

**pv&pvc&statefulSet**

**一、k8s-volumes**

**1、什么要用volumes?**

k8s 中容器中的磁盘的生命周期是短暂的, 这就带来了一些列的问题

1. 当一个容器损坏之后, kubelet会重启这个容器, 但是容器中的文件将丢失----容器以干净的状态 重新启动
2. 当很多容器运行在同一个pod中时, 很多时候需要数据文件的共享
3. 在 k8s 中，由于 pod 分布在各个不同的节点之上，并不能实现不同节点之间持久性数据的共 享，并且，在节点故障时，可能会导致数据的永久性丢失。

**volumes就是用来解决以上问题的**

Volume 的生命周期独立于容器，Pod 中的容器可能被销毁和重建，但 Volume 会被保留。

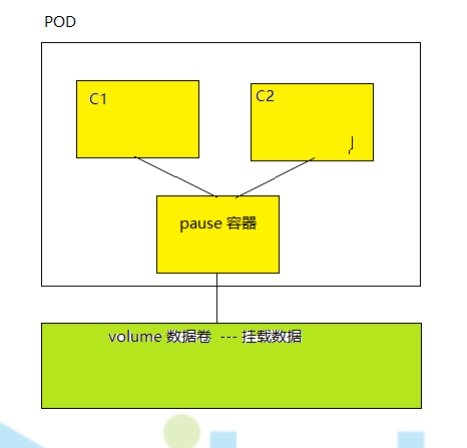
注意：docker磁盘映射的数据将会被保留,和kubernetes有一些不一样

**2、什么是volume？**

volume用来对容器的数据进行挂载，存储容器运行时所需的一些数据。当容器被重新创建时，实 际上我们发现volume挂载卷并没有发生变化。

kubernetes中的卷有明确的寿命————与封装它的pod相同。所以，卷的生命比pod中的所有容 器都长，当这个容器重启时数据仍然得以保存。

当然，当pod不再存在时，卷也不复存在，也许更重要的是kubernetes支持多种类型的卷，pod可 以同时使用任意数量的卷



**3、卷的类型**

kubenetes卷的类型：

**第一种就是本地卷**

像hostPath类型与docker里面的bind mount类型，就是直接挂载到宿主机文件的类型 像 emptyDir是这样本地卷，也就是类似于volume类型 这两点都是绑定node节点的

**第二种就是网络数据卷**

比如Nfs、ClusterFs、Ceph，这些都是外部的存储都可以挂载到k8s上

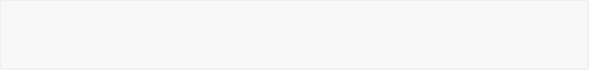
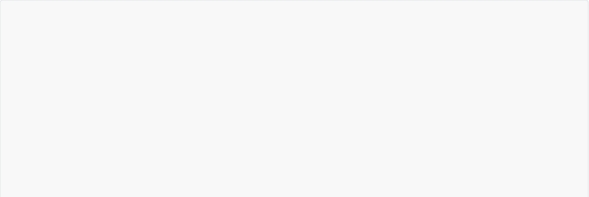
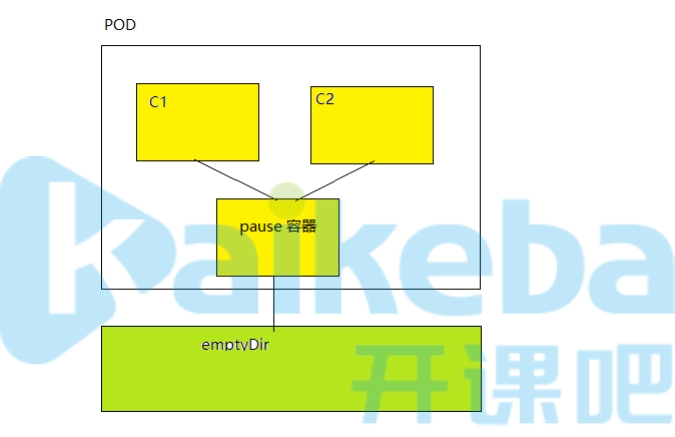
**第三种就是云盘**

比如AWS、微软(azuredisk)

**第四种就是k8s自身的资源**

比如secret、conﬁgmap、downwardAPI

**4、emptyDir**



1. 暂存空间，例如用于基于磁盘的合并排序
2. 用作长时间计算崩溃恢复时候的检查点
3. web服务器容器提供数据时，保存内容管理器容器提取的文件

先来看一下本地卷 像emptyDir类似与docker的volume，而docker删除容器，数据卷还会存在， 而emptyDir删除容器，数据卷也会丢失，一般这个只做临时数据卷来使用

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

当pod被分配给节点时，首先创建emptyDir卷，并且只要该pod在该节点上运行，该卷就会存在。 正如卷的名字所述，它最初是空的，pod中的容器可以读取和写入emptyDir卷中的相同文件，尽管该卷 可以挂载到每个容器中的相同或者不同路径上。当处于任何原因从节点删除pod时，emptyDir中的数据 将被永久删除

注意：容器崩溃不会从节点中移除pod，因此emptyDir卷中的数据在容器崩溃时是安全的

emptyDir的用法

**5、一个例子**

apiVersion: v1

kind: Pod

metadata:

name: test-pod

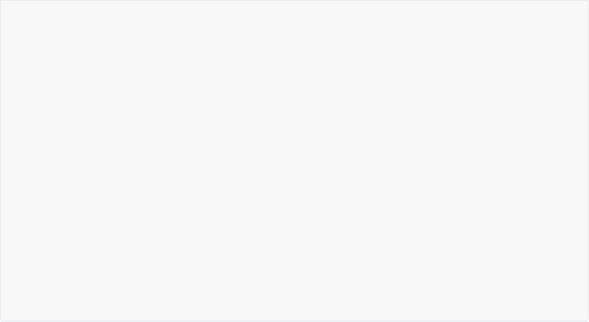
spec:

containers :

- image: hub.kaikeba.com/library/myapp :v1   name: test-container

  volumeMounts :

  - mountPath: /cache



    name: cache-volume

volumes:

- name: cache-volume

  emptyDir: {}

---

apiVersion: v1

kind: Pod

metadata:

name: test-pod

spec:

containers :

- image: hub.kaikeba.com/library/myapp :v1   name: test-container

  volumeMounts :

  - mountPath: /cache

    name: cache-volume

- name: test-1

  image: hub.kaikeba.com/library/busybox :v1   command: ["/bin/sh","-c","sleep 6000s"]   imagePullPolicy : IfNotPresent

  volumeMounts :

  - mountPath: /cache

    name: cache-volume

volumes:

- name: cache-volume

  emptyDir: {}

**6、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 ;

**7、一个例子**

apiVersion: v1

kind: Pod

metadata:

name: test-pod

spec:

containers :

- image: hub.kaikeba.com/library/myapp :v1   name: test-container

  volumeMounts :

  - mountPath: /cache

    name: cache-volume

volumes:

- name: cache-volume

  hostPath:

    path: /data

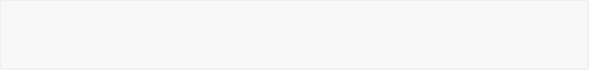
    type: Directory



[root@nfs ~]# systemctl start nfs



yum install nfs-utils -y



mkdir /opt/k8s

vim /etc/exports

/opt/k8s 192.168.30.0/24(rw,no\_root\_squash)

这里创建的数据和我们被分配的node节点的数据都是一样的，创建的数据都会更新上去，删除容 器，不会删除数据卷的数据。

**type类型**

除了所需的path属性职位，用户还可以为hostPath卷指定type.

**8、NFS网络存储**

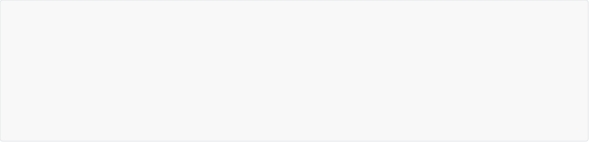
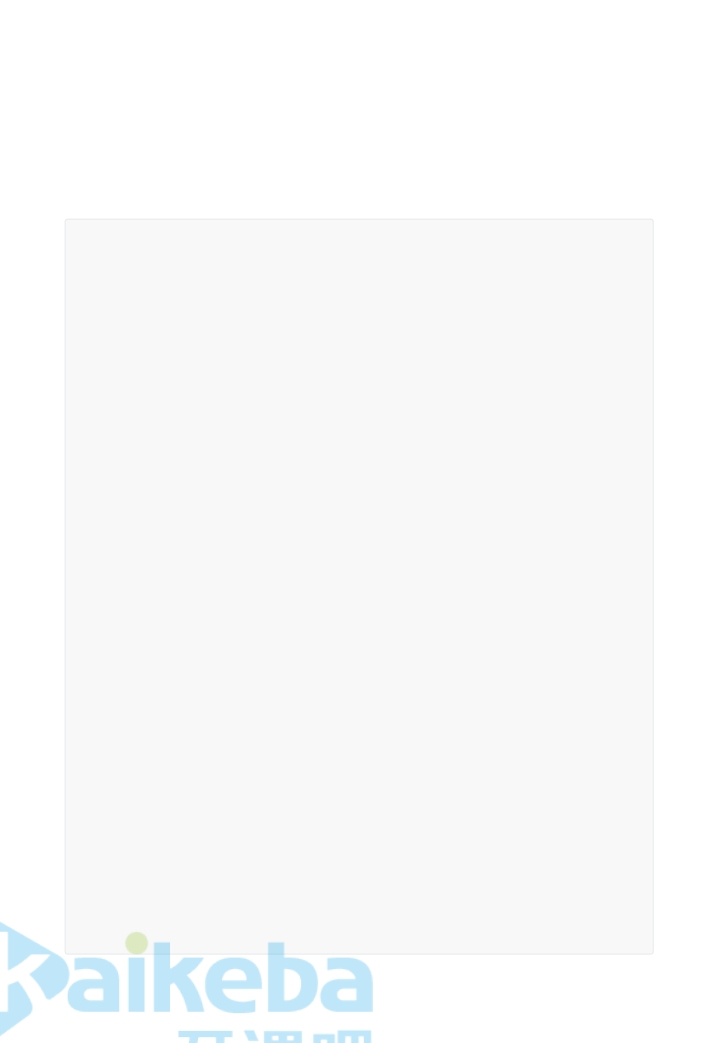
**Kubernetes进阶之PersistentVolume 静态供给实现NFS网络存储**

NFS是一种很早的技术，单机的存储在服务器方面还是非常主流的，但nfs唯一的就是缺点比较大就 是没有集群版，做集群化还是比较费劲的，文件系统做不了，这是一个很大的弊端，大规模的还是需要 选择一些分布式的存储，nfs就是一个网络文件存储服务器，装完nfs之后，共享一个目录，其他的服务 器就可以通过这个目录挂载到本地了，在本地写到这个目录的文件，就会同步到远程服务器上，实现一 个共享存储的功能，一般都是做数据的共享存储，比如多台web服务器，肯定需要保证这些web服务器 的数据一致性，那就会用到这个共享存储了，要是将nfs挂载到多台的web服务器上，网站根目录下， 网站程序就放在nfs服务器上，这样的话。每个网站，每个web程序都能读取到这个目录，一致性的数 据，这样的话就能保证多个节点，提供一致性的程序了。

1. 单独拿一台服务器做nfs服务器，我们这里先搭建一台NFS服务器用来存储我们的网页根目录
2. 暴露目录，让是让其他服务器能挂载这个目录

给这个网段加上权限，可读可写

找个节点去挂载测试一下,只要去共享这个目录就要都去安装这个客户端



#其他节点也需要安装nfs

yum install nfs-utils -y

mount -t nfs 192.168.30.27:/opt/k8s /mnt

1. /mnt
2. -h

192.168.30.27:/opt/k8s   36G  5.8G   30G   17% /mnt

touch a.txt

去服务器端查看已经数据共享过来了

删除nfs服务器的数据也会删除 接下来怎么将K8s进行使用 我们把网页目录都放在这个目录下 # mkdir wwwroot

# vim nfs.yaml

apiVersion: apps/v1beta1

kind: Deployment

metadata:

name: nfs

spec:

replicas: 3

template:

  metadata:

    labels:

      app: nginx

  spec:

    containers :

    - name: nginx

      image: hub.kaikeba.com/library/myapp :v1

      volumeMounts :

      - name: wwwroot

        mountPath: /usr/share/nginx/html

      ports:

      - containerPort: 80

    volumes:

    - name: wwwroot

      nfs:

        server: 192.168.66.13

        path: /opt/k8s/wwwroot

---

apiVersion: v1

kind: Service

metadata:

name: nginx-service

labels:

  app: nginx

spec:

ports:

- port: 80

  targetPort : 80

selector:

  app: nginx

type: NodePort

我们在源pod的网页目录下写入数据，并查看我们的nfs服务器目录下也会共享



**二、PV&PVC**

**1、pv&pvc说明**

管理 存储 和管理 计算 有着明显的不同。 PersistentVolume 给用户和管理员提供了一套API，抽象 出 存储 是如何 提供和消耗的细节 。在这里，我们介绍两种新的API资源： PersistentVolume （简称PV） 和 PersistentVolumeClaim （简称PVC） 。

PersistentVolume（持久卷，简称PV）是集群内，由管理员提供的网络存储的一部分。就像 集群中的节点一样，PV也是集群中的一种资源。它也像Volume一样，是一种volume插件， 但是它的生命周期却是和使用它的Pod相互独立的。PV这个API对象，捕获了诸如NFS、 ISCSI、或其他云存储系统的实现细节。

PersistentVolumeClaim （持久卷声明，简称PVC）是用户的一种存储请求。它和Pod类似， Pod消耗Node资源，而PVC消耗PV资源。Pod能够请求特定的资源（如CPU和内存）。PVC能 够请求指定的大小和访问的模式（可以被映射为一次读写或者多次只读）。

PVC允许用户消耗抽象的存储资源，用户也经常需要各种属性（如性能）的PV。集群管理员需要提 供各种各样、不同大小、不同访问模式的PV，而不用向用户暴露这些volume如何实现的细节。因为这 种需求，就催生出一种 StorageClass 资源。

StorageClass 提供了一种方式，使得管理员能够描述他提供的存储的等级。集群管理员可以将不 同的等级映射到不同的服务等级、不同的后端策略。

**K8s为了做存储的编排** 数据持久卷PersistentVolume 简称pv/pvc主要做容器存储的编排

* 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 pv 都是运维去考虑，用来管理外部存储的
* 静态 ：提前创建好pv，比如创建一个100G 的pv,200G的pv,让有需要的人拿去用，就是说pvc连接 pv,就是知道pv创建的是多少，空间大小是多少，创建的名字是多少，有一定的可匹配性
* 动态
* PersistentVolumeClaim （PVC）：让用户不需要关心具体的Volume实现细节 使用多少个容量来 定义，比如开发要部署一个服务要使用10个G，那么就可以使用pvc这个资源对象来定义使用10个G， 其他的就不用考虑了

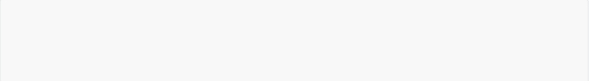
**pv&pvc区别**

PersistentVolume （持久卷） 和 PersistentVolumeClaim （持久卷申请） 是k8s提供的两种API资 源，用于抽象存储细节。

管理员关注如何通过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。

pvc和pv的关系与pod和node关系类似，前者消耗后者的资源。pvc可以向pv申请指定大小的存储 资源并设置访问模式,这就可以通过Provision -> Claim 的方式，来对存储资源进行控制。

**2、生命周期**



#vim pod.yaml

apiVersion: v1

kind: Pod

metadata:

volume 和 claim 的生命周期，PV是集群中的资源，PVC是对这些资源的请求，同时也是这些资源 的“提取证”。PV和PVC的交互遵循以下生命周期：

**供给**

有两种PV提供的方式：静态和动态。

**静态**

集群管理员创建多个PV，它们携带着真实存储的详细信息，这些存储对于集群用户是可用 的。它们存在于Kubernetes API中，并可用于存储使用。

**动态**

当管理员创建的静态PV都不匹配用户的PVC时，集群可能会尝试专门地供给volume给PVC。 这种供给基于StorageClass：PVC必须请求这样一个等级，而管理员必须已经创建和配置过这 样一个等级，以备发生这种动态供给的情况。请求等级配置为“”的PVC，有效地禁用了它自身 的动态供给功能。

**绑定**

用户创建一个PVC（或者之前就已经就为动态供给创建了），指定要求存储的大小和访问模 式。master中有一个控制回路用于监控新的PVC，查找匹配的PV（如果有），并把PVC和PV 绑定在一起。如果一个PV曾经动态供给到了一个新的PVC，那么这个回路会一直绑定这个PV 和PVC。另外，用户总是至少能得到它们所要求的存储，但是volume可能超过它们的请求。 一旦绑定了，PVC绑定就是专属的，无论它们的绑定模式是什么。

如果没找到匹配的PV，那么PVC会无限期得处于unbound未绑定状态，一旦PV可用了，PVC 就会又变成绑定状态。比如，如果一个供给了很多50G的PV集群，不会匹配要求100G的 PVC。直到100G的PV添加到该集群时，PVC才会被绑定。

**使用**

Pod使用PVC就像使用volume一样。集群检查PVC，查找绑定的PV，并映射PV给Pod。对于 支持多种访问模式的PV，用户可以指定想用的模式。一旦用户拥有了一个PVC，并且PVC被绑 定，那么只要用户还需要，PV就一直属于这个用户。用户调度Pod，通过在Pod的volume块 中包含PVC来访问PV。

**释放**

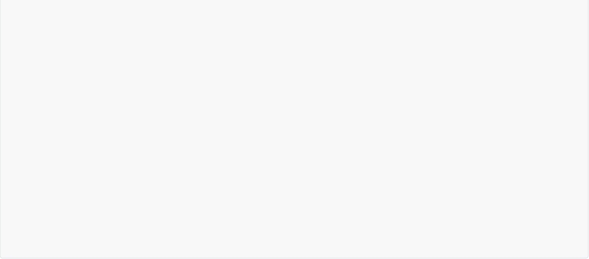
当用户使用PV完毕后，他们可以通过API来删除PVC对象。当PVC被删除后，对应的PV就被认 为是已经是“released”了，但还不能再给另外一个PVC使用。前一个PVC的属于还存在于该PV 中，必须根据策略来处理掉。

**回收**

PV的回收策略告诉集群，在PV被释放之后集群应该如何处理该PV。当前，PV可以被 Retained （保留）、 Recycled（再利用）或者Deleted（删除）。保留允许手动地再次声明 资源。对于支持删除操作的PV卷，删除操作会从Kubernetes 中移除PV对象，还有对应的外部 存储（如AWS EBS，GCE PD，Azure Disk，或者Cinder volume）。动态供给的卷总是会被 删除。

**3、POD&PVC**

先创建一个容器应用



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

卷需求yaml,这里的名称一定要对应，一般两个文件都放在一块 # vim pvc.yaml

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: my-pvc

spec:

accessModes :

  - ReadWriteMany

resources:

  requests:

    storage: 5Gi

接下来就是运维出场了，提前创建好pv

# vim pv1.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: my-pv1

spec:

capacity:

  storage: 5Gi

accessModes :

  - ReadWriteMany

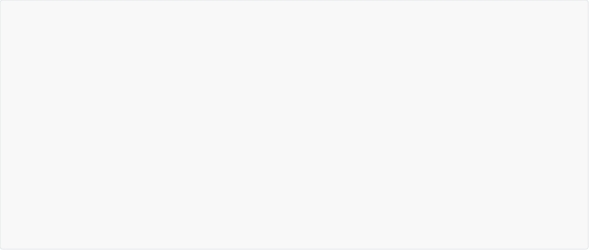
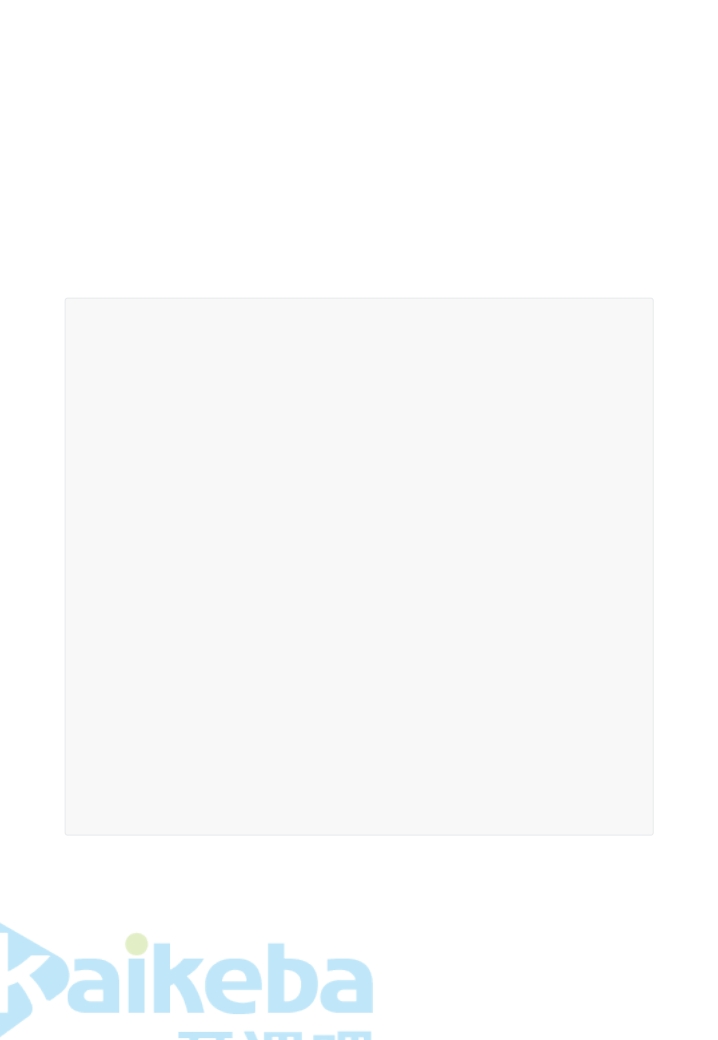
nfs:

  path: /opt/k8s/demo1

  server: 192.168.66.13

提前创建好pv,以及挂载目录

我再创建一个pv，在nfs服务器提前把目录创建好,名称修改一下



# vim pv2.yaml

apiVersion: v1

kind: PersistentVolume

metadata:

name: my-pv2

spec:

capacity:

  storage: 10Gi

accessModes :

  - ReadWriteMany

nfs:

  path: /opt/k8s/demo2

  server: 192.168.66.13

然后现在创建一下我们的pod和pvc,这里我写在一起了

# vim 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

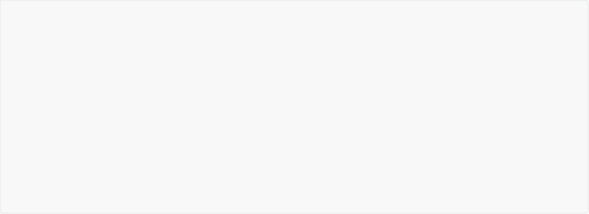
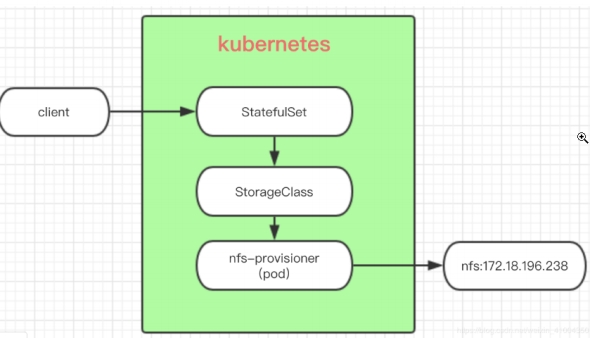
resources:

  requests:

    storage: 5Gi

**4、StatefulSet**

用户通过yaml创建StatefulSet，StatefulSet找到StorageClass ，StorageClass指定到nfs- provisioner为nfs的pv提供者，这是一个pod的服务，用来自动生成pv的，此pod来绑定到对应的nfs服 务。以此来通过nfs服务进行动态的pv生成，然后通过StatefulSet的pvc，与pod进行绑定，实现数据的 持久化存储。



systemctl start rpcbind

systemctl start nfs

yum -y install nfs-common nfs-utils rpcbind mkdir /nfs //创建共享目录

chmod 777 /nfs/

chown nfsnobody /nfs/

vim /etc/exports

/nfs \*(rw,no\_root\_squash ,no\_all\_squash ,sync) /nfs1 \*(rw,no\_root\_squash ,no\_all\_squash ,sync) /nfs2 \*(rw,no\_root\_squash ,no\_all\_squash ,sync)

**持久化演示说明————NFS** 1、安装NFS服务器

在每台节点安装

yum -y install nfs-utils rpcbind

#启动

systemctl start rpcbind

systemctl start nfs

#测试

mkdir /test

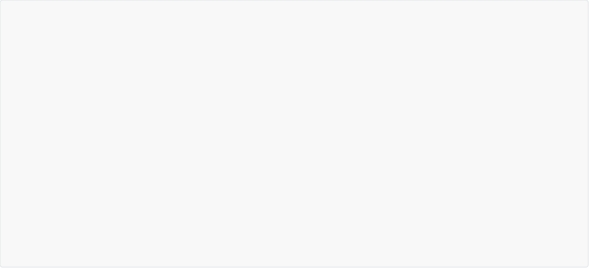
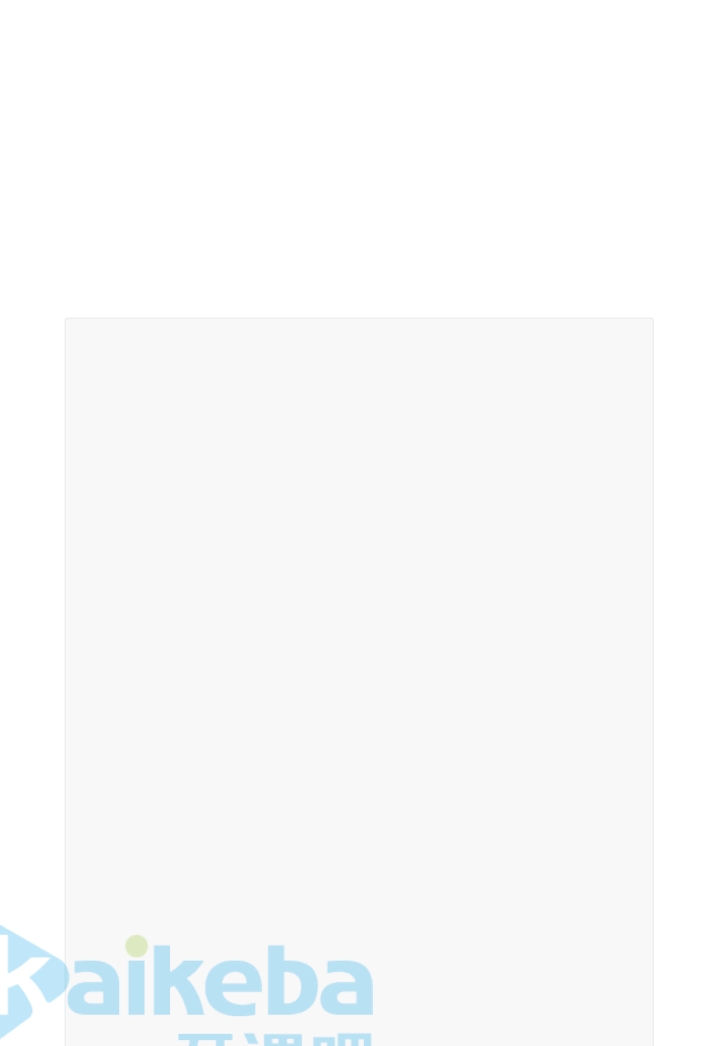
showmount -e 192.168.241.130

mount -t nfs 192.168.241.130:/nfs /test cd /test/

touch 1

umount /etst

创建pv



apiVersion: v1

kind: PersistentVolume

metadata:

name: nfspv1

spec:

capacity:

  storage: 10Gi

accessModes :

  - ReadWriteOnce

persistentVolumeReclaimPolicy : Retain

storageClassName : nfs

nfs:

  path: /nfs

  server: 192.168.66.13

创建pvc

apiVersion: v1

kind: Service

metadata:

name: nginx

labels:

  app: nginx

spec:

ports:

- port: 80

  name: web

clusterIP: None

selector:

  app: nginx

---

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: web

spec:

selector:

  matchLabels :

    app: nginx

serviceName : nginx

replicas: 2

template:

  metadata:

    labels:

      app: nginx

  spec:

    containers :

    - name: nginx

      image: nginx

      ports:

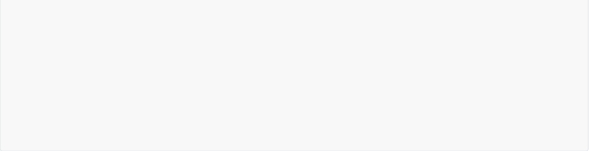
      - containerPort: 80

        name: web

      volumeMounts :

      - name: www

        mountPath: /usr/share/nginx/html volumeClaimTemplates :



- metadata:

    name: www

  spec:

    accessModes : [ "ReadWriteOnce" ]

    storageClassName : nfs

    resources:

      requests:

        storage: 1Gi

作业：部署MySQL 部署。

关于statefulSet\*\*

匹配Pod name(网络标识)的模式为:(statefulset名称)-(序号)，比如上面的示例：web-0， web-1，web-2

statefulSet为每个pod副本创建了一个DNS域名，这个域名的格式为：(podname).(headless server name)，也就意味着服务间是通过pod域名来通信而非pod的ip，因为当pod所在node发 生故障时，pod会被飘逸到其他node上，pod ip会发生变化，但是pod域名不会发生变化

statefulSet使用headless 服务来控制pod的域名，这个域名的FQDN为(servicename). (namespace).svc.cluster.local其中cluster.local指的是集群的域名

根据volumeClaimTemplates 为给个pod创建一个pvc，pvc的命名规则匹配模式

([volumeClaimTemplates.name)-(pod\_name](https://links.jianshu.com/go?to=http%3A%2F%2FvolumeClaimTemplates.name) -(pod\_name))，比如上面的 [volumeMounts.name=www](https://links.jianshu.com/go?to=http%3A%2F%2FvolumeMounts.name%3Dwww) ，podname=web(0-2)，因此创建出来的pvc是www-web-0，www- web-1，www-web-2

删除pod不会删除pvc，手动删除pvc将自动释放pv

StatefulSet的启停顺序

有序部署：部署statefulset时，如果有多个pod副本，他们会被顺序的创建(从0到N-1) 并且， 在下一个pod运行之前所有之前的pod必须都是running和ready状态

有序删除：当pod被删除时，他们被终止的顺序是从N-1到0

有序扩展：当pod执行扩展操作时，与部署一样，它前面的pod必须都处于running 和ready 状态

Statefulset使用场景：稳定的持久化存储，即pod重新调度后还是能访问到相同的持久化数 据，基于pvc来实现 稳定的网络标识符，即pod重新调度后其podname和hostname不变 有序部 署，有序扩展，基于init containers来实现 有序收缩