K8S持久化的两种方法PV与SC

> 2024.5.29日更新
> 本文对K8s的持久化方案进行简单介绍，即PVC、PV、Storageclass。并以NAS配置NFS为例演示如何将pod与SC绑定。
> 环境配置版本：cri-containerd:v1.6.24 ==,kubeadm kubeclt kubelet:v1.28.2,Helm:v3.15.1==
> 配置进行中可能会出现一些问题如，Not superuser、的权限问题，切换到root即可解决。文件权限问题使用 ls -ihl查看，chmod修改，k8s问题请查看日志和其他文档.
> 截至2024年5月29日helm的nfs源https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/可以使用

目录

1K8S持久化方案

2NFS配置（NAS）

3Helm部署nfs

4K8S-NFS使用

1K8S持久化方案

容器中的文件在磁盘上是临时存放的，也有很多场景下应用程序都需要对某些数据进行持久存储，避免在容器奔溃时造成数据丢失。

在kubernetes中，提供了挂载卷（Volume）的能力，卷的类型有很多种，例如还有跟云厂商关联的awsElasticBlockStore、azureDisk、azureFile等，具体可以参考官方文档。
主要的常用卷类型包括：
emptyDir：卷最初是空的，在pod在节点运行时创建，pod删除时数据也会永久删除；
configMap：可以将configMap中的数据作为卷挂在到pod中；
secret：可以将secret中的数据作为卷挂载到pod中；
downwardAPI：将pod的元数据信息注入到pod中；
hostPath：能将主机节点文件系统上的文件或目录挂载到 Pod 中；
nfs：将 NFS (网络文件系统) 挂载到 Pod，可以多挂；
kubernetes的一个重要的基本理念是：**向应用开发者隐藏真实的基础设施，使他们不需要关心基础设施的具体状况信息，并使应用程序可以在不同的云服务商之前进行迁移、切换。**因此，kubernetes提出了PV和PVC的概念，使开发人员可以在创建pod需要使用持久化存储时，就像请求CPU\MEM等资源一样来向kubernetes集群请求持久存储。

1.1PVC

PersistentVolumeClaim (PVC) 是对 PV 的申请 (Claim)。PVC 通常由普通用户创建和维护。需要为 Pod 分配存储资源时，用户可以创建一个 PVC，指明存储资源的容量大小和访问模式（比如只读）等信息，Kubernetes 会查找并提供满足条件的 PV。
有了 PersistentVolumeClaim，用户只需要告诉 Kubernetes 需要什么样的存储资源，而不必关心真正的空间从哪里分配，如何访问等底层细节信息。这些 Storage Provider 的底层信息交给管理员来处理，只有管理员才应该关心创建 PersistentVolume 的细节信息。
例如

```
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-nfs-pvc
spec:
  storageClassName: nfs-sc
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
```

声明了一个名为test-nfs-pvc的pvc使用的是名为nfs-sc的storageclass可读写，其申请了1G的持久化卷。

```
部署pvc
kubectl apply -f test-nfs-pvc.yaml
查看
kubectl get pvc -o wide
```

pod使用方法

```
kind: Pod
apiVersion: v1
metadata:
  name: test-nfs-pod
spec:
  containers:
  - name: test-nfs-pod
    image: busybox:stable
```

```
    command:
      - "/bin/sh"
    args:
      - "-c"
      - "touch /mnt/SUCCESS && sleep 3600"
    volumeMounts:
    - name: nfs-pvc
      mountPath: "/mnt"
  restartPolicy: "Never"
  volumes:
    - name: nfs-pvc
      persistentVolumeClaim:
        claimName: test-nfs-pvc
```

```
部署
kubectl apply -f test-nfs-pod.yaml
进入容器查看卷
kubectl exec test-nfs-pod -n nfs-system -- df -h
```

1.2PV

PersistentVolume (PV) 是外部存储系统中的一块存储空间，由管理员创建和维护。与 Volume 一样，PV 具有持久性，生命周期独立于 Pod。
pv文件示例

```
kind: PersistentVolume
apiVersion: v1
metadata:
        name: pv-volume-consul-0
        namespace: consul
        labels:
                type: local
spec:
        storageClassName: ""
        capacity:
                storage: 10Gi
        accessModes:
                - ReadWriteOnce
        hostPath:
```

```
                path: "/consul/data"
```

pv创建为静态存储卷，需要注意存储容量和存储配置类型需要自己维护。一般使用sc而不是pv。

1.3Storageclass

前面的例子中，我们提前创建了 PV，然后通过 PVC 申请 PV 并在 Pod 中使用，这种方式叫做静态供给（Static Provision）。
与之对应的是动态供给（Dynamical Provision），即如果没有满足 PVC 条件的 PV，会动态创建 PV。相比静态供给，动态供给有明显的优势：不需要提前创建 PV，减少了管理员的工作量，效率高。
动态供给是通过 StorageClass 实现的，StorageClass 定义了如何创建 PV。
在一个大规模的Kubernetes集群里，可能有成千上万个PVC，这就意味着运维人员必须实现创建出这个多个PV，此外，随着项目的需要，会有新的PVC不断被提交，那么运维人员就需要不断的添加新的，满足要求的PV，否则新的Pod就会因为PVC绑定不到PV而导致创建失败。而且通过 PVC 请求到一定的存储空间也很有可能不足以满足应用对于存储设备的各种需求。
而且不同的应用程序对于存储性能的要求可能也不尽相同，比如读写速度、并发性能等，为了解决这一问题，Kubernetes 又为我们引入了一个新的资源对象：StorageClass，通过 StorageClass 的定义，管理员可以将存储资源定义为某种类型的资源，比如快速存储、慢速存储等，用户根据 StorageClass 的描述就可以非常直观的知道各种存储资源的具体特性了，这样就可以根据应用的特性去申请合适的存储资源了。
声明sc的例子，详细介绍参看官网文档或网上资源。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate
```

2NFS配置（NAS）

2.1NFS简介

NFS就是Network File System的缩写，它最大的功能就是可以**通过网络，让不同的机器、不同的操作系统可以共享彼此的文件。**
NFS这个服务器的端口开在2049,但由于文件系统非常复杂。因此NFS还有其他的程序去启动额外的端口，这些额外的用来传输数据的端口是随机选择的，是小于1024的端口；既然是随机的那么客户端又是如何知道NFS服务器端到底使用的是哪个端口呢？这时就需要通过远程过程调用（Remote Procedure Call,RPC）协议来实现了.RPC服务（portmap 或rpcbind服务）

2.2 NAS配置

Nas配置参看[K8S使用群晖DS218+的NFS-腾讯云开发者社区-腾讯云 (tencent.com)]

2.3 NFS 简单使用

注意下面介绍是NAS作为NFS-server，其他linux设备作为client，如果其他设备作为server需要先安装nfs-utils和portmap，作为server的配置不在本文介绍中。
client使用方法

```
sudo mount -t nfs ip:server的挂载目录 client的挂载目录
sudo mount -t nfs 192.168.3.64:/volume4/iiip mnt/nfs
```

随后进入client的mnf/nfs就可以随意创建文件了。

3Helm部署nfs

helm的nfs仓库地址[GitHub - kubernetes-sigs/nfs-subdir-external-provisioner: Dynamic sub-dir volume provisioner on a remote NFS server.]具体value.yaml配置请参看，下面演示安装步骤

```
添加helm仓库
helm repo add nfs-subdir-external-provisioner https://kubernetes-sigs.github.io/nfs-subdir-external-provisioner/
查看可用的nfs包，注意有一些随着时间被遗弃的包是无法使用的
helm search repo nfs
安装nfs，注意nfs需要自己修改。下面会给出一个参看value.yaml
helm install nfs-subdir-external-provisioner nfs-subdir-external-provisioner/nfs-subdir-external-provisioner -f values.yaml
```

参考value.yaml

```
image:
  repository: registry.cn-hangzhou.aliyuncs.com/k-prometheus/nfs  #使用的是实验室阿里云镜像网站
```

```
  tag: v4.0.18
nfs:
  server: 192.168.3.64    #指定 NFS 服务器的地址
  path: /volume4/iiip           #指定 NFS 导出的共享数据目录
storageClass:
  defaultClass: true     #是否设置为默认的 StorageClass，本示例没设置，有需要的可以设置为 true
  name: nfs-sc            #指定 storageClass 的名字
```

查看创建的资源

```
kubectl get sc  -o wide
kubectl get pod -o wide
等待pod running即为部署成功
```

4K8S-NFS使用

4.1pod

```
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: test-nfs-pvc
spec:
  storageClassName: nfs-sc
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 1Gi
---
kind: Pod
apiVersion: v1
metadata:
  name: test-nfs-pod
spec:
  containers:
  - name: test-nfs-pod
```

```
    image: busybox:stable
    command:
      - "/bin/sh"
    args:
      - "-c"
      - "touch /mnt/SUCCESS && sleep 3600"
    volumeMounts:
      - name: nfs-pvc
        mountPath: "/mnt"
  restartPolicy: "Never"
  volumes:
    - name: nfs-pvc
      persistentVolumeClaim:
        claimName: test-nfs-pvc
```

部署此pod即使用的是nas-nfs。登录nas的iip文件即可看到SUCCESS文件

4.2consul的nfs持久化配置

参见注册组件/consul/3.2章