链滴

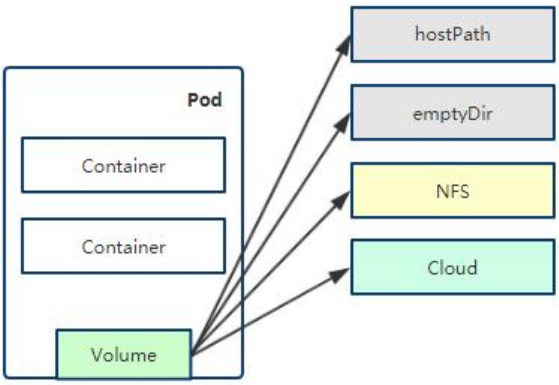
kubernetes 存储（上）

作者：Leif160519
原文链接：https://ld246.com/article/1594471387702
来源网站：链滴

1.为什么需要存储卷

容器部署过程中一般有以下三种数据：

- 启动时需要的初始数据，可以是配置文件
- 启动过程中产生的临时数据，该临时数据需要多个容器间共享
- 启动过程中产生的持久化数据



2.数据卷概述

- Kubernetes中的Volume提供了在容器中 挂载外部存储的能力
- Pod需要设置 卷来源（spec.volume）和挂载点（spec.containers.volumeMounts）两个信息才可以使用相应的Volume

官方给出了k8s支持的所有卷类型，点击查看：Volumes | Kubernetes

Volume 的类型

Kubernetes 支持下列类型的卷：

- awsElasticBlockStore
- azureDisk
- azureFile
- cephfs
- cinder
- configMap
- csi
- downwardAPI
- emptyDir
- fc (fibre channel)
- flexVolume
- flocker
- gcePersistentDisk
- gitRepo (deprecated)
- glusterfs
- hostPath
- iscsi
- local
- nfs
- persistentVolumeClaim
- projected
- portworxVolume
- quobyte
- rbd
- scaleIO
- secret
- storageos
- vsphereVolume

存储分为以下几类：

本地：hostPath，emptyDir

网络：nfs，cephfs，rbd，glusterfs

公有云：awsElasticBlockStore，azureDisk

k8s本身存储：secret，configMap，downwardAPI

3.临时存储卷：emptyDir

创建一个空卷，挂载到Pod中的容器。Pod删除该卷也会被删除。

应用场景：Pod中容器之间数据共享

pod实现：网络共享（infra container），存储共享（emptyDir）

示例程序emptyDir.yaml：

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: write
    image: centos
    command: ["bash","-c","for i in {1..100};do echo $i >> /data/hello;sleep 1;done"]
    volumeMounts:
      - name: data
        mountPath: /data
  - name: read
    image: centos
    command: ["bash","-c","tail -f /data/hello"]
    volumeMounts:
      - name: data
        mountPath: /data

  volumes:
  - name: data
    emptyDir: {}
```

卷挂载是针对容器的，故volumeMounts要与container同级

yaml应用完成后，k8s会在宿主机创建一个空目录，路径在/var/lib/kubelet/pods/<pod-id>/volums/kubernetes.io~empty-dir/data/,所有的文件都会挂载在这个空目录中，下面介绍查找方法：

- 1.查看pod在哪个节点上

```
kubectl get pod -o wide
```

```
NAME                    READY   STATUS    RESTARTS   AGE    IP                NODE        NOMINATED NODE   READINESS GATES
my-pod                  2/2     Running   1          4m5s   10.244.36.116     k8s-node1   <none>           <none>
web-5b9bff6674-8wjlq    1/1     Running   24         28d    10.244.169.191    k8s-node2   <none>           <none>
web-5b9bff6674-99ltd    1/1     Running   24         28d    10.244.169.186    k8s-node2   <none>           <none>
web-5b9bff6674-skgsj    1/1     Running   24         28d    10.244.36.108     k8s-node1   <none>           <none>
```

- 2.去node1执行：

```
docker ps | grep my-pod
```

```
[root@k8s-node1 ~]# docker ps | grep my-pod
a241e62e4090        centos                                                  "bash -c 'for i in {…"   52 seconds ago      Up 52 seconds       k8s_write_my-pod_default_37a7c719-31b6-4a4f-bfa0-acacea1fac22_2
bb3716fd0a8c        centos                                                  "bash -c 'tail -f /d…"   4 minutes ago       Up 4 minutes        k8s_read_my-pod_default_37a7c719-31b6-4a4f-bfa0-acacea1fac22_0
e310e2615fa8        registry.aliyuncs.com/google_containers/pause:3.2       "/pause"                 5 minutes ago       Up 5 minutes        k8s_POD_my-pod_default_37a7c719-31b6-4a4f-bfa0-acacea1fac22_0
```

- 3.找到pod-id：　37a7c719-31b6-4a4f-bfa0
- 4.接着去目录下查看执行结果：

```
cd /var/lib/kubelet/pods/37a7c719-31b6-4a4f-bfa0-acacea1fac22/volumes/kubernetes.io~empty-dir/data
```

可以发现文件夹下产生了一个hello文件.文件内容可以直接cat查看或者通过kubectl查看：kubectl log -f my-pod read

4.节点存储卷：hostPath

挂载Node文件系统上文件或者目录到Pod中的容器。

应用场景：Pod中容器需要访问宿主机文件

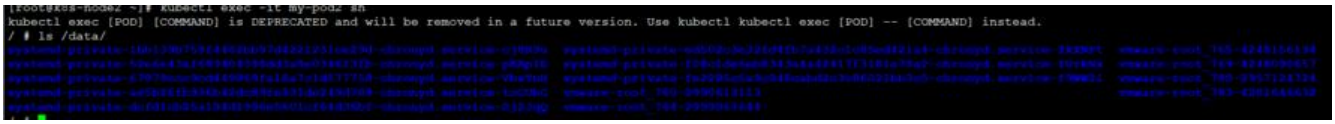
示例:hostPath.yaml

将宿主机/tmp目录挂载到pod的/data目录中

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod2
spec:
  containers:
  - name: busybox
    image: busybox
    args:
    - /bin/sh
    - -c
    - sleep 36000
    volumeMounts:
    - name: data
      mountPath: /data

  volumes:
  - name: data
    hostPath:
      path: /tmp
      type: Directory
```

挂载的目录为当前节点宿主机目录，不是其他节点的也不是远程存储的目录



5. 网络存储卷：NFS

示例程序nfs.yaml：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 3
```

```
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
        - name: wwwroot
          mountPath: /usr/share/nginx/html
        ports:
        - containerPort: 80
      volumes:
      - name: wwwroot
        nfs:
          server: 192.168.0.6
          path: /data/nfs
```

在nfs服务器上新建一个index.html，内容为123：

```
echo 123 > index.html
```

选择一个pod，exec进去看看：

```
[root@k8s-master k8s]# kubectl get pod
NAME                                READY   STATUS             RESTARTS   AGE
my-pod                              1/2     CrashLoopBackOff   15         87m
my-pod2                             1/1     Running            0          70m
nginx-deployment-ff959c8d4-8qbm8    1/1     Running            0          16m
nginx-deployment-ff959c8d4-j7bsj    1/1     Running            0          16m
nginx-deployment-ff959c8d4-wqxd5    1/1     Running            0          16m
web-5b9bff6674-8wjlq                1/1     Running            24         28d
web-5b9bff6674-99ltd                1/1     Running            24         28d
web-5b9bff6674-skgsj                1/1     Running            24         28d
[root@k8s-master k8s]# kubectl exec -it nginx-deployment-ff959c8d4-8qbm8 bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl kubectl exec [POD] -- [COMMAND] instead.
root@nginx-deployment-ff959c8d4-8qbm8:/# cat /usr/share/nginx/html/index.html
123
```

以上说明，nfs挂载没问题！其他两个pod中也会有同样的index.html

访问pod的ip：

```
[root@k8s-master k8s]# kubectl get pod -o wide
NAME                                READY   STATUS             RESTARTS   AGE   IP               NODE        NOMINATED NODE   READINESS GATES
my-pod                              1/2     CrashLoopBackOff   16         91m   10.244.36.116    k8s-node1   <none>           <none>
my-pod2                             1/1     Running            0          74m   10.244.169.190   k8s-node2   <none>           <none>
nginx-deployment-ff959c8d4-8qbm8    1/1     Running            0          20m   10.244.169.130   k8s-node2   <none>           <none>
nginx-deployment-ff959c8d4-j7bsj    1/1     Running            0          20m   10.244.36.113    k8s-node1   <none>           <none>
nginx-deployment-ff959c8d4-wqxd5    1/1     Running            0          20m   10.244.169.136   k8s-node2   <none>           <none>
web-5b9bff6674-8wjlq                1/1     Running            24         28d   10.244.169.191   k8s-node2   <none>           <none>
web-5b9bff6674-99ltd                1/1     Running            24         28d   10.244.169.186   k8s-node2   <none>           <none>
web-5b9bff6674-skgsj                1/1     Running            24         28d   10.244.36.108    k8s-node1   <none>           <none>
[root@k8s-master k8s]# curl 10.244.169.130
123
[root@k8s-master k8s]# curl 10.244.36.113
123
[root@k8s-master k8s]# curl 10.244.169.136
123
```

6.持久卷

持久卷对上述配置磁盘(nfs)的角色进行划分，并且抽象出来两个资源：

- PersistentVolume（PV）：对存储资源创建和使用的抽象，使得存储作为集群中的资源管理

1.静态

2.动态

● PersistentVolumeClaim（PVC）：让用户不需要关心具体的Volume实现细节

解决的痛点：专人做专事，运维只负责提供存储空间，用户不需要关心怎么实现，直接用配置好的存即可。

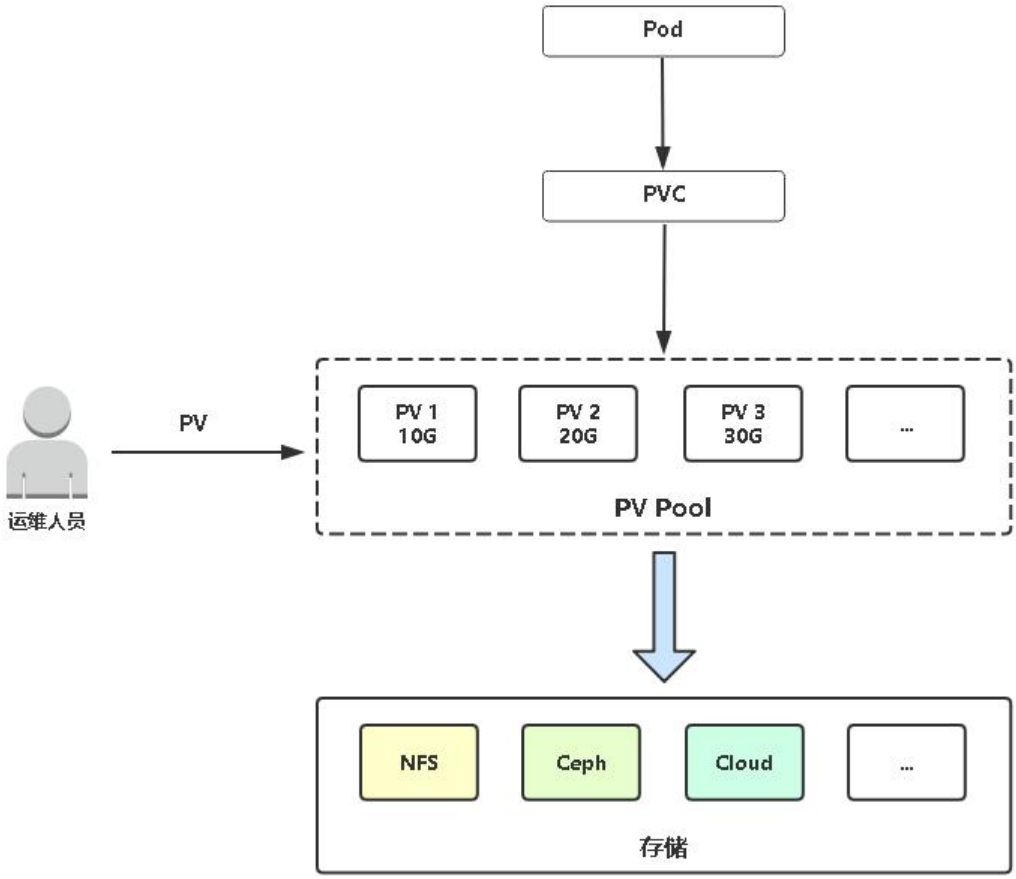
pv与pvc一一对应，用户申请一块容量，会匹配一个pv，该pv会与这个pvc绑定

示例：

作为开发者，需要关注前两个yaml，先写部署的镜像信息，再指定数据卷类型(pvc)（左图），然后指定需要的容量和访问模式(中间图)，它们通过pvc的名称关联；这两块一般写在同一个yaml中（左和中图）；右图一般是运维人员提前准备好的pv，里面包含了卷的来源，访问模式和大小

6.1 PV静态供给

原理图：



解释：中间部分为运维人员准备的一些存储池，存储池中有很多的pv，当开发者在k8s中申请一块存时，执行完yaml之后，pvc会自动查找与其定义的访问模式或者容量相匹配的pv并与之绑定，只有绑了pv成功申请了存储之后，pod才会显示running状态，否则一直显示pending状态

注意pv与pvc一一对应并且绑定

总结：

PVC与PV：

- 一对一的关系
- PV可以是多个后端存储
- PV一般都是k8s运维创建（存储池）

示例

- my-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
```

```
    ports:
    - containerPort: 80
    volumeMounts:
    - name: www
      mountPath: /usr/share/nginx/html
  volumes:
    - name: www
      persistentVolumeClaim:
        claimName: my-pvc

---

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

- my-pvc.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: my-pv
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  nfs:
    path: /data/nfs
    server: 192.168.0.6
```

效果

```
[root@k8s-master pvc]# kubectl get pod
NAME                    READY   STATUS    RESTARTS   AGE
my-pod                  1/1     Running   0          35s
web-5b9bff6674-8wjlq    1/1     Running   24         28d
web-5b9bff6674-99ltd    1/1     Running   24         28d
web-5b9bff6674-skgsj    1/1     Running   24         28d
[root@k8s-master pvc]# kubec my-pget^C
[root@k8s-master pvc]# kubectl get pvc
NAME      STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
my-pvc    Bound    my-pv    5Gi        RWX                           44s
[root@k8s-master pvc]# kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM            STORAGECLASS   REASON   AGE
my-pv   5Gi        RWX            Retain           Bound    default/my-pvc                           3m5s
```

补充：

Kubernetes支持持久卷的存储插件：

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

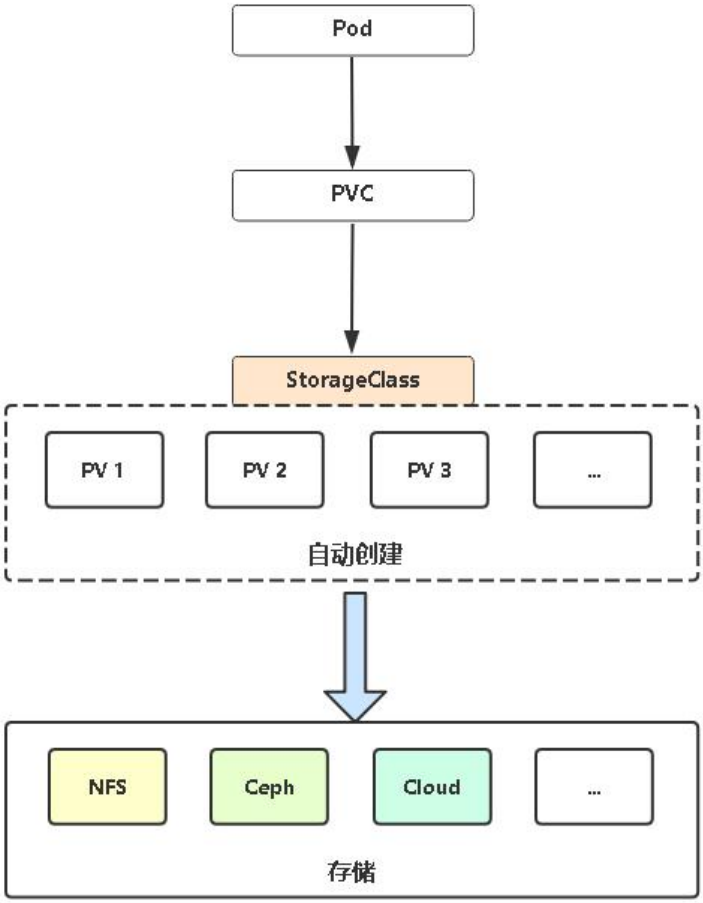
PV与PVC匹配：

1、访问模式（三种访问方式： ReadWriteOnce(RWO)， ReadOnlyMany(ROX)， ReadWriteMany RWX)）

2、存储容量（PVC如果没有找到精确对应的PV时，会根据就近PV匹配）

6.2 PV动态供给

当pvc去申请容量时，它会根据存储类自动创建pv，pvc申请多大空间，pv就会创建多大空间

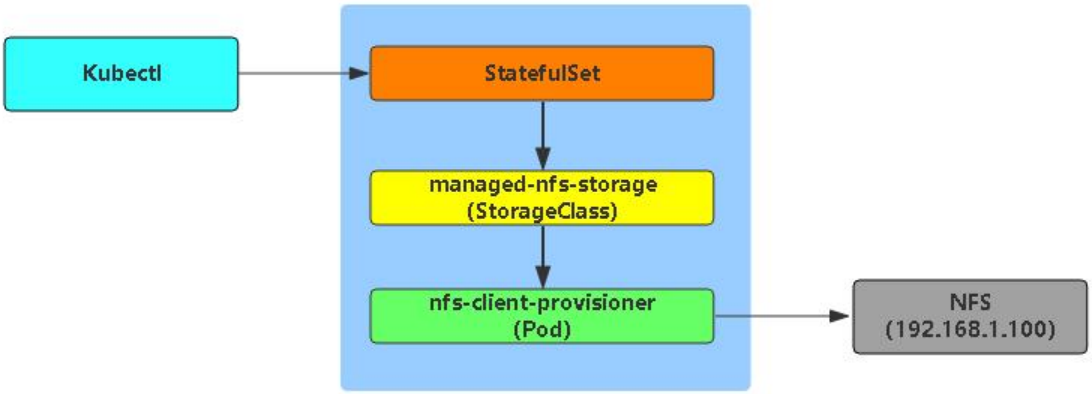


- Dynamic Provisioning机制工作的核心在于StorageClass的API对象。
- StorageClass声明存储插件，用于自动创建PV。

Kubernetes支持动态供给的存储插件：
https://kubernetes.io/docs/concepts/storage/storage-classes/

由此可见，k8s并不支持nfs的pv动态供给，故需要安装插件让其支持，插件地址为：
https://github.com/kubernetes-incubator/external-storage

动态供给-NFS



下载安装存储插件：点击下载

```
[root@k8s-master nfs-client]# ll
总用量 12
-rw-r--r-- 1 root root  225 10月 30 2019 class.yaml
-rw-r--r-- 1 root root  994 10月 30 2019 deployment.yaml
-rw-r--r-- 1 root root 1526 10月 30 2019 rbac.yaml
```

注意：deployment.yml中的nfs地址需要手动更改！！！路径不存在会自动创建。

```
cd nfs-client
kubectl apply -f .
```

```
[root@k8s-master nfs-client]# kubectl get deployment
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
nfs-client-provisioner   1/1     1            1           14m
web                      3/3     3            3           28d
[root@k8s-master nfs-client]# kubectl get deploy
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
nfs-client-provisioner   1/1     1            1           15m
web                      3/3     3            3           28d
[root@k8s-master nfs-client]# kubectl get pod
NAME                                     READY   STATUS    RESTARTS   AGE
my-pod                                   1/1     Running   0          98m
nfs-client-provisioner-65c9f49679-hd9sb  1/1     Running   0          10m
web-5b9bff6674-8wjlq                     1/1     Running   24         28d
web-5b9bff6674-99ltd                     1/1     Running   24         28d
web-5b9bff6674-skgsj                     1/1     Running   24         28d
[root@k8s-master nfs-client]# kubectl get sc
NAME                  PROVISIONER      RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
managed-nfs-storage   fuseim.pri/ifs   Delete          Immediate           false                  15m
```

示例

- my-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
    volumeMounts:
    - name: www
```

```
      mountPath: /usr/share/nginx/html
  volumes:
    - name: www
      persistentVolumeClaim:
        claimName: my-pvc

---

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-pvc
spec:
  storageClassName: "managed-nfs-storage"  # 指定存储类
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

```
[root@k8s-master pvc]# kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM            STORAGECLASS          REASON   AGE
my-pv                                      5Gi        RWX            Retain           Available                                                    113m
pvc-3efdcaa8-7809-40c7-9f18-b1f9856728e8   5Gi        RWX            Delete           Bound       default/my-pvc   managed-nfs-storage           21m
[root@k8s-master pvc]# kubectl get pvc
NAME     STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS          AGE
my-pvc   Bound    pvc-3efdcaa8-7809-40c7-9f18-b1f9856728e8   5Gi        RWX            managed-nfs-storage   113m
```