[**理解 OpenStack + Ceph （5）：OpenStack 与 Ceph 之间的集成 [OpenStack Integration with Ceph]**](http://www.cnblogs.com/sammyliu/p/4838138.html)

理解 OpenStack + Ceph 系列文章：

（1）[安装和部署](http://www.cnblogs.com/sammyliu/p/4804037.html%20)

（2）[Ceph RBD 接口和工具](http://www.cnblogs.com/sammyliu/p/4838139.html%20)

（3）[Ceph 物理和逻辑结构](http://www.cnblogs.com/sammyliu/p/4836014.html%20)

（4）[Ceph 的基础数据结构](http://www.cnblogs.com/sammyliu/p/4843812.html%20)

（5）[Ceph 与 OpenStack 集成的实现](http://www.cnblogs.com/sammyliu/p/4838138.html)

（6）[QEMU-KVM 和 Ceph RBD 的 缓存机制总结](http://www.cnblogs.com/sammyliu/p/5066895.html)

（7）[Ceph 的基本操作和常见故障排除方法](http://www.cnblogs.com/sammyliu/p/5555218.html%20)

**1. Glance 与 Ceph RBD 集成**

**1.1 代码**

Kilo 版本中，glance-store 代码被从 glance 代码中分离出来了，地址在 <https://github.com/openstack/glance_store>。

Glance 中与 Ceph 相关的配置项：

|  |  |  |
| --- | --- | --- |
| 配置项 | 含义 | 默认值 |
| rbd\_pool | 保存rbd 卷的ceph pool 名称 | images |
| rbd\_user | rbd user id，仅仅在使用 cephx 认证时使用 | none。让 librados 根据 ceph 配置文件决定 |
| rbd\_ceph\_conf | Ceph 配置文件的完整路径 | /etc/ceph/ceph.conf |
| rbd\_store\_chunk\_size | 卷会被分成对象的大小（单位为 MB） | 64 |

Glance\_store 使用 rbd 和 rados 模块：

    import rados  
    import rbd

Glance\_store 中新增了文件 /glance\_store/\_drivers/rbd.py，其中实现了三个类：

* class StoreLocation(location.StoreLocation) #表示 Glance image 在 RBD 中的location，格式为 "rbd://<FSID>/<POOL>/<IMAGE>/<SNAP>"
* class ImageIterator(object) #实现从 RBD image 中读取数据块
* class Store(driver.Store): #实现将 RBD 作为Nova 镜像后端（backend）

class Store(driver.Store) 实现的主要方法：

（1）获取 image 数据：根据传入的Glance image location，获取 image 的 IO Interator

def get(self, location, offset=0, chunk\_size=None, context=None):

return (ImageIterator(loc.pool, loc.image, loc.snapshot, self), self.get\_size(location))

1. 根据 location，定位到 rbd image

2. 调用 image.read 方法，按照 chunk size 读取 image data

3. 将 image data 返回调用端

（2）获取 image 的 size

def get\_size(self, location, context=None):

1. 找到 store location：loc = location.store\_location

2. 使用 loc 中指定的 pool 或者配置的默认pool

3. 建立 connection 和 打开 IO Context：with rbd.Image(ioctx, loc.image, snapshot=loc.snapshot) as image

4. 获取 image.stat() 并获取 “size”

（3）添加 image

def add(self, image\_id, image\_file, image\_size, context=None):

1. 将 image\_id 作为 rbd image name：image\_name = str(image\_id)

2. 创建 rbd image：librbd.create(ioctx, image\_name, size, order, old\_format=False, features=rbd.RBD\_FEATURE\_LAYERING)。如果 rbd 支持 RBD\_FEATURE\_LAYERING 的话，创建一个 clonable snapshot：rbd://fsid/pool/image/snapshot；否则，创建一个 rbd image：rbd://image

3. 调用 image.write 方法将 image\_file 的内容按照 chunksize 依次写入 rbd image

4. 如果要创建 snapshot 的话，调用 image.create\_snap(loc.snapshot) 和 image.protect\_snap(loc.snapshot) 创建snapshot

（4）删除 image

def delete(self, location, context=None):

1. 根据 location 计算出 rbd image，snapshot 和 pool

2. 如果它是一个 snapshot （location 是 rbd://fsid/pool/image/snapshot），则 unprotect\_snap，再 remove\_snap，然后删除 image；这时候有可能出错，比如存在基于该 image 的 volume。

3. 如果它不是一个 snapshot （location 是 rbd://image），直接删除 rbd image。这操作也可能会出错。

**1.2 使用**

（1）使用 Ceph RDB 作为后端存储创建一个 Glance image：

root@controller:~/s1# glance image-show 71dc76da-774c-411f-a958-1b51816ec50f  
+------------------+----------------------------------------------------------------------------------+  
| Property         | Value                                                                            |  
+------------------+----------------------------------------------------------------------------------+  
| checksum         | 56730d3091a764d5f8b38feeef0bfcef                                                 |  
| container\_format | bare                                                                             |  
| created\_at       | 2015-09-18T10:22:49Z                                                             |  
| direct\_url         | rbd://4387471a-ae2b-47c4-b67e-9004860d0fd0/images/71dc76da-774c-                 |

|                        | 411f-a958-1b51816ec50f/snap

可见该Glance image 在 Ceph 中其实是个 RBD snapshot，其父 image 为 rbd://4387471a-ae2b-47c4-b67e-9004860d0fd0/images/71dc76da-774c-411f-a958-1b51816ec50f/，snapshot 名称为 “snap”。

查看 rdb：

[复制代码](javascript:void(0);)

root@ceph1:~# rbd info images/71dc76da-774c-411f-a958-1b51816ec50f

rbd image '71dc76da-774c-411f-a958-1b51816ec50f':

size 40162 kB in 5 objects

order 23 (8192 kB objects)

block\_name\_prefix: rbd\_data.103d2246be0e

format: 2

features: layering

[复制代码](javascript:void(0);)

Glance 自动创建了它的 snapshot：

root@ceph1:~# rbd snap ls images/71dc76da-774c-411f-a958-1b51816ec50f

SNAPID NAME SIZE

2 snap 40162 kB

查看该 snapshot：

[复制代码](javascript:void(0);)

root@ceph1:~# rbd info images/71dc76da-774c-411f-a958-1b51816ec50f@snap

rbd image '71dc76da-774c-411f-a958-1b51816ec50f':

size 40162 kB in 5 objects

order 23 (8192 kB objects)

block\_name\_prefix: rbd\_data.103d2246be0e

format: 2

features: layering

protected: True

[复制代码](javascript:void(0);)

基于该 Glance image 创建的 Cinder volume 都是该 snapshot 的 clone：

root@ceph1:~# rbd children images/71dc76da-774c-411f-a958-1b51816ec50f@snap

volumes/volume-65dbaf38-0b9d-4654-bba4-53f12cc906e3

volumes/volume-6868a043-1412-4f6c-917f-bbffb1a8d21a

此时如果试着去删除该 image，则会报错：

The image cannot be deleted because it is in use through the backend store outside of Glance.

这是因为该 image 的 rbd image 的 snapshot 被使用了。

**2. Cinder 与 Ceph RBD 的集成**

OpenStack Cinder 组件和 Ceph RBD 集成的目的是将 Cinder 卷（volume）保存在 Ceph RBD 中。当使用 Ceph RBD 作为 Cinder 的后端存储时，你不需要单独的一个 Cinder-volume 节点.

**2.1 配置项**

|  |  |  |
| --- | --- | --- |
| 配置项 | 含义 | 默认值 |
| rbd\_pool | 保存rbd 卷的ceph pool 名称 | rbd |
| rbd\_user | 访问 RBD 的用户的 ID，仅仅在使用 cephx 认证时使用 | none |
| rbd\_ceph\_conf | Ceph 配置文件的完整路径 | ‘’，表示使用 librados 的默认ceph 配置文件 |
| rbd\_secret\_uuid | rbd secret uuid |  |
| rbd\_flatten\_volume\_from\_snapshot | RBD Snapshot 在底层会快速复制一个元信息表，但不会产生实际的数据拷贝，因此当从 Snapshot 创建新的卷时，用户可能会期望不要依赖原来的 Snapshot，这个选项开启会在创建新卷时对原来的 Snapshot 数据进行拷贝来生成一个不依赖于源 Snapshot 的卷。 | false |
| rbd\_max\_clone\_depth | 卷克隆的最大层数，超过的话则使用 fallter。设为 0 的话，则禁止克隆。  与上面这个选项类似的原因，RBD 在支持 Cinder 的部分 API (如从 Snapshot 创建卷和克隆卷)都会使用 rbd clone 操作，但是由于 RBD 目前对于多级卷依赖的 IO 操作不好，多级依赖卷会有比较严重的性能问题。因此这里设置了一个最大克隆值来避免这个问题，一旦超出这个阀值，新的卷会自动被 flatten。 | 5 |
| rbd\_store\_chunk\_size | 每个 RBD 卷实际上就是由多个对象组成的，因此用户可以指定一个对象的大小来决定对象的数量，默认是 4 MB | 4 |
| rados\_connect\_timeout | 连接 ceph 集群的超时时间，单位为秒。如果设为负值，则使用默认 librados 中的值 | -1 |

从这里也能看出来，

（1）Cinder 不支持单个 volume 的条带化参数设置，而只是使用了公共配置项 rbd\_store\_chunk\_size 来指定 order。

（2）Cinder 不支持卷被附加到客户机时设置缓存模式。

**2.2 代码**

Cinder 使用的就是之前介绍过的 rbd phthon 模块：

import rados

import rbd

它实现了以下主要接口。

**2.2.1 与 RBD 的连接**

（1）初始化连接到 RBD 的连接

[复制代码](javascript:void(0);)

def initialize\_connection(self, volume, connector):

hosts, ports = self.\_get\_mon\_addrs() #调用 args = ['ceph', 'mon', 'dump', '--format=json'] 获取 monmap，再获取 hosts 和 ports

data = {

'driver\_volume\_type': 'rbd',

'data': {

'name': '%s/%s' % (self.configuration.rbd\_pool, volume['name']),

'hosts': hosts,

'ports': ports,

'auth\_enabled': (self.configuration.rbd\_user is not None),

'auth\_username': self.configuration.rbd\_user,

'secret\_type': 'ceph',

'secret\_uuid': self.configuration.rbd\_secret\_uuid, }

}

[复制代码](javascript:void(0);)

（2）连接到 ceph rados

client = self.rados.Rados(rados\_id=self.configuration.rbd\_user, conffile=self.configuration.rbd\_ceph\_conf)

client.connect(timeout= self.configuration.rados\_connect\_timeout)

ioctx = client.open\_ioctx(pool)

（3）断开连接

ioctx.close()

client.shutdown()

**2.2.2 创建卷**

Cinder API：create\_volume

调用的是 RBD 的 create 方法来创建 RBD image：

[复制代码](javascript:void(0);)

with RADOSClient(self) as client:

self.rbd.RBD().create(client.ioctx,

encodeutils.safe\_encode(volume['name']),

size,

order,

old\_format=old\_format,

features=features)

[复制代码](javascript:void(0);)

**2.2.3 克隆卷**

[复制代码](javascript:void(0);)

#创建克隆卷

def create\_cloned\_volume(self, volume, src\_vref):   
  
# 因为 RBD 的 clone 方法是基于 snapshot 的，所有 cinder 会首先创建一个 snapshot，再创建一个 clone。

if CONF.rbd\_max\_clone\_depth <= 0: #如果设置的 rbd\_max\_clone\_depth 为负数，则做一个完整的 rbd image copy

vol.copy(vol.ioctx, dest\_name)

depth = self.\_get\_clone\_depth(client, src\_name) #判断 volume 对应的 image 的 clone depth，如果已经达到 CONF.rbd\_max\_clone\_depth，则需要做 flattern

src\_volume = self.rbd.Image(client.ioctx, src\_name) #获取 source volume 对应的 rbd image

#如果需要 flattern，

\_pool, parent, snap = self.\_get\_clone\_info(src\_volume, src\_name) #获取 parent 和 snapshot

src\_volume.flatten() # 将 parent 的data 拷贝到该 clone 中

parent\_volume = self.rbd.Image(client.ioctx, parent) #获取 paraent image

parent\_volume.unprotect\_snap(snap) #将 snap 去保护

parent\_volume.remove\_snap(snap) #删除 snapshot

src\_volume.create\_snap(clone\_snap) #创建新的 snapshot

src\_volume.protect\_snap(clone\_snap) #将 snapshot 加保护

self.rbd.RBD().clone(client.ioctx, src\_name, clone\_snap, client.ioctx, dest\_name, features=client.features) #在 snapshot 上做clone

self.\_resize(volume) #如果 clone 的size 和 src volume 的size 不一样，则 resize

[复制代码](javascript:void(0);)

**3. Nova 与 Ceph 的集成**

常规地，Nova 将虚机的镜像文件放在本地磁盘或者Cinder 卷上。为了与 Ceph 集成，Nova 中添加了新的代码来将镜像文件保存在 Ceph 中。

**3.1 Nova 与 Ceph 集成的配置项**

|  |  |  |
| --- | --- | --- |
| 配置项 | 含义 | 默认值 |
| images\_type | 其值可以设为下面几个选项中的一个：   * raw：实现了 class Raw(Image) 类，在 CONF.instances\_path 指定的目录中保存 raw 格式的 镜像文件 * qcow2：实现了 class Qcow2(Image) 类，在 CONF.instances\_path 指定的目录中创建和保存 qcow2 格式的镜像文件 ['qemu-img', 'create', '-f', 'qcow2'] * lvm：实现了 class Lvm(Image) 类，在 CONF.libvirt.images\_volume\_group 指定的 LVM Volume Group 中创建 local vm 来存放 vm images * rbd：实现了 class Rbd(Image) 类 * default：使用 use\_cow\_images 配置项 | default |
| images\_rbd\_pool | 存放 vm 镜像文件的 RBD pool | rbd |
| images\_rbd\_ceph\_conf | Ceph 配置文件的完整路径 | ‘’ |
| hw\_disk\_discard | 设置使用或者不使用discard 模式，使用的话需要 Need Libvirt(1.0.6)、 Qemu1.5 (raw format) 和 Qemu1.6(qcow2 format)') 的支持  "unmap" : Discard requests("trim" or "unmap") are passed to the filesystem. "ignore": Discard requests("trim" or "unmap") are ignored and aren't passed to the filesystem. | none |
| rbd\_user | rbd user ID |  |
| rbd\_secret\_uuid | rbd secret UUID |  |

关于 discard 模式（详情见[这篇文章](http://specs.openstack.org/openstack/nova-specs/specs/juno/implemented/libvirt-disk-discard-option.html)）：

    Discard，在 SSD 上称为 trim，是在磁盘上回收不使用的数据块的机制。默认地 RBD image 是稀疏的（thin provision），这意味着只有在你写入数据时物理空间才会被占用。在 OpenStack 虚机中使用 discard 机制需要满足两个条件：

（1）虚机使用 Virtio-scsi 作为存储接口，而不是使用传统的 Virtio-blk。这是因为 Virtio-scsi 中实现了如下的新的功能：

* 设备直接暴露给客户机（device pass-through to directly expose physical storage devices to guests）
* 更好的性能（better performance and support for true SCSI device）
* 标准的设备命名方法（common and standard device naming identical to the physical world thus virtualising physical applications is made easier）
* 更好的扩展性（better scalability of the storage where virtual machines can attach more device (more LUNs etc…)）

要让虚机使用该接口，需要设置 Glance image 的属性：

$ glance image-update --property hw\_scsi\_model=virtio-scsi --property hw\_disk\_bus=scsi

（2）在 nova.conf 中配置 hw\_disk\_discard = unmap

注意目前 cinder 尚不支持 discard。

**3.2 Nova 中实现的 RBD image 操作**

Nova 在 \nova\virt\libvirt\imagebackend.py 文件中添加了支持 RBD 的新类 class Rbd(Image) 来支持将虚机的image 放在 RBD 中。其主要方法包括：

（1）image create API

def create\_image(self, prepare\_template, base, size, \*args, \*\*kwargs):

# 调用 'rbd', 'import' 命令将 image file 的数据保存到 rbd image 中

rbd import 命令：

import [–image-format format-id] [–order bits] [–stripe-unit size-in-B/K/M –stripe-count num] [–image-feature feature-name]... [–image-shared] src-path[image-spec]

Creates a new image and imports its data from path (use - for stdin). The import operation will try to create sparse rbd images if possible. For import from stdin, the sparsification unit is the data block size of the destination image (1 << order).

The –stripe-unit and –stripe-count arguments are optional, but must be used together.

虚机创建成功后，在 RBD 中查看创建出来的 image：

[复制代码](javascript:void(0);)

root@ceph1:~# rbd ls vms

74cbdb41-3789-4eae-b22e-5085de8caba8\_disk.local

root@ceph1:~# rbd info vms/74cbdb41-3789-4eae-b22e-5085de8caba8\_disk.local

rbd image '74cbdb41-3789-4eae-b22e-5085de8caba8\_disk.local':

size 1024 MB in 256 objects

order 22 (4096 kB objects)

block\_name\_prefix: rbd\_data.11552ae8944a

format: 2

features: layering

[复制代码](javascript:void(0);)

查看虚机的 xml 定义文件，能看到虚机的系统盘、临时盘和交换盘的镜像文件都在 RBD 中，而且可以使用特定的 cache 模式（由 CONF.libvirt.disk\_cachemodes 配置项指定）和 discard 模式（由 CONF.libvirt.hw\_disk\_discard 配置项设置）：

[复制代码](javascript:void(0);)

<devices>

<disk type="network" device="disk">

<driver type="raw" cache="writeback" discard="unmap"/>

<source protocol="rbd" name="vms/74cbdb41-3789-4eae-b22e-5085de8caba8\_disk.local">

<host name="9.115.251.194" port="6789"/>

<host name="9.115.251.195" port="6789"/>

<host name="9.115.251.218" port="6789"/>

</source>

<auth username="cinder">

<secret type="ceph" uuid="e21a123a-31f8-425a-86db-7204c33a6161"/>

</auth>

<target bus="virtio" dev="vdb"/>

</disk>

<disk type="network" device="disk">

<driver name="qemu" type="raw" cache="writeback"/>

<source protocol="rbd" name="volumes/volume-6868a043-1412-4f6c-917f-bbffb1a8d21a">

<host name="9.115.251.194" port="6789"/>

<host name="9.115.251.195" port="6789"/>

<host name="9.115.251.218" port="6789"/>

</source>

<auth username="cinder">

<secret type="ceph" uuid="e21a123a-31f8-425a-86db-7204c33a6161"/>

</auth>

<target bus="virtio" dev="vda"/>

<serial>6868a043-1412-4f6c-917f-bbffb1a8d21a</serial>

</disk>

[复制代码](javascript:void(0);)

    关于 libvirt.disk\_cachemodes 配置项，可以指定镜像文件的缓存模式，其值的格式为 ”A=B"，其中：

* A 可以为 'file'，'block'，'network'，'mount'。其中，file 是针对 file-backend 的disk（比如使用 qcow2 格式的镜像文件），block 是针对块设备的disk（比如使用 cinder volume 或者 lvm），network 是针对通过网络连接的设备的disk（比如 Ceph）。
* B 可以为 none,writethrough,writeback,directsync,unsafe。writethrough 和 writeback 可以参考 [理解 OpenStack + Ceph （2）：Ceph 的物理和逻辑结构](http://www.cnblogs.com/sammyliu/p/4836014.html)，其它的请自行google。
* 比如，disk\_cachemodes="network=writeback"，disk\_cachemodes="block=writeback"。

   上面的虚机 XML 定义文件是在Nova 配置为 disk\_cachemodes="network=writeback" 和 hw\_disk\_discard = unmap 的情形下生成的。

（2）image clone API

def clone(self, context, image\_id\_or\_uri):

1. 通过 Glance API 获取 image 的 rbd location

2. 检查 rbd image 是否可以被克隆（检查它是不是在本 ceph cluster 内、是不是 raw 格式、是不是可以访问等）

3. 调用 rbd 的 clone 方法来创建 clone

**3.3 镜像 clone 而非下载**

    使用传统存储作为 image 的后端存储时，在创建虚机的过程中的创建镜像文件时，都是调用  \_try\_fetch\_image\_cache 方法来从 Glance 中将镜像文件下载到本地（第一次会缓存，以后就直接读缓存而不用下载），然后再创建镜像文件的方法。而在使用 RBD 作为镜像的后端存储时，如果 Glance 镜像文件被保存在 RBD 中，那么该过程将是重复的（先通过 Glance 从 RDB 中倒出镜像，然后再由 Nova 放到RBD中），而且是非常耗时的。针对这种情况，Nova 实现了[一种新的办法](http://specs.openstack.org/openstack/nova-specs/specs/juno/implemented/rbd-clone-image-handler.html)，具体见下面的蓝色字体部分：

在 Spawn 虚机过程中，一个步骤是 create image，如下的方法会被调用：

[复制代码](javascript:void(0);)

def \_create\_image(self, context, instance,

disk\_mapping, suffix='',

disk\_images=None, network\_info=None,

block\_device\_info=None, files=None,

admin\_pass=None, inject\_files=True,

fallback\_from\_host=None):

if not booted\_from\_volume: #如果不是从 volume 启动虚机

root\_fname = imagecache.get\_cache\_fname(disk\_images, 'image\_id')

size = instance.root\_gb \* units.Gi

backend = image('disk')

if backend.SUPPORTS\_CLONE: #如果 image backend 支持 clone 的话（目前的各种 image backend，只有 RBD 支持 clone）

def clone\_fallback\_to\_fetch(\*args, \*\*kwargs):

backend.clone(context, disk\_images['image\_id']) #直接调用 backend 的 clone 函数做 image clone

fetch\_func = clone\_fallback\_to\_fetch

else:

fetch\_func = libvirt\_utils.fetch\_image #否则走常规的 image 下载-导入过程

self.\_try\_fetch\_image\_cache(backend, fetch\_func, context, root\_fname, disk\_images['image\_id'], instance, size, fallback\_from\_host)

[复制代码](javascript:void(0);)

    可见，当 nova 后端使用 ceph 时，nova driver 调用 RBD imagebackend 命令，直接在 ceph 存储层完成镜像拷贝动作（无需消耗太多的nova性能，也无需将镜像下载到hypervisor本地，再上传镜像到ceph），如此创建虚拟机时间将会大大提升。当然，这个的前提是 image 也是保存在 ceph 中，而且 image 的格式为 raw，否则 clone 过程会报错。 具体过程如下：

（1）这是 Glance image 对应的 rbd image：

[复制代码](javascript:void(0);)

root@ceph1:~# rbd info images/0a64fa67-3e34-42e7-b7b0-423c11850e18

rbd image '0a64fa67-3e34-42e7-b7b0-423c11850e18':

size 564 MB in 71 objects

order 23 (8192 kB objects)

block\_name\_prefix: rbd\_data.16d21e1d755b

format: 2

features: layering

[复制代码](javascript:void(0);)

（2）使用该 image 创建第一个虚机

[复制代码](javascript:void(0);)

root@ceph1:~# rbd info vms/982b8eac-6bcc-4a21-bd04-b67e26188be0\_disk

rbd image '982b8eac-6bcc-4a21-bd04-b67e26188be0\_disk':

size 3072 MB in 384 objects

order 23 (8192 kB objects)

block\_name\_prefix: rbd\_data.130a36a6b435

format: 2

features: layering

parent: images/0a64fa67-3e34-42e7-b7b0-423c11850e18@snap

overlap: 564 MB

root@ceph1:~# rbd info vms/982b8eac-6bcc-4a21-bd04-b67e26188be0\_disk.local

rbd image '982b8eac-6bcc-4a21-bd04-b67e26188be0\_disk.local':

size 2048 MB in 512 objects

order 22 (4096 kB objects)

block\_name\_prefix: rbd\_data.a69b2ae8944a

format: 2

features: layering

root@ceph1:~# rbd info vms/982b8eac-6bcc-4a21-bd04-b67e26188be0\_disk.swap

rbd image '982b8eac-6bcc-4a21-bd04-b67e26188be0\_disk.swap':

size 102400 kB in 25 objects

order 22 (4096 kB objects)

block\_name\_prefix: rbd\_data.a69e74b0dc51

format: 2

features: layering

[复制代码](javascript:void(0);)

（3）创建第二个虚机

[复制代码](javascript:void(0);)

root@ceph1:~# rbd info vms/a9670d9a-8aa7-49ba-baf5-9d7a450172f3\_disk

rbd image 'a9670d9a-8aa7-49ba-baf5-9d7a450172f3\_disk':

size 3072 MB in 384 objects

order 23 (8192 kB objects)

block\_name\_prefix: rbd\_data.13611f6abac6

format: 2

features: layering

parent: images/0a64fa67-3e34-42e7-b7b0-423c11850e18@snap

overlap: 564 MB

[复制代码](javascript:void(0);)

（4）会看到 Glance image 对应的 RBD image 有两个克隆，分别是上面虚机的系统盘

root@ceph1:~# rbd children images/0a64fa67-3e34-42e7-b7b0-423c11850e18@snap

vms/982b8eac-6bcc-4a21-bd04-b67e26188be0\_disk

vms/a9670d9a-8aa7-49ba-baf5-9d7a450172f3\_disk

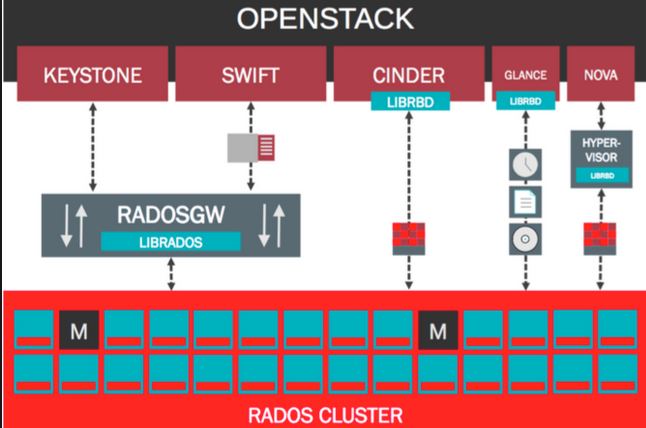
**4. 其它集成**

除了上面所描述的 Cinder、Nova 和 Glance 与 Ceph RBD 的集成外，OpenStack 和 Ceph 之间还有其它的集成点：

（1）使用 Ceph 替代 Swift 作为对象存储 （网络上有很多比较 Ceph 和 Swift 的文章，比如 [1](http://blogs.rdoproject.org/6427/ceph-and-swift-why-we-are-not-fighting),2,3,）

（2）CephFS 作为 Manila 的后端（backend）

（3）Keystone 和  Ceph Object Gateway 的集成，具体可以参考文章 [（1）](http://www.smanjara.in/integrating-ceph-rados-gateway-with-openstack-keystone/)[（2）](http://docs.ceph.com/docs/v0.86/radosgw/keystone/)



 参考文档：

<http://www.sebastien-han.fr/blog/2015/02/02/openstack-and-ceph-rbd-discard/>

[http://www.sebastien-han.fr/blog/2013/11/26/back-from-icehouse-openstack-summit-ceph-slash-openstack-integration/](http://www.sebastien-han.fr/blog/2013/11/26/back-from-icehouse-openstack-summit-ceph-slas)

<http://www.wzxue.com/openstack-cinder-advance/>