概念

`PersistentVolume`（PV）

是由管理员设置的存储，它是群集的一部分。就像节点是集群中的资源一样，PV 也是集群中的资源。 PV 是 Volume 之类的卷插件，但具有独立于使用 PV 的 Pod 的生命周期。此 API 对象包含存储实现的细节，即 NFS、iSCSI 或特定于云供应商的存储系统

`PersistentVolumeClaim`（PVC）

是用户存储的请求。它与 Pod 相似。Pod 消耗节点资源，PVC 消耗 PV 资源。Pod 可以请求特定级别的资源（CPU 和内存）。声明可以请求特定的大小和访问模式（例如，可以以读/写一次或 只读多次模式挂载）

静态 pv

集群管理员创建一些 PV。它们带有可供群集用户使用的实际存储的细节。它们存在于 Kubernetes API 中，可用于消费

动态

当管理员创建的静态 PV 都不匹配用户的 `PersistentVolumeClaim` 时，集群可能会尝试动态地为 PVC 创建卷。此配置基于 `StorageClasses`：PVC 必须请求 [存储类]，并且管理员必须创建并配置该类才能进行动态创建。声明该类为 `""` 可以有效地禁用其动态配置

要启用基于存储级别的动态存储配置，集群管理员需要启用 API server 上的 `DefaultStorageClass` [准入控制器]。例如，通过确保 `DefaultStorageClass` 位于 API server 组件的 `--admission-control` 标志，使用逗号分隔的有序值列表中，可以完成此操作

绑定

master 中的控制环路监视新的 PVC，寻找匹配的 PV（如果可能），并将它们绑定在一起。如果为新的 PVC 动态调配 PV，则该环路将始终将该 PV 绑定到 PVC。否则，用户总会得到他们所请求的存储，但是容量可能超出要求的数量。一旦 PV 和 PVC 绑定后，`PersistentVolumeClaim` 绑定是排他性的，不管它们是如何绑定的。 PVC 跟 PV 绑定是一对一的映射

持久化卷声明的保护

PVC 保护的目的是确保由 pod 正在使用的 PVC 不会从系统中移除，因为如果被移除的话可能会导致数据丢失

当启用PVC 保护 alpha 功能时，如果用户删除了一个 pod 正在使用的 PVC，则该 PVC 不会被立即删除。PVC 的删除将被推迟，直到 PVC 不再被任何 pod 使用

持久化卷类型

`PersistentVolume` 类型以插件形式实现。Kubernetes 目前支持以下插件类型：

- GCEPersistentDisk AWSElasticBlockStore AzureFile AzureDisk FC (Fibre Channel)
- FlexVolume Flocker NFS iSCSI RBD (Ceph Block Device) CephFS
- Cinder (OpenStack block storage) Glusterfs VsphereVolume Quobyte Volumes
- HostPath VMware Photon Portworx Volumes ScaleIO Volumes StorageOS

持久卷演示代码

```
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
```

PV 访问模式

`PersistentVolume` 可以以资源提供者支持的任何方式挂载到主机上。如下表所示，供应商具有不同的功能，每个 PV 的访问模式都将被设置为该卷支持的特定模式。例如，NFS 可以支持多个读/写客户端，但特定的 NFS PV 可能以只读方式导出到服务器上。每个 PV 都有一套自己的用来描述特定功能的访问模式

- ReadWriteOnce——该卷可以被单个节点以读/写模式挂载
- ReadOnlyMany——该卷可以被多个节点以只读模式挂载
- ReadWriteMany——该卷可以被多个节点以读/写模式挂载

在命令行中，访问模式缩写为：

- RWO - ReadWriteOnce
- ROX - ReadOnlyMany
- RWX - ReadWriteMany

-

Volume 插件	ReadWriteOnce	ReadOnlyMany	ReadWriteMany
AWSElasticBlockStoreAWSElasticBlockStore	✓	-	-
AzureFile	✓	✓	✓
AzureDisk	✓	-	-
CephFS	✓	✓	✓
Cinder	✓	-	-
FC	✓	✓	-
FlexVolume	✓	✓	-
Flocker	✓	-	-
GCEPersistentDisk	✓	✓	-
Glusterfs	✓	✓	✓
HostPath	✓	-	-
iSCSI	✓	✓	-
PhotonPersistentDisk	✓	-	-
Quobyte	✓	✓	✓
NFS	✓	✓	✓
RBD	✓	✓	-
VsphereVolume	✓	-	-（当 pod 并列时有效）
PortworxVolume	✓	-	✓
ScaleIO	✓	✓	-
StorageOS	✓	-	-

回收策略

- **Retain（保留）——手动回收**
- **Recycle（回收）——基本擦除（`rm -rf /thevolume/*`）**
- **Delete（删除）——关联的存储资产（例如 AWS EBS、GCE PD、Azure Disk 和 OpenStack Cinder 卷）将被删除**

当前，只有 NFS 和 HostPath 支持回收策略。AWS EBS、GCE PD、Azure Disk 和 Cinder 卷支持删除策略

状态

卷可以处于以下的某种状态：

- **Available（可用）——一块空闲资源还没有被任何声明绑定**
- **Bound（已绑定）——卷已经被声明绑定**
- **Released（已释放）——声明被删除，但是资源还未被集群重新声明**

- **Failed（失败）——该卷的自动回收失败**

命令行会显示绑定到 PV 的 PVC 的名称

持久化演示说明 - NFS

Ⅰ、安装 NFS 服务器

```
yum install -y nfs-common nfs-utils  rpcbind
mkdir /nfsdata
chmod 666 /nfsdata
chown nfsnobody /nfsdata
cat /etc/exports
    /nfsdata *(rw,no_root_squash,no_all_squash,sync)
systemctl start rpcbind
systemctl start nfs
```

Ⅱ、部署 PV

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfspv1
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: nfs
  nfs:
    path: /data/nfs
    server: 10.66.66.10
```

Ⅲ、创建服务并使用 PVC

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  ports:
  - port: 80
    name: web
  clusterIP: None
  selector:

    app: nginx
```

```yaml
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: web
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: k8s.gcr.io/nginx-slim:0.8
        ports:
        - containerPort: 80
          name: web
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "nfs"
      resources:
        requests:
          storage: 1Gi
```

关于 StatefulSet

- **匹配 Pod name (网络标识) 的模式为：$(statefulset名称)-$(序号)，比如上面的示例：web-0，web-1，web-2**
- **StatefulSet 为每个 Pod 副本创建了一个 DNS 域名，这个域名的格式为： $(podname).(headless server name)，也就意味着服务间是通过Pod域名来通信而非 Pod IP，因为当Pod所在Node发生故障时， Pod 会被飘移到其它 Node 上，Pod IP 会发生变化，但是 Pod 域名不会有变化**
- **StatefulSet 使用 Headless 服务来控制 Pod 的域名，这个域名的 FQDN 为：$(service name).$(namespace).svc.cluster.local，其中，“cluster.local” 指的是集群的域名**
- **根据 volumeClaimTemplates，为每个 Pod 创建一个 pvc，pvc 的命名规则匹配模式：(volumeClaimTemplates.name)-(pod_name)，比如上面的 volumeMounts.name=www， Pod name=web-[0-2]，因此创建出来的 PVC 是 www-web-0、www-web-1、www-web-2**
- **删除 Pod 不会删除其 pvc，手动删除 pvc 将自动释放 pv**

Statefulset的启停顺序：

- **有序部署：部署StatefulSet时，如果有多个Pod副本，它们会被顺序地创建（从0到N-1）并且，在下一个Pod运行之前所有之前的Pod必须都是Running和Ready状态。**
- **有序删除：当Pod被删除时，它们被终止的顺序是从N-1到0。**
- **有序扩展：当对Pod执行扩展操作时，与部署一样，它前面的Pod必须都处于Running和Ready状态。**

StatefulSet使用场景：

- **稳定的持久化存储，即Pod重新调度后还是能访问到相同的持久化数据，基于 PVC 来实现。**
- **稳定的网络标识符，即 Pod 重新调度后其 PodName 和 HostName 不变。**
- **有序部署，有序扩展，基于 init containers 来实现。**
- **有序收缩。**