

原创

# Kubernetes核心概念之Volume存储数据卷详解

 奋斗的寒霜

2018-07-02 16:53:1030068人阅读 · 0人评论

在Docker中就有数据卷的概念，当容器删除时，数据也一起会被删除，想要持久化使用数据，需要把主机上的目录挂载到Docker中去，在K8S中，数据卷是通过Pod实现持久化的，如果Pod删除，数据卷也会一起删除，k8s的数据卷是docker数据卷的扩展，K8S适配各种存储系统，包括本地存储EmptyDir,HostPath,网络存储NFS,GlusterFS,PV/PVC等，下面就详细介绍下K8S的存储如何实现。

## 一.本地存储

### 1, EmptyDir

#### ①编辑EmptyDir配置文件

vim emptydir.yaml

```
apiVersion: v1
kind: Pod          #类型是Pod
metadata:
  labels:
    name: redis
    role: master    #定义为主redis
    name: redis-master
spec:
  containers:
    - name: master
      image: redis:latest
      env:           #定义环境变量
        - name: MASTER
          value: "true"
      ports:         #容器内端口
        - containerPort: 6379
      volumeMounts:  #容器内挂载点
        - mountPath: /data
          name: redis-data      #必须有名称
  volumes:
    - name: redis-data      #跟上面的名称对应
      emptyDir: {}          #宿主机挂载点
```



#### ②创建Pod

kubectl create -f emptydir.yaml

```
[root@docker200 volume]# kubectl get pod
NAME      READY   STATUS    RESTARTS   @51CTO博客
redis-master 1/1     Running   1          53s
```

此时Emptydir已经创建成功，在宿主机上的访问路径为/var/lib/kubelet/pods/<pod uid>/volumes/kubernetes.io~empty-dir/redis-data,如果在此目录中创建删除文件，都将对容器中的/data目录有影响，如果删除Pod，文件将全部删除，即使是在宿主机上创建的文件也是如此，在宿主机上删除容器则k8s会再自动创建一个容器，此时文件仍然存在。

### 2.HostDir

在宿主机上指定一个目录，挂载到Pod的容器中，其实跟上面的写法不尽相同，这里只截取不同的部分，当pod删除时，本地仍然保留文件

```
...
volumes:
  - name: redis-data      #跟上面的名称对应
    hostPath:
      path: /data         #宿主机挂载点
```



①编辑一个使用NFS的Pod的配置文件

vim nfs.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: nfs-web
spec:
  containers:
    - name: web
      image: nginx
      imagePullPolicy: Never #如果已经有镜像，就不需要再拉取镜像
      ports:
        - name: web
          containerPort: 80
          hostPort: 80 #将容器的80端口映射到宿主机的80端口
      volumeMounts:
        - name: nfs #指定名称必须与下面一致
          mountPath: "/usr/share/nginx/html" #容器内的挂载点
  volumes:
    - name: nfs #指定名称必须与上面一致
      nfs: #nfs存储
        server: 192.168.66.50 #nfs服务器ip或是域名
        path: "/test" #nfs服务器共享的目录
```

②创建Pod

kubectl create -f nfs.yaml

```
[root@docker200 kubemange]# kubectl get pod
NAME      READY   STATUS    RESTARTS   AGE
nfs-web   1/1     Running   0           1m
```

在节点端可以用mount命令查询挂载情况

```
192.168.66.50:/test on /var/lib/kubelet/pods/e3709a9f-7dcc-11e8-8f6b-000c2901ff20/volumes/kubernetes.io~nfs/nfs-type= nfs
```

因为我映射的是代码目录，在/test目录中创建index.html文件后，这个文件也将在容器中生效，当Pod删除时，文件不受影响，实现了数据持久化。



三.Persistent Volume(PV)和Persistent Volume Claim(PVC)

其实这两种数据卷也属于网络数据卷，单拎出来是因为我觉得这个比前面的数据卷要酷多了，有种大数据，云平台的意思，当用户要使用数据存储的时候他是否还需要知道是什么类型的数据存储，答案是不需要，用户只想要安全可靠的数据存储，而且实现起来很简单，管理员建立一个存储平台，用户按自己的需求消费就可以了，下面就来实现PV/PVC架构。

1.Persistent Volume(PV)

①编辑PV配置文件

vim persistent-volume.yaml

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nfs-pv
  labels:
    type: nfs #指定类型是NFS
spec:
  capacity: #指定访问空间是15G
    storage: 15Gi
  accessModes: #指定访问模式是能在多节点上挂载，并且访问权限是读写执行
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle #指定回收模式是自动回收，当空间被释放时，K8S自动清理
  nfs:
    server: 192.168.66.50
    path: /test
```

②创建PV

kubectl create -f persistent-volume.yaml

```
NAME      CAPACITY  ACCESSMODES  RECLAIMPOLICY  STATUS    CLAIM  REASON  AGE
nfs-pv    15Gi      RWX          Retain         Available             5m
```



```
[root@docker200 volume]# kubectl describe pv
Name:          nfs-pv
Labels:        type=nfs
StorageClass:
Status:        Available
Claim:
Reclaim Policy: Retain
Access Modes:  RWX
Capacity:      15Gi
Message:
Source:
  Type:        NFS (an NFS mount that lasts the lifetime of a pod)
  Server:      192.168.66.50
  Path:        /test
  ReadOnly:    false
No events.
```

状态已经变成可用

2.Persistent Volume Claim(PVC)

①编辑PVC配置文件

vim test-pvc.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: test-pvc
spec:
  accessModes:
    - ReadWriteMany
  resources:      #指定请求的资源，存储3G
    requests:
      storage: 3Gi
```

如果当前有两个PV,一个10G，一个2G，请求资源为3G,那么将直接使用10GPV

②创建PVC

kubectl create -f test-pvc.yaml

```
[root@docker200 volume]# kubectl get pvc
NAME      STATUS   VOLUME   CAPACITY   ACCESSMODES   AGE
test-pvc  Bound    nfs-pv2   3Gi        RWX            1m
```

```
[root@docker200 volume]# kubectl describe pvc
Name:          test-pvc
Namespace:     default
StorageClass:
Status:        Bound
Volume:        nfs-pv2
Labels:        <none>
Capacity:      3Gi
Access Modes:  RWX
No events.
```



因为我之前又创建了一个3G可回收的PV，所以自动选择这个卷了，在PVC选择PV后，不管PV有多少空间都会直接占满所有虚拟空间，实际使用则由Pod来完成

NAME	CAPACITY	ACCESSMODES	RECLAIMPOLICY	STATUS	CLAIM	REASON	AGE
nfs-pv	15Gi	RWX	Retain	Available			13m
nfs-pv2	3Gi	RWX	Recycle	Bound	default/test-pvc		8m

3.创建Pod以使用平台空间

vim pv-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: redis111
  labels:
    app: redis111
spec:
  containers:
    - name: redis
      image: redis
      imagePullPolicy: Never
      volumeMounts:
        - mountPath: "/data"
          name: data
      ports:
        - containerPort: 6379
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: test-pvc
```



当前Pod可用空间为3G，如果超过3G，则需要再创建存储来满足需求，因为是网络数据卷，如果需要扩展空间，直接删除Pod再建立一个即可。

© 著作权归作者所有：来自51CTO博客作者奋斗的寒霜的原创作品，如需转载，请注明出处，否则将追究法律责任


k8s docker volume

虚拟化技术

1

收藏 分享

上一篇：Zabbix利用JMX监控多实例... 下一篇：MMM架构实现MySQL高可用读...



奋斗的寒霜

25篇文章, 38W+人气, 0粉丝


死灰复燃，坚持到底



提问和评论都可以，用心的回复会被更多人看到和认可

Ctrl+Enter 发布 取消 发布

推荐专栏

- 


基于Python的  
**DevOps**  
实战  
运维开发全流程

**基于Python的DevOps实战**

自动化运维开发新概念

共20章 | 抚琴煮酒

¥ 51.00 289人订阅

订 阅
- 


最近更新  
**全局视角**  
看大型园区网

**全局视角看大型园区网**

路由交换+安全+无线+优化+运维

共40章 | 51CTO夏杰

¥ 51.00 767人订阅

订 阅
- 


最近更新  
**网工2.0晋级**  
零基础入门Python/Ansible

**网工2.0晋级攻略——零基础入门Python/Ansible**

网络工程师2.0进阶指南

共30章 | 姜汁啤酒

¥ 51.00 1161人订阅

订 阅
- 


高并发架构之路  
**负载均衡**  
高手炼成记

**负载均衡高手炼成记**

高并发架构之路

共15章 | sery

¥ 51.00 396人订阅

订 阅
- 

**带你玩转高可用**

**带你玩转高可用**

前百度高级工程师的架构高可用实战

共15章 | 曹林华

¥ 51.00 388人订阅

订 阅



猜你喜欢 ☆ 1

奋斗的寒霜

- MMM架构实现MySQL高可用读写分离（进阶版,包含Am...
- GitLab在docker和Kubernetes之间折腾
- ELK收集Apache的json格式访问日志并按状态码绘制图表
- gitlab服务器
- Zabbix利用JMX监控多实例Tomcat运行状态
- Zabbix4.0历史数据的持久化
- 基于Docker-compose部署wiki-confluence6.10
- jenkins部署三种构建方式的详细步骤

