## Volumes

Container（容器）中的磁盘文件是短暂的，当容器崩溃时，kubelet会重新启动容器，但最初的文件将丢失，Container会以最干净的状态启动。另外，当一个Pod运行多个Container时，各个容器可能需要共享一些文件。Kubernetes Volume可以解决这两个问题。

### 卷的类型

Kubernetes支持的卷的类型有很多，以下为常用的卷。

#### ConfigMap

ConfigMap卷也可以作为volume使用，存储在ConfigMap中的数据可以通过ConfigMap类型的卷挂载到Pod中，然后使用该ConfigMap中的数据。引用ConfigMap对象时，只需要在volume中引用ConfigMap的名称即可，同时也可以自定义ConfigMap的挂载路径。

例如，将名称为log-config的ConfigMap挂载到Pod的/etc/config目录下，挂载的文件名称为path指定的值，当前为log\_level：

apiVersion: v1

kind: Pod

metadata:

name: configmap-pod

spec:

containers:

- name: test

image: busybox

volumeMounts:

- name: config-vol

mountPath: /etc/config

volumes:

- name: config-vol

configMap:

name: log-config

items:

- key: log\_level

path: log\_level

注 意

ConfigMap需要提前创建。

#### emptyDir

和上述volume不同的是，如果删除Pod，emptyDir卷中的数据也将被删除，一般emptyDir卷用于Pod中的不同Container共享数据。它可以被挂载到相同或不同的路径上。

默认情况下，emptyDir卷支持节点上的任何介质，可能是SSD、磁盘或网络存储，具体取决于自身的环境。可以将emptyDir.medium字段设置为Memory，让Kubernetes使用tmpfs（内存支持的文件系统），虽然tmpfs非常快，但是tmpfs在节点重启时，数据同样会被清除，并且设置的大小会被计入到Container的内存限制当中。

使用emptyDir卷的示例，直接指定emptyDir为{}即可：

apiVersion: v1

kind: Pod

metadata:

name: test-pd

spec:

containers:

- image: k8s.gcr.io/test-webserver

name: test-container

volumeMounts:

- mountPath: /cache

name: cache-volume

volumes:

- name: cache-volume

emptyDir: {}

#### hostPath

hostPath卷可将节点上的文件或目录挂载到Pod上，用于Pod自定义日志输出或访问Docker内部的容器等。

使用hostPath卷的示例。将主机的/data目录挂载到Pod的/test-pd目录：

apiVersion: v1

kind: Pod

metadata:

name: test-pd

spec:

containers:

- image: k8s.gcr.io/test-webserver

name: test-container

volumeMounts:

- mountPath: /test-pd

name: test-volume

volumes:

- name: test-volume

hostPath:

# directory location on host

path: /data

# this field is optional

type: Directory

hostPath卷常用的type（类型）如下：

* type为空字符串：默认选项，意味着挂载hostPath卷之前不会执行任何检查。
* DirectoryOrCreate：如果给定的path不存在任何东西，那么将根据需要创建一个权限为0755的空目录，和Kubelet具有相同的组和权限。
* Directory：目录必须存在于给定的路径下。
* FileOrCreate：如果给定的路径不存储任何内容，则会根据需要创建一个空文件，权限设置为0644，和Kubelet具有相同的组和所有权。
* File：文件，必须存在于给定路径中。
* Socket：UNIX套接字，必须存在于给定路径中。
* CharDevice：字符设备，必须存在于给定路径中。
* BlockDevice：块设备，必须存在于给定路径中。

#### NFS

NFS卷也是一种网络文件系统，同时也可以作为动态存储，和GFS类似，删除Pod时，NFS中的数据不会被删除。NFS可以被多个写入同时挂载。

#### persistentVolumeClaim

persistentVolumeClaim卷用于将PersistentVolume（持久化卷）挂载到容器中，PersistentVolume分为动态存储和静态存储，静态存储的PersistentVolume需要手动提前创建PV，动态存储无需手动创建PV。

#### Secret

Secret卷和ConfigMap卷类似，见上述。

#### SubPath

有时可能需要将一个卷挂载到不同的子目录，此时使用volumeMounts.subPath可以实现不同子目录的挂载。

本示例为一个LAMP共享一个卷，使用subPath卷挂载不同的目录：

apiVersion: v1

kind: Pod

metadata:

name: my-lamp-site

spec:

containers:

- name: mysql

image: mysql

env:

- name: MYSQL\_ROOT\_PASSWORD

value: "rootpasswd"

volumeMounts:

- mountPath: /var/lib/mysql

name: site-data

subPath: mysql

- name: php

image: php:7.0-apache

volumeMounts:

- mountPath: /var/www/html

name: site-data

subPath: html

volumes:

- name: site-data

persistentVolumeClaim:

claimName: my-lamp-site-data

更多volume可参考：

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

## PV&PVC

PersistentVolume（简称PV）是由管理员设置的存储，它同样是集群中的一类资源，PV是容量插件，如Volumes（卷），但其生命周期独立使用PV的任何Pod，PV的创建可使用NFS、iSCSI、GFS、CEPH等。

PersistentVolumeClaim（简称PVC）是用户对存储的请求，类似于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可以请求特定的大小和访问模式。例如，可以以一次读/写或只读多次的模式挂载。

虽然PVC允许用户使用抽象存储资源，但是用户可能需要具有不同性质的PV来解决不同的问题，比如使用SSD硬盘来提高性能。所以集群管理员需要能够提供各种PV，而不仅是大小和访问模式，并且无须让用户了解这些卷的实现方式，对于这些需求可以使用StorageClass资源实现。

目前PV的提供方式有两种：静态或动态。

静态PV由管理员提前创建，动态PV无需提前创建，只需指定PVC的StorageClasse即可。

（1）回收策略

当用户使用完卷时，可以从API中删除PVC对象，从而允许回收资源。回收策略会告诉PV如何处理该卷，目前卷可以保留、回收或删除。

* Retain：保留，该策略允许手动回收资源，当删除PVC时，PV仍然存在，volume被视为已释放，管理员可以手动回收卷。
* Recycle：回收，如果volume插件支持，Recycle策略会对卷执行rm -rf清理该PV，并使其可用于下一个新的PVC，但是本策略已弃用，建议使用动态配置。
* Delete：删除，如果volume插件支持，删除PVC时会同时删除PV，动态卷默认为Delete。

（2）创建PV

在使用持久化时，需要先创建PV，然后再创建PVC，PVC会和匹配的PV进行绑定，然后Pod即可使用该存储。

创建一个基于NFS的PV：

apiVersion: v1

kind: PersistentVolume

metadata:

name: pv0003

spec:

capacity:

storage: 5Gi

volumeMode: Filesystem

accessModes:

- ReadWriteOnce

persistentVolumeReclaimPolicy: Recycle

storageClassName: slow

mountOptions:

- hard

- nfsvers=4.1

nfs:

path: /tmp

server: 172.17.0.2

说明：

* capacity：容量。
* accessModes：访问模式。包括以下3种：
* ReadWriteOnce：可以被单节点以读写模式挂载，命令行中可以被缩写为RWO。
* ReadOnlyMany：可以被多个节点以只读模式挂载，命令行中可以被缩写为ROX。
* ReadWriteMany：可以被多个节点以读写模式挂载，命令行中可以被缩写为RWX。
* storageClassName：PV的类，一个特定类型的PV只能绑定到特定类别的PVC。
* persistentVolumeReclaimPolicy：回收策略。
* mountOptions：非必须，新版本中已弃用。
* nfs：NFS服务配置。包括以下两个选项：
* path：NFS上的目录
* server：NFS的IP地址

创建的PV会有以下几种状态：

* Available（可用），没有被PVC绑定的空间资源。
* Bound（已绑定），已经被PVC绑定。
* Released（已释放），PVC被删除，但是资源还未被重新使用。
* Failed（失败），自动回收失败。

可以创建一个基于hostPath的PV：

kind: PersistentVolume

apiVersion: v1

metadata:

name: task-pv-volume

labels:

type: local

spec:

storageClassName: manual

capacity:

storage: 10Gi

accessModes:

- ReadWriteOnce

hostPath:

path: "/mnt/data"

（3）创建PVC

创建PVC需要注意的是，各个方面都符合要求PVC才能和PV进行绑定，比如accessModes、storageClassName、volumeMode都需要相同才能进行绑定。

创建PVC的示例如下：

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: myclaim

spec:

accessModes:

- ReadWriteOnce

volumeMode: Filesystem

resources:

requests:

storage: 8Gi

storageClassName: slow

selector:

matchLabels:

release: "stable"

matchExpressions:

- {key: environment, operator: In, values: [dev]}

比如上述基于hostPath的PV可以使用以下PVC进行绑定，storage可以比PV小：

kind: PersistentVolumeClaim

apiVersion: v1

metadata:

name: task-pv-claim

spec:

storageClassName: manual

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 3Gi

然后创建一个Pod指定volumes即可使用这个PV：

kind: Pod

apiVersion: v1

metadata:

name: task-pv-pod

spec:

volumes:

- name: task-pv-storage

persistentVolumeClaim:

claimName: task-pv-claim

containers:

- name: task-pv-container

image: nginx

ports:

- containerPort: 80

name: "http-server"

volumeMounts:

- mountPath: "/usr/share/nginx/html"

name: task-pv-storage

注 意

claimName需要和上述定义的PVC名称task-pv-claim一致。