

**SandStone MOSFS**

**CSI Plugin操作手册**

**深圳市杉岩数据技术有限公司**

**2022年8月**

目录

[1 背景 1](#_Toc651658266)

[2 说明 2](#_Toc253876962)

[3 部署 5](#_Toc2007577392)

[3.1 准备 5](#_Toc90165680)

[3.2 导入镜像 7](#_Toc1438612625)

[3.3 部署CSI Plugin 7](#_Toc244006802)

[3.4 创建StorageClass 9](#_Toc1476039091)

[3.4.1 对接类型为桶的文件系统 9](#_Toc57912293)

[3.4.2 对接类型为逻辑池的文件系统 9](#_Toc521816360)

[4 测试 11](#_Toc1991831819)

[4.1 对接类型为桶的文件系统 11](#_Toc1742292497)

[4.1.1 创建pvc 11](#_Toc1770470234)

[4.1.2 创建pod挂载pvc 11](#_Toc759810006)

[4.1.3 创建快照 11](#_Toc1189005780)

[4.1.4 从快照创建pvc 12](#_Toc1284809125)

[4.1.5 克隆pvc 12](#_Toc838893290)

[4.1.6 扩容pvc 12](#_Toc1049382475)

[4.1.7 删除测试资源 13](#_Toc1835548161)

[4.2 对接类型为逻辑池的文件系统 13](#_Toc1455352772)

[4.3 使用静态卷 14](#_Toc275299674)

[4.3.1 静态卷配置 14](#_Toc1281845280)

[5 可选配置 15](#_Toc1874662196)

[5.1 配置mountpod的资源限制 15](#_Toc1714943035)

[5.2 配置DNS 15](#_Toc1669562858)

[6 删除CSI Plugin 19](#_Toc1089425207)

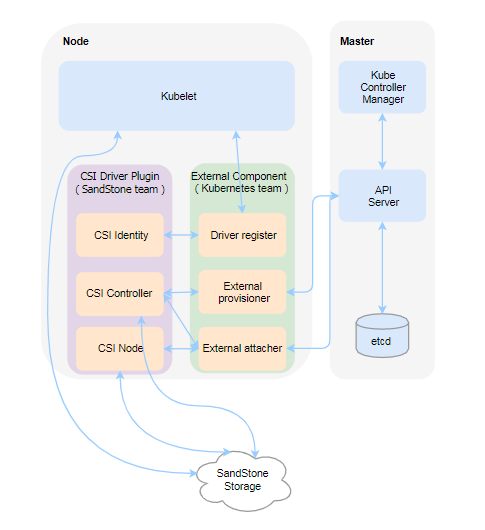
1. 背景

Kubernetes原来的存储插件与核心Kubernetes二进制文件链接、编译、构建，这样会导致以下结果：

* 存储插件开发与Kubernetes发行版本紧密耦合。
* 不利于社区/开发人员测试、维护。
* 插件中的错误会导致kubernetes组件崩溃。
* 插件能获取Kubernetes全部特权。
* 插件开发人员被迫提供插件源码。

容器存储接口（CSI）用于解决上述问题，它是来自不同CO的社区成员之间的合作而产生的规范，Kubernetes对外开放了存储接口，实现这个接口即可集成到Kubernetes系统中。另外CSI特性正式GA，同时社区也宣布未来将不再对In tree/Out of tree继续开发，并将已有功能全部迁移到CSI上，以后CSI存储插件开发与使用是趋势。

1. 说明
2. 整体设计



整体设计如上，蓝色模块是K8S自带的模块，绿色模块与紫色模块为我们部署的模块，底层为我们的存储服务，下面主要介绍要部署的这两个模块：

* 如图绿色部分，Kubernetes 团队会提供以下可复用的sidecar（辅助）容器，负责连接SandStone CSI插件以及和Kubernetes集群交互。
* Driver-registrar：使用Kubelet注册CSI驱动程序的sidecar容器，并将 NodeId（通过GetNodeID调用检索到CSI endpoint）添加到Kubernetes Node API对象的annotation里面。
* External-provisioner：监听Kubernetes PersistentVolumeClaim对象的sidecar容器，并触发对CSI端点的CreateVolume 和DeleteVolume操作。
* External-attacher：可监听Kubernetes VolumeAttachment对象并触发ControllerPublish和ControllerUnPublish操作的sidecar容器，负责attache/detache卷到node节点上。
* 如图紫色部分，CSI插件在Kubernetes和存储资源池起到了纽带作用，通过实现CSI中identity、controller、node三类接口，Kubernetes平台可以动态（Storageclass）或者静态（PV）调用存储的资源。
* identity接口，其主要作用上报存储厂商提供的基本信息。

例如GetPluginInfo可以得到插件的版本号，以及插件CSI的规范版本，Get Plugin Capabilities可以找到存储插件支持的功能点，例如存储插件如果要支持storageclass必须支持CreateVolume、DeleteVolume两个接口，开发者还可以通过实现Probe接口来支持对存储插件的健康检测。

* Controller类接口，主要实现存储资源和存储卷进行相关管理的接口。

如存储卷的创建和删除，需要实现Controller接口的Create Volume接口和Delete Volume接口，对存储卷进行创建和删除的操作。将存储卷从某台主机挂载或将存储卷上卸载，需要实现ControllerPublishVolume和ControllerUnpublishVolume这两个接口。

* Node类接口，主要对主机上的存储卷进行操作。

例如存储卷的分区和格式化，将存储卷挂载到指定POD上，或者将存储卷从指定POD上卸载。存储卷分区格式化，需要调用NodeStageVolume和NodeUnstageVolume来实现。存储卷挂载至某个路径或者将存储卷从某个容器路径中卸载，需要实现NodePublishVolume和NodeUnpublishVolume这两个接口。

* mosfs-csi特性支持：

1. 支持静态卷(static provisioning)；
2. 支持动态卷(Dynamic provisioning)；
3. 支持ReadWriteMany，ReadOnlyMany
4. 支持Subpath
5. 支持卷快照，卷克隆以及卷扩容
6. 支持qos配置

下面将介绍如何将sidecar容器 与SandStone MOSFS插件结合进行部署。

1. 部署

SandStone MOSFS CSI插件是基于Kubernetes Container Storage Interface (CSI) Driver标准开发的，用于对接Kubernetes和杉岩分布式统一存储的CSI插件。

在开始之前，请提前安装SandStone MOSFS服务并获取最新版本的SandStone MOSFS CSI驱动程序。

1. 准备集群

| **集群** | **版本** |
| --- | --- |
| SandStone MOS | 6.3.5或以上版本 |
| Kubernetes | 1.17-1.25 |

* 1. 准备

通过OM界面创建用于CSI的MOSFS文件系统并获取以下信息，注意文件系统有两种类型：对接桶和对接逻辑池。

* 文件系统名。
* Console token。

在元数据节点执行以下命令：

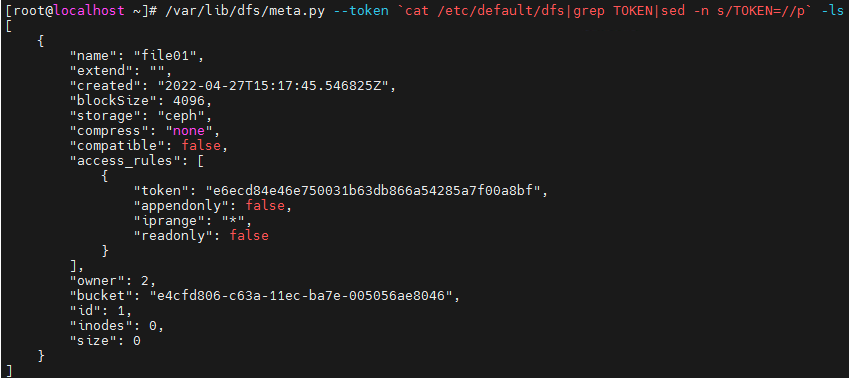
cat /etc/default/mosfs | grep -i token



* 文件系统Token。

在元数据节点执行以下命令，找到对应文件系统的token。

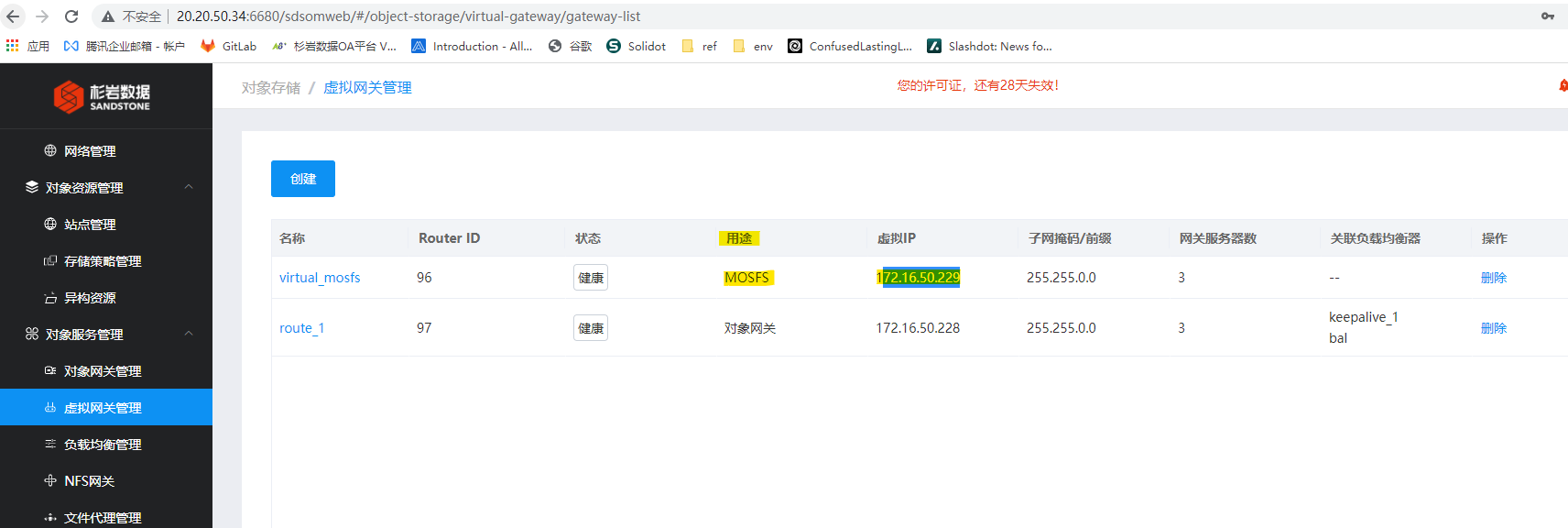
/var/lib/mosfs/meta.py --token `cat /etc/default/mosfs | grep TOKEN|sed -n s/TOKEN=//p` -ls



* MOSFS Console的IP端口。

Console的IP为创建MOSFS时配置的控制台虚拟IP，可以在虚拟网关管理页面查看。

1. 查看MOSFS Console的IP



在console安装节点执行以下命令可以查看console的端口。示例图中console端口为9400。

grep CONSOLE\_PORT /var/lib/mosfs/etc/mosfs/console/mosfs-console-1/.env



* accesskey/secretkey

对接桶需要提前获取桶所属用户的accesskey/secretkey，对接逻辑池则不需要。

* 1. 导入镜像

1. 导入插件镜像：

docker load -i sandstone-mosfsplugin.tar

注意：如果容器运行时由docker替换为了containerd(例如k8sv1.24)，则使用如下命令进行镜像导入：

$ ctr -n k8s.io images import sandstone-mosfsplugin.tar

$ ctr -n k8s.io images import sidecar.tar

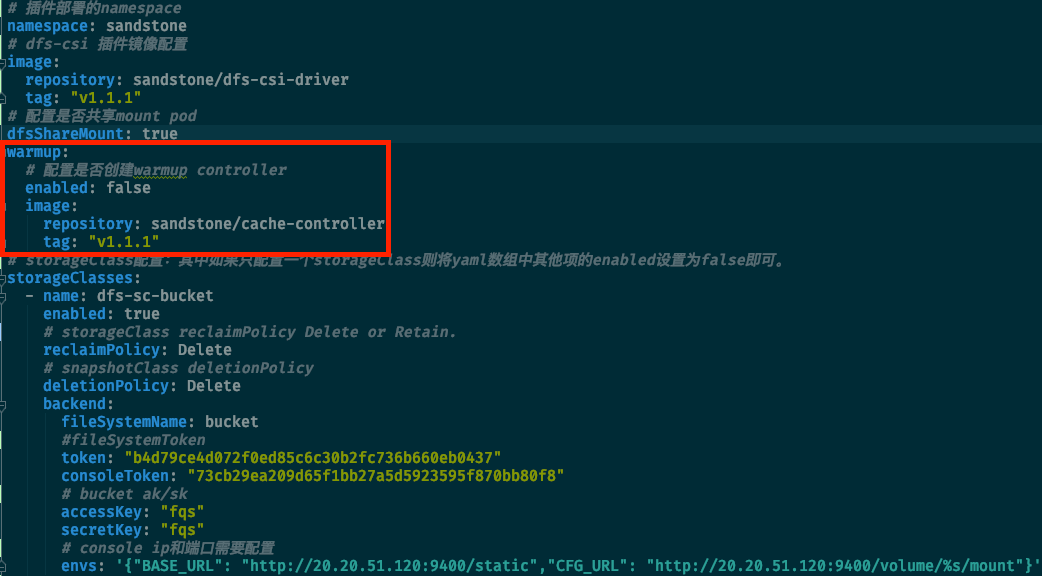
1. 进入不同k8s版本目录，导入sidecar镜像：

docker load -i sidecar.tar

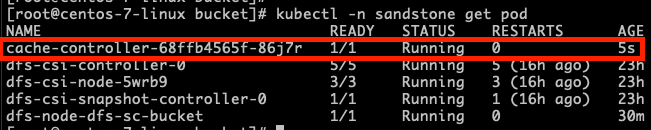
注意：需在所有k8s节点导入镜像。

* 1. 配置warmup
     1. 配置是否开启warmup功能

如下图所示，enabled为true，则开启warmup功能：



部署时会额外启动一个warmup-controller：



* 1. 配置StorageClass
     1. 对接类型为桶的文件系统

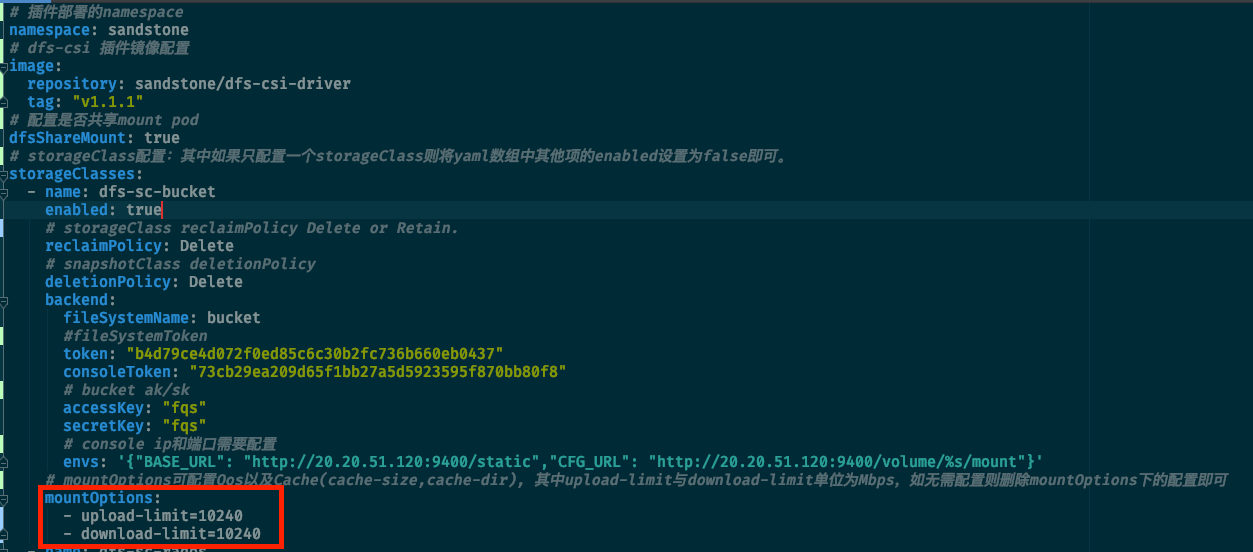
在config.yaml配置文件中，用准备阶段准备的信息修改目录中的secret.yaml的相关参数：

* name：文件系统名。
* token：文件系统名token。
* consoleToken：console token。
* accesskey/secretkey：访问文件系统的存储桶的accesskey/secretkey。
* env中的ip端口为console的ip端口。
  + 1. 对接类型为逻辑池的文件系统

注意：对接mosfs逻辑池类型，需要保证与mosfs rados网络平面可以互通

* name：文件系统名。
* token：文件系统名token。
* consoleToken：console token。
* accesskey：ceph。
* secretkey：client.admin。
* envs中的ip端口为console的ip端口。
  + 1. 配置Qos

编辑config.yaml文件，在mountOptions栏目增加配置”upload-limit”和“download-limit”可配置Qos,单位Mbps。如下所示：

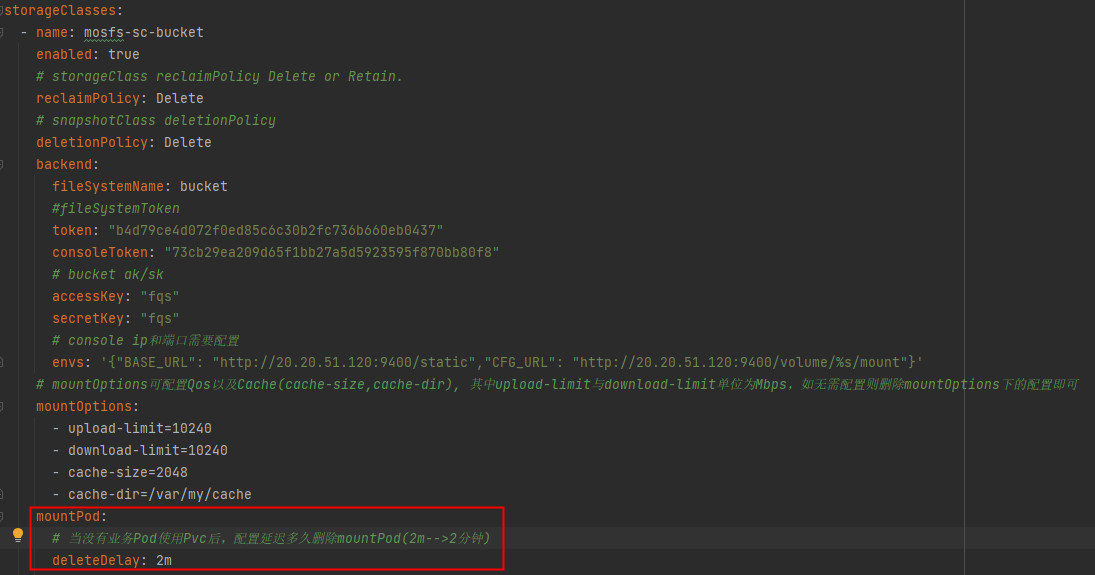


* + 1. 配置cache

编辑config.yaml文件，在mountOptions栏目增加”cache-size”，”cache-dir”，如下所示可配置cache(cache-dir默认：/var/jfsCache):



* + 1. 配置mountPod延迟删除

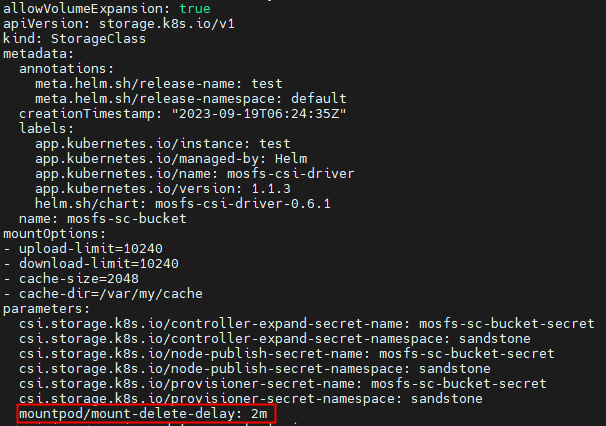


Mount Pod 是支持复用的，由CSI Node Service 以引用计数的方式进行管理：当没有任何应用 Pod 在使用该 Mount Pod 创建出来的 PV 时，CSI Node Service 会删除 Mount Pod。

但在 Kubernetes 不少场景中，容器转瞬即逝，调度极其频繁，这时可以为 Mount Pod 配置延迟删除，这样一来，如果短时间内还有新应用 Pod 使用相同的 Volume，Mount Pod 能够被继续复用，免除了反复销毁创建的开销。

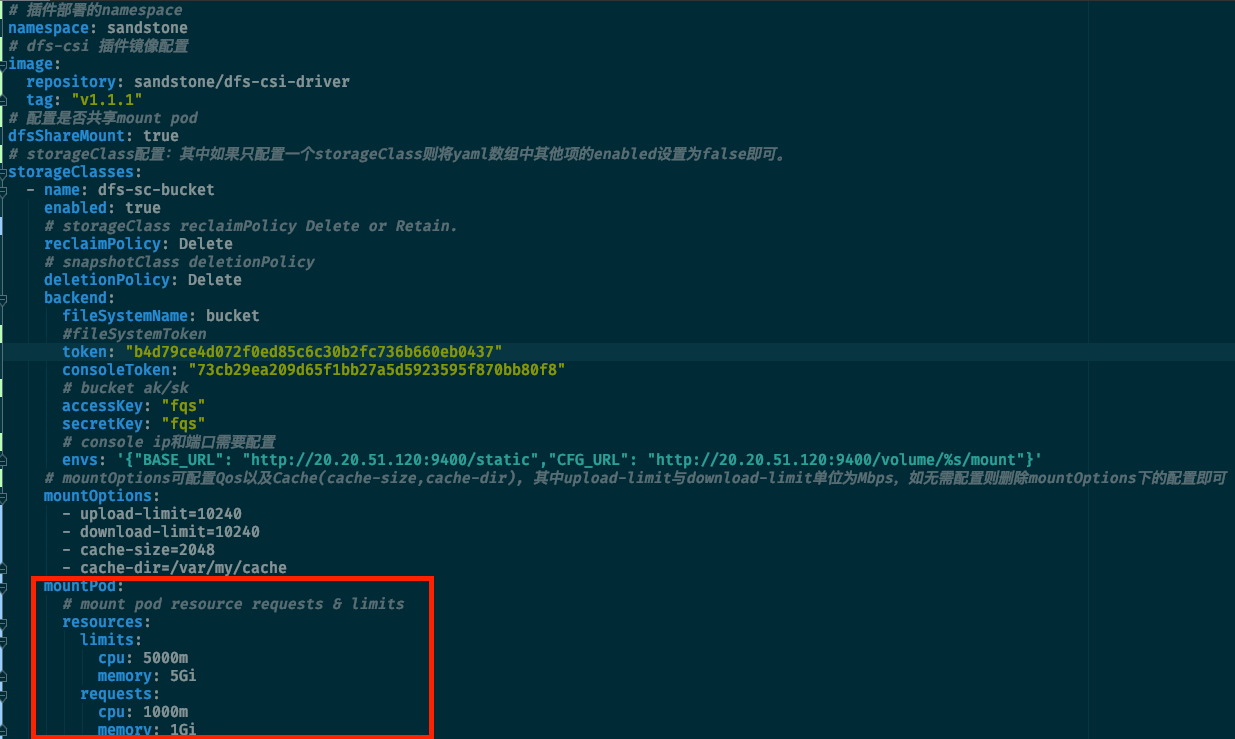
控制延迟删除 Mount Pod 的配置项形如 mountpod/mount-delete-delay: 2m，单位支持 m（分钟）、h（小时）。 配置好延迟删除后，当引用计数归零，Mount Pod 会被打上 mosfs-delete-at 的注解（annotation），标记好删除时间，到达设置的删除时间后，Mount Pod 才会被删除。但如果在此期间有新的应用 Pod 欲使用该 PV，注解 mosfs-delete-at 就被清空，Mount Pod 的删除计划随之取消，得以继续复用。

注意：如果须在已安装的环境增加以上配置，可直接重新创建storageClass，增加如下配置：



* + 1. 配置mountPod资源

编辑config.yaml文件，增加mountPod栏目，如下：



注意：如果是共享mountPod则每个节点相同storageclass的卷共享一个mountPod。非共享模式则是每个卷单独一个mountPod。共享模式下，共享同一mountPod的所有卷的”只读”和”读写”，由使用mountPod时第一个卷的读写模式决定。

以上属于storageClass的配置，在使用helm安装后都会应用到k8s storageClass对象中，如需修改storageClass对象，需要先删除再创建（storageClass对象由于k8s限制不支持更新）

* 1. 部署CSI Plugin

1. 安装包解压后，进入对应k8s版本子目录，其中**mosfs-csi-driver**为**helm chart**配置目录，**config.yaml**文件为helm install时的配置文件。编辑config.yaml文件，在backend字段中填入相关环境配置，具体含义见storageClass配置说明。
2. config.yaml中如果是非default命令空间，则namespace需手动创建，使用如下命令

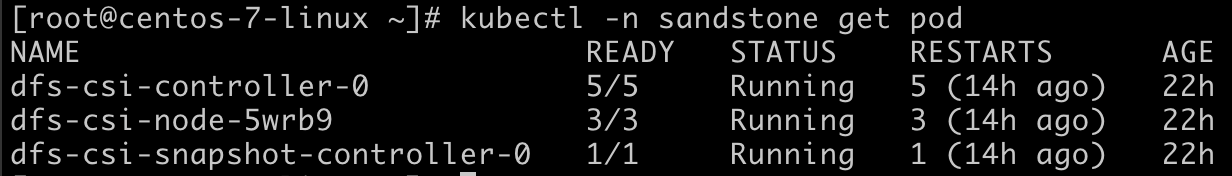
kubectl create namespace <your-namespace>

1. 安装前需至少配置storageClass中backend下的相关内，配置完毕后，使用如下命令进行安装：

helm install <helm-instance-name> --set cephConf=$(curl -sS http://<consoleIP:port>/static/Linux/ceph.conf|base64 -w 0) -f config.yaml ./mosfs-csi-driver/

1. 执行安装命令后如无报错信息则代表安装成功，使用如下命令查看mosfs-csi插件状态

kubectl -n <your-namespace> get pod



1. 测试
   1. 对接类型为桶的文件系统

cd到example/bucket目录（注意：**客户端节点需要能与服务端元数据ip互通**）。

* + 1. 创建pvc

1. 执行以下命令，创建pvc。

kubectl create -f pvc.yaml

1. 执行以下命令，查看pvc。

kubectl get pvc mosfs-pvc

* + 1. 创建pod挂载pvc

1. 执行以下命令，创建pod挂载pvc。

kubectl create -f pod.yaml

1. 执行以下命令，查看pod成功创建。

kubectl get pod|grep mosfs

* + 1. 创建快照

1. 执行以下命令，创建快照。

kubectl create -f snapshotclass.yaml

kubectl create -f snapshot.yaml

1. 执行以下命令，查看快照。

kubectl get volumesnapshot

* + 1. 从快照创建pvc

1. 执行以下命令，从快照创建pvc。

kubectl create -f pvc-restore.yaml

1. 执行以下命令，查看从快照创建的pvc。

kubectl get pvc mosfs-pvc-restore

* + 1. 克隆pvc

1. 执行以下命令，克隆pvc。

kubectl create -f pvc-clone.yaml

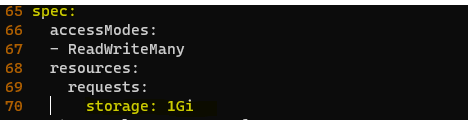
1. 执行以下命令，查看克隆pvc。

kubectl get pvc mosfs-pvc-clone

* + 1. 扩容pvc

1. 执行以下命令，修改spec下“storage”的大小。

kubectl edit pvc mosfs-pvc



1. 保存后退出。
2. 执行以下命令，查看扩容后的pvc。

kubectl get pvc mosfs-pvc

* + 1. 删除测试资源
* 删除克隆卷：

kubectl delete -f pvc-clone.yaml

* 删除快照恢复卷：

kubectl delete -f pvc-restore.yaml

* 删除快照：

kubectl delete -f snapshot.yaml

* 删除测试pod：

kubectl delete -f pod.yaml

* 删除卷：

kubectl delete -f pvc.yaml

* 1. 对接类型为逻辑池的文件系统

操作命令与对接类型为桶的文件系统一样，只把“yaml”文件改为“example/radospool”目录的下的对应文件即可。

* 1. 使用warmup

进入example/warmup目录中，配置cache.yaml。cache需配置和warmup的pvc同namespace，且在spec字段中填入对应pvcName，如果需要预热卷中的子目录，则通过path字段填写。

创建该cache对象可对指定的pvc挂载路径进行预热，如果预热成功，查看该cache资源的status字段cached则为true。

* 1. 使用静态卷
     1. 静态卷配置

1. 在安装包example/static目录中配置pv.yaml，通过指定nodePublishSecretRef为基于桶的文件系统secret（mosfs-sc-bucket-secret）或者基于池的文件系统secret(mosfs-sc-rados-secret)，来选择桶或者池。Namespace需要配置和插件安装命名空间一致。
2. 通过subPath字段来配置挂载时的子目录
3. 在安装包example/static目录下执行执行如下命令创建相关资源：

kubectl create -f pv.yaml

kubectl create -f pvc.yaml

kubectl create -f pod.yaml

1. 静态卷需要自行在mosfs ui中配置配额，才能限制pv的使用容量。配额配置在subPath上，如subPath=”abc”，则配额配置为“/abc”
2. 使用完毕后，安装包example/static目录下，使用如下命令删除相关资源：(注意：mosfs中创建的卷目录/mosfs/<pvName>/<subPath>，以及配额资源需手动删除)

kubectl create -f pod.yaml

kubectl create -f pvc.yaml

kubectl create -f pv.yaml

1. 可选配置
   1. 配置mountpod的资源限制

一个pv在一个node上挂载会创建一个挂载pod。可以通过在storageclass的parameters里配置以下参数限制挂载pod的资源（配置方式参考pod资源配置）。

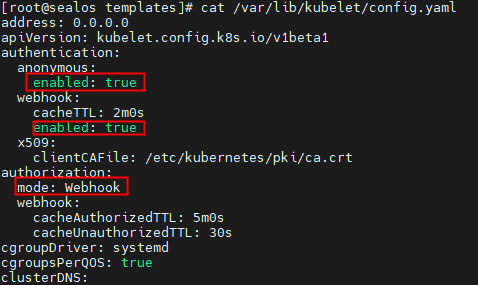
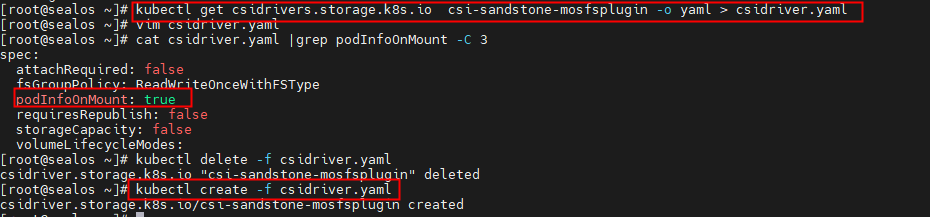
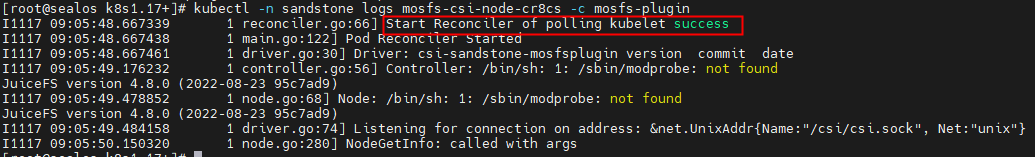
mountpod/cpu-limit

mountpod/memory-limit

mountpod/cpu-request

mountpod/memory-request

* 1. 启用 Kubelet 认证鉴权
     1. 说明
     2. csi node 在维护mountPod时会监听业务pod的挂载情况，默认情况下通过list apiServer进行业务pod状态查询。在集群负载较大的情况下，直接list APIServer 会造成额外的压力。因此支持配置通过查询kubelet进行list pod以减少apiServer压力，且提高csi响应速度。
     3. 操作

1. 配置kubelet 支持匿名访问
   1. 编辑kubelet配置文件，默认路径为:/var/lib/kubelet/config.yaml，如下所示：
      * 1. 
        2. 红框内为开启匿名访问的配置。
   2. 执行命令重启kubelet: **systemctl restart kubelet**
2. 配置csidriver对象
   1. 执行如下命令更新（如下图所示，先保存csidriver对象，然后修改yaml，最后重新创建该对象）：
      * 1. 
3. 验证
   * + 1. 进行上述配置后，查看csi node pod启动日志，如下：
       2. 

如上所示为配置成功后的日志

* + 1. 验证
  1. 配置DNS

不做该配置不影响功能，影响如下：

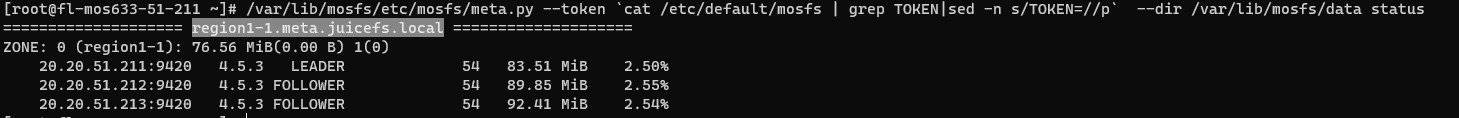
* 不能更换元数据节点IP。
* 客户端不断的写无法解析meta域名的warning日志。

获取meta dns域名

1. 在任一元数据节点执行以下命令。

/var/lib/mosfs/meta.py --token `cat /etc/default/mosfs | grep TOKEN|sed -n s/TOKEN=//p` --dir /var/lib/mosfs/data\_0 status

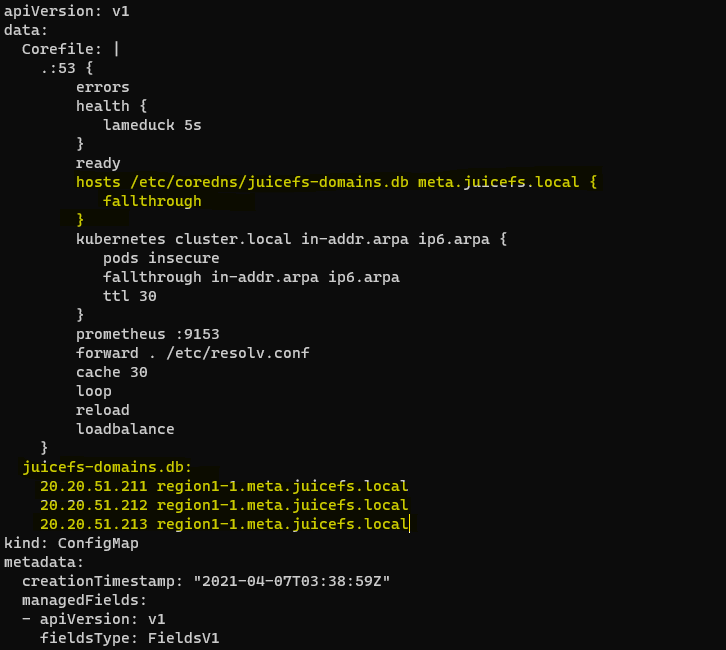
示例图中的meta域名为“region1-1.meta.juicefs.local”。



1. 假设meta ip在20.20.51.211~20.20.51.213，meta dns域名为“region1-1.meta.juicefs.local”，执行以下命令。

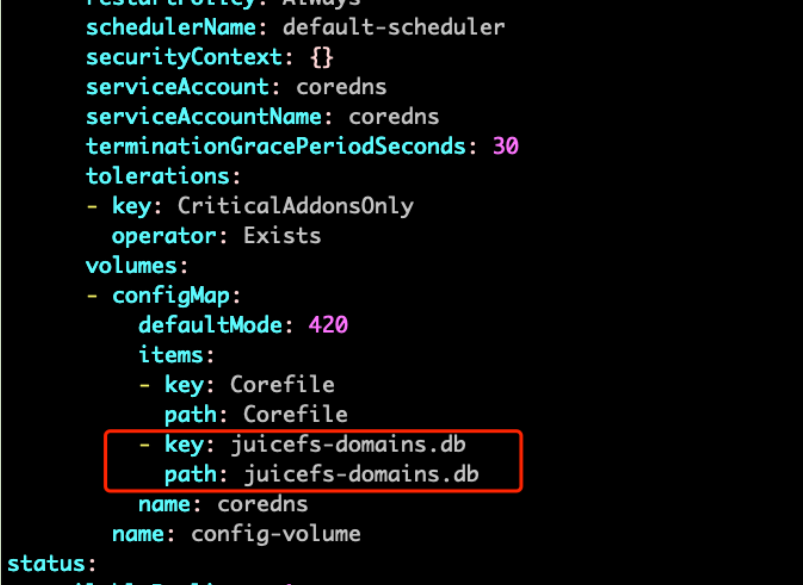
kubectl edit cm coredns -n kube-system

新增黄色部分内容：



1. 执行以下命令，将ConfigMap coredns增加的“juicefs-domains.db”挂载到CoreDNS的Pod⾥。

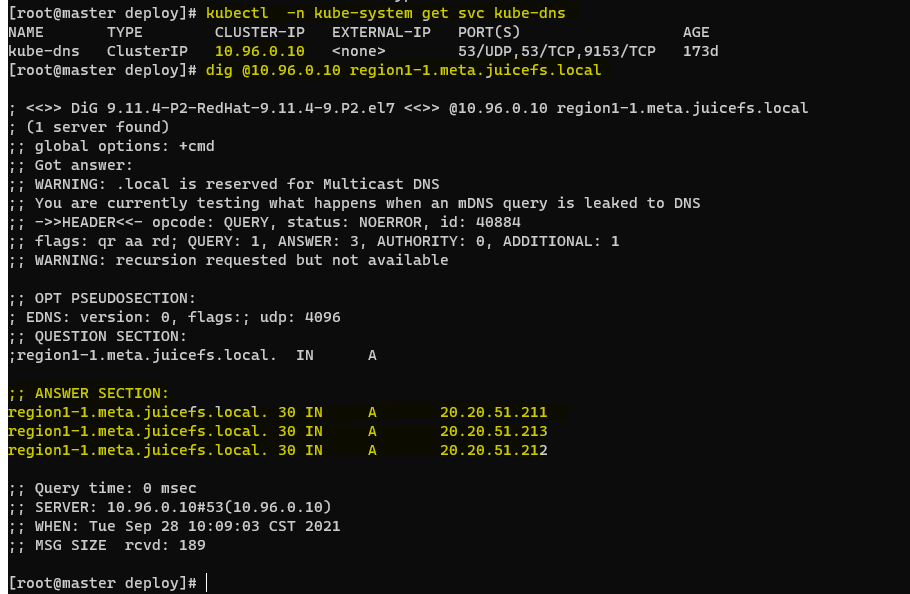
kubectl edit -n kube-system deployment coredns



1. 找到“spec:template:spec:containers:volumes:configMap:items:”里的“Corefile”，在下⾯加上如上红框内容，其它内容保持不变。
2. 做完以上两步后，CoreDNS会重启。获取集群DNS服务地址并使⽤这个地址解析主机记录成功，表示DNS配置成功。

kubectl -n kube-system get svc kube-dns

dig @10.96.0.10 region1-1.meta.juicefs.local



1. 删除CSI Plugin
2. 进入对应k8s版本目录，执行以下命令。

helm uninstall <helm-instance-name>

****

**版权声明**

本手册中的所有内容及格式的版权属于深圳市杉岩数据技术有限公司

（以下简称“杉岩数据”）所有，未经杉岩数据许可，任何人不得仿制、拷贝、转译或任意引用。

版权所有 不得翻印 © 2014 - 2022杉岩数据

**商标声明**

本手册中所谈及的产品名称仅做识别之用，手册中涉及的

其他公司的注册商标或是版权属各商标注册人所有，恕不逐一列明。