**加我微信可进群学习交流：**

**微信号：**

luckylucky421302

也可通过扫描下面二维码添加



**课程更新的知识点会通过微信公众号免费分享给大家，可以关注我的公众号**

****

**1.pv**

PersistentVolume（PV）是群集中的一块存储，由管理员配置或使用存储类动态配置。 它是集群中的资源，就像节点是集群资源一样。 PV是容量插件，如Volumes，但其生命周期独立于使用PV的任何单个pod。 此API对象捕获存储实现的详细信息，包括NFS，iSCSI或特定于云提供程序的存储系统

**2.pvc**

PersistentVolumeClaim（PVC）是一个持久化存储卷，我们在创建pod时可以定义这个类型的存储卷。 它类似于一个pod。 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）。 pvc在申请pv的时候也可以请求特定的大小和访问模式（例如，可以一次读/写或多次只读）。

**3.pv和pvc**

**3.1 pv和pvc的生命周期**

PV是群集中的资源。 PVC是对这些资源的请求，并且还充当对资源的索赔检查。 PV和PVC之间的相互作用遵循以下生命周期：

**（1）pv的供应方式**

可以通过两种方式配置PV：静态或动态。

**静态的**

集群管理员创建了许多PV。它们包含可供群集用户使用的实际存储的详细信息。它们存在于Kubernetes API中，可供使用。

**动态的**

当管理员创建的静态PV都不匹配用户的PersistentVolumeClaim时，群集可能会尝试为PVC专门动态配置卷。此配置基于StorageClasses：PVC必须请求存储类，管理员必须已创建并配置该类，以便进行动态配置。

**（2）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

**（3）使用**

a）需要找一个存储服务器，把它划分成多个存储空间；

b）k8s管理员可以把这些存储空间定义成多个pv；

c）在pod中使用pvc类型的存储卷之前需要先创建pvc，通过定义需要使用的pv的大小和对应的访问模式，找到合适的pv；

d）pvc被创建之后，就可以当成存储卷来使用了，我们在定义pod时就可以使用这个pvc的存储卷

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

f）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）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可能被废弃了）**

**Delete**

* **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，管理员能够通过下面的步骤手工回收存储卷：

1）删除PV：在PV被删除后，在外部设施中相关的存储资产仍然还在；

2）手工删除遗留在外部存储中的数据；

3）手工删除存储资产，如果需要重用这些存储资产，则需要创建新的PV。

* **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，例如AWS EBS, 或者Cinder存储卷。

**3.2 创建pv**

**（1）在nfs中导出多个存储目录，在nfs服务器上操作（这里是k8s的master1节点）**

mkdir /data/volume\_test/v{1,2,3,4,5,6,7,8,9,10} -p

cat /etc/exports

/data/volume\_test/v1 192.168.0.0/24(rw,no\_root\_squash)

/data/volume\_test/v2 192.168.0.0/24(rw,no\_root\_squash)

/data/volume\_test/v3 192.168.0.0/24(rw,no\_root\_squash)

/data/volume\_test/v4 192.168.0.0/24(rw,no\_root\_squash)

/data/volume\_test/v5 192.168.0.0/24(rw,no\_root\_squash)

/data/volume\_test/v6 192.168.0.0/24(rw,no\_root\_squash)

/data/volume\_test/v7 192.168.0.0/24(rw,no\_root\_squash)

/data/volume\_test/v8 192.168.0.0/24(rw,no\_root\_squash)

/data/volume\_test/v9 192.168.0.0/24(rw,no\_root\_squash)

/data/volume\_test/v10 192.168.0.0/24(rw,no\_root\_squash)

exportfs -arv 使配置文件生效

service nfs restart

**（2）把上面的存储目录做成pv**

kubectl explain pv 查看pv的创建方法

kubectl explain pv.spec.nfs 查看怎么把nfs定义成pv

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

**（3）创建pv（pv是集群级别的资源，不需要定义namespace）**

**cat pv.yaml**

apiVersion: v1

kind: PersistentVolume

metadata:

name: v1

spec:

capacity:

storage: 1Gi

accessModes: ["ReadWriteOnce"]

nfs:

path: /data/volume\_test/v1

server: master1

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: v2

spec:

capacity:

storage: 2Gi

accessModes: ["ReadWriteMany"]

nfs:

path: /data/volume\_test/v2

server: master1

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: v3

spec:

capacity:

storage: 3Gi

accessModes: ["ReadOnlyMany"]

nfs:

path: /data/volume\_test/v3

server: master1

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: v4

spec:

capacity:

storage: 4Gi

accessModes: ["ReadWriteOnce","ReadWriteMany"]

nfs:

path: /data/volume\_test/v4

server: master1

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: v5

spec:

capacity:

storage: 5Gi

accessModes: ["ReadWriteOnce","ReadWriteMany"]

nfs:

path: /data/volume\_test/v5

server: master1

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: v6

spec:

capacity:

storage: 6Gi

accessModes: ["ReadWriteOnce","ReadWriteMany"]

nfs:

path: /data/volume\_test/v6

server: master1

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: v7

spec:

capacity:

storage: 7Gi

accessModes: ["ReadWriteOnce","ReadWriteMany"]

nfs:

path: /data/volume\_test/v7

server: master1

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: v8

spec:

capacity:

storage: 8Gi

accessModes: ["ReadWriteOnce","ReadWriteMany"]

nfs:

path: /data/volume\_test/v8

server: master1

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: v9

spec:

capacity:

storage: 9Gi

accessModes: ["ReadWriteOnce","ReadWriteMany"]

nfs:

path: /data/volume\_test/v9

server: master1

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: v10

spec:

capacity: #pv的存储空间容量

storage: 10Gi

accessModes: ["ReadWriteOnce","ReadWriteMany"]

nfs:

path: /data/volume\_test/v10 #把nfs的存储空间创建成pv

server: master1

---

kubectl apply -f pv.yaml

kubectl get pv 显示如下：

**（4）创建pvc**

**cat pvc.yaml**

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: my-pvc

spec:

accessModes: ["ReadWriteMany"]

resources:

requests:

storage: 2Gi

**（5）创建pod**

**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: my-pvc

kubectl apply -f pod-pvc.yaml

**注：**

(1)我们每次创建pvc的时候，需要事先有划分好的pv，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.