV 资源，那么 PVC 的规格必须与该 PV 资源的规格完全匹配。如果 PV 规格中包含的存储空间或访问模式不满足 PVC 规格中声明的需求，则 PVC 将无法匹配到该 PV。

当 Pod 容器需要使用持久化存储时，可以通过 PVC 申请使用 PV 资源。Kubernetes 集群会为 PVC 寻找一个合适的 PV 资源并将其分配给该 PVC。Pod 容器在启动时，只需要将该 PVC 挂载到预定目录即可使用 PV 提供的持久化存储服务。

总之，PVC 为 Kubernetes 内的 Pod 容器提供了管理存储卷的方式，通过声明式的方法为 Pod 引入了新的持久存储块。它将对存储资源的请求隔离开来，无需在 Pod 定义中耦合具体的存储参数细节，并优化了资源的灵活使用和管理。

5.1.3、k8s PVC 和 PV 工作原理

在 Kubernetes 中，PVC 和 PV 是协同工作的。PVC 用于向 Kubernetes 请求持久性存储资源，而 PV 则用于提供这些存储资源的实际物理实现。

以下是 PVC 和 PV 的工作原理：

1、PV 定义

PV 是 Kubernetes 集群中的可用存储资源，PV 是提供持久化存储的实际物理实现，例如 NFS、iSCSI 等。管理员需要在控制台中定义 PV。在这里，可以指定存储类型、访问模式、存储容量等信息。

2、PVC 定义

在创建 PVC 的 YAML 文件中，我们可以定义需要的存储容量和访问模式。Kubernetes 将使用 PVC 规范来查找并匹配合适的 PV。

3、PVC 和 PV 的绑定

用户创建 pvc 并指定需要的资源和访问模式。在找到可用 pv 之前，pvc 会保持未绑定状态。绑定是通过 PVC 规范中的 spec.selector 字段实现的，它用于绑定与 PVC 要求最接近的 PV。

pvc 和 pv 它们是一一对应的关系，pv 如果被 pvc 绑定了，就不能被其他 pvc 使用了。

我们在创建 pvc 的时候，应该确保和底下的 pv 能绑定，如果没有合适的 pv，那么 pvc 就会处于 pending 状态。

4、Pod 与 PVC 的挂载

随着 PVC 和 PV 的绑定，管理员将把 PV 暴露给 Kubernetes 中的 Pods。Pod 指定 PVC 名称，Kubernetes 确定哪个 PVC 绑定了哪个 PV，并将 PV 添加到 Pod 的文件系统中。Pod 然后在容器中挂载 PV，以使容器可以访问它。

5、回收策略

当我们创建 pod 时如果使用 pvc 做为存储卷，那么它会和 pv 绑定，当删除 pod，pvc 和 pv 绑定就会解除，解除之后和 pvc 绑定的 pv 卷里的数据需要怎么处理，目前，卷可以保留，回收或删除：Retain、Recycle（不推荐使用，1.15 被废弃了）、Delete。

1、Retain

当删除 pvc 的时候，pv 仍然存在，处于 released 状态，但是它不能被其他 pvc 绑定使用，里面的数据还是存在的，当我们下次再使用的时候，数据还是存在的，这个是默认的回收策略。

2、Delete

删除 pvc 时即会从 Kubernetes 中移除 PV，也会从相关的外部设施中删除存储资产。

总结来说，通过 PVC 和 PV 的协作，Kubernetes 简化了存储资源使用和管理。管理员只需要定义 PV，而开发人员在需要持久性存储时可以使用 PVC。Kubernetes 的任务是匹配 PVC 和 PV；节点和 Pod 容器都可以直接访问 PV，以存储和访问持久性数据。

5.2、PV 与 PVC 语法与应用

5.2.1、PV 语法说明与创建

在 kubernetes 集群中，PersistentVolume 资源是用于定义 PV 的。

- PersistentVolume 语法说明

PersistentVolume 语法可以通过如下命令进行查询：

```
[root@k8s-master01 ~]# kubectl explain PersistentVolume
KIND:     PersistentVolume
VERSION:  v1

DESCRIPTION:
     PersistentVolume (PV) is a storage resource provisioned by an
     administrator. It is analogous to a node. More info:
     https://kubernetes.io/docs/concepts/storage/persistent-volumes

FIELDS:
```

```
      apiVersion   <string>
      kind <string>
      metadata <Object>
      spec <Object>
      status   <Object>
```

下面主要说一下 spec 字段下的参数：

属性名称	取值类型	是否必选	取值说明
accessModes	[]string		用于指定 PV/PVC 的访问模式。该字段可选的访问模式包括： ● ReadWriteOnce（RWO）：可被单个节点以读写方式挂载。 ● ReadOnlyMany（ROX）：可被多个节点以只读方式挂载； ● ReadWriteMany（RWX）：可被多个节点以读写方式挂载。 针对不同的访问模式，PV 对应的后端存储资源必须支持相应的模式。下面是每种访问模式的更详细解释： ● ReadWriteOnce（RWO）：该模式要求 PV/PVC 只能被一个节点以读写方式挂载。这意味着，当 PV/PVC 与 Pod 绑定时，Pod 可以在一个节点上以读写模式使用该 PV/PVC，但不允许在其他节点上以相同方式使用。 ● ReadOnlyMany（ROX）：该模式要求 PV/PVC 能够被多个节点以只读方式挂载。这意味着，当 PV/PVC 与 Pod 绑定时，Pod 可以在多个节点上以只读模式使用该 PV/PVC，但不允许在任何一个节点上进行写操作。 ● ReadWriteMany（RWX）：该模式要求 PV/PVC 可以被多个节点以读写方式挂载。这意味着，当 PV/PVC 与 Pod 绑定时，Pod 可以在多个节点上以读写模式使用该 PV/PVC。
capacity	map[string] string		用于指定 PV 或 PVC 的存储容量大小。该字段必须是以字符串形式指定的数字，并附带相应的单位（如 Gi、G、Mi、M 等）。 例如，要在 PV 上定义 1GiB 的存储容量，可以设置 spec.capacity.storage 字段为 1Gi。
persistentVolumeReclaimPolicy	string		PersistentVolume.spec.persistentVolumeReclaimPolicy 是指在删除使用完的 PersistentVolume（PV）后，PV 上残留的数据应该如何处理的策略。这个策略可以在 PV 创建时指定。 可能用到的参数如下：

			● Retain：保留 PersistentVolume 删除之后的数据，不进行再利用。需要手动清理 PV 上的数据。 ● Recycle：自动清空 PersistentVolume 上面的数据，PV 又可以重复使用。可进行标准的文件系统格式化和清空操作，但不适用于多个 PV 共享的场景。 ● Delete：直接删除 PersistentVolume。PV 上的数据会被同步删除。 其中，默认的策略为 Retain，即保留删除之后的数据。在实际使用时，需要根据应用场景进行选择和配置。例如，对于 One-Time 使用的数据可以使用 Delete 策略，对于需要保留数据并长期存储的应用可以使用 Retain 策略。

然后就是指定存储类型，例如：hostPath、nfs、cephfs、rbd 等。

● 创建 PV（这里使用 nfs 方式提供存储，使用上小节已经安装好的 NFS）

（1）创建资源清单文件

```
[root@k8s-master01 ~]# vi pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: test
spec:
  capacity:
    storage: 5Gi
  accessModes: ["ReadOnlyMany"]
  nfs:
    path: /data/volumes
    server: 192.168.128.11
```

对上面的文件说明：

这是一个 Kubernetes PV 的 YAML 示例，其中包含以下字段：

kind：对象类型为 PersistentVolume。

apiVersion：Kubernetes API 版本。

metadata.name：PV 的名称。

spec.capacity.storage：PV 的存储容量为 5Gi。

spec.accessModes：PV 的访问模式为 ReadOnlyMany，即支持只读多访问模式。

spec.nfs.path：PV 存储的路径为/data/volumes。

spec.nfs.server：提供 NFS 服务的 IP 地址，即 192.168.128.11。

这个 YAML 文件定义了一个 PV，在 Kubernetes 中，该 PV 将表示一个具有 5G 存储容量、只读多访问模式、NFS 存储后端以及其它存储相关的特征。Kubernetes 管理员只需将该 PV 用于 Pod 容器即可。

需要注意的是，PV 一旦创建并绑定到 PVC 上后，它的存储容量及访问模式就无法再进行更改。因此，在设置 PV 的时候需要仔细评估存储需求，并选择对应的存储类型与容量，以满足业务需求。

（2）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f pv.yaml
persistentvolume/v1 created
```

（3）查看 pv

```
[root@k8s-master01 ~]# kubectl get pv
```

```
[root@k8s-master01 ~]# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
test   5Gi        ROX            Retain           Available                                   11s
[root@k8s-master01 ~]#
```

对上面的输出说明：

NAME：名称
CAPACITY：容量大小
ACCESS MODES：访问模式
RECLAIM POLICY：资源回收策略
STATUS：资源状态，Available 表示有空的，没有被 PVC 绑定。
CLAIM：由哪个供应商自动创建的 PV
STORAGECLASS：存储类型
AGE：运行时间

5.2.2、PVC 语法说明与创建

在 kubernetes 集群中，PersistentVolumeClaim 资源是用于定义 PVC 的。

● PersistentVolumeClaim 语法说明

PersistentVolumeClaim 语法可以通过如下命令进行查询：

```
[root@k8s-master01 ~]# kubectl explain PersistentVolumeClaim
```

下面主要说一下 spec 字段下的参数：

属性名称	取值类型	是否必选	取值说明
accessModes	[]string	required	指定访问模式。
resources	Object	required	用于指定 PV 或 PVC 的存储容量大小。
resources.limits	map[string]string		限制允许请求的最大容量大小。
resources.requests	map[string]string		请求所需的最小资源量。也就是最小容量。
storageClassName	string		storageClassName 是声明所需的 StorageClass 的名称。
volumeMode	string		用于指定 PV 和 PVC 的卷模式。卷模式决定了卷内

			的内容如何被呈现给容器。 Kubernetes 目前支持两种卷模式： ● Filesystem：该模式表示卷将被格式化为一个文件系统，并作为常规挂载点提供给容器。这是默认的卷模式。 ● Block：该模式表示卷被格式化为一个块设备卷，通常用于需要性能较高且需要显式控制块设备的应用程序。
volumeName	string		用于引用现有的 PersistentVolume（PV），即将该 PVC 绑定到指定的 PV 上。 请求的访问模式和存储容量大小必须要和 pv 匹配上，否则指定了 PV 进行绑定，也会绑定失败。

● 创建 PVC

（1）创建资源清单文件

```
[root@k8s-master01 ~]# vi pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test-pvc
spec:
  accessModes: ["ReadOnlyMany"]
  resources:
    requests:
      storage: 5Gi
```

对上面的文件说明：

这是一个 Kubernetes YAML 文件，用于创建一个名为 test-pvc 的 PersistentVolumeClaim（PVC）资源。配置信息如下：

apiVersion：v1：使用的 API 版本为 v1。

kind：PersistentVolumeClaim：指定这是 PersistentVolumeClaim 资源。

metadata.name：PVC 的名称为 test-pvc。

spec.accessMode：PVC 要求的访问模式为 ReadOnlyMany，即只读访问（多个节点只读）。

spec.resources.requests.storage：PVC 请求的存储资源大小为 5GiB。

根据这份配置文件，Kubernetes 将在系统中为 PVC 创建 5GiB 的只读存储，并将其分配给 Pod 使用。注意，这里的 PVC 没有指定数据源，它会用默认的存储类 provisioner 来创建 PV，并将 PV 与 PVC 进行绑定。

（2）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f pvc.yaml
persistentvolumeclaim/test-pvc created
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

（3）查看 PVC

```
[root@k8s-master01 ~]# kubectl get pvc
[root@k8s-master01 ~]# kubectl get pvc
NAME       STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
test-pvc   Bound    test     5Gi        ROX                           12s
[root@k8s-master01 ~]#
```

对上面的输出说明：

```
NAME：PVC 名称
STATUS：Bound 表示此 PVC 已经和 PV 绑定；未绑定则为 Pending 状态。
VOLUME：绑定的 PV 名称
CAPACITY：卷的大小
ACCESS MODES：访问模式。
STORAGECLASS：存储类型
AGE：运行时间
```

（4）查看 PV

```
[root@k8s-master01 ~]# kubectl get pv
[root@k8s-master01 ~]# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM              STORAGECLASS   REASON   AGE
test   5Gi        ROX            Retain           Bound    default/test-pvc                           11m
[root@k8s-master01 ~]#
```

可以看到此 PV 已经是绑定的状态，绑定到了 default 命名空间下的 test-pvc pvc。

5.2.3、挂载 PVC 给 Pod

小节目标：将上面创建好的 pv 和 pvc 挂载到 pod 内，测试删除 pod 数据是否还在，pv 和 pvc 是否有被删除。

（1）创建资源清单文件

```
[root@k8s-master01 ~]# cat pod-pvc.yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-pvc
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - name: nginx-html
      mountPath: /usr/share/nginx/html
  volumes:
  - name: nginx-html
```

```
      persistentVolumeClaim:
        claimName: test-pvc
```

（2）应用资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f pod-pvc.yaml
pod/pod-pvc created
```

（3）查看 Pod

```
[root@k8s-master01 ~]# kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
pod-pvc   1/1     Running   0          41s
```

（4）创建一个测试数据

```
[root@k8s-master01 ~]# cd /data/volumes/
[root@k8s-master01 volumes]# ls
[root@k8s-master01 volumes]# echo "This is test web" > index.html
```

（5）进入 Pod，测试挂载点是否可以读写

```
[root@k8s-master01 ~]# kubectl exec -it pod-pvc -- bash
root@pod-pvc:/# cd /usr/share/nginx/html/
root@pod-pvc:/usr/share/nginx/html# ls
index.html
root@pod-pvc:/usr/share/nginx/html# cat index.html
This is test web
root@pod-pvc:/usr/share/nginx/html# echo "I am zyf" > name.txt
root@pod-pvc:/usr/share/nginx/html# ls
index.html  name.txt
root@pod-pvc:/usr/share/nginx/html# cat name.txt
I am zyf
```

（6）查看本地数据

```
[root@k8s-master01 ~]# ls /data/volumes/
index.html  name.txt
[root@k8s-master01 ~]# cat /data/volumes/name.txt
I am zyf
```

（7）删除 Pod，查看数据是否还在，检查 PVC 状态

```
[root@k8s-master01 ~]# kubectl delete -f pod-pvc.yaml
pod "pod-pvc" deleted
[root@k8s-master01 ~]# ls /data/volumes/
index.html  name.txt
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 ~]# kubectl get pv
```

```
[root@k8s-master01 ~]# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM             STORAGECLASS   REASON   AGE
test   5Gi        ROX            Retain           Bound    default/test-pvc                          26m
[root@k8s-master01 ~]#
```

```
[root@k8s-master01 ~]# kubectl get pvc
```

```
[root@k8s-master01 ~]# kubectl get pvc
NAME       STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
test-pvc   Bound    test     5Gi        ROX                           20m
[root@k8s-master01 ~]#
```

可以看到删除 Pod 不会影响 PV 和 PVC，PV 和 PVC 还可以继续挂载到 pod 内进行使用，并且数据还在存在的。

5.3、PV 与 PVC 实战应用

目标：

从头开始安装 NFS，创建 10 个 PV，创建一个 PVC，观察 PV 与 PVC 绑定的关系。

（1）安装 NFS，创建 NFS 共享目录

```
1、安装 NFS
    [root@k8s-master01 ~]# yum -y install nfs-utils

2、创建 NFS 共享目录
    [root@k8s-master01 ~]# mkdir /data/data_v{1..10} -p
    [root@k8s-master01 ~]# ls /data/ -l
    total 0
    drwxr-xr-x 2 root root 6 Jun  3 06:03 data_v1
    drwxr-xr-x 2 root root 6 Jun  3 06:03 data_v10
    drwxr-xr-x 2 root root 6 Jun  3 06:03 data_v2
    drwxr-xr-x 2 root root 6 Jun  3 06:03 data_v3
    drwxr-xr-x 2 root root 6 Jun  3 06:03 data_v4
    drwxr-xr-x 2 root root 6 Jun  3 06:03 data_v5
    drwxr-xr-x 2 root root 6 Jun  3 06:03 data_v6
    drwxr-xr-x 2 root root 6 Jun  3 06:03 data_v7
    drwxr-xr-x 2 root root 6 Jun  3 06:03 data_v8
    drwxr-xr-x 2 root root 6 Jun  3 06:03 data_v9

3、修改 NFS 配置文件
    [root@k8s-master01 ~]# vi /etc/exports
    /data/data_v1 192.168.128.0/24(rw,no_root_squash)
```

```
    /data/data_v2 192.168.128.0/24(rw,no_root_squash)
    /data/data_v3 192.168.128.0/24(rw,no_root_squash)
    /data/data_v4 192.168.128.0/24(rw,no_root_squash)
    /data/data_v5 192.168.128.0/24(rw,no_root_squash)
    /data/data_v6 192.168.128.0/24(rw,no_root_squash)
    /data/data_v7 192.168.128.0/24(rw,no_root_squash)
    /data/data_v8 192.168.128.0/24(rw,no_root_squash)
    /data/data_v9 192.168.128.0/24(rw,no_root_squash)
    /data/data_v10 192.168.128.0/24(rw,no_root_squash)

4、启动 nfs 服务
    [root@k8s-master01 ~]# systemctl restart nfs && systemctl enable nfs &&
systemctl status nfs

5、检查
    [root@k8s-master01 ~]# showmount -e 127.0.0.1
    Export list for 127.0.0.1:
    /data/data_v10 192.168.128.0/24
    /data/data_v9  192.168.128.0/24
    /data/data_v8  192.168.128.0/24
    /data/data_v7  192.168.128.0/24
    /data/data_v6  192.168.128.0/24
    /data/data_v5  192.168.128.0/24
    /data/data_v4  192.168.128.0/24
    /data/data_v3  192.168.128.0/24
    /data/data_v2  192.168.128.0/24
    /data/data_v1  192.168.128.0/24
```

（2）创建 pv 资源清单文件

```
    [root@k8s-master01 ~]# vi data-pv.yaml
    apiVersion: v1
    kind: PersistentVolume
    metadata:
      name: v1
    spec:
      capacity:
        storage: 1Gi
      accessModes: ["ReadWriteOnce"]
      nfs:
        path: /data/data_v1
        server: 192.168.128.11
    ---
```

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: v2
spec:
  capacity:
    storage: 2Gi
  accessModes: ["ReadWriteMany"]
  nfs:
    path: /data/data_v2
    server: 192.168.128.11
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: v3
spec:
  capacity:
    storage: 3Gi
  accessModes: ["ReadOnlyMany"]
  nfs:
    path: /data/data_v3
    server: 192.168.128.11
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: v4
spec:
  capacity:
    storage: 4Gi
  accessModes: ["ReadWriteOnce","ReadWriteMany"]
  nfs:
    path: /data/data_v4
    server: 192.168.128.11
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: v5
spec:
  capacity:
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    storage: 5Gi
  accessModes: ["ReadWriteOnce","ReadWriteMany"]
  nfs:
    path: /data/data_v5
    server: 192.168.128.11
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: v6
spec:
  capacity:
    storage: 6Gi
  accessModes: ["ReadWriteOnce","ReadWriteMany"]
  nfs:
    path: /data/data_v6
    server: 192.168.128.11
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: v7
spec:
  capacity:
    storage: 7Gi
  accessModes: ["ReadWriteOnce","ReadWriteMany"]
  nfs:
    path: /data/data_v7
    server: 192.168.128.11
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: v8
spec:
  capacity:
    storage: 8Gi
  accessModes: ["ReadWriteOnce","ReadWriteMany"]
  nfs:
    path: /data/data_v8
    server: 192.168.128.11
---
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: v9
spec:
  capacity:
    storage: 9Gi
  accessModes: ["ReadWriteOnce","ReadWriteMany"]
  nfs:
    path: /data/data_v9
    server: 192.168.128.11
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: v10
spec:
  capacity:
    storage: 10Gi
  accessModes: ["ReadWriteOnce","ReadWriteMany"]
  nfs:
    path: /data/data_v10
    server: 192.168.128.11
```

（3）更新 pv 资源清单文件，并查看 pv 资源

```
[root@k8s-master01 ~]# kubectl apply -f data-pv.yaml
persistentvolume/v1 created
persistentvolume/v2 created
persistentvolume/v3 created
persistentvolume/v4 created
persistentvolume/v5 created
persistentvolume/v6 created
persistentvolume/v7 created
persistentvolume/v8 created
persistentvolume/v9 created
persistentvolume/v10 created

[root@k8s-master01 ~]# kubectl get pv
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
[root@k8s-master01 ~]# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
v1     1Gi        RWO            Retain           Available                                   11s
v10    10Gi       RWO,RWX        Retain           Available                                   11s
v2     2Gi        RWX            Retain           Available                                   11s
v3     3Gi        ROX            Retain           Available                                   11s
v4     4Gi        RWO,RWX        Retain           Available                                   11s
v5     5Gi        RWO,RWX        Retain           Available                                   11s
v6     6Gi        RWO,RWX        Retain           Available                                   11s
v7     7Gi        RWO,RWX        Retain           Available                                   11s
v8     8Gi        RWO,RWX        Retain           Available                                   11s
v9     9Gi        RWO,RWX        Retain           Available                                   11s
[root@k8s-master01 ~]#
```

（4）创建 PVC 资源清单文件

```
[root@k8s-master01 ~]# cat data-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: data-pvc
spec:
  accessModes: ["ReadWriteMany"]
  resources:
    requests:
      storage: 7Gi
```

（5）更新 PVC 资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f data-pvc.yaml
persistentvolumeclaim/data-pvc created

[root@k8s-master01 ~]# kubectl get pvc
```

```
[root@k8s-master01 ~]# kubectl get pvc
NAME       STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-pvc   Bound    v7       7Gi        RWO,RWX                       14s
```

```
[root@k8s-master01 ~]# kubectl get pv
```

```
[root@k8s-master01 ~]# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM             STORAGECLASS   REASON   AGE
v1     1Gi        RWO            Retain           Available                                           6m15s
v10    10Gi       RWO,RWX        Retain           Available                                           6m15s
v2     2Gi        RWX            Retain           Available                                           6m15s
v3     3Gi        ROX            Retain           Available                                           6m15s
v4     4Gi        RWO,RWX        Retain           Available                                           6m15s
v5     5Gi        RWO,RWX        Retain           Available                                           6m15s
v6     6Gi        RWO,RWX        Retain           Available                                           6m15s
v7     7Gi        RWO,RWX        Retain           Bound       default/data-pvc                        6m15s
v8     8Gi        RWO,RWX        Retain           Available                                           6m15s
v9     9Gi        RWO,RWX        Retain           Available                                           6m15s
[root@k8s-master01 ~]#
```

可以看到 pvc 申请的访问模式为 ReadWriteMany，pv v7 的访问模式为 ReadWriteOnce 和 ReadWriteMany，v7 pv 满足此 pvc，他们绑定在一起之后，具体的访问模式权限以 PV 为准。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

（6）创建 pod，挂载 pvc

```
1、创建资源清单文件
    [root@k8s-master01 ~]# vi data-pod.yaml
    apiVersion: v1
    kind: Pod
    metadata:
      name: pod-pvc
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
        - name: nginx-html
          mountPath: /usr/share/nginx/html
      volumes:
      - name: nginx-html
        persistentVolumeClaim:
          claimName: data-pvc

2、更新资源清单文件
    [root@k8s-master01 ~]# kubectl apply -f data-pod.yaml
    pod/pod-pvc created
```

（6）进入 pod-pvc 创建测试数据

```
    [root@k8s-master01 ~]# kubectl exec -it pod-pvc -- bash
    root@pod-pvc:/# cd /usr/share/nginx/html/
    root@pod-pvc:/usr/share/nginx/html#  echo  "hello,  xiaozhang"  >
index.html
    root@pod-pvc:/usr/share/nginx/html# cat index.html
    hello, xiaozhang
```

（7）删除 pod，测试数据是否还在

```
    [root@k8s-master01 ~]# kubectl delete -f data-pod.yaml
    pod "pod-pvc" deleted
    [root@k8s-master01 ~]#
    [root@k8s-master01 ~]# ls /data/data_v7/
    index.html
    [root@k8s-master01 ~]# cat /data/data_v7/index.html
    hello, xiaozhang
```

（8）删除 PVC 测试

```
1、删除 pvc
```

```
[root@k8s-master01 ~]# kubectl delete -f data-pvc.yaml
persistentvolumeclaim "data-pvc" deleted
```

2、查看 pv 状态

```
[root@k8s-master01 ~]# kubectl get pv
```

```
[root@k8s-master01 ~]# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM              STORAGECLASS   REASON   AGE
v1     1Gi        RWO            Retain           Available                                             15m
v10    10Gi       RWO,RWX        Retain           Available                                             15m
v2     2Gi        RWX            Retain           Available                                             15m
v3     3Gi        ROX            Retain           Available                                             15m
v4     4Gi        RWO,RWX        Retain           Available                                             15m
v5     5Gi        RWO,RWX        Retain           Available                                             15m
v6     6Gi        RWO,RWX        Retain           Available                                             15m
v7     7Gi        RWO,RWX        Retain           Released    default/data-pvc                          15m
v8     8Gi        RWO,RWX        Retain           Available                                             15m
v9     9Gi        RWO,RWX        Retain           Available                                             15m
[root@k8s-master01 ~]#
```

```
# 删除 pvc 之后，pv 会处于 released 状态，想要继续使用这个 pv，需要手动删除 pv，kubectl delete pv pv_name，然后再重新创建此 PV。
```

3、查看数据是否还在

```
[root@k8s-master01 ~]# ls /data/data_v7/
index.html
```

（9）再次创建 pvc，查看 pvc 状态

1、再次创建 pvc

```
[root@k8s-master01 ~]# kubectl apply -f data-pvc.yaml
persistentvolumeclaim/data-pvc created
```

2、查看 pvc 状态

```
[root@k8s-master01 ~]# kubectl get pvc
```

```
[root@k8s-master01 ~]# kubectl get pvc
NAME       STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-pvc   Bound    v8       8Gi        RWO,RWX                       3s
[root@k8s-master01 ~]#
```

```
# 可以看到 pvc 和 pv v8 绑定到了一块
```

3、查看 pv 状态

```
[root@k8s-master01 ~]# kubectl get pv
```

```
[root@k8s-master01 ~]# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM              STORAGECLASS   REASON   AGE
v1     1Gi        RWO            Retain           Available                                             18m
v10    10Gi       RWO,RWX        Retain           Available                                             18m
v2     2Gi        RWX            Retain           Available                                             18m
v3     3Gi        ROX            Retain           Available                                             18m
v4     4Gi        RWO,RWX        Retain           Available                                             18m
v5     5Gi        RWO,RWX        Retain           Available                                             18m
v6     6Gi        RWO,RWX        Retain           Available                                             18m
v7     7Gi        RWO,RWX        Retain           Released    default/data-pvc                          18m
v8     8Gi        RWO,RWX        Retain           Bound       default/data-pvc                          18m
v9     9Gi        RWO,RWX        Retain           Available                                             18m
[root@k8s-master01 ~]#
```

（10）若想使用 v7，则必须删除再创建

1、重新创建所有 pv 和 pvc

```
[root@k8s-master01 ~]# kubectl delete -f data-pvc.yaml
[root@k8s-master01 ~]# kubectl delete -f data-pv.yaml
[root@k8s-master01 ~]# kubectl apply -f data-pv.yaml
[root@k8s-master01 ~]# kubectl apply -f data-pvc.yaml
```

2、查看 pv

```
[root@k8s-master01 yaml]# kubectl get pv
```

```
[root@k8s-master01 ~]# kubectl get pv
NAME   CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM              STORAGECLASS   REASON   AGE
v1     1Gi        RWO            Retain           Available                                             53s
v10    10Gi       RWO,RWX        Retain           Available                                             53s
v2     2Gi        RWX            Retain           Available                                             53s
v3     3Gi        ROX            Retain           Available                                             53s
v4     4Gi        RWO,RWX        Retain           Available                                             53s
v5     5Gi        RWO,RWX        Retain           Available                                             53s
v6     6Gi        RWO,RWX        Retain           Available                                             53s
v7     7Gi        RWO,RWX        Retain           Bound       default/data-pvc                          53s
v8     8Gi        RWO,RWX        Retain           Available                                             53s
v9     9Gi        RWO,RWX        Retain           Available                                             53s
[root@k8s-master01 ~]#
```

3、查看 pvc

```
[root@k8s-master01 ~]# kubectl get pvc
```

```
[root@k8s-master01 ~]# kubectl get pvc
NAME       STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
data-pvc   Bound    v7       7Gi        RWO,RWX                       65s
[root@k8s-master01 ~]#
```

4、查看数据

```
[root@k8s-master01 ~]# ls /data/data_v7/
index.html
[root@k8s-master01 ~]# cat /data/data_v7/index.html
hello, xiaozhang
```

为什么删除 pv 时，数据不会丢失？

因为默认回收策略是 Retain（保留），所以数据不会被删除。

注意：使用 pvc 和 pv 的注意事项

1、我们每次创建 pvc 的时候，需要事先有划分好的 pv，这样可能不方便，那么可以在创建 pvc 的时候直接动态创建一个 pv 这个存储类，pv 事先是不存在的。

2、pvc 和 pv 绑定，如果使用默认的回收策略 retain，那么删除 pvc 之后，pv 会处于 released 状态，我们想要继续使用这个 pv，需要手动删除 pv，kubectl delete pv pv_name，删除 pv，不会删除 pv 里的数据，当我们重新创建 pvc 时还会和这个最匹配的 pv 绑定，数据还是原来数据，不会丢失。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

6、k8s 存储类：storageclass

6.1、什么是 storageclass

上面介绍的 PV 和 PVC 模式都是需要先创建好 PV，然后定义好 PVC 和 pv 进行一对一的 Bound（绑定），但是如果 PVC 请求成千上万，那么就需要创建成千上万的 PV，对于运维人员来说维护成本很高，Kubernetes 提供一种自动创建 PV 的机制，叫 StorageClass，它的作用就是创建 PV 的模板。k8s 集群管理员通过创建 storageclass 可以动态生成一个存储卷 pv 供 k8s pvc 使用。

具体来说，StorageClass 会定义以下两部分：

1、PV 的属性，比如存储的大小、类型等。

2、创建这种 PV 需要使用到的存储插件，比如 Ceph、NFS 等。

有了这两部分信息，Kubernetes 就能够根据用户提交的 PVC，找到对应的 StorageClass，然后 Kubernetes 就会调用 StorageClass 声明的存储插件，创建出需要的 PV。

6.2、storageclass 语法说明

在 kubernetes 集群中，StorageClass 资源用于快速创建 PV。

● StorageClass 整体语法说明

StorageClass 语法可以通过如下命令进行查询：

```
[root@k8s-master01 ~]# kubectl explain StorageClass
KIND:     StorageClass
VERSION:  storage.k8s.io/v1
DESCRIPTION:
     StorageClass describes the parameters for a class of storage for which
     PersistentVolumes can be dynamically provisioned.
     StorageClasses are non-namespaced; the name of the storage class according
     to etcd is in ObjectMeta.Name.
FIELDS:
   allowVolumeExpansion <boolean>
   allowedTopologies    <[]Object>
   apiVersion   <string>
   kind <string>
   metadata <Object>
   mountOptions <[]string>
   parameters   <map[string]string>
   provisioner  <string> -required-
   reclaimPolicy    <string>
   volumeBindingMode    <string>
```

- provisioner 字段说明：

每个 StorageClass 都有一个制备器（provisioner），provisioner 用来确定我们使用什么样的存储来创建 pv，常见的 provisioner 如下：

参考地址：https://kubernetes.io/zh/docs/concepts/storage/storage-classes/

下图来自官网（Kubernetes 1.28 版本支持的卷插件）：

存储制备器

每个 StorageClass 都有一个制备器（Provisioner），用来决定使用哪个卷插件制备 PV。该字段必须指定。

卷插件	内置制备器	配置示例
AzureFile	✓	Azure File
CephFS	-	-
FC	-	-
FlexVolume	-	-
iSCSI	-	-
NFS	-	NFS
RBD	✓	Ceph RBD
VsphereVolume	✓	vSphere
PortworxVolume	✓	Portworx Volume
Local	-	Local

provisioner 打对勾的表示可以由内部供应商提供，也可以由外部供应商提供。

如果是外部供应商可以参考如下提供的方法创建：

https://github.com/kubernetes-retired/external-storage

https://github.com/kubernetes-sigs/sig-storage-lib-external-provisioner

- allowVolumeExpansion 字段说明：

allowVolumeExpansion：允许卷扩展，PersistentVolume 可以配置成可扩展。将此功能设置为 true 时，允许用户通过编辑相应的 PVC 对象来调整卷大小。以下类型的卷支持卷扩展：

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

卷类型	Kubernetes 版本要求
gcePersistentDisk	1.11
awsElasticBlockStore	1.11
Cinder	1.11
glusterfs	1.11
rbd	1.11
Azure File	1.11
Azure Disk	1.11
Portworx	1.11
FlexVolume	1.13
CSI	1.14 (alpha), 1.16 (beta)

注意：此功能仅用于扩容卷，不能用于缩小卷。

- StorageClass 整体语法说明

属性名称	取值类型	是否必选	取值说明
allowVolumeExpansion	boolean		指定了存储卷是否可以动态扩展其容量。这意味着如果存储卷的容量已经满了，可以通过修改StorageClass的请求容量来扩展现有的存储卷的容量，而不必创建一个新的存储卷。 扩展存储卷的能力取决于底层存储插件是否支持扩容，以及存储卷所连接的节点上是否有足够的可用存储资源。

allowedTopologies	[]Object		指定了存储卷可以被动态分配的节点拓扑域。
			示例：
			apiVersion: storage.k8s.io/v1
			kind: StorageClass
			metadata:
			  name: standard
			provisioner: kubernetes.io/gce-pd
			parameters:
			  type: pd-standard
			volumeBindingMode: WaitForFirstConsumer
			allowedTopologies:
			- matchLabelExpressions:
			  - key: kubernetes.io/hostname

matchLabelExpressions.matchLabelExpressions.key	string	required	
matchLabelExpressions.matchLabelExpressions.values	[]string	required	

			values: - k8s-node01 - k8s-node02 这是一个 Kubernetes 的 StorageClass 的 yaml 文件，属性包括： ● volumeBindingMode：WaitForFirstConsumer 表示存储卷的绑定模式为 Waiting。即等待 Pod 被调度到节点上再绑定存储卷。 ● allowedTopologies：表示存储卷可以被动态分配的节点拓扑域。 ● matchLabelExpressions：表示节点拓扑域的匹配。这里使用的是 matchLabelExpressions 来匹配节点。 ● key：kubernetes.io/hostname 表示匹配的节点标签。 ● values：表示匹配标签值的节点是 k8s-node01 和 k8s-node02。
volumeBindingMode	string		指定了 Kubernetes 如何绑定动态分配的存储卷。 Kubernetes 中存在两种存储卷绑定模式： ● Immediate 模式：在创建 Pod 的过程中立即绑定存储卷。 若使用此模式，应设置值为“Immediate”。 ● Waiting 模式：等待 Pod 被调度到节点上，然后再绑定存储卷。 若使用此模式，应设置值为“WaitForFirstConsumer”。 默认情况下，Kubernetes 使用 Immediate 模式来绑定动态分配的存储卷。这意味着在创建 Pod 时，存储卷将会立即被绑定并用于该 Pod。然而，在某些情况下，可能需要使用 Waiting 模式来等待 Pod 被调度到节点上再绑定存储卷。例如，如果使用的存储插件需要等到 Pod 被调度到指定的节点上，才能将存储卷绑定到 Pod 上。
provisioner	string	required	provisioner 用来确定我们使用什么样的存储来创建 pv。
reclaimPolicy	string		定义 Persistent Volume（PV）的回收策略，当该 PV 被释放时是否要对其进行清除操作。 可以设置以下三种回收策略： ● Retain：保留存储资源，当 PV 释放时不会执行清除操作，需要手动清理对应的存储资源。 ● Delete：在 PV 释放时将其对应的存储资源删除，并释放存储区块。 ● Recycle：PV 的回收策略为回收并重用 PV。

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

6.3、实战：使用 NFS provisioner 动态生成 PV

6.1、NFS-Subdir-External-Provisioner 简介

NFS-Subdir-External-Provisioner 是 Kubernetes 中的一个应用程序，它可以自动为一个或多个 Kubernetes Persistent Volume Claims（PVCs）创建 NFS 共享，并将它们挂载到相应的 Pod 中。这个程序主要是为了解决 Kubernetes 环境中动态 PV 创建的问题。使用这个程序可以大大降低 PV 的管理配置工作量。

该程序通过监控 Kubernetes 的 StorageClasses 和 PersistentVolumeClaims 资源，自动创建和删除 PersistentVolum 资源。这个程序使用了 NFS 共享服务器提供的卷。当一个 PVC 请求被创建时，该程序会新建一个与其相容的 PV，并将其插入到 Kubernetes Persistent Volume 树中。当 PVC 资源被删除时，将会自动删除 PV 资源。

GitHub 地址：（下面部署皆参考此）

https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner

6.2、安装 nfs provisioner

（1）安装 NFS Server 并创建数据存放目录

```
1、安装 nfs
    [root@k8s-master01 ~]# yum install nfs-utils -y

2、创建 NFS 需要的共享目录
    [root@k8s-master01 ~]# mkdir /nfs/data -p

3、配置 nfs 共享服务器，共享/data/volumes 目录
    [root@k8s-master01 ~]# vi /etc/exports
    /nfs/data 192.168.128.0/24(rw,no_root_squash)
    # no_root_squash: 用户具有根目录的完全管理访问权限

4、启动 nfs 服务
    [root@k8s-master01 ~]# systemctl restart nfs-server && systemctl enable
nfs-server && systemctl status nfs-server
```

（2）创建运行 nfs-provisioner 需要的 sa 账号

```
    [root@k8s-master01 ~]# vi nfs-serviceaccount.yaml
    apiVersion: v1
    kind: ServiceAccount
    metadata:
      name: nfs-client-provisioner
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    # replace with namespace where provisioner is deployed
    namespace: default

[root@k8s-master01 ~]# kubectl apply -f nfs-serviceaccount.yaml
serviceaccount/nfs-client-provisioner created

[root@k8s-master01 ~]# kubectl get sa nfs-client-provisioner
NAME                     SECRETS   AGE
nfs-client-provisioner   0         16s
```

对上面的 yaml 文件说明：

这是一个 Kubernetes ServiceAccount 的 YAML 文件，用于创建一个名为 nfs-client-provisioner 的服务帐户，并将其部署到名为 default 的命名空间中。

扩展：什么是 sa？

sa 的全称是 serviceaccount。

serviceaccount 是为了方便 Pod 里面的进程调用 Kubernetes API 或其他外部服务而设计的。

在创建 Pod 的时候指定 serviceaccount，那么当 Pod 运行后，他就拥有了我们指定账号的权限了。

（3）对 sa 授权

```
[root@k8s-master01 ~]# vi nfs-rbac.yaml
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: nfs-client-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "delete"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
```

```
    resources: ["events"]
    verbs: ["create", "update", "patch"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: run-nfs-client-provisioner
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    # replace with namespace where provisioner is deployed
    namespace: default
roleRef:
  kind: ClusterRole
  name: nfs-client-provisioner-runner
  apiGroup: rbac.authorization.k8s.io
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  # replace with namespace where provisioner is deployed
  namespace: default
rules:
  - apiGroups: [""]
    resources: ["endpoints"]
    verbs: ["get", "list", "watch", "create", "update", "patch"]
---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: leader-locking-nfs-client-provisioner
  # replace with namespace where provisioner is deployed
  namespace: default
subjects:
  - kind: ServiceAccount
    name: nfs-client-provisioner
    # replace with namespace where provisioner is deployed
    namespace: default
roleRef:
  kind: Role
  name: leader-locking-nfs-client-provisioner
```

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。

```
    apiGroup: rbac.authorization.k8s.io

[root@k8s-master01 ~]# kubectl apply -f nfs-rbac.yaml
clusterrole.rbac.authorization.k8s.io/nfs-client-provisioner-runner created
clusterrolebinding.rbac.authorization.k8s.io/run-nfs-client-provisioner created
role.rbac.authorization.k8s.io/leader-locking-nfs-client-provisioner created
rolebinding.rbac.authorization.k8s.io/leader-locking-nfs-client-provisioner created
```

对上面的 yaml 文件说明：

首先是一个 ClusterRole YAML 文件，它定义了 nfs-client-provisioner-runner 角色，可以访问一些资源，比如节点、持久化卷、持久化卷声明、存储类和事件，并且可以执行查看和监听操作。除此之外还可以创建、删除、更新、补丁操作和更新 PV。

接下来是一个 ClusterRoleBinding YAML 文件，它将上述角色绑定到 nfs-client-provisioner Service Account。

接下来是一个 Role YAML 文件，它定义了 leader-locking-nfs-client-provisioner 角色，该角色具有 get、list、watch、create、update 和 patch 操作的权限。

接下来，是一个 RoleBinding YAML 文件，它将上述角色绑定到 nfs-client-provisioner Service Account。

（4）安装 nfs-provisioner 程序

```
[root@k8s-master01 ~]# vi nfs-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nfs-client-provisioner
  labels:
    app: nfs-client-provisioner
  # replace with namespace where provisioner is deployed
  namespace: default
spec:
  replicas: 1
  strategy:
    type: Recreate
  selector:
    matchLabels:
      app: nfs-client-provisioner
  template:
    metadata:
      labels:
```

```
            app: nfs-client-provisioner
        spec:
          serviceAccountName: nfs-client-provisioner
          containers:
            - name: nfs-client-provisioner
              image:
registry.cn-hangzhou.aliyuncs.com/lfy_k8s_images/nfs-subdir-external-provisione
r:v4.0.2
              # resources:
              #    limits:
              #      cpu: 10m
              #    requests:
              #      cpu: 10m
              volumeMounts:
                - name: nfs-client-root
                  mountPath: /persistentvolumes
              env:
                - name: PROVISIONER_NAME  #供应商名称，创建 SC 的时候需要指定
                  value: kubernetes.test/nfs
                - name: NFS_SERVER  #指定 nfs 服务器地址
                  value: 192.168.128.11
                - name: NFS_PATH  #指定 nfs 服务器共享的目录
                  value: /nfs/data
          volumes:
            - name: nfs-client-root
              nfs:   #使用 nfs 方式进行挂载，挂载到容器内
                server: 192.168.128.11
                path: /nfs/data

    [root@k8s-master01 ~]# kubectl apply -f nfs-deployment.yaml
    deployment.apps/nfs-client-provisioner created

    [root@k8s-master01 ~]# kubectl get pods
    [root@k8s-master01 ~]# kubectl get pods
    NAME                                     READY   STATUS    RESTARTS
    nfs-client-provisioner-596b57b9bf-n9s5v   1/1     Running   0
    [root@k8s-master01 ~]#
```

6.3、创建 storageclass，动态供给 pv

（1）创建资源清单文件

```
    [root@k8s-master01 ~]# vi nfs-storageclass.yaml
    kind: StorageClass
```

```
apiVersion: storage.k8s.io/v1
metadata:
  name: nfs-storage
provisioner: kubernetes.test/nfs
parameters:
  archiveOnDelete: "true"

[root@k8s-master01 ~]# kubectl apply -f nfs-storageclass.yaml
storageclass.storage.k8s.io/nfs-storage created
```

对上面的 yaml 文件说明：

这个 Kubernetes 的 YAML 配置文件用来定义 NFS 存储类 `nfs-storage`。

metadata 属性这里用来指定 NFS 存储类的名称为 nfs-storage。

provisioner 属性指定了用来提供持久化卷的插件名称，这里是 kubernetes.test/nfs。即使用刚刚部署的 nfs-client-provisioner 容器来提供持久化卷。

parameters 属性用来指定一些额外参数，这里使用了一个名为 archiveOnDelete 的参数，将其值设置为 true。这意味着当 PVC 被删除时，NFS 服务器将会把存储的数据存档，而不是直接删除。

（2）查看 storageclass 资源

```
[root@k8s-master01 ~]# kubectl get storageclass
```

```
[root@k8s-master01 ~]# kubectl get storageclass
NAME          PROVISIONER           RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
nfs-storage   kubernetes.test/nfs   Delete          Immediate           false                  5m47s
[root@k8s-master01 ~]#
```

显示以上内容，说明 storageclass 创建成功了。

6.4、创建 pvc，通过 storageclass 动态生成 pv

（1）创建资源清单文件

```
[root@k8s-master01 ~]# vi nfs-pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: nfs-test-pvc
spec:
  accessModes: ["ReadWriteMany"]
  resources:
    requests:
      storage: 1Gi
  storageClassName: nfs-storage
```

（2）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f nfs-pvc.yaml
```

```
persistentvolumeclaim/nfs-test-pvc created
```

（3）查看是否动态生成了 pv，pvc 是否创建成功，并和 pv 绑定

```
[root@k8s-master01 ~]# kubectl get pvc
NAME          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   AGE
nfs-test-pvc  Bound    pvc-728a1bed-d0ff-4372-bf2d-baf7019f7354   1Gi        RWX            nfs-storage    38s
[root@k8s-master01 ~]#
```

```
[root@k8s-master01 ~]# kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                 STORAGECLASS   REASON   AGE
pvc-728a1bed-d0ff-4372-bf2d-baf7019f7354   1Gi        RWX            Delete           Bound    default/nfs-test-pvc  nfs-storage             59s
[root@k8s-master01 ~]#
```

（4）查看 NFS 共享目录

```
[root@k8s-master01 ~]# ll /nfs/data/
total 0
drwxrwxrwx 2 root root 6 Jun  4 03:22 default-nfs-test-pvc-pvc-a9111d19-3dde-4bba-b2e0-1e5eb71a0cde
[root@k8s-master01 ~]#
```

这里的目录结构为：“命名空间-PVC 名称-PV 名称”。所以一般在生产环境中，我们一般创建 PVC 时，指定 PVC 的名称和 Pod 名称一样，就可以方便我们维护。

6.5、创建 pod，挂载 storageclass 动态生成的 pvc

（1）创建资源清单文件

```
[root@k8s-master01 ~]# vi nfs-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nfs-test
spec:
  containers:
  - name: nfs-test
    image: nginx
    volumeMounts:
      - name: nfs-pvc
        mountPath: /usr/share/nginx/html
  restartPolicy: "Never"
  volumes:
    - name: nfs-pvc
      persistentVolumeClaim:
        claimName: nfs-test-pvc
```

（2）更新资源清单文件

```
[root@k8s-master01 ~]# kubectl apply -f nfs-pod.yaml
pod/nfs-test created
```

（3）查看 pod 是否创建成功：

```
[root@k8s-master01 ~]# kubectl get pods nfs-test
NAME       READY   STATUS    RESTARTS   AGE
nfs-test   1/1     Running   0          77s
```

（4）创建测试数据

```
1、进入到 pod 中创建测试数据
    [root@k8s-master01 ~]# kubectl exec -it nfs-test -- /bin/sh
    # cd /usr/share/nginx/html
    # echo "This is test web" > index.html

2、查看 nfs 共享目录下的数据。
    [root@k8s-master01                          ~]#                          ls
/nfs/data/default-nfs-test-pvc-pvc-a9111d19-3dde-4bba-b2e0-1e5eb71a0cde/
    index.html
```

（5）删除 pod 和 pvc

```
[root@k8s-master01 ~]# kubectl delete -f nfs-pod.yaml
pod "nfs-test" deleted

[root@k8s-master01 ~]# kubectl delete -f nfs-pvc.yaml
persistentvolumeclaim "nfs-test-pvc" deleted
```

（6）查看 pv

```
[root@k8s-master01 ~]# kubectl get pv
No resources found
```

会发现删除 pv 所绑定的 pvc 之后，pv 也会被删除。

（7）进入到共享目录中，会发现，目录前加了 archived

```
1、查看 nfs 共享目录下的文件
    [root@k8s-master01 ~]# cd /nfs/data/
    [root@k8s-master01 data]# ls
    archived-default-nfs-test-pvc-pvc-a9111d19-3dde-4bba-b2e0-1e5eb71
a0cde

2、进入到目录中查看数据
```

```
[root@k8s-master01 data]# cd archived-default-nfs-test-pvc-pvc-a9111d19-3dde-4bba-b2e0-1e5eb71a0cde/
[root@k8s-master01 archived-default-nfs-test-pvc-pvc-a9111d19-3dde-4bba-b2e0-1e5eb71a0cde]# ls
index.html
[root@k8s-master01 archived-default-nfs-test-pvc-pvc-a9111d19-3dde-4bba-b2e0-1e5eb71a0cde]# cat index.html
This is test web
```

可以发现数据是没有被删除的。如果过时的数据或者无用的数据，我们可以手动去做清理。（自动清理风险太高）

版权声明，本文档全部内容及版权归“张岩峰”老师所有，只可用于自己学习使用，禁止私自传阅，违者依法追责。