Ceph RBD

目录

** 版本修订记录 **

版本号	*修订时间*	*修订内容*
v1.0	*2018-08-12*	*初版修订*
v1.1	*2018-10-24*	*+ iSCSI 网关/+ QEMU /+ libvirt /+ RBD 缓存/+ RBD REPLAY /+镜像/+ iSCSI 配置/+iSCSI 依赖包安装/*

一.简介

RBD(*RADOS Block Device*).即 Ceph 集群提供的块设备功能；Ceph 块设备是轻量级的，可调大小的并且将数据条带化存储到多个 OSD 上，利用了 RADOS 提供的快照功能，多副本功能和一致性功能使得 Ceph 块设备具有高性能，高可用性等特性，它主要使用以下两种方式和 Ceph 集群进行交互：

- Kernel rbd:建立好 Ceph 块设备后，将其映射到操作系统内核中，可像其他物理块设备一样进行格式化并挂载使用，它的设备文件格式为/dev/rbd*;
- librbd:提供给依赖 libvirt 和 QEMU 的虚拟化软件如，OpenStack,CloudStack 的后端块设备；

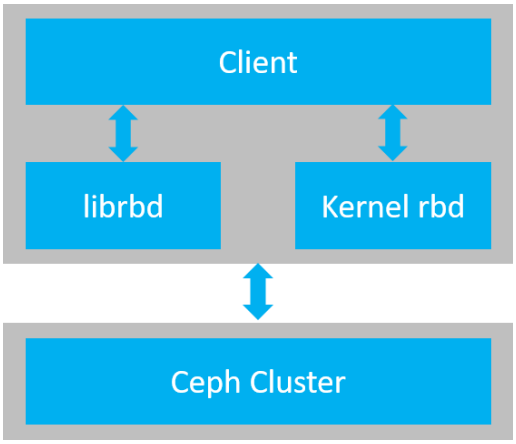


Figure 1 RBD 交互图

二．原理

2.1 架构

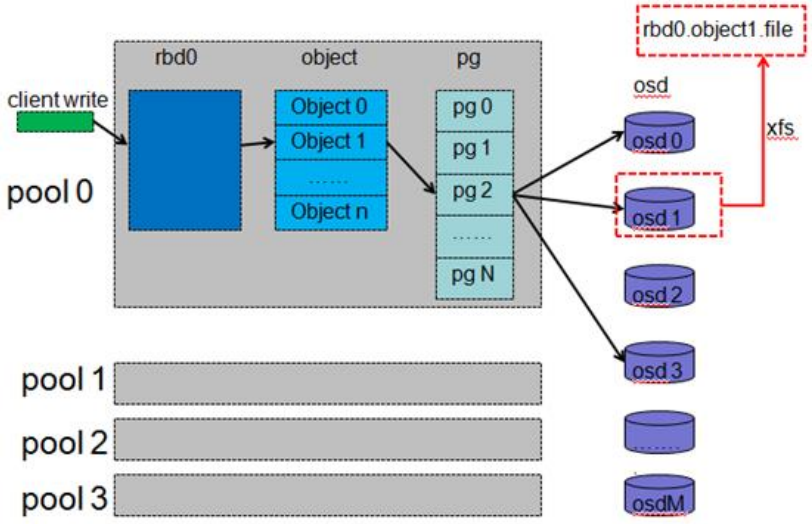
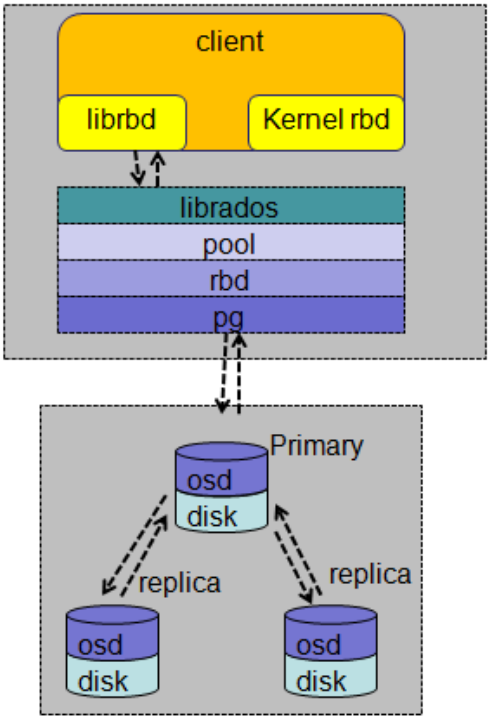


Figure 2RBD 架构图

如图是 RBD 的写流程；



2.2 快照

快照是特定时间点跨设备的只读复制副本，Ceph 还支持快照分层，这允许使用 rbd 命令或许多高层级的应用诸如 QEMU,libvirt,OpenCloud 和 CloudStack 快速地导出块设备；注意在对块设备打快照的时候，需要先关闭块设备的 IO，以免打出的快照中缺少数据；如果块设备上有文件系统，可以使用 fsfreeze 命令关闭设备上的 IO；

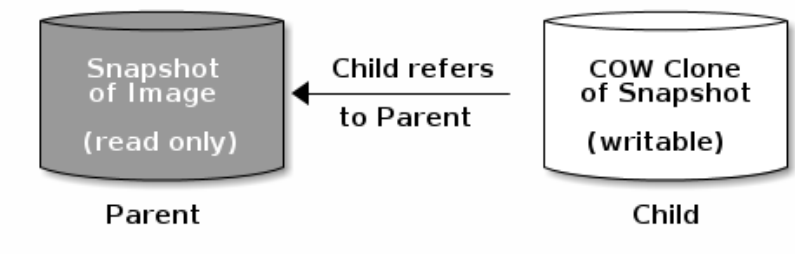


Figure 3 RBD 分层特性

Ceph 支持 COW(Copy-on-wright 写拷贝)特性，在对块设备创建快照后，该快照是只读的，基于该快照，可以创建多个可写的克隆块设备，由于其是基于 Parent 快照的，所以 Parent 快照是不能删

除，这会导致克隆的 Child 块设备数据丢失，所以需要对只读的 Parent 快照进行保护操作；克隆的镜像可以读取，写入，再克隆或重新设置大小，并且可以将一个池中的镜像克隆到另一个池中；

2.3 镜像

RBD 的 Image 可以在两个 Ceph 集群之间进行镜像同步，该功能可以在 Jewel 版本之后使用，镜像需要在每个集群中的池上进行配置，配置时可以指定需要镜像的 Image 或镜像该池中的所有 Image,集群中的 rbd-mirror 进程负责将 Image 的变化发送到远程 Image 并将其进行恢复；

镜像有两种复制模式：

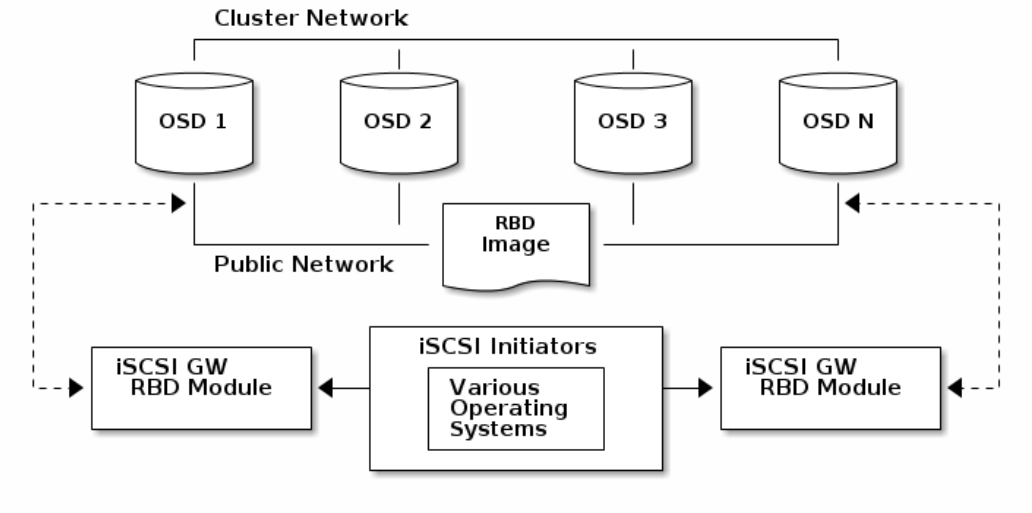
- One-way Replication(单边复制)：数据仅从主集群同步到次集群，rbd-mirror 进程仅在次集群上运行；
- Two-way Replication(双边复制)：当数据从一个集群上的主 Image 同步到另一个集群上的次 Image(反之亦可)，此时，双边都需要 rbd-mirror 进程的运行；

镜像有两种配置模式：

- Pool(池镜像)：镜像池内的所有 Image
- Image(Image 模式)：仅镜像指定的 Image，该模式仅需要启动一个 rbd-mirror 进程，依赖与 Image 的日志特性 journaling，需要将该特性进行使能；并且该特性依赖于 exclusive-lock 特性的使能；

2.4 iSCSI 网关

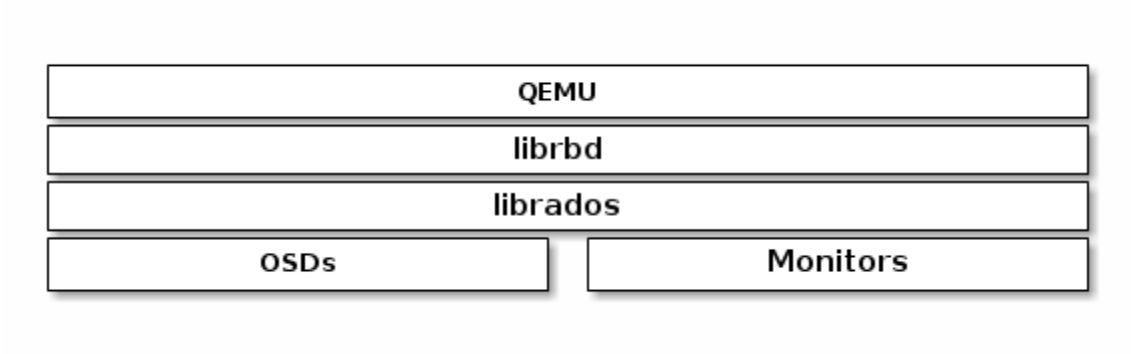
iSCSI 网关集成了 Ceph 存储和 iSCSI 标准用于提供高可用的 iSCSI Target，它将 RBD 镜像导出为 SCSI 磁盘，iSCSI 客户端可以通过 TCP/IP 网络发送命令到 RBD 镜像。



如上图为相应的架构图。每一个 iSCSI 网关运行 Linux IO(LIO)子系统来提供对 iSCSI 协议的支持，LIO 利用用户空间直通(TCMU)与 Ceph 的 librbd 库进行交互，并将 RBD 镜像暴露给 iSCSI 客户端;

2.5 QEMU

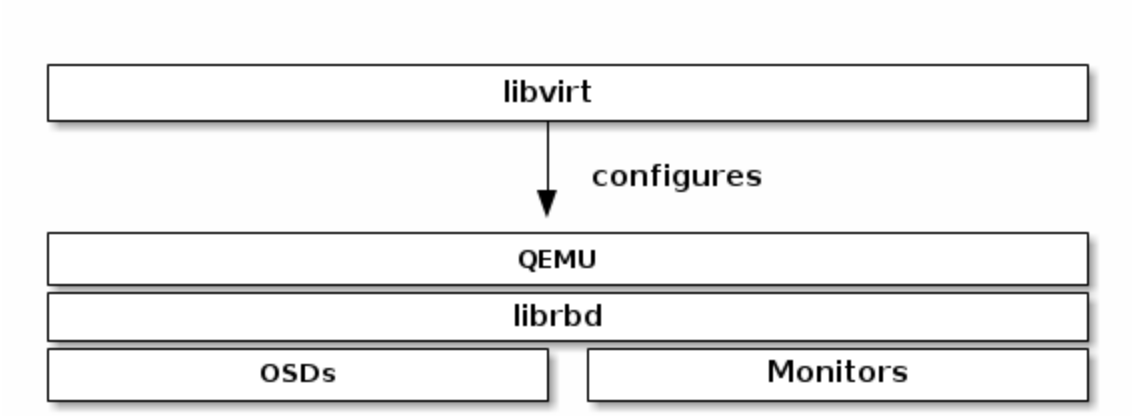
RBD 最常用的场景时为虚拟机提供块设备，如下图是 Ceph 和 QEMU 相应的架构关系:



QEMU 默认使用的配置文件为/etc/ceph/$cluster.conf，默认使用的用户为 client.admin。

2.6 libvirt

RBD 支持 QEMU/KVM，所以 Ceph 可以作为和 libvirt 交互的虚拟管理组件的后端存储，libvirt 常用的场景是为云解决方案如 OpenStack 或 CloudStack 提供 Ceph 块设备，它们使用 libvirt 和 QEMU/KVM 进行交互，QEME/KVM 通过 librbd 和 Ceph 块设备交互，如下为相应的架构图:



2.7 RBD 缓存

使用内核驱动驱动的 RBD 能够使用缓存页来提高性能，而用户空间实现(librbd)的 RBD 则不能使用缓存页的快速优势，所以 RBD 实现了自己的缓存机制---“RBD caching”，它和磁盘缓存的效果是一

样的，当操作系统发送更新请求时，缓存中的所有脏数据都会写回到 OSDs 中，它使用 LRU(最近最少使用)算法，并且在回写模式下它可以合并连续的请求以获取更高的吞吐量；

2.8 RBD REPLAY

RBD REPLAY 是一系列捕获 RBD 负载的工具集合，如果需要使用则需要安装 lttng-tools。并且 librbd 的版本在 v0.87 以上。

三．操作

3.1 创建 RBD

```
ceph osd pool create {pool-name} {pg-num} {pgp-num} # 如果在创建时没有指定 pool-name,则默认会将 rbd 的数据存储在 rbd 池中；
rbd pool init {pool-name}
rbd create --size {megabytes} {pool-name}/{image-name} # 创建块设备，首先需要创建 Pool, --size <M/G/T>，默认单位为 M(1024)

# 默认情况下会使用 admin 用户来进行集群认证，admin 用户拥有集群中的所有权限，所以可以创建新的用户用于 rbd 的操作
ceph auth get-or-create client.{ID} mon 'profile rbd' osd 'profile {profile name} [pool={pool-name}][, profile ...]'
```

3.2 查询 RBD

```
rbd ls {pool-name}
rbd trash ls {pool-name}
rbd info {pool-name}/{image-name}

案列：
[cpu@mon ~]$ rbd ls rbdp
rbd
[cpu@mon ~]$ rbd trash ls rbdp
170496b8b4567 rbd
[cpu@mon ~]$ rbd info rbdp/rbd
rbd image 'rbd':
        size 10 GiB in 2560 objects
        order 22 (4 MiB objects)
        id: 170496b8b4567
        block_name_prefix: rbd_data.170496b8b4567
        format: 2
        features: layering, exclusive-lock, object-map, fast-diff, deep-flatten
        op_features:
```

```
    flags:
    create_timestamp: Thu Oct 18 09:17:48 2018
```

3.3 删除 RBD

```
rbd rm {pool-name}/{image-name}
rbd trash mv {pool-name}/{image-name}
rbd trash rm {pool-name}/{image-name}
```

3.4 恢复 RBD

```
rbd trash restore {image-id} # 恢复 rbd pool 的块设备
rbd trash resotre {pool-name}/{image-id } # 恢复其他数据池的块设备
rbd trash restore {pool-name}/{image-id }  --image {image-new-name} # 恢复镜像时可以重命名
```

3.5 重设置 RBD 大小

```
rbd resize --size {megabytes} {pool-name}/{image-name} # 增加 rbd 的大小
rbd resize --size {megabytes} {pool-name}/{image-name} --allow-shrink # 减少 rbd 的大小
```

3.6 映射 RBD 设备

```
rbd list # 查看块设备
sudo rbd device map {pool-name}/{image-name} --id {user-name} --keyring {keyring-file} # 映射块设备到内核空间，映射时默认使用 admin 用户;
rbd device list # 查看设备的映射情况
sudo rbd device unmap /dev/rbd/{pool-name}/{image-name}
sudo rbd unmap {pool-name}/{image-name}

案列:
# rbd 在映射时有些特性不支持，需要取消掉
[cpu@mon ~]$ rbd info rbdp/rbd
rbd image 'rbd':
    size 10 GiB in 2560 objects
    order 22 (4 MiB objects)
    id: 170496b8b4567
    block_name_prefix: rbd_data.170496b8b4567
    format: 2
    features: layering, exclusive-lock, object-map, fast-diff, deep-flatten
    op_features:
    flags:
```

```
        create_timestamp: Thu Oct 18 09:17:48 2018
[cpu@mon ~]$ rbd feature disable rbdp/rbd object-map fast-diff deep-flatten
[cpu@mon ~]$ rbd info rbdp/rbd
rbd image 'rbd':
        size 10 GiB in 2560 objects
        order 22 (4 MiB objects)
        id: 170496b8b4567
        block_name_prefix: rbd_data.170496b8b4567
        format: 2
        features: layering, exclusive-lock
        op_features:
        flags:
        create_timestamp: Thu Oct 18 09:17:48 2018
[cpu@mon ~]$ sudo rbd device map rbdp/rbd
/dev/rbd0
[cpu@mon ~]$ ll /dev/rbd0
brw-rw---- 1 root disk 251, 0 Oct 18 16:02 /dev/rbd0

# rbd 在映射到内核之后依然同步调整大小
[cpu@mon ~]$ rbd resize --size 12G rbdp/rbd
Resizing image: 100% complete...done.
[cpu@mon ~]$ fdisk -l /dev/rbd0
fdisk: cannot open /dev/rbd0: Permission denied
[cpu@mon ~]$ sudo fdisk -l /dev/rbd0

Disk /dev/rbd0: 12.9 GB, 12884901888 bytes, 25165824 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
[cpu@mon ~]$ rbd resize --size 10G rbdp/rbd --allow-shrink
Resizing image: 100% complete...done.
[cpu@mon ~]$ sudo fdisk -l /dev/rbd0

Disk /dev/rbd0: 10.7 GB, 10737418240 bytes, 20971520 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

# 取消映射
[cpu@mon ~]$ sudo rbd device unmap rbdp/rbd
[cpu@mon ~]$ sudo rbd device list
[cpu@mon ~]$
```

3.7 RBD 快照

```
rbd snap create {pool-nam}/{image-name}@{snap-name}
```

```
rbd snap ls {pool-name}/{image-name}
rbd snap roolback {pool-name}/{image-name}@{snap-name} # 官方推荐使用克隆而不是恢复的方式来获得一个之前的块设备；因为它更为节省时间;
rbd snap  protect {pool-name}/{image-name}@{snap-name}
rbd snap rm {pool-name}/{image-name}@{snap-name}
rbd snap purge {pool-name}{image-name}

案例:
[cpu@mon ~]$ rbd snap create rbdp/rbd@snap1
[cpu@mon ~]$ rbd snap ls rbdp/rbd
SNAPID NAME    SIZE TIMESTAMP
    4 snap1 10 GiB Fri Oct 19 15:49:03 2018
[cpu@mon ~]$ rbd snap rollback rbdp/rbd@snap1
Rolling back to snapshot: 100% complete...done.
[cpu@mon ~]$ rbd snap ls rbdp/rbd
SNAPID NAME    SIZE TIMESTAMP
    4 snap1 10 GiB Fri Oct 19 15:49:03 2018
[cpu@mon ~]$ rbd snap protect rbdp/rbd@snap1 # 保护的对象时不能删除的
[cpu@mon ~]$ rbd snap rm rbdp/rbd@snap1
Removing snap: 100% complete...done.
[cpu@mon ~]$ rbd snap ls rbdp/rbd
[cpu@mon ~]$ rbd snap purge rbdp/rbd  # 删除所有为保护的快照
rbd: error removing snapshot(s) 'snap1', which is protected - these must be unprotected with `rbd snap unprotect`.
Removing all snapshots: 0% complete...failed.
```

3.8 克隆快照

```
rbd snap protect {pool-name}/{image-name}@{snap-name}
rbd clone {pool-name}/{image-name}@{snap-name} {pool-name}/{child-image-name}
rbd snap unprotect {pool-name}/{image-name}@{snap-name}
rbd children {pool-name}/{image-name}@{snapshot-name}
rbd flatten {pool-name}/{image-name}

案列:
[cpu@mon ~]$ rbd clone rbdp/rbd@snap1 rbdp/rbdc
cpu@mon ~]$ rbd children rbdp/rbd@snap1
rbdp/rbdc
[cpu@mon ~]$ rbd flatten rbdp/rbdc
Image flatten: 100% complete...done.
[cpu@mon ~]$
```

3.9 镜像

```
# 池模式
rbd mirror pool enable {pool-name} {mode} # 使能进行，mode 为镜像模式，包括 pool 和 image 两种
rbd mirror pool disable {pool-name}
rbd mirror pool peer add {pool-name} {client-name}@{cluster-name} # /etc/ceph/{cluster-name}.conf 该配置文件查找远程的用户名，/etc/ceph/{cluster-name}.{client-name}.keyring 该配置文件查找知道用用户 cliet-name 的 kering;
rbd mirror pool peer remove {pool-name} {peer-uuid}

# Image 模式
rbd feature enable {pool-name}/{image-name} {feature-name} # 需要使能 journaling 特性;
rbd mirror image enable {pool-name}/{image-name}
rbd mirror image disable {pool-name}/{image-name}

# 主备切换
rbd mirror image demote {pool-name}/{image-name} # 降级 Image
rbd mirror pool demote {pool-name} # 降级池
rbd mirror image promote [--force] {pool-name}/{image-name} # 提升池
rbd mirror pool promote [--force] {pool-name}

# 强制同步
rbd mirror image resync {pool-name}/{image-name} # 当 rbd-mirror 进程检测到脑裂事件时，关联的镜像不会在进行同步，需要将主 Image 进行降级后然后在强制同步;
rbd mirror image status {pool-name}/{image-name} # 查看镜像的状态;
rbd mirror pool status {pool-name}

# rbd-mirror 进程
ceph auth get-or-create client.rbd-mirror.{unique id} mon 'profile rbd-mirror' osd 'profile rbd'
systemctl enable ceph-rbd-mirror@rbd-mirror.{unique id}
rbd-mirror -f --log-file={log_path} # 指定前台运行
```

3.10 iSCSI 配置

```
# 为了 iSCSI 网关高可用解决方案的使用，推荐配置使用两到四个网关节点；可以同 OSD 节点一同部署，也可以单独部署；如果硬件条件允许，可以分开配置 iSCSI 的前端流量和后端 Ceph 流量;
# 内核系统支持参考 https://shaman.ceph.com/repos/kernel/ceph-iscsi-test/
# 推荐的 OSD 心跳配置
#配置文件更新
[osd]
osd heartbeat grace = 20
osd heartbeat interval = 5
# 或在线 Monitor 更新
```

```
ceph tell osd.0 config set osd_heartbeat_grace 20
ceph tell osd.0 config set osd_heartbeat_interval 5
# 或在线 OSD 更新
ceph daemon osd.0 config set osd_heartbeat_grace 20
ceph daemon osd.0 config set osd_heartbeat_interval 5

#####################################################
##########                非高可用模式            ##########
#####################################################
# 创建相应的镜像
[root@mon ~]# rbd ls rbdp
rbd
[root@mon ~]# rbd info rbdp/rbd
rbd image 'rbd':
        size 10 GiB in 2560 objects
        …
# 使用 targetcli 创建相应后端存储为 Ceph 块设备的 Target
[root@mon ~]# yum -y install scsi-target-utils
[root@mon ~]# targetcli
/> cd backstores/user:rbd/
/backstores/user:rbd> create cfgstring=rbdp/rbd name=disk0 size=10G
## /backstores/user:rbd> create cfgstring=PoolName/ImageName name=diak0 size=10G
## cfgstring=ool_name/image_name[;osd_op_timeout=N;conf=N;id=N]
## osd_op_timeout is optional and N is in seconds
## conf is optional and N is the path to the conf file
## id is optional and N is the id to connect to the cluster
/backstores/user:rbd> ls
o- user:rbd ..... [Storage Objects: 1]
  o- disk0 ..... [rbdp/rbd (10.0GiB) deactivated]
    o- alua ..... [ALUA Groups: 1]
      o- default_tg_pt_gp ..... [ALUA state: Active/optimized]
# 配置 Target
# 配置配置文件
# IQN 名字格式为：Iqn.yyyy-mm.<reversed domain name>:identifier
/> cd /iscsi/
/iscsi> create iqn.2003-01.com.redhat.iscsi-gw:iscsi-igw
/iscsi> cd iqn.2003-01.com.redhat.iscsi-gw:iscsi-igw/tpg1/
/iscsi/iqn.20...scsi-igw/tpg1> cd luns/
/iscsi/iqn.20...igw/tpg1/luns> create /backstores/user:rbd/disk0
/iscsi/iqn.20...igw/tpg1/luns> cd ../portals/
/iscsi/iqn.20.../tpg1/portals> create 10.64.37.164
/iscsi/iqn.20.../tpg1/portals> cd ..
/iscsi/iqn.20...scsi-igw/tpg1> ls
o- tpg1 ..... [no-gen-acls, no-auth]
  o- acls ..... [ACLs: 0]
  o- luns ..... [LUNs: 1]
  | o- lun0 ..... [user/disk0 (default_tg_pt_gp)]
```

```
  o- portals ..... [Portals: 1]
    o- 10.64.37.164:3260 ..... [OK]
# 查看导出的 RDB 设备
[root@mon ~]# iscsiadm -m discovery -t sendtargets -p 10.64.37.164:3260
10.64.37.164:3260,1 iqn.2003-01.com.redhat.iscsi-gw:iscsi-igw

#######################################################
##########                高可用模式                ##########
#######################################################
# Target 端配置
## Ansible 安装配置
## Requirements:
    • A running Ceph Luminous (12.2.x) cluster or newer
    • RHEL/CentOS 7.5; Linux kernel v4.16 or newer; or the Ceph iSCSI client test kernel
    • The ceph-iscsi-config package installed on all the iSCSI gateway nodes
## git clone https://github.com/ceph/ceph-ansible.git 下载相应 Ceph-Ansible 源码，配置需要安装 iSCSI 的主机即可;

## 命令行安装
## Requirements:
    • A running Ceph Luminous or later storage cluster
    • RHEL/CentOS 7.5; Linux kernel v4.16 or newer; or the Ceph iSCSI client test kernel
    • The following packages must be installed from your Linux distribution's software repository:
        ○ targetcli-2.1.fb47 or newer package
        ○ python-rtslib-2.1.fb64 or newer package
        ○ tcmu-runner-1.3.0 or newer package
        ○ ceph-iscsi-config-2.4 or newer package
        ○ ceph-iscsi-cli-2.5 or newer package

# 自定义安装可以参考附录 iSCSI 依赖包安装 部分
[root@mon ~]# yum -y install targetcli python-rtslib

# 使用 gwcli 配置 iSCSI Target
[root@mon ~]# gwcli
/> cd /iscsi-target
/iscsi-target>  create iqn.2003-01.com.redhat.iscsi-gw:iscsi-igw
/iscsi-target> cd iqn.2003-01.com.redhat.iscsi-gw:iscsi-igw/gateways
/iscsi-target...-igw/gateways>  create hostname1  hostip_address1 # 必须多余两个网关
/iscsi-target...-igw/gateways>  create hostname2  hostip_address2
/iscsi-target...-igw/gateways> cd /disks
/disks> create pool=rbd image=disk_1 size=20G
/disks> cd /iscsi-target/iqn.2003-01.com.redhat.iscsi-gw:iscsi-igw/hosts
/iscsi-target...eph-igw/hosts>  create iqn.1994-05.com.redhat:rh7-client
/iscsi-target...at:rh7-client>  auth chap=myiscsiusername/myiscsipassword
/iscsi-target...at:rh7-client> disk add rbd.disk_1
```

```
# 配置 Initiators
[root@mon ~]# yum install iscsi-initiator-utils
[root@mon ~]# yum install device-mapper-multipath
[root@mon ~]# mpathconf --enable --with_multipathd y
[root@mon ~]# vim /etc/multipath/mutipath.conf
## 将下面内容添加到/etc/multipath/mutipath.conf
## 发现 target
[root@mon ~]# iscsiadm -m discovery -t -st 10.64.37.164
# 登陆进入 Target
[root@mon ~]# iscsiadm -m node -T iqn.2003-01.org.linux-iscsi.rheln1 -l
[root@mon ~]# multipath –ll
```

3.11 RBD Replay 配置

```
[cpu@mon ~]$ mkdir -p traces
[cpu@mon ~]$ lttng create -o traces/ librbd
# lttng create [NAME] [-o output_PATH ]
[cpu@mon ~]$ lttng enable-event -u 'librbd:*'
UST event librbd:* created in channel channel0
[cpu@mon ~]$ lttng add-context -u -t pthread_id
UST context pthread_id added to all channels
[cpu@mon ~]$ lttng start
Tracing started for session librbd
[cpu@mon traces]$ lttng stop
Waiting for data availability
Tracing stopped for session librbd
[cpu@mon traces]$ rbd-replay-prep traces/ust/uid/*/* replay.bin
[cpu@mon traces]$ rbd-replay --read-only replay.bin
[cpu@mon traces]$
```

3.12 QEMU 配置

```
# 安装 QEMU
[root@mon ~]# yum -y install qemu-kvm
# 如下如果不支持 rbd，下面的命令没有输出，则需要重新编译
[root@mon ~]# qemu-img --help | grep rbd
[root@mon ~]#
[cpu@mon QEMU]$ wget https://download.qemu.org/qemu-3.0.0.tar.bz2
[cpu@mon QEMU]$ bzip2 -dkv qemu-3.0.0.tar.bz2
[cpu@mon QEMU]$ tar -xvf qemu-3.0.0.tar
[cpu@mon QEMU]$ mkdir qemu3.0 # 创建安装目录
[cpu@mon QEMU]$ cd qemu-3.0.0
[cpu@mon qemu-3.0.0]$ sudo yum -y install pixman pixman-devel
[cpu@mon qemu-3.0.0]$ ./configure --enable-rbd --prefix=/home/cpu/QEMU/qemu3.0/
```

```
[cpu@mon qemu-3.0.0]$ make
[cpu@mon qemu-3.0.0]$ make installs
[cpu@mon qemu-3.0.0]$ cd ../qemu3.0/bin/
# 输出如下表示支持 rbd
[cpu@mon bin]$ ./qemu-img --help | grep rbd
Supported formats: blkdebug blklogwrites blkreplay blkverify bochs cloop copy-on-read dmg file
gluster host_cdrom host_device iscsi iser luks nbd null-aio null-co nvme parallels qcow qcow2 qed
quorum raw rbd replication sheepdog throttle vdi vhdx vmdk vpc vvfat
[cpu@mon bin]$

# QEMU 创建 RBD
[cpu@mon bin]$ ./qemu-img create -f raw rbd:rbd/qemu 10G
Formatting 'rbd:rbd/qemu', fmt=raw size=10737418240
[cpu@mon bin]$ rbd ls
qemu
[cpu@mon bin]$ rbd info qemu
rbd image 'qemu':
        size 10 GiB in 2560 objects
        order 22 (4 MiB objects)
        id: 1be2e6b8b4567
        block_name_prefix: rbd_data.1be2e6b8b4567
        format: 2
        features: layering, exclusive-lock, object-map, fast-diff, deep-flatten
        op_features:
        flags:
        create_timestamp: Mon Oct 29 11:34:26 2018
[cpu@mon bin]# ./qemu-img info -f rbd rbd:rbd/qemu
image: json:{"pool": "rbd", "image": "qemu", "driver": "rbd"}
file format: rbd
virtual size: 10G (10737418240 bytes)
disk size: unavailable
cluster_size: 4194304

# QEMU 重设置大小
[cpu@mon bin]$ ./qemu-img resize -f raw rbd:rbd/qemu 20G
Image resized.
[cpu@mon bin]$ rbd info qemu
rbd image 'qemu':
        size 20 GiB in 5120 objects
        order 22 (4 MiB objects)
        id: 1be2e6b8b4567
        block_name_prefix: rbd_data.1be2e6b8b4567
        format: 2
        features: layering, exclusive-lock, object-map, fast-diff, deep-flatten
        op_features:
        flags:
        create_timestamp: Mon Oct 29 11:34:26 2018
```

```
# 缩小的 Image 大小的时候注意卷上是否有重要的数据，以免数据丢失。缩小镜像时使用—
shrink 参数;
[cpu@mon bin]$ ./qemu-img resize --shrink -f raw rbd:rbd/qemu 10G

# 指定某个镜像上的顺序 benchmark 测试。默认为读，有-w 参数时表示为写。
[cpu@mon bin]$ ./qemu-img bench -c 102400 -S 4096 -d 32 -f raw rbd:rbd/qemu
Sending 102400 read requests, 4096 bytes each, 32 in parallel (starting at offset 0, step size 4096)
Run completed in 10.103 seconds.
```

3.13 libvirt 配置

```
# 创建池
[cpu@mon bin]$ ceph osd pool create libvirt-pool 128 128
pool 'libvirt-pool' created
[cpu@mon bin]$ ceph osd lspools
46 libvirt-pool
[cpu@mon bin]$ rbd pool init libvirt-pool
# libvirt 默认和 ceph 使用的用户为 libvirt.创建用户并配置 keyring 文件
[cpu@mon bin]$ ceph auth get-or-create client.libvirt mon 'profile rbd' osd 'profile rbd pool=libvirt-
pool'
[cpu@mon bin]$ ceph auth get client.libvirt
exported keyring for client.libvirt
[client.libvirt]
        key = AQD/t9ZbzcETCxAAXwzwPpfgz14FlVgc34g4AQ==
        caps mon = "profile rbd"
        caps osd = "profile rbd pool=libvirt-pool"
[cpu@mon bin]$ sudo vim /etc/ceph/ceph.client.libvirt.keyring # 写入上调命令的结果
# 使用 qemu 创建镜像并查看镜像信息
[cpu@mon bin]$ ./qemu-img create -f rbd rbd:libvirt-pool/new-libvirt-image 4G
Formatting 'rbd:libvirt-pool/new-libvirt-image', fmt=rbd size=4294967296
[cpu@mon bin]$ ./qemu-img info -f rbd rbd:libvirt-pool/new-libvirt-image
image: json:{"pool": "libvirt-pool", "image": "new-libvirt-image", "driver": "rbd"}
file format: rbd
virtual size: 4.0G (4294967296 bytes)
disk size: unavailable
cluster_size: 4194304
[cpu@mon bin]$ rbd info libvirt-pool/new-libvirt-image
rbd image 'new-libvirt-image':
        size 4 GiB in 1024 objects
        order 22 (4 MiB objects)
        id: 1be6c6b8b4567
        block_name_prefix: rbd_data.1be6c6b8b4567
        format: 2
        features: layering, exclusive-lock, object-map, fast-diff, deep-flatten
        op_features:
        flags:
```

```
      create_timestamp: Mon Oct 29 15:37:10 2018
# 安装 virt-manager
[cpu@mon bin]$ sudo yum -y install virt-manager libvirt-daemon libvirt-daemon-lxc
[root@mon system]# systemctl enable libvirtd
[root@mon system]# systemctl start libvirtd
[root@mon system]# systemctl status libvirtd
● libvirtd.service - Virtualization daemon
   Loaded: loaded (/usr/lib/systemd/system/libvirtd.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2018-10-29 16:10:55 CST; 5s ago
     Docs: man:libvirtd(8)
           https://libvirt.org
 Main PID: 9012 (libvirtd)
    Tasks: 16 (limit: 32768)
   Memory: 15.0M
   CGroup: /system.slice/libvirtd.service
           └─9012 /usr/sbin/libvirtd

Oct 29 16:10:55 mon systemd[1]: Starting Virtualization daemon...
Oct 29 16:10:55 mon systemd[1]: Started Virtualization daemon.
# 启动 virt-manager,注意如果为远程连接启动时需要设置 DISPLAY 环境变量
[root@mon bin]# export DISPLAY=10.65.42.37:0
[root@mon bin]# sudo virt-manager
# 创建虚拟机
```

四．参考资料

【1】ceph 的数据存储之路(2) ----- rbd 到 osd 的数据映射

【2】Ceph Block Device

【3】官方文档 RBD 部分

【4】Ceph-Ansible 配置文档

【5】RBD API

五．附录

5.1 命令行

```
rbd bench # rbd 简单的测试工具
rbd copy (cp) # 复制镜像，源镜像和复制镜像具有相同的大小，对象大小，Image 格式；
rbd deep copy (deep cp) # 深度复制镜像，源镜像和复制镜像具有相同的大小，对象大小，Image 格式以及快照和克隆；
```

rbd disk-usage (du) # 查看 RBD 的使用情况
rbd create # 创建 RBD
rbd pool init # RBD 池初始化
rbd feature disable # 关闭镜像的某些特性
rbd feature enable # 使能镜像特性
rbd remove (rm) # 删除镜像
rbd rename (mv) # 重命名镜像
rbd resize # 重置镜像大小
rbd status # 查看镜像状态
rbd list (ls) # 查看镜像列表
rbd watch # 监控 RBD 上的事件

rbd snap create (snap add) # 创建快照
rbd snap list (snap ls) # 查看镜像的快照
rbd snap protect # 保护快照，不能被删除
rbd snap unprotect # 去除快照的保护
rbd snap rollback (snap revert) # 回滚镜像
rbd snap rename # 重命名快照，池和镜像名必须相同
rbd snap remove (snap rm) # 删除快照
rbd snap snap purge # 删除所有非保护的对象
rbd clone # 克隆快照
rbd flatten # 用父级的数据填充克隆镜像，让其独立；

rbd trash list (trash ls) # 查看回收箱
rbd trash move (trash mv) # 将镜像放入回收箱
rbd trash purge # 删除回收箱的所有镜像
rbd trash remove (trash rm) # 删除回收箱中指定的对象
rbd trash restore # 恢复回收箱中指定的对象

rbd export # 导出镜像或快照
rbd export-diff # 导出镜像的增量部分
rbd import # 导入镜像或快照
rbd import-diff # 导入镜像的增量部分

rbd journal client disconnect
rbd journal export # 导出镜像归档日志
rbd journal import # 导入镜像归档日志
rbd journal info # 查看镜像归档信息、
rbd journal inspect # 检查日志是否存在错误
rbd journal reset #重置日志
rbd journal status # 查看镜像日志状态

rbd image-meta get # 获取镜像元数据的值
rbd image-meta list (image-meta ls) # 查看镜像元数据列表

```
rbd image-meta remove (image-meta rm) # 镜像元数据删除
rbd image-meta set # 镜像元数据设置

rbd group create # 创建组
rbd group image add # 把镜像添加到组
rbd group image list (group image ls) # 查看组中的镜像
rbd group image remove (group image rm) # 将镜像从组中移除
rbd group list (group ls) # 查看组
rbd group remove (group rm) # 删除组
rbd group rename # 组重命名
rbd group snap create # 创建组快照
rbd group snap list (group snap ls) # 组快照查看
rbd group snap remove (group snap rm) # 组快照删除
rbd group snap rename # 组快照重命名
```

5.2 iSCSI 依赖包安装

- tcmu-runner

```
[root@mon tcmu-runner]# git clone https://github.com/open-iscsi/tcmu-runner
[root@mon tcmu-runner]# cd tcmu-runner/
[root@mon tcmu-runner]# cmake -Dwith-glfs=false -Dwith-qcow=false -DSUPPORT_SYSTEMD=ON -
DCMAKE_INSTALL_PREFIX=/usr
[root@mon tcmu-runner]# make install
[root@mon tcmu-runner]# systemctl daemon-reload
[root@mon tcmu-runner]# systemctl enable tcmu-runner
Created symlink from /etc/systemd/system/multi-user.target.wants/tcmu-runner.service to
/usr/lib/systemd/system/tcmu-runner.service.
[root@mon tcmu-runner]# systemctl status tcmu-runner
● tcmu-runner.service - LIO Userspace-passthrough daemon
  Loaded: loaded (/usr/lib/systemd/system/tcmu-runner.service; enabled; vendor preset: disabled)
  Active: active (running) since Mon 2018-10-22 16:02:31 CST; 5s ago
 Main PID: 29535 (tcmu-runner)
   Tasks: 6
  Memory: 23.2M
  CGroup: /system.slice/tcmu-runner.service
          └─29535 /usr/bin/tcmu-runner
…
```

- rtslib-fb

```
[root@mon ceph-iscsi-cli]# git clone https://github.com/open-iscsi/rtslib-fb.git
[root@mon ceph-iscsi-cli]# cd rtslib-fb/
[root@mon rtslib-fb]# python setup.py install
```

- configshell-fb

```
[root@mon ceph-iscsi-cli]# git clone https://github.com/open-iscsi/targetcli-fb.git
[root@mon ceph-iscsi-cli]# cd targetcli-fb/
[root@mon ceph-iscsi-cli]# python setup.py install
[root@mon ceph-iscsi-cli]# mkdir /etc/target
[root@mon ceph-iscsi-cli]# mkdir /var/target
```

- ceph-iscsi-config

```
[root@mon ceph-iscsi-cli]# git clone https://github.com/ceph/ceph-iscsi-config.git
[root@mon ceph-iscsi-cli]# cd ceph-iscsi-config/
[root@mon ceph-iscsi-config-2.6]# python setup.py install --install-scripts=/usr/bin/
[root@mon ceph-iscsi-config-2.6]# cp usr/lib/systemd/system/rbd-target-gw.service
/lib/systemd/system/
[root@mon ceph-iscsi-config-2.6]# systemctl daemon-reload
[root@mon ceph-iscsi-config-2.6]# systemctl enable rbd-target-rgw
[root@mon ceph-iscsi-config-2.6]# systemctl start rbd-target-rgw
[root@mon ceph-iscsi-config-2.6]# systemctl status rbd-target-rgw
```

- ceph-iscsi-cli

```
[root@mon ceph-iscsi-cli]# git clone https://github.com/ceph/ceph-iscsi-cli.git
[root@mon ceph-iscsi-cli]# cd ceph-iscsi-cli/
[root@mon ceph-iscsi-cli]# python setup.py install --install-scripts=/usr/bin/
[root@mon ceph-iscsi-cli]# cp usr/lib/systemd/system/rbd-target-api.service /lib/systemd/system/
[root@mon ceph-iscsi-cli]# systemctl daemon-reload
[root@mon ceph-iscsi-cli]# systemctl enable rbd-target-api
# 编辑配置文件/etc/ceph/iscsi-gateway.cfg
[config]
# Name of the Ceph storage cluster. A suitable Ceph configuration file allowing
# access to the Ceph storage cluster from the gateway node is required, if not
# colocated on an OSD node.
cluster_name = ceph

# Place a copy of the ceph cluster's admin keyring in the gateway's /etc/ceph
# drectory and reference the filename here
gateway_keyring = ceph.client.admin.keyring


# API settings.
# The API supports a number of options that allow you to tailor it to your
# local environment. If you want to run the API under https, you will need to
# create cert/key files that are compatible for each iSCSI gateway node, that is
# not locked to a specific node. SSL cert and key files *must* be called
# 'iscsi-gateway.crt' and 'iscsi-gateway.key' and placed in the '/etc/ceph/' directory
# on *each* gateway node. With the SSL files in place, you can use 'api_secure = true'
# to switch to https mode.

# To support the API, the bear minimum settings are:
api_secure = false
```

```
# Additional API configuration options are as follows, defaults shown.
# api_user = admin
# api_password = admin
# api_port = 5001
# trusted_ip_list = 192.168.0.10,192.168.0.11

# 启动服务
[root@mon ceph-iscsi-cli]# systemctl start rbd-target-api
[root@mon ceph-iscsi-cli]# systemctl status rbd-target-api
```

5.3 RBD 性能测试方法

5.4 RBD bench 性能测试

5.5 API(python)

- rbd.RBD

```
clone(self, p_ioctx, p_name, p_snapname, c_ioctx, c_name, features=None, order=None,
stripe_unit=None, stripe_count=None, data_pool=None)

config_list(self, ioctx)

create(self, ioctx, name, size, order=None, old_format=True, features=None, stripe_unit=None,
stripe_count=None, data_pool=None)

group_create(self, ioctx, name)

group_list(self, ioctx)

group_remove(self, ioctx, name)

group_rename(self, ioctx, src, dest)

list(self, ioctx)
```

```
migration_abort(self, ioctx, image_name)

migration_commit(self, ioctx, image_name)

migration_execute(self, ioctx, image_name)

migration_prepare(self, ioctx, image_name, dest_ioctx, dest_image_name, features=None,
order=None, stripe_unit=None, stripe_count=None, data_pool=None)

migration_status(self, ioctx, image_name)

mirror_image_status_list(self, ioctx)

mirror_image_status_summary(self, ioctx)

mirror_mode_get(self, ioctx)

mirror_mode_set(self, ioctx, mirror_mode)

mirror_peer_add(self, ioctx, cluster_name, client_name)

mirror_peer_list(self, ioctx)

mirror_peer_remove(self, ioctx, uuid)

mirror_peer_set_client(self, ioctx, uuid, client_name)

mirror_peer_set_cluster(self, ioctx, uuid, cluster_name)

pool_metadata_get(self, ioctx, key)

pool_metadata_list(self, ioctx)

pool_metadata_remove(self, ioctx, key)

pool_metadata_set(self, ioctx, key, value)

remove(self, ioctx, name)

rename(self, ioctx, src, dest)

trash_get(self, ioctx, image_id)

trash_list(self, ioctx)

trash_move(self, ioctx, name, delay=0)

trash_remove(self, ioctx, image_id, force=False)
```

```
trash_restore(self, ioctx, image_id, name)

version(self)
```

- rbd.Image(ioctx, name=None, snapshot=None, read_only=False, image_id=None)

```
access_timestamp(self)

aio_discard(self, offset, length, oncomplete)

aio_flush(self, oncomplete)

aio_read(self, offset, length, oncomplete, fadvise_flags=0)

aio_write(self, data, offset, oncomplete, fadvise_flags=0)

block_name_prefix(self)

break_lock(self, client, cookie)

close(self)

config_list(self)

copy(self, dest_ioctx, dest_name, features=None, order=None, stripe_unit=None, stripe_count=None,
data_pool=None)

create_snap(self, name)

create_timestamp(self)

data_pool_id(self)

deep_copy(self, dest_ioctx, dest_name, features=None, order=None, stripe_unit=None,
stripe_count=None, data_pool=None)

diff_iterate(self, offset, length, from_snapshot, iterate_cb, include_parent=True, whole_object=False)

discard(self, offset, length)

features(self)

flags(self)
```

```
flatten(self)

flush(self)

get_name(self)

get_snap_limit(self)

get_snap_timestamp(self, snap_id)

group(self)

id(self)

invalidate_cache(self)

is_exclusive_lock_owner(self)

is_protected_snap(self, name)

list_children(self)

list_children2(self)

list_lockers(self)

list_snaps(self)

lock_acquire(self, lock_mode)

lock_break(self, lock_mode, lock_owner)

lock_exclusive(self, cookie)

lock_get_owners(self)

lock_release(self)

lock_shared(self, cookie, tag)

metadata_get(self, key)

metadata_list(self)

metadata_remove(self, key)

metadata_set(self, key, value)
mirror_image_demote(self)
```

```
mirror_image_disable(self, force)
mirror_image_enable(self)
mirror_image_get_info(self)
mirror_image_get_status(self)
mirror_image_promote(self, force)
mirror_image_resync(self)
modify_timestamp(self)
old_format(self)
op_features(self)
overlap(self)
parent_id(self)
parent_info(self
protect_snap(self, name)
read(self, offset, length, fadvise_flags=0)
rebuild_object_map(self)
remove_snap(self, name)
remove_snap2(self, name, flags)
remove_snap_by_id(self, snap_id)
remove_snap_limit(self)
rename_snap(self, srcname, dstname)
resize(self, size, allow_shrink=True)
rollback_to_snap(self, name)
set_snap(self, name)
set_snap_by_id(self, snap_id)
```

```
set_snap_limit(self, limit)

size(self)

snap_get_group_namespace(self, snap_id)

snap_get_namespace_type(self, snap_id)

snap_get_trash_namespace(self, snap_id)

stripe_count(self)

stripe_unit(self)

unlock(self, cookie)

unprotect_snap(self, name)

update_features(self, features, enabled)

watchers_list(self)

write(self, data, offset, fadvise_flags=0)
```

- rbd.SnapIterator(Image image)

```
# 属性
id (int)
size (int)
name (str)
namespace (int)
group (dict)
trash (dict)
```

5.6 librbd python 案列

- 创建镜像并写入数据

```
#!/usr/bin/python
#-*- coding: UTF-8 -*-
```

```
import rados
import rbd


def main():
    cluster = rados.Rados(conffile = 'ceph.conf')
    try:
        print('connect to ceph cluster successfully.')
        cluster.connect()
        ioctx = cluster.open_ioctx('rbdp')
        try:
            print('open pool rbdp successfully.')
            rbd_inst = rbd.RBD()
            image_list = rbd_inst.list(ioctx)
            if 'myImage' not in image_list:
                size = 10 * 1024 ** 3
                rbd_inst.create(ioctx, 'myImage', size)
                print('create image \'myImage\' successfully.')
            else:
                print('image \'myImage\' already exists.')
            image = rbd.Image(ioctx, 'myImage')
            try:
                print('start to write data to image.')
                data = 'foo' * 200
                image.write(data, 0)
                print('data write into image successfully.')
            finally:
                image.close()
        finally:
            ioctx.close()
    finally:
       cluster.shutdown()


if __name__ == '__main__':
    print('start to run.')
    main()
    print('end.')
```