Kubernetes存储

前言

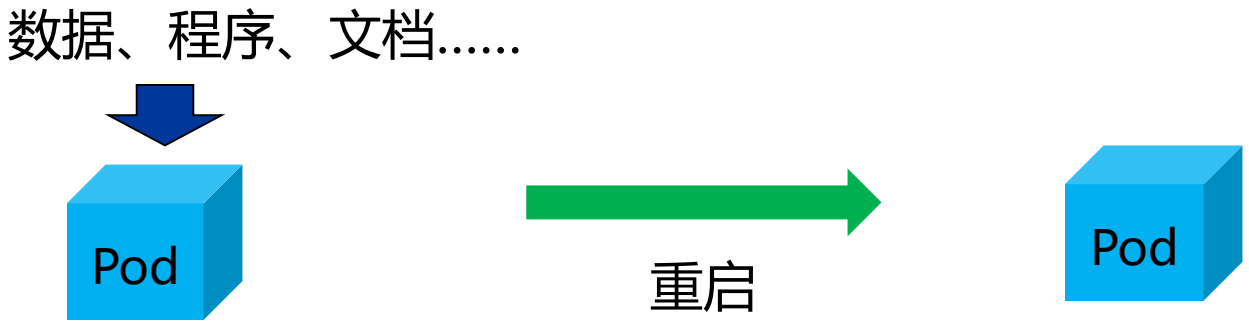
- 本章节介绍了在Kubernetes中使用存储的多种方式，包括：
  - emptyDir
  - HostPath
  - PV和PVC

目标

- 学完本课程后，您将能够：
  - 区分不同volume使用方式
  - 使用emptyDir
  - 使用HostPath
  - 使用PV和PVC为Pod提供存储

目录

Volume



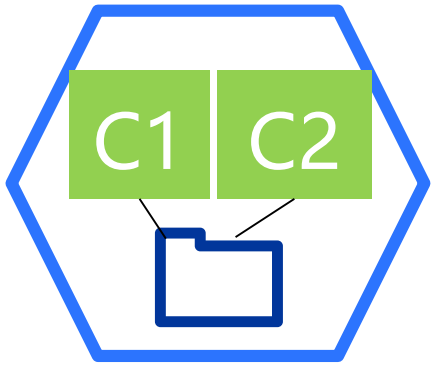
- 在一个Pod使用过程中，会进行很多操作，如修改文件、安装程序等。但当重启容器之后，会发现容器往往又回到了初始的状态，所有的修改都丢失了。
- 除了希望数据不在Pod重启后丢失，有时候也需要在Pod间共享文件。
- 因此，Kubernetes抽象出了Volume对象来解决这两个问题。

Volume类型

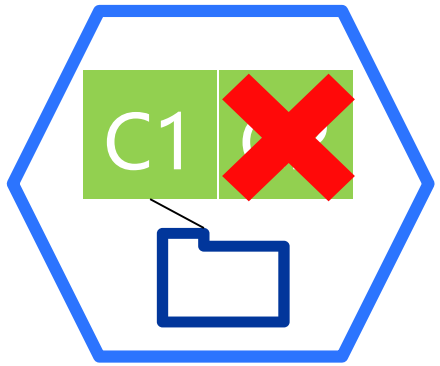
- Kubernetes支持的卷类型非常丰富，包括：
  - NFS文件系统。
  - Cephfs等分布式存储系统。
  - awsElasticBlockStore，azureDisk等公有云存储服务。
  - emptyDir，configMap，hostPath等Kubernetes内置存储类型。
  - ISCSI，FC等等……

EmptyDir

- 当Pod指定到某个节点上时，首先创建的是一个emptyDir卷，并且只要Pod在该节点上运行，卷就一直存在。就像它的名称表示的那样，卷最初是空的。尽管Pod中的容器挂载emptyDir卷的路径可能相同也可能不同，但是这些容器都可以读写emptyDir卷中相同的文件。当Pod因为某些原因被从节点上删除时，emptyDir卷中的数据也会永久删除。



创建时挂载emptyDir



容器故障不影响数据

创建一个使用EmptyDir的Pod

- 创建使用emptyDir时需要配置两个参数：
  - Spec.containers.volumeMounts: 设置volume的挂载点。
  - Spec.volumes: 配置volume。
- 如果保持emptyDir的默认配置，格式如右。如果限制emptyDir容量（如1G），需配置如下：

```
volumes:
  - name: cache-volume
    emptyDir:
      sizeLimit: 1Gi
```

```
apiVersion: v1
kind: Pod
metadata:
  name: em
spec:
  containers:
  - image: ubuntu
    name: test-container
    volumeMounts:
    - mountPath: /cache
      name: cache-volume
    args:
    - /bin/sh
    - -c
    - sleep 30000
  volumes:
  - name: cache-volume
    emptyDir: {}
```

查看EmptyDir对应目录

- 使用docker inspect可以查看到对应容器的挂载点：

```
 "Mounts": [
                {"Type": "bind",
                 "Source": "/var/lib/kubelet/pods/9b94747e-9628-11e9-ac91-
000c290a92ce/volumes/kubernetes.io~empty-dir/cache-volume",
                 "Destination": "/cache",
                 "Mode": "Z",
                 "RW": true,
                 "Propagation": "rprivate"}
```

- 进入对应目录后，可以发现如果在pod中创建一个文件，在这个目录中也可呈现。
  - Pod中创建一个名为testfile的文件 `# touch testfile`
  - 在该文件夹内查看：

```
[root@k8s-node1 cache-volume]# ls
testfile
```

- emptyDir的生命周期与Pod一致，如果将pod删除，可以看到对应的目录也不复存在。

EmptyDir容量限制

- emptyDir可以进行容量限制，如限制为1G。
- 进入Pod查看分配文件夹的大小，可以看到空间是Host存储空间大小，并非1G。

```
# df -h
Filesystem                Size  Used Avail Use% Mounted on
overlay                    44G  4.3G   40G  10% /
tmpfs                      64M     0   64M   0% /dev
tmpfs                     3.9G     0  3.9G   0% /sys/fs/cgroup
/dev/mapper/centos-root    44G  4.3G   40G  10% /cache
```

- 尝试在容器内写入一个2G的文件

```
# dd if=/dev/zero of=/cache/test2g bs=1M count=2048
```

- 再次查看容器状态发现进入Evicted状态，无法使用。

```
[root@k8s-master probe]# kubectl get pod
NAME      READY   STATUS    RESTARTS   AGE
test-pd   0/1     Evicted   0          10m
```

目录

HUAWEI

HostPath

- hostPath卷能将主机节点文件系统上的文件或目录挂载到Pod中。
- 比如希望Pod使用一些docker引擎或系统已经包含的内部程序的时候，会使用到这种方式。如，以下为kube-proxy中配置的hostPath。

```
 volumes:
- hostPath:
      path: /run/xtables.lock
      type: FileOrCreate
    name: xtables-lock
  - hostPath:
      path: /lib/modules
      type: ""
    name: lib-modules
```

创建使用hostPath的Pod

- hostPath配置项与emptyDir类似，但类型需指定为① hostPath。
  - ② path参数需要配置为主机上已存在的目录。
  - ③ type指定为目录，本实验中挂载给pod的是一个文件夹。

```
apiVersion: v1
kind: Pod
metadata:
  name: hppod
spec:
  containers:
  - image: ubuntu
    name: hp-container
    volumeMounts:
    - mountPath: /hp-dir
      name: hp-volume
    args:
    - /bin/sh
    - -c
    - sleep 30000
  volumes:
  - name: hp-volume
  ① hostPath:
   ②  path: /root/runfile/hostpathdir
    ③ type: Directory
```

HostPath的类型

- 创建hostPath时，需要指定类型（type），可用的type如右表。
- 如果选择类型不正确，或主机上不存在对应资源（如不存在指定文件夹），Kubernetes系统将无法继续创建Pod，创建步骤终止。Pod状态长时间处于ContainerCreating中。

取值	行为
	空字符串（默认）用于向后兼容，这意味着在安装hostPath卷之前不会执行任何检查。
DirectoryOrCreate	如果在给定路径上什么都不存在，那么将根据需要创建空目录，权限设置为0755，具有与Kubelet相同的组和所有权。
Directory	在给定路径上必须存在的目录。
FileOrCreate	如果在给定路径上什么都不存在，那么将在那里根据需要创建空文件，权限设置为0644，具有与Kubelet相同的组和所有权。
File	在给定路径上必须存在的文件。
Socket	在给定路径上必须存在的UNIX套接字。
CharDevice	在给定路径上必须存在的字符设备。
BlockDevice	在给定路径上必须存在的块设备。

目录

PV和PVC概述

- PersistentVolume（pv）和PersistentVolumeClaim（pvc）是Kubernetes提供的两种API资源，用于抽象存储细节。管理员关注于如何通过pv提供存储功能而无需关注用户如何使用，同样的用户只需要挂载pvc到容器中而不需要关注存储卷采用何种技术实现。
  - PV是集群中由管理员配置的一块存储空间。它是集群中的资源，就像节点是集群中的资源一样。PV是卷插件，和之前介绍的volumes类似，但它有一个独立于单个Pod的生命周期。PV的后端可以是NFS，iSCSI或者云存储等。
  - PVC是用户的存储请求。它类似于Pod：Pod消耗节点资源，而PVC消耗PV资源。Pod可以请求特定级别的资源（CPU和内存），PVC可以请求PV特定的接入模式（读写等）和大小。

创建PV (1)

- ① kind选择PersistentVolume。
- ② name命名PV，在PVC中可调用。
- ③ capacity指定PV的容量。
- ④ accessModes指定访问模式。
  - **ReadWriteOnce**：该卷能够以读写模式被加载到一个节点上。
  - **ReadOnlyMany**：该卷能够以只读模式加载到多个节点上。
  - **ReadWriteMany**：该卷能够以读写模式被多个节点同时加载。

```
apiVersion: v1
kind: PersistentVolume ①
metadata:
  name: mypv ②
spec:
  capacity: ③
    storage: 1Gi
  accessModes: ④
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle ⑤
  nfs:
    path: /nfs
    server: 192.168.137.21 ⑥
```

创建PV (2)

- ⑤ persistentVolumeReclaimPolicy指定PV的回收策略
  - Retain（保留），不删除，需要手动回收
  - Recycle（回收），基本擦除，类似rm –rf，使其可供其他PVC申请。
  - Delete（删除），关联存储将被删除，如Azure Disk或OpenStack Cinder卷
- ⑥ nfs字段配置NFS服务器信息，在创建PV前，已在k8s-master节点上搭建完NFS服务器，服务器的IP地址是192.168.137.21，共享的文件夹是/nfs。
  - PV支持的挂载选项包括NFS，iSCSI，Cinder卷，CephFS等。

```
[root@k8s-master volume]# kubectl get pv
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
mypv    1Gi        RWO            Recycle          Available                                   41m
```

创建PVC

- ① Kind，类型指定为PVC
- ② accessModes，保持与PV一致。
- ③ volumeName，使用的PV名称，用于PVC找到正确的PV。
- ④ requests：指定PV的容量，如果不存在满足该容量需求的PV，则PVC无法绑定任何PV。
- ⑤ 创建Pod时，可以使用该方式定义volumes，使用PV和PVC。

```
apiVersion: v1
kind: PersistentVolumeClaim ①
metadata:
  name: mypvc
spec:
  accessModes:          ②
    - ReadWriteOnce
  volumeName: mypv      ③
  resources:
    requests:
      storage: 1Gi      ④
```

```
volumes:                                    ⑤
 - name: pvc-volume
   persistentVolumeClaim:
     claimName: mypvc
```

PV与PVC的状态

- PV状态，创建完成后为Available。

- PVC状态，创建完成后为Pending。

PV回收

- 由于创建时选择的回收策略是Recycle，删除PVC的时候Kubernetes会删除原有PV的数据。它采用的方式是创建一个回收专用Pod来完成这一操作。

```
[root@k8s-master volume]# kubectl delete pvc mypvc && kubectl get pod
persistentvolumeclaim "mypvc" deleted
NAME                  READY   STATUS              RESTARTS   AGE
recycler-for-mypv     0/1     ContainerCreating   0          0s
```

- 如果不希望数据被删除，可以配置回收策略为Retain，这样在删除PVC后，PV的数据仍然存在，PV状态如下：

```
NAME    CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS     CLAIM              REASON   AGE
mypv    1Gi        RWO            Retain           Released   default/mypvc              19s
```

- 数据未被删除，但由于PV处于Released状态，依然无法直接被PVC使用。这是由于PV保存了之前关联的PVC状态，如需关联新PVC，需要删除其中ClaimRef参数。

PV和PVC的绑定

- 如果我们重复前文的PV和PVC实验，将PVC中volumeName：mypv这一参数删除，可以发现PVC仍能申请到PV。那么PVC是以什么机制找到匹配的PV呢？
  - PVC首先根据筛选条件，如容量大小和访问模式筛选掉不符合条件的PV。
  - 筛选掉不符合volumeName的PV。在前文的实验实例中我们使用的是这个方式。
  - 筛选掉不符合StorageClass的PV。
  - 根据其他条件筛选符合的PV。

在创建PV和PVC时使用StorageClass

- 创建PV时指定storageClassName，可以在PVC中申请该PV。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mypv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: nfs
  nfs:
    path: /nfs
    server: 192.168.137.21
```

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mypvc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: nfs
  resources:
    requests:
      storage: 1Gi
```

动态卷供给

- storageClass除了上述用法之外，主要使用场景是动态供给PV。
- 由于Kubernetes集群中存在大量的Pod，也就意味着很可能有大量的PV和PVC，如果需要一个一个手工创建无疑是一个巨大的工程，也不符合自动化的特点。
- 以nfs为例，使用动态供给功能需要完成以下几个步骤：
  - 创建nfs-provisioner，provisioner用于动态创建符合要求的PV。
  - 创建StorageClass，在配置时指定使用的provisioner。
  - 创建PVC，只需要指定StorageClass，容量及访问模式即可。
- 如果将一个StorageClass标注为default，则PVC在申请时可以不指定StorageClass而默认使用该defaultStorageClass。

实验&实训任务

- 实验任务
  - 请按照实验手册2.10章节完成Kubernetes存储实验，包括：
    - 创建emptyDir
    - 使用emptyDir容量限制功能
    - 使用hostPath
    - 使用PV和PVC
    - 使用StorageClass方式关联PV和PVC
- 实训任务
  - 请灵活使用本章节课程及实验手册中学到的知识，按照实验手册的2.10.6章节完成Kubernetes存储实训任务。

本章总结

- 本章节介绍了实现Pod卷供给的多种方式，包括：
  - emptyDir
  - hostPath
  - PV和PVC


谢 谢
www.huawei.com