CEPH之块存储

一、官方文档

```
https://docs.ceph.com/en/latest/
http://docs.ceph.org.cn/rbd/rbd/
```

二、块存储

块存储简称（RADOS Block Device）,是一种有序的字节序块，也是Ceph三大存储类型中最为常用的存储方式，Ceph的块存储时基于RADOS的，因此它也借助RADOS的快照，复制和一致性等特性提供了快照，克隆和备份等操作。Ceph的块设备值一种精简置备模式，可以拓展块存储的大小且存储的数据以条带化的方式存储到Ceph集群中的多个OSD中。

2.1、创建pool

```
官方文档：http://docs.ceph.org.cn/rados/operations/pools/
# 1、 查看pool命令
[root@node1 ceph-deploy]# ceph osd lspools

# 2、首先得创建一个pool（名字为ceph-demo pg数量64  pgp数量64 副本数【replicated】默认是3）
[root@node1 ceph-deploy]# ceph osd pool create ceph-demo 64 64
pool 'ceph-demo' created

# 3、查看pool
[root@node1 ceph-deploy]# ceph osd lspools
1 ceph-demo

# 4、查看pg、pgp、副本的数量
[root@node1 ceph-deploy]# ceph osd pool get ceph-demo pg_num
pg_num: 64
[root@node1 ceph-deploy]# ceph osd pool get ceph-demo pgp_num
pgp_num: 64
[root@node1 ceph-deploy]# ceph osd pool get ceph-demo size
size: 3

# 5、查看调度算法
[root@node1 ceph-deploy]# ceph osd pool get ceph-demo crush_rule
crush_rule: replicated_rule

# 6、调整就用set
[root@node1 ceph-deploy]# ceph osd pool set ceph-demo pg_num  128
set pool 1 pg_num to 128
[root@node1 ceph-deploy]# ceph osd pool set ceph-demo pgp_num  128
set pool 1 pgp_num to 128

# 7、查看调整后的pg、pgp
```

```
[root@node1 ceph-deploy]# ceph osd pool get ceph-demo pg_num
pg_num: 128
[root@node1 ceph-deploy]# ceph osd pool get ceph-demo pgp_num
pgp_num: 128
```

2.2、创建块存储文件

```
官方文档：http://docs.ceph.org.cn/rbd/rados-rbd-cmds/
# 1、创建方式（2种方式都可）  -p 指定pool名称、--image 指定image（块名字）名字
[root@node1 ~]# rbd create -p ceph-demo --image rbd-demo.img --size 1G
[root@node1 ~]# rbd create ceph-demo/rbd-demo1.img --size 1G

# 2、查看列表
[root@node1 ~]# rbd -p ceph-demo ls
rbd-demo.img
rbd-demo1.img

# 3、查看某个块的信息（可以看到一个块被分成了256个objects）
[root@node1 ~]# rbd info ceph-demo/rbd-demo.img
rbd image 'rbd-demo.img':
        size 1 GiB in 256 objects
        order 22 (4 MiB objects)
        snapshot_count: 0
        id: 14a48cb7303ce
        block_name_prefix: rbd_data.14a48cb7303ce
        format: 2
        features: layering, exclusive-lock, object-map, fast-diff, deep-flatten  # 等会把这几个features都去掉
        op_features:
        flags:
        create_timestamp: Sun Jan 24 14:13:37 2021
        access_timestamp: Sun Jan 24 14:13:37 2021
        modify_timestamp: Sun Jan 24 14:13:37 2021

# 4、删除块
[root@node1 ~]#  rbd rm ceph-demo/rbd-demo1.img
Removing image: 100% complete...done.
```

2.3、使用块存储文件

```
# 1、查看当前pool有几个块文件
[root@node1 ~]# rbd list  ceph-demo
rbd-demo.img

# 2、直接使用会报错，因为内核级别的一些东西不支持
[root@node1 ~]# rbd map ceph-demo/rbd-demo.img
rbd: sysfs write failed
```

```
RBD image feature set mismatch. You can disable features unsupported by the kernel with
"rbd feature disable ceph-demo/rbd-demo.img object-map fast-diff deep-flatten".
In some cases useful info is found in syslog - try "dmesg | tail".
rbd: map failed: (6) No such device or address

# 3、disable 模块
[root@node1 ~]# rbd feature disable ceph-demo/rbd-demo.img deep-flatten
[root@node1 ~]# rbd feature disable ceph-demo/rbd-demo.img fast-diff
[root@node1 ~]# rbd feature disable ceph-demo/rbd-demo.img object-map
[root@node1 ~]# rbd feature disable ceph-demo/rbd-demo.img

# 4、查看是否成功禁用
[root@node1 ~]# rbd info ceph-demo/rbd-demo.img
rbd image 'rbd-demo.img':
        size 1 GiB in 256 objects
        order 22 (4 MiB objects)
        snapshot_count: 0
        id: 14a48cb7303ce
        block_name_prefix: rbd_data.14a48cb7303ce
        format: 2
        features: layering   # 看到这里是layering状态就可以测试挂载了
        op_features:
        flags:
        create_timestamp: Sun Jan 24 14:13:37 2021
        access_timestamp: Sun Jan 24 14:13:37 2021
        modify_timestamp: Sun Jan 24 14:13:37 2021

# 5、再次使用刚刚创建的块存储文件
[root@node1 ~]# rbd map ceph-demo/rbd-demo.img
/dev/rbd0

# 6、查看device
[root@node1 ~]# rbd device  list
id pool      namespace image        snap device
0  ceph-demo           rbd-demo.img -    /dev/rbd0  (这就相当于我们本地的一块磁盘一样，可以进行
分区格式化操作)

# 7、fdisk查看可以查看到相应信息
[root@node1 ~]# fdisk -l | grep rbd0
Disk /dev/rbd0: 1073 MB, 1073741824 bytes, 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 4194304 bytes / 4194304 bytes

# 8、比如格式化
[root@node1 ~]# mkfs.ext4 /dev/rbd0

# 9、然后挂载
[root@node1 ~]# mkdir /mnt/rbd-demo
```

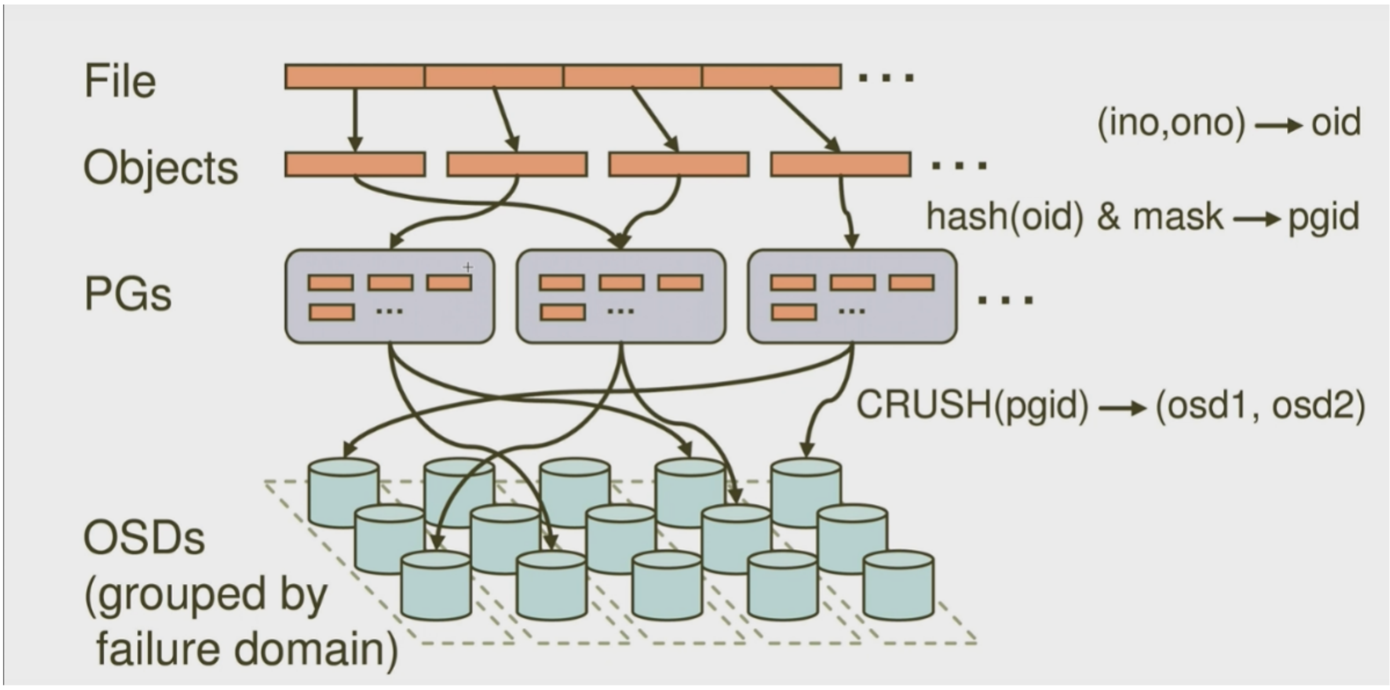
```
[root@node1 ~]# mount /dev/rbd0 /mnt/rbd-demo

# 10、df查看
[root@node1 ~]# df -h
Filesystem               Size  Used Avail Use% Mounted on
/dev/rbd0                976M  2.6M  907M   1% /mnt/rbd-demo
```

2.4、块存储扩容

```
# 1、就拿之前创建的盘来操作
[root@node1 ~]# rbd -p ceph-demo ls
rbd-demo.img

# 2、查看它的信息
[root@node1 ~]# rbd -p ceph-demo info --image rbd-demo.img
rbd image 'rbd-demo.img':
        size 1 GiB in 256 objects    （目前是一个g）
        order 22 (4 MiB objects)
        snapshot_count: 0
        id: 14a48cb7303ce
        block_name_prefix: rbd_data.14a48cb7303ce
        format: 2
        features: layering
        op_features:
        flags:
        create_timestamp: Sun Jan 24 14:13:37 2021
        access_timestamp: Sun Jan 24 14:13:37 2021
        modify_timestamp: Sun Jan 24 14:13:37 2021

# 3、扩容（缩容也可，但是不建议）
[root@node1 ~]# rbd resize ceph-demo/rbd-demo.img --size 2G
Resizing image: 100% complete...done.

# 4、扩容后查看
[root@node1 ~]# rbd -p ceph-demo info --image rbd-demo.img
rbd image 'rbd-demo.img':
        size 2 GiB in 512 objects
        order 22 (4 MiB objects)
        snapshot_count: 0
        id: 14a48cb7303ce
        block_name_prefix: rbd_data.14a48cb7303ce
        format: 2
        features: layering
        op_features:
        flags:
        create_timestamp: Sun Jan 24 14:13:37 2021
        access_timestamp: Sun Jan 24 14:13:37 2021
        modify_timestamp: Sun Jan 24 14:13:37 2021
```

```
# 5、此时的磁盘大小是扩上去了，可是文件系统挂载的是不会自动扩的
[root@node1 ~]# fdisk -l | grep rbd0
Disk /dev/rbd0: 2147 MB, 2147483648 bytes, 4194304 sectors
[root@node1 ~]# df -h
Filesystem                Size  Used Avail Use% Mounted on
/dev/rbd0                 976M  2.6M  907M   1% /mnt/rbd-demo  # 此处还是1个G

# 6、扩容文件系统（注意不建议对这种磁盘进行分区，云上的也是一样，建议多买几块）
[root@node1 ~]# blkid
/dev/sr0: UUID="2018-11-25-23-54-16-00" LABEL="CentOS 7 x86_64" TYPE="iso9660"
PTTYPE="dos"
/dev/sdb: UUID="k4g1pw-rOvV-NG7w-ajnZ-qipH-kXwq-h0jY0o" TYPE="LVM2_member"
/dev/sda1: UUID="ccb430ea-66c9-4c91-a4b4-ba870ca15943" TYPE="xfs"
/dev/sda2: UUID="NegVJw-3XZn-BJeZ-NfKW-VROQ-roSa-LcKgoy" TYPE="LVM2_member"
/dev/mapper/centos-root: UUID="59c5d6b6-e34d-4149-b28a-8a3b9c32536d" TYPE="xfs"
/dev/sdc: UUID="h65OWm-ELDd-R5pO-HaQ0-Ejjs-cUWn-8Ejcpq" TYPE="LVM2_member"
/dev/mapper/centos-swap: UUID="25f54a98-e472-4438-9641-eac952a46e3e" TYPE="swap"
/dev/rbd0: UUID="911aadb8-bbf4-48d2-a62d-d86886af79dc" TYPE="ext4"
# 扩它
[root@node1 ~]# resize2fs /dev/rbd0
```

2.5、RBD数据写入流程



2.6、解决告警排查

```
# 1、发现问题
[root@node1 ~]# ceph -s
  cluster:
    id:     081dc49f-2525-4aaa-a56d-89d641cef302
    health: HEALTH_WARN
            application not enabled on 1 pool(s)
  services:
    mon: 3 daemons, quorum node1,node2,node3 (age 3h)
    mgr: node2(active, since 3h), standbys: node3, node1
    osd: 3 osds: 3 up (since 3h), 3 in (since 13h)

  data:
    pools:   1 pools, 128 pgs
    objects: 22 objects, 38 MiB
    usage:   3.1 GiB used, 57 GiB / 60 GiB avail
    pgs:     128 active+clean

# 2、通过他提供的命令查看
[root@node1 ~]# ceph health detail
HEALTH_WARN application not enabled on 1 pool(s)
POOL_APP_NOT_ENABLED application not enabled on 1 pool(s)
    application not enabled on pool 'ceph-demo'   # 看这句
    use 'ceph osd pool application enable <pool-name> <app-name>', where <app-name> is 'cephfs', 'rbd', 'rgw', or freeform for custom applications.       # 这句是提示你怎么操作（就是把这个资源类型进行分类就行）

# 3、解决命令
[root@node1 ~]# ceph osd pool application enable ceph-demo rbd
enabled application 'rbd' on pool 'ceph-demo'

# 4、再次查看
[root@node1 ~]# ceph -s
  cluster:
    id:     081dc49f-2525-4aaa-a56d-89d641cef302
    health: HEALTH_OK
```

CEPH之对象存储

一、官方文档

```
http://docs.ceph.org.cn/
http://docs.ceph.org.cn/radosgw/
```

二、安装 CEPH 对象网关

自从 firefly (v0.80) 版本开始，Ceph 对象网关运行在 Civetweb 上（已经集成进守护进程 `ceph-radosgw` ），而不再是 Apache 和 FastCGI 之上。使用 Civetweb简化了Ceph对象网关的安装和配置。

```
# 操作文档
http://docs.ceph.org.cn/install/install-ceph-gateway/
```

2.1、部署

```
# 1、在管理节点的工作目录下，给 Ceph 对象网关节点安装Ceph对象所需的软件包
[root@node1 ~]# yum install -y ceph-radosgw  # 之前在三个节点都安装过了

# 2、部署rgw
[root@node1 ~]# cd /app/ceph-deploy/ceph-deploy/
[root@node1 ceph-deploy]# ceph-deploy rgw create node1

# 3、报错解决（查看日志）
[root@node1 ceph-deploy]# tail -f /var/log/ceph/ceph-client.rgw.node1.log
###
ceph.conf 配置文件加上
mon_max_pg_per_osd = 1000
###

# 4、推送配置文件，然后重启
[root@node1 ceph-deploy]# ceph-deploy  --overwrite-conf  admin node1 node2 node3  #(node1执行)
[root@node1 ceph-deploy]# sudo systemctl restart ceph.target      #  （这个三个节点都重启）

# 5、端口检查
[root@node1 ceph-deploy]# ss -ntl | grep 7480
LISTEN     0      128          *:7480                     *:*
LISTEN     0      128         :::7480                    :::*

# 6、curl一下（出现一下情况是正常的了）
[root@node1 ceph-deploy]# curl http://node1:7480
<?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner><ID>anonymous</ID><DisplayName></DisplayName></Owner><Buckets></Buckets></ListAllMyBucketsResult
```

2.2、在任何时候如果你遇到麻烦，而你也想重新来一次，执行下面的命令来清除配置

```
ceph-deploy purge <gateway-node1> [<gateway-node2>]
ceph-deploy purgedata <gateway-node1> [<gateway-node2>]
```

2.3、修改网关默认端口

```
# 1、在ceph.conf 末尾加上
[root@node1 ceph-deploy]# vim ceph.conf
[client.rgw.node1]
rgw_frontends = "civetweb port=80"

# 2、推送文件
[root@node1 ceph-deploy]# ceph-deploy --overwrite-conf config push node1 node2 node3

# 3、重启
[root@node1 ceph-deploy]# systemctl restart ceph.target      #  （这个三个节点都重启）

# 4、测试是否更改成功
[root@node1 ceph-deploy]# curl http://node1:80
<?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult
xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner><ID>anonymous</ID><DisplayName>
</DisplayName></Owner><Buckets></Buckets></ListAllMyBucketsResult>[root@node1
```

2.4、使用对象存储

为了使用 REST 接口，首先需要为S3接口创建一个初始 Ceph 对象网关用户。然后，为 Swift 接口创建一个子用户。然后你需要验证创建的用户是否能够访问网关。

2.4.1、创建用户,在 `gateway host` 上执行下面的命令

```
[root@node3 ~]# radosgw-admin user create --uid="ceph-s3-user" --display-name="Ceph S3
User Demo"
{
    "user_id": "ceph-s3-user",
    "display_name": "Ceph S3 User Demo",
    "email": "",
    "suspended": 0,
    "max_buckets": 1000,
    "subusers": [],
    "keys": [
        {     # 这三个信息后面会要，很重要
            "user": "ceph-s3-user",
            "access_key": "7W1JL897CI54324HN4GH",
            "secret_key": "pBqTKmOTBOPaCulchBzusudlIFJrlKqpH7L0mKcE"
        }
    ],
    "swift_keys": [],
    "caps": [],
    "op_mask": "read, write, delete",
    "default_placement": "",
    "default_storage_class": "",
    "placement_tags": [],
    "bucket_quota": {
        "enabled": false,
        "check_on_raw": false,
        "max_size": -1,
        "max_size_kb": 0,
        "max_objects": -1
    },
    "user_quota": {
        "enabled": false,
        "check_on_raw": false,
        "max_size": -1,
        "max_size_kb": 0,
        "max_objects": -1
    },
    "temp_url_keys": [],
    "type": "rgw",
    "mfa_ids": []
}

# 其他命令
[root@node3 ~]# radosgw-admin user list
[
    "ceph-s3-user"
]
[root@node3 ~]# radosgw-admin user info  --uid   ceph-s3-user
```

2.4.3、测试 S3 访问

为了验证 S3 访问，你需要编写并运行一个 Python 测试脚本。S3 访问测试脚本将连接 `radosgw`, 新建一个新的 bucket 并列出所有的 buckets。 `aws_access_key_id` 和 `aws_secret_access_key` 的值来自于命令 `radosgw_admin` 的返回值 `access_key` 和 `secret_key`

执行下面的步骤:

1. 你需要安装 `python-boto` 包:

   ```
   sudo yum install python-boto
   ```

2. 新建 Python 脚本文件:

   ```
   vi s3test.py
   ```

3. 将下面的内容添加到文件中:

   ```
   import boto
   import boto.s3.connection

   access_key = '7W1JL897CI54324HN4GH'
   secret_key = 'pBqTKmOTBOPaCulchBzusudlIFJrlKqpH7L0mKcE+ZDA'
   conn = boto.connect_s3(
   ```

```
    aws_access_key_id = access_key,
    aws_secret_access_key = secret_key,
    host = '{hostname}', port = {port},
    is_secure=False, calling_format =
boto.s3.connection.OrdinaryCallingFormat(),
    )

bucket = conn.create_bucket('my-new-bucket')
  for bucket in conn.get_all_buckets():
      print "{name}".format(
          name = bucket.name,
          created = bucket.creation_date,
 )
```

将 `{hostname}` 替换为你配置了网关服务的节点的主机名。比如 `gateway host`. 将 {port} 替换为 Civetweb 所使用的端口。

4. 运行脚本:

```
python s3test.py
```

输出类似下面的内容:

```
my-new-bucket 2015-02-16T17:09:10.000Z
```

OSD纵向扩容

一、纵向扩容（增加磁盘块数）

```
# 1、查看节点磁盘个数
[root@node1 ~]# cd /app/ceph-deploy/ceph-deploy/
[root@node1 ceph-deploy]# ceph-deploy disk list node1

# 2、清理分区表，也可以手动dd
[root@node1 ceph-deploy]# ceph-deploy disk zap node1 /dev/sdb

# 3、纵向扩容（增加磁盘块数）
[root@node1 ceph-deploy]# ceph-deploy osd create node1 --data /dev/sdb
```

CephFS文件储存

一、官方文档

```
http://docs.ceph.org.cn/
http://docs.ceph.org.cn/cephfs/

# 操作文档
```

二、部署MDS集群

```
# 1、创建三个节点
[root@node1 ~]# cd /app/ceph-deploy/ceph-deploy/
[root@node1 ceph-deploy]# ceph-deploy mds create node1 node2 node3

# 2、查看集群状态
[root@node1 ceph-deploy]# ceph -s
  cluster:
    id:     081dc49f-2525-4aaa-a56d-89d641cef302
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum node1,node2,node3 (age 25m)
    mgr: node3(active, since 25m), standbys: node1, node2
    mds:  3 up:standby  # 现在没有文件系统，所以是standby的状态
    osd: 3 osds: 3 up (since 25m), 3 in (since 45m)
    rgw: 2 daemons active (node1, node2)
```

三、部署CephFS文件系统

3.1、创建CephFS文件系统

```
官方文档：http://docs.ceph.org.cn/cephfs/createfs/
```

一个 Ceph 文件系统需要至少两个 RADOS 存储池，一个用于数据、一个用于元数据。配置这些存储池时需考虑：

- 为元数据存储池设置较高的副本水平，因为此存储池丢失任何数据都会导致整个文件系统失效。
- 为元数据存储池分配低延时存储器（像 SSD ），因为它会直接影响到客户端的操作延时。

关于存储池的管理请参考 *存储池* 。例如，要用默认设置为文件系统创建两个存储池，你可以用下列命令：

```
# 1、创建2个pool
[root@node1 ceph-deploy]# ceph osd pool create cephfs_data 8 8
pool 'cephfs_data' created
[root@node1 ceph-deploy]# ceph osd pool create cephfs_metadata 8 8
pool 'cephfs_metadata' created
[root@node1 ceph-deploy]# ceph osd lspools
1 cephfs_data
2 cephfs_metadata
```

```
# 2、创建CephFS
[root@node1 ceph-deploy]# ceph fs new cephfs-demo cephfs_metadata cephfs_data
new fs with metadata pool 7 and data pool 6
[root@node1 ceph-deploy]# ceph fs ls
name: cephfs-demo, metadata pool: cephfs_metadata, data pools: [cephfs_data ]

# 3、查看集群状态
[root@node1 ceph-deploy]# ceph -s
  cluster:
    id:     081dc49f-2525-4aaa-a56d-89d641cef302
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum node1,node2,node3 (age 35m)
    mgr: node3(active, since 36m), standbys: node1, node2
    mds: cephfs-demo:1 {0=node1=up:active} 2 up:standby   # 变成了一个active
    osd: 3 osds: 3 up (since 35m), 3 in (since 55m)
    rgw: 2 daemons active (node1, node2)
```

3.2、用内核驱动挂载 CEPH 文件系统

要挂载 Ceph 文件系统，如果你知道监视器 IP 地址可以用 `mount` 命令、或者用 `mount.ceph` 工具来自动解析监视器 IP 地址。例如：

```
sudo mkdir /mnt/mycephfs
sudo mount -t ceph 192.168.1.129:6789:/ /mnt/mycephfs
```

要挂载启用了 `cephx` 认证的 Ceph 文件系统，你必须指定用户名、密钥。

```
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs -o
name=admin,secret=AQATSKdNGBnwLhAAnNDKnH65FmVKpXZJVasUeQ==

# secret 可以不写
[root@node1 ~]# sudo mount -t ceph 192.168.1.129:6789:/ /mnt/mycephfs -o name=admin

[root@node1 ~]# cd /mnt/mycephfs
[root@node1 mycephfs]# ll
total 0
[root@node1 mycephfs]# echo aaa > aaa
[root@node1 mycephfs]# ls
aaa
```

前述用法会把密码遗留在 Bash 历史里，更安全的方法是从文件读密码。例如：

```
sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs -o
name=admin,secretfile=/etc/ceph/admin.secret
```

关于 cephx 参见认证。

要卸载 Ceph 文件系统，可以用 `unmount` 命令，例如：

```
sudo umount /mnt/mycephfs
```

OSD删除+Ceph守护服务进程+Ceph服务日志分析

```
# 1、性能查看
[root@node1 ceph-deploy]# ceph osd perf
osd commit_latency(ms) apply_latency(ms)
  0                  0                 0
  2                  0                 0
  1                  0                 0

# 2、模拟某块盘GG了
[root@node3 ~]# systemctl stop ceph-osd@2

# 3、out
[root@node1 ~]# ceph osd out osd.2

# 4、crush rm
[root@node1 ~]# ceph osd crush rm osd.2

# 5、rm
[root@node1 ~]# ceph osd rm osd.2

# 6、auth rm
[root@node1 ~]# ceph osd auth rm osd.2
```

Ceph守护服务进程

```
# 1、管理所有守护进程
systemctl start ceph.target
systemctl start ceph-osd@1

# 2、服务类型分类
systemctl start ceph-osd.target
systemctl start ceph-mon.target
systemctl start ceph-mds.target

# 3、根据id区分、主机名区分
systemctl start ceph-osd@1
systemctl start ceph-mon@node1
systemctl start ceph-mds@node2
```

Ceph服务日志分析

```
# 1、日志目录、每个守护进程对应一个log
[root@node1 ceph]# cd /var/log/ceph/
[root@node1 ceph]# ll
total 9852
-rw------- 1 ceph ceph  121188 Jan 24 20:40 ceph.audit.log
-rw-r--r-- 1 ceph ceph   32009 Jan 24 18:44 ceph-client.rgw.node1.log
-rw------- 1 ceph ceph 2612353 Jan 24 20:53 ceph.log
-rw-r--r-- 1 ceph ceph    2403 Jan 24 19:31 ceph-mds.node1.log
-rw-r--r-- 1 ceph ceph   69137 Jan 24 20:41 ceph-mgr.node1.log
-rw-r--r-- 1 ceph ceph 2996628 Jan 24 20:53 ceph-mon.node1.log
-rw-r--r-- 1 ceph ceph 3835428 Jan 24 20:44 ceph-osd.1.log
-rw-r--r-- 1 root ceph  271058 Jan 24 19:57 ceph-volume.log
-rw-r--r-- 1 root ceph   49821 Jan 24 18:38 ceph-volume-systemd.log
```

RBD介绍

一、RBD回收

```
# 1、创建一块盘
[root@node1 ~]# rbd create ceph-demo/ceph-trash.img --size 1G

# 2、info
[root@node1 ~]# rbd info ceph-demo/ceph-trash.img
rbd image 'ceph-trash.img':
        size 1 GiB in 256 objects
        order 22 (4 MiB objects)
        snapshot_count: 0
        id: 283b35df6c430
        block_name_prefix: rbd_data.283b35df6c430
        format: 2
        features: layering, exclusive-lock, object-map, fast-diff, deep-flatten
        op_features:
        flags:
        create_timestamp: Mon Jan 25 17:46:29 2021
        access_timestamp: Mon Jan 25 17:46:29 2021
        modify_timestamp: Mon Jan 25 17:46:29 2021

# 3、删除（如果这样删，直接就没了）
[root@node1 ~]# rbd rm ceph-demo/ceph-trash.img
Removing image: 100% complete...done.

# 4、回收站查看，也是没有的
[root@node1 ~]# rbd -p  ceph-demo trash ls
```

```
# 5、再次创建，然后通过回收站的形式删除
[root@node1 ~]# rbd create ceph-demo/ceph-trash.img --size 1G
--expires-at  指定过期时间
[root@node1 ~]# rbd trash  move ceph-demo/ceph-trash.img --expires-at 20201126

# 6、查看
[root@node1 ~]# rbd -p  ceph-demo trash ls
2cff636e77644（回收就通过这个id回收） ceph-trash.img

# 7、回收操作
[root@node1 ~]# rbd trash restore -p ceph-demo 2cff636e77644
[root@node1 ~]# rbd -p  ceph-demo trash ls
[root@node1 ~]# rbd -p  ceph-demo ls  # 恢复
ceph-trash.img
rbd-demo.img
```

二、RBD快照

```
也是一种备份的形式
[root@node1 ~]# rbd create ceph-demo/rbd-test.img --image-feature layering --size 1G

[root@node1 ~]# rbd info  ceph-demo/rbd-test.img
rbd image 'rbd-test.img':
        size 1 GiB in 256 objects
        order 22 (4 MiB objects)
        snapshot_count: 0
        id: 2d00bff5c0ac4
        block_name_prefix: rbd_data.2d00bff5c0ac4
        format: 2
        features: layering
        op_features:
        flags:
        create_timestamp: Mon Jan 25 18:14:45 2021
        access_timestamp: Mon Jan 25 18:14:45 2021
        modify_timestamp: Mon Jan 25 18:14:45 2021

# 3、映射到本地文件系统，然后挂载起来
[root@node1 ~]# rbd device map  ceph-demo/rbd-test.img
/dev/rbd0
[root@node1 ~]# mkfs.ext4  /dev/rbd0
[root@node1 ~]# mkdir /mnt/test1
[root@node1 ~]# mount /dev/rbd0 /mnt/test1

# 4、进到这里问价夹，创建一些文件，然后打快照，删除原来的，再恢复
[root@node1 ~]# cd /mnt/test1; echo 111 > test

# 5、打快照
```

```
[root@node1 ~]# rbd snap create ceph-demo/rbd-test.img@snap_2021125
[root@node1 ~]# rbd snap ls ceph-demo/rbd-test.img
SNAPID   NAME         SIZE        PROTECTED TIMESTAMP
   4     snap_2021125 1 GiB     Mon Jan 25 19:03:38 2021

# 6、误删数据、然后恢复数据（恢复之前的快照）
[root@node1 test1]# rm -rf test # 模拟删除
[root@node1 test1]# rbd snap rollback ceph-demo/rbd-test.img@snap_2021125   # 回滚
Rolling back to snapshot: 100% complete...done.

# 7、查看文件是否恢复（显然没有，因为文件回滚的时候是离线的）。得卸载这个盘，再挂载才有
[root@node1 test1]# ll
[root@node1 ~]# umount /mnt/test1/
[root@node1 ~]# mount /dev/rbd0  /mnt/test1/

# 8、克隆快照、父子关系解除
```

三、RBD备份与恢复

```
# 1、查看现在有的快照
[root@node1 ~]# rbd snap ls ceph-demo/rbd-test.img
SNAPID   NAME         SIZE        PROTECTED TIMESTAMP
   4     snap_2021125 1 GiB     Mon Jan 25 19:03:38 2021

# 2、导出快照、备份快照
[root@node1 ~]# rbd export ceph-demo/rbd-test.img@snap_2021125  /app/rbd-test.img
Exporting image: 100% complete...done.

# 3、导入快照、恢复快照
[root@node1 ~]# rbd import /app/rbd-test.img ceph-demo/rbd-test-new.img
Importing image: 100% complete...done.
```

四、RBD增量备份与增量恢复

```
[root@node1 ~]# rbd export-diff ceph-demo/rbd-test.img@snap_2021125  /app/rbd-test1.img

Exporting image: 100% complete...done.
[root@node1 ~]# ll -h /app/rbd-test1.img
-rw-r--r-- 1 root root 5.8M Jan 25 19:54 /app/rbd-test1.img

[root@node1 ~]# rbd import-diff /app/rbd-test1.img ceph-demo/rbd-test-new.img

# 3、查看文件是否恢复（显然没有，因为文件回滚的时候是离线的）。得卸载这个盘，再挂载才有
[root@node1 test1]# ll
[root@node1 ~]# umount /mnt/test1/
[root@node1 ~]# mount /dev/rbd0  /mnt/test1/
```

CEPH高可用配置

`就是再增加一个对象网关rgw`

一、查看集群现在状态

```
[root@node1 ~]#  ceph -s
    rgw: 1 daemons active (node1)  # 现在只有一个rgw节点
```

二、部署第二个RGW

```
# 1、在管理节点的工作目录下，给 Ceph 对象网关节点安装Ceph对象所需的软件包
[root@node1 ~]# yum install -y ceph-radosgw  # 之前在三个节点都安装过了

# 2、部署rgw
[root@node1 ~]# cd /app/ceph-deploy/ceph-deploy/
[root@node1 ceph-deploy]# ceph-deploy rgw create node2

# 3、端口检查
[root@node1 ceph-deploy]# ss -ntl | grep 7480
LISTEN     0      128          *:7480                     *:*
LISTEN     0      128         :::7480                    :::*

# 4、curl一下（出现一下情况是正常的了）
[root@node1 ceph-deploy]# curl http://node1:7480
<?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult
xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner><ID>anonymous</ID><DisplayName>
</DisplayName></Owner><Buckets></Buckets></ListAllMyBucketsResult

# 5、查看集群状态
[root@node1 ceph-deploy]# ceph -s
    rgw: 2 daemons active (node1, node2)  # 已经2个rgw节点了
```

三、修改网关默认端口

```
# 1、在ceph.conf 末尾加上
[root@node1 ceph-deploy]# vim ceph.conf
[client.rgw.node2]  # 原本是有个[client.rgw.node1]的表示修改node的端口
rgw_frontends = "civetweb port=80"

# 2、推送文件
[root@node1 ceph-deploy]# ceph-deploy --overwrite-conf config push node1 node2 node3

# 3、重启
[root@node2 ceph-deploy]# systemctl restart ceph.target      # （重启node2）
```

```
# 4、测试是否更改成功
[root@node1 ceph-deploy]# curl http://node2:80
<?xml version="1.0" encoding="UTF-8"?><ListAllMyBucketsResult
xmlns="http://s3.amazonaws.com/doc/2006-03-01/"><Owner><ID>anonymous</ID><DisplayName>
</DisplayName></Owner><Buckets></Buckets></ListAllMyBucketsResult>[root@node1
```

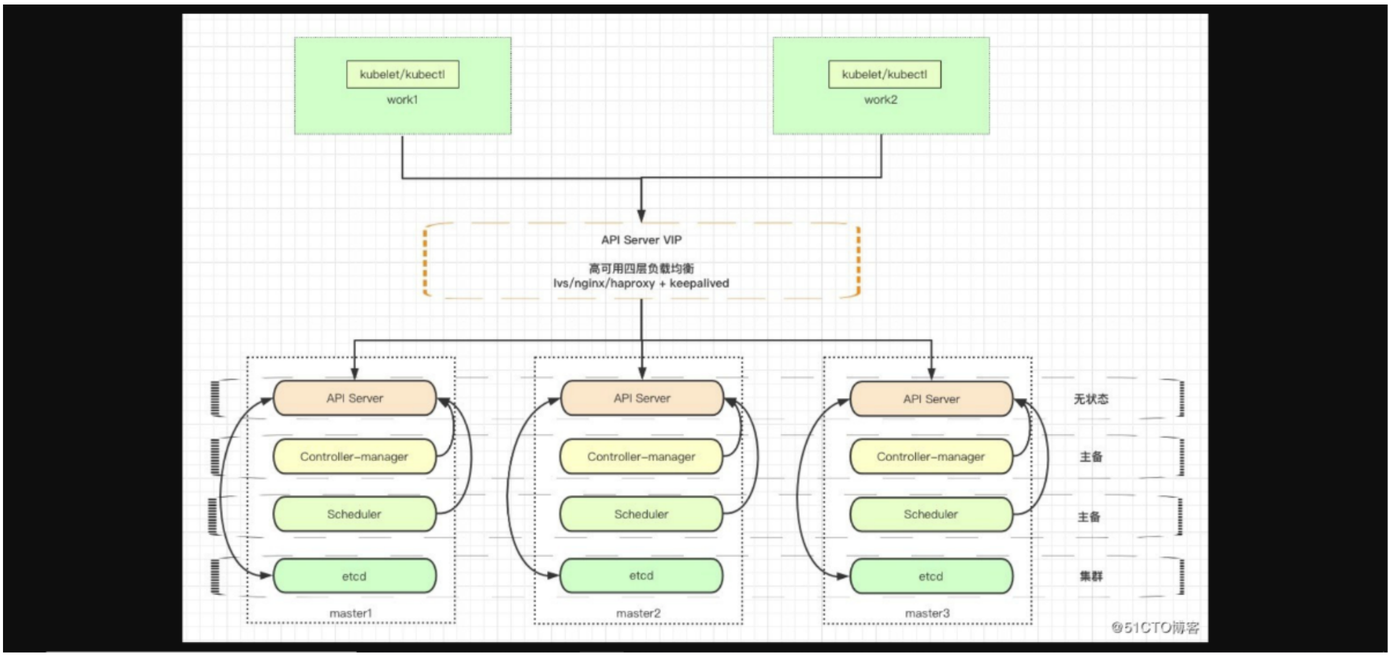
四、haproxy+keepalived构建RGW高可用集群

4.1、安装haproxy（node1、node2）

```
[root@node1 ~]# yum install haproxy -y
```

Ceph跟K8S进行集成

```
# github官网：
  https://github.com/kubernetes/examples/tree/master/volumes/cephfs/

# k8s官网：
  https://kubernetes.io/docs/concepts/storage/volumes/#cephfs

# 参考
  https://blog.51cto.com/tryingstuff/2386821
```



一、K8S连接Ceph步骤

1.1、首先得在kubernetes的主机上安装ceph客户端（要访问ceph的节点）

```
所有节点安装ceph-common
添加ceph的yum源：

[Ceph]
name=Ceph packages for $basearch
baseurl=https://mirrors.aliyun.com/ceph/rpm-mimic/el7/$basearch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://download.ceph.com/keys/release.asc

[Ceph-noarch]
name=Ceph noarch packages
baseurl=https://mirrors.aliyun.com/ceph/rpm-mimic/el7/noarch
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://download.ceph.com/keys/release.asc

[ceph-source]
name=Ceph source packages
baseurl=https://mirrors.aliyun.com/ceph/rpm-mimic/el7/SRPMS
enabled=1
gpgcheck=1
type=rpm-md
gpgkey=https://download.ceph.com/keys/release.asc
安装ceph-common：

yum install ceph-common -y
如果安装过程出现依赖报错，可以通过如下方式解决：

yum install -y yum-utils && \
yum-config-manager --add-repo https://dl.fedoraproject.org/pub/epel/7/x86_64/ && \
yum install --nogpgcheck -y epel-release && \
rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-EPEL-7 && \
rm -f /etc/yum.repos.d/dl.fedoraproject.org*

yum -y install ceph-common
```

1.2、配置ceph配置文件（不用也行）

```
将ceph配置文件拷贝到各个k8s的node节点
[root@node1 ~]# scp /etc/ceph k8s-node:/etc/
```

1.3、ceph新建pool、user、image（ceps集群上）

```
# 1、新建一个pool
[root@node1 ~]# ceph osd pool create kubernetes 64 64
pool 'kubernetes' created

# 2、创建一个名字为client.k8s的用户，且只能使用指定的pool（kubernetes）
[root@node1 ~]# ceph auth get-or-create client.k8s mon 'allow r' osd 'allow class-read object_prefix rbd_children,allow rwx pool=kubernetes'
[client.k8s]
        key = AQD+pg9gjreqMRAAtwq4dQnwX0kX4Vx6TueAJQ==

# 3、查看用户
[root@node1 ~]# ceph auth  ls
client.k8s
        key: AQD+pg9gjreqMRAAtwq4dQnwX0kX4Vx6TueAJQ==
        caps: [mon] allow r
        caps: [osd] allow class-read object_prefix rbd_children,allow rwx pool=kubernetes

# 4、对这个用户进行加密，后面的secret会用到
[root@node1 ~]# echo AQD+pg9gjreqMRAAtwq4dQnwX0kX4Vx6TueAJQ== | base64
QVFEK3BnOWdqcmVxTVJBQXR3cTRkUW53WDBrWDRWeDZUdWVBSlE9PQo=

# 5、新建一个块
[root@node1 ~]# rbd create -p kubernetes --image rbd.img --size 3G
[root@node1 ~]# rbd -p kubernetes ls
rbd.img

# 6、disable 模块
[root@node1 ~]# rbd feature disable  kubernetes/rbd1.img deep-flatten
[root@node1 ~]# rbd feature disable  kubernetes/rbd1.img fast-diff
[root@node1 ~]# rbd feature disable  kubernetes/rbd1.img object-map
[root@node1 ~]# rbd feature disable  kubernetes/rbd1.img exclusive-lock

# 7、查看mon地址
[root@node1 ~]# ceph mon dump
dumped monmap epoch 3
epoch 3
fsid 081dc49f-2525-4aaa-a56d-89d641cef302
last_changed 2021-01-24 02:09:24.404424
created 2021-01-24 02:00:36.999143
min_mon_release 14 (nautilus)
```

```
0: [v2:192.168.1.129:3300/0,v1:192.168.1.129:6789/0] mon.node1
1: [v2:192.168.1.130:3300/0,v1:192.168.1.130:6789/0] mon.node2
2: [v2:192.168.1.131:3300/0,v1:192.168.1.131:6789/0] mon.node3
```

二、volumes集成Ceph

```
在上面的步骤，我们已经在ceph集群用创建了我们所需的信息，现在在k8s集群中使用起来
# 1、新建目录
[root@master1 ~]# mkdir /app/ceph_test -p ;cd  /app/ceph_test

# 2、创建secret的yaml文件
[root@master1 ceph_test]# cat ceph-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: ceph-secret
type: "kubernetes.io/rbd"
data:
  key: QVFEK3BnOWdqcmVxTVJBQXR3cTRkUW53WDBrWDRWeDZUdWVBSlE9PQo=  # the base64-encoded string of the already-base64-encoded key `ceph auth get-key` outputs

# 3、create这个secret
[root@master1 ceph_test]# kubectl apply -f ceph-secret.yaml
secret/ceph-secret created
[root@master1 ceph_test]# kubectl get secret
NAME                  TYPE                                  DATA   AGE
ceph-secret           kubernetes.io/rbd                     1      4m1s

# 4、创建Pod文件
[root@master1 ceph_test]# cat cephfs.yaml
apiVersion: v1
kind: Pod
metadata:
  name: rbd-demo
spec:
  containers:
  - name: rbd-demo-nginx
    image: nginx
    volumeMounts:
    - mountPath: "/data"
      name: rbd-demo
  volumes:
  - name: rbd-demo
    rbd:
      monitors:        # 指定mom的集群地址
      - 192.168.1.129:6789
      - 192.168.1.130:6789
      - 192.168.1.131:6789
      pool: kubernetes # pool name
      image: rbd.img   # 块 name
      user: k8s        # 刚刚新建的用户名
      secretRef:
        name: ceph-secret

# 5、创建pod
[root@master1 ceph_test]# kubectl apply -f cephfs.yaml
pod/rbd-demo created

# 6、验证pod是否成功把块rbd.img挂载到容器中
[root@master1 ceph_test]# kubectl exec -it   rbd-demo -- bash
root@rbd-demo:/# df -h
Filesystem               Size  Used Avail Use% Mounted on
overlay                   47G  4.1G   43G   9% /
tmpfs                     64M     0   64M   0% /dev
tmpfs                    2.0G     0  2.0G   0% /sys/fs/cgroup
/dev/rbd0                2.9G  9.0M  2.9G   1% /data      # 已经可以看到成功挂载进来了
/dev/mapper/centos-root   47G  4.1G   43G   9% /etc/hosts
shm                       64M     0   64M   0% /dev/shm
tmpfs                    2.0G   12K  2.0G   1%
/run/secrets/kubernetes.io/serviceaccount
tmpfs                    2.0G     0  2.0G   0% /proc/acpi
tmpfs                    2.0G     0  2.0G   0% /proc/scsi
tmpfs                    2.0G     0  2.0G   0% /sys/firmware
# 然后创建一些文件
root@rbd-demo:/# cd /data/
root@rbd-demo:/data# echo aaa > test

# 6、Pod是创建在node01节点上（k8s的node01）
[root@node1 ~]# df -h | grep rbd
/dev/rbd0                2.9G  9.1M  2.9G   1%
/var/lib/kubelet/plugins/kubernetes.io/rbd/mounts/kubernetes-image-rbd.img
```

三、PV、PVC集成Cpeh

```
前提：参考步骤1.3，新建pool、image、用户、还有secret的创建
```

3.1、创建PV

```
# 1、创建PV
[root@master1 ceph_test]# cat ceph-rbd-pv.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-rbd-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  rbd:
    monitors:        # 指定mom的集群地址
      - 192.168.1.129:6789
      - 192.168.1.130:6789
      - 192.168.1.131:6789
    pool: kubernetes # pool name
    image: rbd1.img   # 块 name
    user: k8s        # 刚刚新建的用户名
    secretRef:
      name: ceph-secret
    fsType: ext4
    readOnly: false
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: rbd

# 2、创建PV
[root@master1 ceph_test]# kubectl apply -f ceph-rbd-pv.yaml
persistentvolume/ceph-rbd-pv created

# 3、查看PV
[root@master1 ceph_test]# kubectl get pv ceph-rbd-pv
NAME          CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM
STORAGECLASS   REASON   AGE
ceph-rbd-pv   1Gi        RWO            Recycle          Available                  rbd
             51s
```

3.2、创建PVC

```
# 1、创建PVC
[root@master1 ceph_test]# cat ceph-rbd-pvc.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ceph-rbd-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  volumeName: ceph-rbd-pv
  resources:
    requests:
      storage: 1Gi
  storageClassName: rbd

# 2、apply PVC
[root@master1 ceph_test]# kubectl apply -f  ceph-rbd-pvc.yaml
persistentvolumeclaim/ceph-rbd-pv-claim created
```

```
# 3、查看PVC
[root@master1 ceph_test]# kubectl get pvc ceph-rbd-pv-claim
NAME                STATUS   VOLUME        CAPACITY   ACCESS MODES   STORAGECLASS   AGE
ceph-rbd-pv-claim   Bound    ceph-rbd-pv   1Gi        RWO            rbd            30s
```

3.3、Pod使用PVC

```
# 1、创建一个Pod的yaml文件
[root@master1 ceph_test]# cat pod-demo.yaml
apiVersion: v1
kind: Pod
metadata:
  name: rbd-nginx
spec:
  containers:
    - image: nginx
      name: rbd-rw
      ports:
      - name: www
        protocol: TCP
        containerPort: 80
      volumeMounts:
      - name: rbd-pvc
        mountPath: /mnt
  volumes:
    - name: rbd-pvc
      persistentVolumeClaim:
        claimName: ceph-rbd-pv-claim

# 2、apply pod
[root@master1 ceph_test]# kubectl apply -f  pod-demo.yaml
pod/rbd-nginx created

# 3、查看Pod
[root@master1 ceph_test]# kubectl get pod rbd-nginx
NAME        READY   STATUS    RESTARTS   AGE
rbd-nginx   1/1     Running   0          22s
```

四、StorageClass集成Ceph（最终的方案）

```
CEPH官网:
  https://docs.ceph.com/en/latest/rbd/rbd-kubernetes/?highlight=CSI
```

CSI架构图：

image-20210126163736834

4.1、集成步骤（ceph集群上）

```
# 1、创建池
# 默认情况下，Ceph块设备使用该rbd池。为Kubernetes卷存储创建一个池。确保您的Ceph集群正在运行，然后创建池。
[root@node1 ~]# ceph osd pool create kubernetes
[root@node1 ~]# rbd pool init kubernetes

# 2、设置CEPH客户端身份验证
ceph auth get-or-create client.k8s mon 'profile rbd' osd 'profile rbd pool=kubernetes' mgr 'profile rbd pool=kubernetes'
[client.kubernetes]
    key = AQD+pg9gjreqMRAAtwq4dQnwX0kX4Vx6TueAJQ==

# 3、查看CEPH- CSI CONFIGMAP
[root@node1 ~]# ceph mon dump
dumped monmap epoch 3
epoch 3
fsid 081dc49f-2525-4aaa-a56d-89d641cef302
last_changed 2021-01-24 02:09:24.404424
created 2021-01-24 02:00:36.999143
min_mon_release 14 (nautilus)
0: [v2:192.168.1.129:3300/0,v1:192.168.1.129:6789/0] mon.node1
1: [v2:192.168.1.130:3300/0,v1:192.168.1.130:6789/0] mon.node2
2: [v2:192.168.1.131:3300/0,v1:192.168.1.131:6789/0] mon.node3
```

4.2、k8s节点上

```
# 1、创建ConfigMap
[root@master1 ~]# mkdir /app/csi -p; cd /app/csi
[root@master1 csi]# cat csi-config-map.yaml
apiVersion: v1
kind: ConfigMap
data:
  config.json: |-
    [
      {
        "clusterID": "081dc49f-2525-4aaa-a56d-89d641cef302",  # ceph mon dump获取
        "monitors": [
          "192.168.1.129:6789",
          "192.168.1.130:6789",
          "192.168.1.131:6789"
        ]
      }
    ]
```

```
metadata:
  name: ceph-csi-config

# 2、apply且查看这个ConfigMap
[root@master1 csi]# kubectl apply -f csi-config-map.yaml
configmap/ceph-csi-config created
[root@master1 csi]# kubectl get cm ceph-csi-config
NAME              DATA   AGE
ceph-csi-config   1      19s

# 3、生成CEPH- CSI CEPHX秘密
[root@master1 csi]# cat csi-rbd-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: csi-rbd-secret
  namespace: default
stringData:
  userID: k8s  # ceph集群创建的username    # 这里用admin的要不然下面会报错!！！！！！！！！！！！！
  userKey: AQD+pg9gjreqMRAAtwq4dQnwX0kX4Vx6TueAJQ== # 这个key不需要base64加密!！！！！！！！！！！！！！！！！

# 4、创建且查看
[root@master1 csi]# kubectl apply -f  csi-rbd-secret.yaml
secret/csi-rbd-secret created
[root@master1 csi]# kubectl get secret csi-rbd-secret
NAME             TYPE     DATA   AGE
csi-rbd-secret   Opaque   2      14s
```

4.2.0、再创建一个ConfigMap

```
[root@master1 csi]# cat kms-config.yaml
apiVersion: v1
kind: ConfigMap
data:
  config.json: |-
    {
      "vault-test": {
        "encryptionKMSType": "vault",
        "vaultAddress": "http://vault.default.svc.cluster.local:8200",
        "vaultAuthPath": "/v1/auth/kubernetes/login",
        "vaultRole": "csi-kubernetes",
        "vaultPassphraseRoot": "/v1/secret",
        "vaultPassphrasePath": "ceph-csi/",
        "vaultCAVerify": "false"
      }
    }
metadata:
  name: ceph-csi-encryption-kms-config

# 创建、查看
[root@master1 csi]# kubectl apply -f  kms-config.yaml
configmap/ceph-csi-encryption-kms-config created
[root@master1 csi]# kubectl get cm
NAME                             DATA   AGE
ceph-csi-config                  1      59m
ceph-csi-encryption-kms-config   1      15s
```

4.2.1、配置CEPH- CSI插件（k8s集群上）

创建所需的ServiceAccount和RBAC ClusterRole / ClusterRoleBinding Kubernetes对象。不一定需要这些对象为您定制Kubernetes环境中，因此可作为-从ceph- CSI 部署YAMLs：

```
$ kubectl apply -f https://raw.githubusercontent.com/ceph/ceph-
csi/master/deploy/rbd/kubernetes/csi-provisioner-rbac.yaml
$ kubectl apply -f https://raw.githubusercontent.com/ceph/ceph-
csi/master/deploy/rbd/kubernetes/csi-nodeplugin-rbac.yaml
# 1、文件查看
[root@master1 csi]# cat csi-nodeplugin-rbac.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: rbd-csi-nodeplugin
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rbd-csi-nodeplugin
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get"]
  # allow to read Vault Token and connection options from the Tenants namespace
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rbd-csi-nodeplugin
subjects:
  - kind: ServiceAccount
    name: rbd-csi-nodeplugin
    namespace: default
roleRef:
  kind: ClusterRole
  name: rbd-csi-nodeplugin
  apiGroup: rbac.authorization.k8s.io
-------------------------------------------------------------------------
[root@master1 csi]# cat csi-provisioner-rbac.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: rbd-csi-provisioner

---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rbd-external-provisioner-runner
rules:
  - apiGroups: [""]
    resources: ["nodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get", "list"]
  - apiGroups: [""]
    resources: ["events"]
    verbs: ["list", "watch", "create", "update", "patch"]
  - apiGroups: [""]
    resources: ["persistentvolumes"]
    verbs: ["get", "list", "watch", "create", "update", "delete", "patch"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "update"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims/status"]
    verbs: ["update", "patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["storageclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshots"]
    verbs: ["get", "list"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshotcontents"]
    verbs: ["create", "get", "list", "watch", "update", "delete"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshotclasses"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["storage.k8s.io"]
```

```
    resources: ["volumeattachments"]
    verbs: ["get", "list", "watch", "update", "patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["volumeattachments/status"]
    verbs: ["patch"]
  - apiGroups: ["storage.k8s.io"]
    resources: ["csinodes"]
    verbs: ["get", "list", "watch"]
  - apiGroups: ["snapshot.storage.k8s.io"]
    resources: ["volumesnapshotcontents/status"]
    verbs: ["update"]
---
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rbd-csi-provisioner-role
subjects:
  - kind: ServiceAccount
    name: rbd-csi-provisioner
    namespace: default
roleRef:
  kind: ClusterRole
  name: rbd-external-provisioner-runner
  apiGroup: rbac.authorization.k8s.io

---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  # replace with non-default namespace name
  namespace: default
  name: rbd-external-provisioner-cfg
rules:
  - apiGroups: [""]
    resources: ["configmaps"]
    verbs: ["get", "list", "watch", "create", "update", "delete"]
  - apiGroups: ["coordination.k8s.io"]
    resources: ["leases"]
    verbs: ["get", "watch", "list", "delete", "update", "create"]

---
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: rbd-csi-provisioner-role-cfg
  # replace with non-default namespace name
  namespace: default
subjects:
  - kind: ServiceAccount
```

```
    name: rbd-csi-provisioner
    # replace with non-default namespace name
    namespace: default
roleRef:
  kind: Role
  name: rbd-external-provisioner-cfg
  apiGroup: rbac.authorization.k8s.io



# 2、创建这2个yaml
[root@master1 csi]# kubectl apply -f csi-nodeplugin-rbac.yaml -f csi-provisioner-rbac.yaml
serviceaccount/rbd-csi-nodeplugin created
clusterrole.rbac.authorization.k8s.io/rbd-csi-nodeplugin created
clusterrolebinding.rbac.authorization.k8s.io/rbd-csi-nodeplugin created
serviceaccount/rbd-csi-provisioner created
clusterrole.rbac.authorization.k8s.io/rbd-external-provisioner-runner created
clusterrolebinding.rbac.authorization.k8s.io/rbd-csi-provisioner-role created
role.rbac.authorization.k8s.io/rbd-external-provisioner-cfg created
rolebinding.rbac.authorization.k8s.io/rbd-csi-provisioner-role-cfg created
```

4.2.2、创建ceph csi供应器和节点插件

最后，创建ceph csi供应器和节点插件。随着的可能是个例外ceph- CSI集装箱发行版本，不一定需要这些对象为您定制Kubernetes环境中，因此可作为-从ceph- CSI部署YAMLs：

```
$ wget https://raw.githubusercontent.com/ceph/ceph-csi/master/deploy/rbd/kubernetes/csi-rbdplugin-provisioner.yaml
$ kubectl apply -f csi-rbdplugin-provisioner.yaml
$ wget https://raw.githubusercontent.com/ceph/ceph-csi/master/deploy/rbd/kubernetes/csi-rbdplugin.yaml
$ kubectl apply -f csi-rbdplugin.yaml
# 1、文件查看
[root@master1 csi]# cat csi-rbdplugin.yaml
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: csi-rbdplugin
spec:
  selector:
    matchLabels:
      app: csi-rbdplugin
  template:
    metadata:
      labels:
        app: csi-rbdplugin
    spec:
      serviceAccount: rbd-csi-nodeplugin
      hostNetwork: true
      hostPID: true
      # to use e.g. Rook orchestrated cluster, and mons' FQDN is
      # resolved through k8s service, set dns policy to cluster first
      dnsPolicy: ClusterFirstWithHostNet
      containers:
        - name: driver-registrar
          # This is necessary only for systems with SELinux, where
          # non-privileged sidecar containers cannot access unix domain socket
          # created by privileged CSI driver container.
          securityContext:
            privileged: true
          image: k8s.gcr.io/sig-storage/csi-node-driver-registrar:v2.0.1
          args:
            - "--v=5"
            - "--csi-address=/csi/csi.sock"
            - "--kubelet-registration-
path=/var/lib/kubelet/plugins/rbd.csi.ceph.com/csi.sock"
          env:
            - name: KUBE_NODE_NAME
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
            - name: registration-dir
              mountPath: /registration
        - name: csi-rbdplugin
          securityContext:
            privileged: true
            capabilities:
              add: ["SYS_ADMIN"]
            allowPrivilegeEscalation: true
          # for stable functionality replace canary with latest release version
          image: quay.io/cephcsi/cephcsi:canary
          args:
            - "--nodeid=$(NODE_ID)"
            - "--type=rbd"
            - "--nodeserver=true"
            - "--endpoint=$(CSI_ENDPOINT)"
            - "--v=5"
            - "--drivername=rbd.csi.ceph.com"
            # If topology based provisioning is desired, configure required
            # node labels representing the nodes topology domain
            # and pass the label names below, for CSI to consume and advertise
            # its equivalent topology domain
            # - "--domainlabels=failure-domain/region,failure-domain/zone"
          env:
            - name: POD_IP
```

```
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
            - name: NODE_ID
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            # - name: POD_NAMESPACE
            #   valueFrom:
            #     fieldRef:
            #       fieldPath: spec.namespace
            # - name: KMS_CONFIGMAP_NAME
            #   value: encryptionConfig
            - name: CSI_ENDPOINT
              value: unix:///csi/csi.sock
          imagePullPolicy: "IfNotPresent"
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
            - mountPath: /dev
              name: host-dev
            - mountPath: /sys
              name: host-sys
            - mountPath: /run/mount
              name: host-mount
            - mountPath: /lib/modules
              name: lib-modules
              readOnly: true
            - name: ceph-csi-config
              mountPath: /etc/ceph-csi-config/
            - name: ceph-csi-encryption-kms-config
              mountPath: /etc/ceph-csi-encryption-kms-config/
            - name: plugin-dir
              mountPath: /var/lib/kubelet/plugins
              mountPropagation: "Bidirectional"
            - name: mountpoint-dir
              mountPath: /var/lib/kubelet/pods
              mountPropagation: "Bidirectional"
            - name: keys-tmp-dir
              mountPath: /tmp/csi/keys
        - name: liveness-prometheus
          securityContext:
            privileged: true
          image: quay.io/cephcsi/cephcsi:canary
          args:
            - "--type=liveness"
            - "--endpoint=$(CSI_ENDPOINT)"
            - "--metricsport=8680"
            - "--metricspath=/metrics"
            - "--polltime=60s"
            - "--timeout=3s"
          env:
            - name: CSI_ENDPOINT
              value: unix:///csi/csi.sock
            - name: POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
          imagePullPolicy: "IfNotPresent"
      volumes:
        - name: socket-dir
          hostPath:
            path: /var/lib/kubelet/plugins/rbd.csi.ceph.com
            type: DirectoryOrCreate
        - name: plugin-dir
          hostPath:
            path: /var/lib/kubelet/plugins
            type: Directory
        - name: mountpoint-dir
          hostPath:
            path: /var/lib/kubelet/pods
            type: DirectoryOrCreate
        - name: registration-dir
          hostPath:
            path: /var/lib/kubelet/plugins_registry/
            type: Directory
        - name: host-dev
          hostPath:
            path: /dev
        - name: host-sys
          hostPath:
            path: /sys
        - name: host-mount
          hostPath:
            path: /run/mount
        - name: lib-modules
          hostPath:
            path: /lib/modules
        - name: ceph-csi-config
          configMap:
            name: ceph-csi-config
        - name: ceph-csi-encryption-kms-config
          configMap:
            name: ceph-csi-encryption-kms-config
        - name: keys-tmp-dir
```

```
          emptyDir: {
            medium: "Memory"
          }
---
# This is a service to expose the liveness metrics
apiVersion: v1
kind: Service
metadata:
  name: csi-metrics-rbdplugin
  labels:
    app: csi-metrics
spec:
  ports:
    - name: http-metrics
      port: 8080
      protocol: TCP
      targetPort: 8680
  selector:
    app: csi-rbdplugin
-----------------------------------------------------------------------------------------
------
[root@master1 csi]# cat csi-rbdplugin-provisioner.yaml
kind: Service
apiVersion: v1
metadata:
  name: csi-rbdplugin-provisioner
  labels:
    app: csi-metrics
spec:
  selector:
    app: csi-rbdplugin-provisioner
  ports:
    - name: http-metrics
      port: 8080
      protocol: TCP
      targetPort: 8680

---
kind: Deployment
apiVersion: apps/v1
metadata:
  name: csi-rbdplugin-provisioner
spec:
  replicas: 3
  selector:
    matchLabels:
      app: csi-rbdplugin-provisioner
  template:
    metadata:
      labels:
        app: csi-rbdplugin-provisioner
    spec:
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                  - key: app
                    operator: In
                    values:
                      - csi-rbdplugin-provisioner
              topologyKey: "kubernetes.io/hostname"
      serviceAccount: rbd-csi-provisioner
      containers:
        - name: csi-provisioner
          image: k8s.gcr.io/sig-storage/csi-provisioner:v2.0.4
          args:
            - "--csi-address=$(ADDRESS)"
            - "--v=5"
            - "--timeout=150s"
            - "--retry-interval-start=500ms"
            - "--leader-election=true"
            #  set it to true to use topology based provisioning
            - "--feature-gates=Topology=false"
            # if fstype is not specified in storageclass, ext4 is default
            - "--default-fstype=ext4"
            - "--extra-create-metadata=true"
          env:
            - name: ADDRESS
              value: unix:///csi/csi-provisioner.sock
          imagePullPolicy: "IfNotPresent"
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: csi-snapshotter
          image: k8s.gcr.io/sig-storage/csi-snapshotter:v3.0.2
          args:
            - "--csi-address=$(ADDRESS)"
            - "--v=5"
            - "--timeout=150s"
            - "--leader-election=true"
          env:
            - name: ADDRESS
              value: unix:///csi/csi-provisioner.sock
          imagePullPolicy: "IfNotPresent"
          securityContext:
            privileged: true
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: csi-attacher
          image: k8s.gcr.io/sig-storage/csi-attacher:v3.0.2
          args:
            - "--v=5"
            - "--csi-address=$(ADDRESS)"
            - "--leader-election=true"
            - "--retry-interval-start=500ms"
          env:
            - name: ADDRESS
              value: /csi/csi-provisioner.sock
          imagePullPolicy: "IfNotPresent"
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: csi-resizer
          image: k8s.gcr.io/sig-storage/csi-resizer:v1.0.1
          args:
            - "--csi-address=$(ADDRESS)"
            - "--v=5"
            - "--timeout=150s"
            - "--leader-election"
            - "--retry-interval-start=500ms"
            - "--handle-volume-inuse-error=false"
          env:
            - name: ADDRESS
              value: unix:///csi/csi-provisioner.sock
          imagePullPolicy: "IfNotPresent"
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
        - name: csi-rbdplugin
          securityContext:
            privileged: true
            capabilities:
              add: ["SYS_ADMIN"]
          # for stable functionality replace canary with latest release version
          image: quay.io/cephcsi/cephcsi:canary
          args:
            - "--nodeid=$(NODE_ID)"
            - "--type=rbd"
            - "--controllerserver=true"
            - "--endpoint=$(CSI_ENDPOINT)"
            - "--v=5"
            - "--drivername=rbd.csi.ceph.com"
            - "--pidlimit=-1"
            - "--rbdhardmaxclonedepth=8"
            - "--rbdsoftmaxclonedepth=4"
          env:
            - name: POD_IP
              valueFrom:
                fieldRef:
                  fieldPath: status.podIP
            - name: NODE_ID
              valueFrom:
                fieldRef:
                  fieldPath: spec.nodeName
            # - name: POD_NAMESPACE
            #   valueFrom:
            #     fieldRef:
            #       fieldPath: spec.namespace
            # - name: KMS_CONFIGMAP_NAME
            #   value: encryptionConfig
            - name: CSI_ENDPOINT
              value: unix:///csi/csi-provisioner.sock
          imagePullPolicy: "IfNotPresent"
          volumeMounts:
            - name: socket-dir
              mountPath: /csi
            - mountPath: /dev
              name: host-dev
            - mountPath: /sys
              name: host-sys
            - mountPath: /lib/modules
              name: lib-modules
              readOnly: true
            - name: ceph-csi-config
              mountPath: /etc/ceph-csi-config/
            - name: ceph-csi-encryption-kms-config
              mountPath: /etc/ceph-csi-encryption-kms-config/
            - name: keys-tmp-dir
              mountPath: /tmp/csi/keys
        - name: csi-rbdplugin-controller
          securityContext:
            privileged: true
            capabilities:
              add: ["SYS_ADMIN"]
          # for stable functionality replace canary with latest release version
          image: quay.io/cephcsi/cephcsi:canary
          args:
            - "--type=controller"
            - "--v=5"
            - "--drivername=rbd.csi.ceph.com"
            - "--drivernamespace=$(DRIVER_NAMESPACE)"
          env:
            - name: DRIVER_NAMESPACE
              valueFrom:
```

```
            fieldRef:
              fieldPath: metadata.namespace
        imagePullPolicy: "IfNotPresent"
        volumeMounts:
          - name: ceph-csi-config
            mountPath: /etc/ceph-csi-config/
          - name: keys-tmp-dir
            mountPath: /tmp/csi/keys
      - name: liveness-prometheus
        image: quay.io/cephcsi/cephcsi:canary
        args:
          - "--type=liveness"
          - "--endpoint=$(CSI_ENDPOINT)"
          - "--metricsport=8680"
          - "--metricspath=/metrics"
          - "--polltime=60s"
          - "--timeout=3s"
        env:
          - name: CSI_ENDPOINT
            value: unix:///csi/csi-provisioner.sock
          - name: POD_IP
            valueFrom:
              fieldRef:
                fieldPath: status.podIP
        volumeMounts:
          - name: socket-dir
            mountPath: /csi
        imagePullPolicy: "IfNotPresent"
      volumes:
        - name: host-dev
          hostPath:
            path: /dev
        - name: host-sys
          hostPath:
            path: /sys
        - name: lib-modules
          hostPath:
            path: /lib/modules
        - name: socket-dir
          emptyDir: {
            medium: "Memory"
          }
        - name: ceph-csi-config
          configMap:
            name: ceph-csi-config
        - name: ceph-csi-encryption-kms-config
          configMap:
            name: ceph-csi-encryption-kms-config
        - name: keys-tmp-dir
```

```
          emptyDir: {
            medium: "Memory"
          }

# 2、创建这2个文件
[root@master1 csi]# kubectl apply -f csi-rbdplugin-provisioner.yaml -f csi-
rbdplugin.yaml
service/csi-rbdplugin-provisioner created
deployment.apps/csi-rbdplugin-provisioner created
daemonset.apps/csi-rbdplugin created
service/csi-metrics-rbdplugin created
```

4.2.3、使用CEPH块设备

```
# 1、会用到这个secret
[root@master1 ~]# kubectl  get secret
NAME                              TYPE                                  DATA   AGE
csi-rbd-secret                    Opaque                                2      99m

# 2、创建StorageClass的yaml文件
[root@master1 csi]# cat csi-rbd-sc.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
   name: csi-rbd-sc
provisioner: rbd.csi.ceph.com   # 驱动
parameters:
   clusterID: b9127830-b0cc-4e34-aa47-9d1a2e9949a8  # ceph集群id (ceph -s 查看)
   pool: kubernetes             # 访问的pool name
   csi.storage.k8s.io/provisioner-secret-name: csi-rbd-secret  # 默认是这些个名字
   csi.storage.k8s.io/provisioner-secret-namespace: default    # 默认在default namespace
   csi.storage.k8s.io/node-stage-secret-name: csi-rbd-secret
   csi.storage.k8s.io/node-stage-secret-namespace: default
reclaimPolicy: Delete
mountOptions:
   - discard

# 3、创建且查看
[root@master1 csi]# kubectl apply -f csi-rbd-sc.yaml
storageclass.storage.k8s.io/csi-rbd-sc created
[root@master1 csi]# kubectl get storageclass.storage.k8s.io/csi-rbd-sc
NAME         PROVISIONER        RECLAIMPOLICY   VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION   AGE
csi-rbd-sc   rbd.csi.ceph.com   Delete          Immediate           false
   17s
```

创建PERSISTENTVOLUMECLAIM （现在创建PVC，会自动创建PV咯。智能化666）

例如，为了创建一个基于块的PersistentVolumeClaim其利用ceph- CSI基StorageClass上面创建，以下YAML可以用来请求来自原始块存储CSI -rbd-SC StorageClass：

```
$ cat <<EOF > raw-block-pvc.yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: raw-block-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Block   # Block
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-rbd-sc
EOF
$ kubectl apply -f raw-block-pvc.yaml
```

以下示例和示例将上述PersistentVolumeClaim绑定 到作为原始块设备的Pod资源：

```
$ cat <<EOF > raw-block-pod.yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-raw-block-volume
spec:
  containers:
    - name: fc-container
      image: fedora:26
      command: ["/bin/sh", "-c"]
      args: ["tail -f /dev/null"]
      volumeDevices:
        - name: data
          devicePath: /dev/xvda
  volumes:
    - name: data
      persistentVolumeClaim:
        claimName: raw-block-pvc
EOF
$ kubectl apply -f raw-block-pod.yaml
```

创建一个基于文件系统PersistentVolumeClaim其利用 ceph- CSI基StorageClass上面创建，以下YAML可以用于请求安装从文件系统（由RBD图像支持）CSI -rbd-SC StorageClass：

```
$ cat <<EOF > pvc.yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: rbd-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem  # Filesystem
  resources:
    requests:
      storage: 1Gi
  storageClassName: csi-rbd-sc
EOF
$ kubectl apply -f pvc.yaml
```

以下示例和示例将上述PersistentVolumeClaim绑定 到作为已挂载文件系统的Pod资源：

```
$ cat <<EOF > pod.yaml
---
apiVersion: v1
kind: Pod
metadata:
  name: csi-rbd-demo-pod
spec:
  containers:
    - name: web-server
      image: nginx
      volumeMounts:
        - name: mypvc
          mountPath: /var/lib/www/html
  volumes:
    - name: mypvc
      persistentVolumeClaim:
        claimName: rbd-pvc
        readOnly: false
EOF
$ kubectl apply -f pod.yaml
```

五、使用StorageClass (最终方式)

Storage Class的作用

简单来说，storage配置了要访问ceph RBD的IP/Port、用户名、keyring、pool，等信息，我们不需要提前创建image；当用户创建一个PVC时，k8s查找是否有符合PVC请求的storage class类型，如果有，则依次执行如下操作：

- 到ceph集群上创建image
- 创建一个PV，名字为pvc-xx-xxx-xxx，大小pvc请求的storage。
- 将上面的PV与PVC绑定，格式化后挂到容器中

通过这种方式管理员只要创建好storage class就行了，后面的事情用户自己就可以搞定了。如果想要防止资源被耗尽，可以设置一下Resource Quota。

当pod需要一个卷时，直接通过PVC声明，就可以根据需求创建符合要求的持久卷。

创建storage class

```
# cat storageclass.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/rbd
parameters:
  monitors: 192.168.20.41:6789
  adminId: admin
  adminSecretName: ceph-secret
  pool: k8s
  userId: admin
  userSecretName: ceph-secret
  fsType: xfs
  imageFormat: "2"           # mun==2
  imageFeatures: "layering"  # 定义是layering
```

创建PVC

RBD只支持 ReadWriteOnce 和 ReadOnlyAll，不支持ReadWriteAll。注意这两者的区别点是，不同nodes之间是否可以同时挂载。同一个node上，即使是ReadWriteOnce，也可以同时挂载到2个容器上的。

创建应用的时候，需要同时创建 pv和pod，二者通过storageClassName关联。pvc中需要指定其storageClassName为上面创建的sc的name（即fast）。

```
# cat pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: rbd-pvc-pod-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 1Gi
  storageClassName: fast
```

创建pod

```
# cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: rbd-pvc-pod
  name: ceph-rbd-sc-pod1
spec:
  containers:
  - name: ceph-rbd-sc-nginx
    image: nginx
    volumeMounts:
    - name: ceph-rbd-vol1
      mountPath: /mnt
      readOnly: false
  volumes:
  - name: ceph-rbd-vol1
    persistentVolumeClaim:
      claimName: rbd-pvc-pod-pvc
```

补充

在使用Storage Class时，除了使用PVC的方式声明要使用的持久卷，还可通过创建一个volumeClaimTemplates进行声明创建（StatefulSets中的存储设置），如果涉及到多个副本，可以使用StatefulSets配置：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "fast"
      resources:
        requests:
          storage: 1Gi
```

但注意不要用Deployment。因为，如果Deployment的副本数是1，那么还是可以用的，跟Pod一致；但如果副本数 >1 ，此时创建deployment后会发现，只启动了1个Pod，其他Pod都在ContainerCreating状态。过一段时间describe pod可以看到，等volume等很久都没等到。使用StorageClass

Storage Class的作用

简单来说，storage配置了要访问ceph RBD的IP/Port、用户名、keyring、pool，等信息，我们不需要提前创建image；当用户创建一个PVC时，k8s查找是否有符合PVC请求的storage class类型，如果有，则依次执行如下操作：

- 到ceph集群上创建image
- 创建一个PV，名字为pvc-xx-xxx-xxx，大小pvc请求的storage。
- 将上面的PV与PVC绑定，格式化后挂到容器中

通过这种方式管理员只要创建好storage class就行了，后面的事情用户自己就可以搞定了。如果想要防止资源被耗尽，可以设置一下Resource Quota。

当pod需要一个卷时，直接通过PVC声明，就可以根据需求创建符合要求的持久卷。

创建storage class

```
# cat storageclass.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/rbd
parameters:
  monitors: 192.168.20.41:6789
  adminId: admin
  adminSecretName: ceph-secret
  pool: k8s
  userId: admin
  userSecretName: ceph-secret
  fsType: xfs
  imageFormat: "2"
  imageFeatures: "layering"
```

创建PVC

RBD只支持 ReadWriteOnce 和 ReadOnlyAll，不支持ReadWriteAll。注意这两者的区别点是，不同nodes之间是否可以同时挂载。同一个node上，即使是ReadWriteOnce，也可以同时挂载到2个容器上的。

创建应用的时候，需要同时创建 pv和pod，二者通过storageClassName关联。pvc中需要指定其storageClassName为上面创建的sc的name（即fast）。

```
# cat pvc.yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: rbd-pvc-pod-pvc
spec:
  accessModes:
    - ReadWriteOnce
  volumeMode: Filesystem
  resources:
    requests:
      storage: 1Gi
  storageClassName: fast
```

创建pod

```
# cat pod.yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: rbd-pvc-pod
  name: ceph-rbd-sc-pod1
spec:
  containers:
  - name: ceph-rbd-sc-nginx
    image: nginx
    volumeMounts:
```

```
    - name: ceph-rbd-vol1
      mountPath: /mnt
      readOnly: false
  volumes:
  - name: ceph-rbd-vol1
    persistentVolumeClaim:
      claimName: rbd-pvc-pod-pvc
```

补充

在使用Storage Class时，除了使用PVC的方式声明要使用的持久卷，还可通过创建一个volumeClaimTemplates进行声明创建（StatefulSets中的存储设置），如果涉及到多个副本，可以使用StatefulSets配置：

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: nginx
spec:
  selector:
    matchLabels:
      app: nginx
  serviceName: "nginx"
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: nginx
        image: nginx
        volumeMounts:
        - name: www
          mountPath: /usr/share/nginx/html
  volumeClaimTemplates:
  - metadata:
      name: www
    spec:
      accessModes: [ "ReadWriteOnce" ]
      storageClassName: "fast"
      resources:
        requests:
          storage: 1Gi
```

但注意不要用Deployment。因为，如果Deployment的副本数是1，那么还是可以用的，跟Pod一致；但如果副本数 >1 ，此时创建deployment后会发现，只启动了1个Pod，其他Pod都在ContainerCreating状态。过一段时间describe pod可以看到，等volume等很久都没等到。