****

**Ceph RBD**

Contents

[**一.简介** 5](#_Toc528572691)

[**二．原理** 5](#_Toc528572692)

[2.1 架构 5](#_Toc528572693)

[2.2 快照 6](#_Toc528572694)

[2.3 镜像 7](#_Toc528572695)

[2.4 iSCSI网关 7](#_Toc528572696)

[2.5 QEMU 8](#_Toc528572697)

[2.6 libvirt 8](#_Toc528572698)

[2.7 RBD缓存 8](#_Toc528572699)

[2.8 RBD REPLAY 9](#_Toc528572700)

[**三．操作** 9](#_Toc528572701)

[3.1 创建RBD 9](#_Toc528572702)

[3.2 查询RBD 9](#_Toc528572703)

[3.3 删除RBD 10](#_Toc528572704)

[3.4 恢复RBD 10](#_Toc528572705)

[3.5 重设置RBD大小 10](#_Toc528572706)

[3.6 映射RBD设备 10](#_Toc528572707)

[3.7 RBD快照 11](#_Toc528572708)

[3.8 克隆快照 12](#_Toc528572709)

[3.9 镜像 12](#_Toc528572710)

[3.10 iSCSI配置 13](#_Toc528572711)

[3.11 RBD Replay配置 16](#_Toc528572712)

[3.12 QEMU配置 16](#_Toc528572713)

[**四．参考资料** 16](#_Toc528572714)

[**五．附录** 17](#_Toc528572715)

[5.1 命令行 17](#_Toc528572716)

[5.2 iSCSI依赖包安装 18](#_Toc528572717)

[5.3 RBD性能测试方法 20](#_Toc528572718)

[5.4 API(python) 20](#_Toc528572719)

[5.5 librbd python案列 26](#_Toc528572720)

***\* 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/* |
|  |  |  |
|  |  |  |
|  |  |  |

***\* Release Copyleft©free \****

**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；

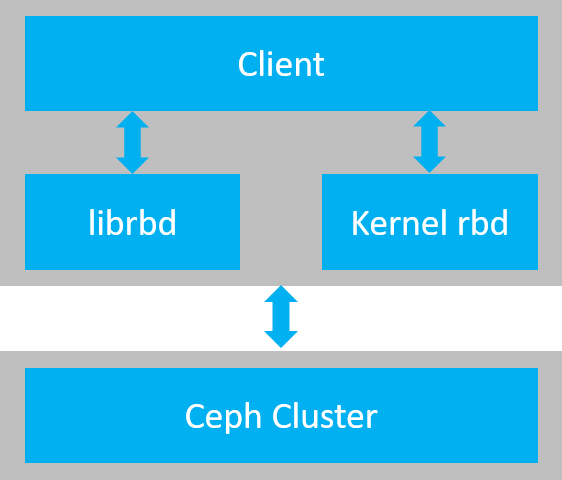


Figure 1 RBD交互图

**二．原理**

2.1 架构

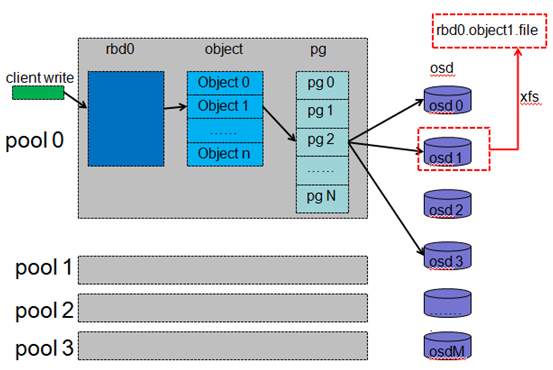
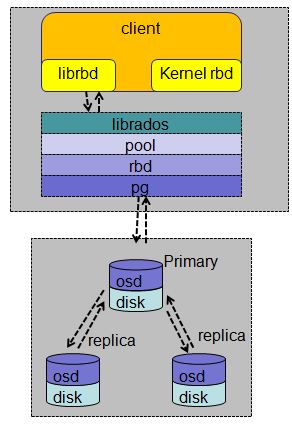


Figure 2RBD架构图

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；

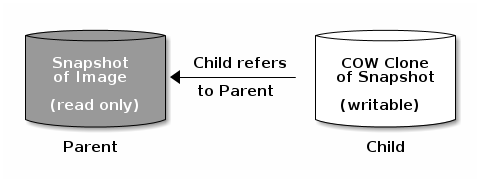


Figure 3 RBD分层特性

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：

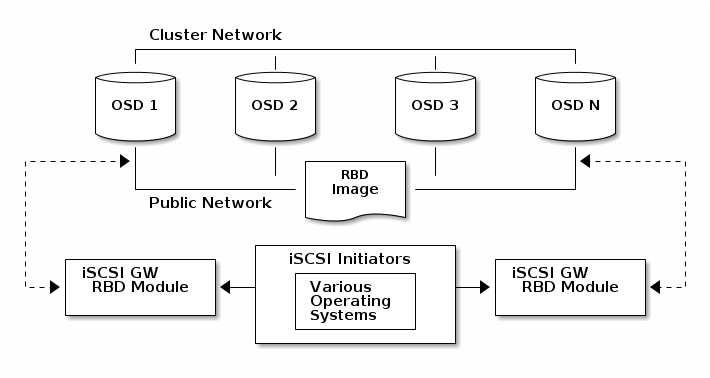
* One-way Replication(单边复制)：数据仅从主集群同步到次集群，rbd-mirror进程仅在次集群上运行；
* Two-way Replication(双边复制)：当数据从一个集群上的主Image同步到另一个集群上的次Image(反之亦可)，此时，双边都需要rbd-mirror进程的运行；

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

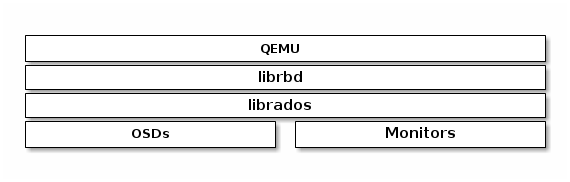
iSCSI网关集成了Ceph存储和iSCSI标准用于提供高可用的iSCSI Target，它将RBD镜像导出为SCSI磁盘，iSCSI客户端可以通过TCP/IP网络发送命令到RBD镜像。



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

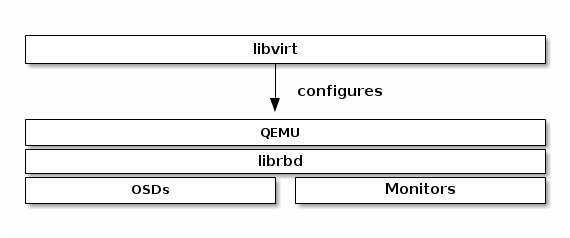
RBD最常用的场景时为虚拟机提供块设备，如下图是Ceph和QEMU相应的架构关系：



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*](#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](https://my.oschina.net/u/2460844/blog/531686)

【2】[Ceph Block Device](http://docs.ceph.com/docs/master/rbd/rbd-snapshot/)

【3】[官方文档RBD部分](http://docs.ceph.com/docs/master/rbd/)

【4】[Ceph-Ansible配置文档](http://docs.ceph.com/ceph-ansible/master/)

【5】[RBD API](http://docs.ceph.com/docs/master/rbd/api/librbdpy/#module-rbd)

**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.') |