nova-cinder交互流程分析

本文主要调研cinder与nova的交互流程，分析了自有块存储系统与nova的整合问题。

# Nova现有API统计

nova已经支持的块设备API可以参考<http://api.openstack.org/api-ref.html>中Volume Attachments，Volume Extension to Compute两个部分的说明。

操作类（所有删除操作都是异步的，需要用户自行调用查询API进行确认）：

1. 创建块设备（包括从快照恢复出块设备）（可以指定块设备AZ）（需要提供用户ID）
2. 删除块设备（需要提供用户ID和块设备ID）
3. 挂载块设备（需要指定用户ID，云主机ID，块设备ID）
4. 卸载块设备（需要指定用户ID，云主机ID，块设备ID）
5. 给块设备建快照（需要提供用户ID和块设备ID）
6. 删除快照（需要提供用户ID和快照ID）

查询类：

1. 列出云主机上挂载的块设备（需要指定用户ID和云主机ID）
2. 根据云主机ID及挂载在其上的块设备ID查询挂载详细信息（需要指定用户ID，云主机ID，块设备ID）
3. 查询用户所有的块设备（需要提供用户ID）
4. 根据块设备ID查询用户某个块设备的详细信息（需要提供用户ID和块设备ID）
5. 查询用户所有的块设备快照（需要提供用户ID）
6. 查询用户所有的块设备快照详细信息（需要提供用户ID和快照ID）

需要新增API：

1. 扩容API（我们这边有新增API的经验，比较容易实现）

# Nova-Cinder交互流程分析

这里只选择两个比较典型的交互过程进行分析。

## 创建块设备cinder流程

创建块设备支持从快照恢复出块设备。

API URL：POST http://localhost:8774/v1.1/{tenant\_id}/os-volumes

Request parameters

Parameter Description

tenant\_id The unique identifier of the tenant or account.

volume\_id The unique identifier for a volume.

Volume A partial representation of a volume that is used to create a volume.

Create Volume Request**:** JSON

**{**

"volume"**:** **{**

"display\_name"**:** "vol-001"**,**

"display\_description"**:** "Another volume."**,**

"size"**:** 30**,**

"volume\_type"**:** "289da7f8-6440-407c-9fb4-7db01ec49164"**,**

"metadata"**:** **{**"contents"**:** "junk"**},**

"availability\_zone"**:** "us-east1"

**}**

**}**

Create Volume Response**:** JSON

**{**

"volume"**:** **{**

"id"**:** "521752a6-acf6-4b2d-bc7a-119f9148cd8c"**,**

"display\_name"**:** "vol-001"**,**

"display\_description"**:** "Another volume."**,**

"size"**:** 30**,**

"volume\_type"**:** "289da7f8-6440-407c-9fb4-7db01ec49164"**,**

"metadata"**:** **{**"contents"**:** "junk"**},**

"availability\_zone"**:** "us-east1"**,**

"snapshot\_id"**:** null**,**

"attachments"**:** **[],**

"created\_at"**:** "2012-02-14T20:53:07Z"

**}**

**}**

# nova\api\openstack\compute\contrib\volumes.py:

VolumeController.create()

@wsgi.serializers**(**xml**=**VolumeTemplate**)**

@wsgi.deserializers**(**xml**=**CreateDeserializer**)**

**def** create**(**self**,** req**,** body**):**

"""Creates a new volume."""

context **=** req**.**environ**[**'nova.context'**]**

authorize**(**context**)**

**if** **not** self**.**is\_valid\_body**(**body**,** 'volume'**):**

**raise** exc**.**HTTPUnprocessableEntity**()**

vol **=** body**[**'volume'**]**

# 卷类型，暂时不支持，参数不传入即可

vol\_type **=** vol**.**get**(**'volume\_type'**,** **None)**

**if** vol\_type**:**

**try:**

vol\_type **=** volume\_types**.**get\_volume\_type\_by\_name**(**context**,**

vol\_type**)**

**except** exception**.**NotFound**:**

**raise** exc**.**HTTPNotFound**()**

metadata **=** vol**.**get**(**'metadata'**,** **None)**

# 如果要从快照恢复卷，传入要被恢复的快照ID即可

snapshot\_id **=** vol**.**get**(**'snapshot\_id'**)**

**if** snapshot\_id **is** **not** **None:**

# 从快照恢复云硬盘需要实现如下方法，self.volume\_api下面会有说明

snapshot **=** self**.**volume\_api**.**get\_snapshot**(**context**,** snapshot\_id**)**

**else:**

snapshot **=** **None**

size **=** vol**.**get**(**'size'**,** **None)**

**if** size **is** **None** **and** snapshot **is** **not** **None:**

size **=** snapshot**[**'volume\_size'**]**

LOG**.**audit**(**\_**(**"Create volume of %s GB"**),** size**,** context**=**context**)**

# 卷AZ信息

availability\_zone **=** vol**.**get**(**'availability\_zone'**,** **None)**

# 云硬盘需要实现如下方法，self.volume\_api下面会有说明

new\_volume **=** self**.**volume\_api**.**create**(**context**,**

size**,**

vol**.**get**(**'display\_name'**),**

vol**.**get**(**'display\_description'**),**

snapshot**=**snapshot**,**

volume\_type**=**vol\_type**,**

metadata**=**metadata**,**

availability\_zone**=**availability\_zone

**)**

# TODO(vish): Instance should be None at db layer instead of

# trying to lazy load, but for now we turn it into

# a dict to avoid an error.

retval **=** \_translate\_volume\_detail\_view**(**context**,** dict**(**new\_volume**))**

result **=** **{**'volume'**:** retval**}**

location **=** '%s/%s' **%** **(**req**.**url**,** new\_volume**[**'id'**])**

**return** wsgi**.**ResponseObject**(**result**,** headers**=**dict**(**location**=**location**))**

# self.volume\_api说明

self**.**volume\_api **=** volume**.**API**()**

volume是**from** nova **import** volume导入的

# nova\volume\\_\_init\_\_.py:

**def** API**():**

importutils **=** nova**.**openstack**.**common**.**importutils

cls **=** importutils**.**import\_class**(**nova**.**flags**.**FLAGS**.**volume\_api\_class**)**

**return** cls**()**

可见self.volume\_api调用的所有方法都是由配置项volume\_api\_class决定的，默认配置是使用nova-volume的API封装类，

cfg.StrOpt('volume\_api\_class',

default='nova.volume.api.API',

help='The full class name of the volume API class to use'),

也可以改用cinder的API封装类，只要把配置改为volume\_api\_class=nova.volume.cinder.API即可，cinder API封装类通过调用封装了创建卷方法的cinder\_client库来调用到cinder的API，云硬盘可以实现一个类似的client库，也可以直接调用已有的API来实现相同的动作（cinder\_client库也是对cinder API调用的封装），云硬盘可以参考nova\volume\cinder.py开发自己的API封装类，供NVS使用，由于API已经开发完成，所以只是封装API，工作量应该不是很大，需要注意的应该是认证问题。

快照相关操作及查询与上述流程没有区别，只要模仿nova\volume\cinder.py即可实现。

## 挂载块设备cinder流程

API URL：POST http://localhost:8774/v2/{tenant\_id}/servers/{server\_id}/os-volume\_attachments

Request parameters

Parameter Description

tenant\_id The ID for the tenant or account in a multi-tenancy cloud.

server\_id The UUID for the server of interest to you.

volumeId ID of the volume to attach.

device Name of the device e.g. /dev/vdb. Use "auto" for autoassign (if supported).

volumeAttachment A dictionary representation of a volume attachment.

Attach Volume to Server Request**:** JSON

**{**

'volumeAttachment'**:** **{**

'volumeId'**:** volume\_id**,**

'device'**:** device

**}**

**}**

Attach Volume to Server Response**:** JSON

**{**

"volumeAttachment"**:** **{**

"device"**:** "/dev/vdd"**,**

"serverId"**:** "fd783058-0e27-48b0-b102-a6b4d4057cac"**,**

"id"**:** "5f800cf0-324f-4234-bc6b-e12d5816e962"**,**

"volumeId"**:** "5f800cf0-324f-4234-bc6b-e12d5816e962"

**}**

**}**

需要注意的是这个API返回是同步的，但挂载卷到虚拟机是异步的。

# nova\api\openstack\compute\contrib\volumes.py:

VolumeAttachmentController.create()

@wsgi.serializers**(**xml**=**VolumeAttachmentTemplate**)**

**def** create**(**self**,** req**,** server\_id**,** body**):**

"""Attach a volume to an instance."""

context **=** req**.**environ**[**'nova.context'**]**

authorize**(**context**)**

**if** **not** self**.**is\_valid\_body**(**body**,** 'volumeAttachment'**):**

**raise** exc**.**HTTPUnprocessableEntity**()**

volume\_id **=** body**[**'volumeAttachment'**][**'volumeId'**]**

device **=** body**[**'volumeAttachment'**].**get**(**'device'**)**

msg **=** \_**(**"Attach volume %(volume\_id)s to instance %(server\_id)s"

" at %(device)s"**)** **%** locals**()**

LOG**.**audit**(**msg**,** context**=**context**)**

**try:**

instance **=** self**.**compute\_api**.**get**(**context**,** server\_id**)**

# nova-compute负责挂载卷到虚拟机

device **=** self**.**compute\_api**.**attach\_volume**(**context**,** instance**,**

volume\_id**,** device**)**

**except** exception**.**NotFound**:**

**raise** exc**.**HTTPNotFound**()**

# The attach is async

attachment **=** **{}**

attachment**[**'id'**]** **=** volume\_id

attachment**[**'serverId'**]** **=** server\_id

attachment**[**'volumeId'**]** **=** volume\_id

attachment**[**'device'**]** **=** device

# NOTE(justinsb): And now, we have a problem...

# The attach is async, so there's a window in which we don't see

# the attachment (until the attachment completes). We could also

# get problems with concurrent requests. I think we need an

# attachment state, and to write to the DB here, but that's a bigger

# change.

# For now, we'll probably have to rely on libraries being smart

# TODO(justinsb): How do I return "accepted" here?

**return** **{**'volumeAttachment'**:** attachment**}**

# nova\compute\api.py:API.attach\_volume()

@wrap\_check\_policy

@check\_instance\_lock

**def** attach\_volume**(**self**,** context**,** instance**,** volume\_id**,** device**=None):**

"""Attach an existing volume to an existing instance."""

# NOTE(vish): Fail fast if the device is not going to pass. This

# will need to be removed along with the test if we

# change the logic in the manager for what constitutes

# a valid device.

**if** device **and** **not** block\_device**.**match\_device**(**device**):**

**raise** exception**.**InvalidDevicePath**(**path**=**device**)**

# NOTE(vish): This is done on the compute host because we want

# to avoid a race where two devices are requested at

# the same time. When db access is removed from

# compute, the bdm will be created here and we will

# have to make sure that they are assigned atomically.

device **=** self**.**compute\_rpcapi**.**reserve\_block\_device\_name**(**

context**,** device**=**device**,** instance**=**instance**)**

**try:**

# 云硬盘需要实现的方法，也可以参考nova\volume\cinder.py

volume **=** self**.**volume\_api**.**get**(**context**,** volume\_id**)**

# 检测卷是否可以挂载

self**.**volume\_api**.**check\_attach**(**context**,** volume**)**

# 预留要挂载的卷，防止并发挂载问题

self**.**volume\_api**.**reserve\_volume**(**context**,** volume**)**

# RPC Cast异步调用到虚拟机所在的宿主机的nova-compute服务进行挂载

self**.**compute\_rpcapi**.**attach\_volume**(**context**,** instance**=**instance**,**

volume\_id**=**volume\_id**,** mountpoint**=**device**)**

**except** Exception**:**

**with** excutils**.**save\_and\_reraise\_exception**():**

self**.**db**.**block\_device\_mapping\_destroy\_by\_instance\_and\_device**(**

context**,** instance**[**'uuid'**],** device**)**

# API在这里返回

**return** device

# nova\compute\manager.py:ComputeManager.attach\_volume()

@exception.wrap\_exception**(**notifier**=**notifier**,** publisher\_id**=**publisher\_id**())**

@reverts\_task\_state

@wrap\_instance\_fault

**def** attach\_volume**(**self**,** context**,** volume\_id**,** mountpoint**,** instance**):**

"""Attach a volume to an instance."""

**try:**

**return** self**.**\_attach\_volume**(**context**,** volume\_id**,**

mountpoint**,** instance**)**

**except** Exception**:**

**with** excutils**.**save\_and\_reraise\_exception**():**

self**.**db**.**block\_device\_mapping\_destroy\_by\_instance\_and\_device**(**

context**,** instance**.**get**(**'uuid'**),** mountpoint**)**

**def** \_attach\_volume**(**self**,** context**,** volume\_id**,** mountpoint**,** instance**):**

# 同上面的volume\_api.get方法

volume **=** self**.**volume\_api**.**get**(**context**,** volume\_id**)**

context **=** context**.**elevated**()**

LOG**.**audit**(**\_**(**'Attaching volume %(volume\_id)s to %(mountpoint)s'**),**

locals**(),** context**=**context**,** instance**=**instance**)**

**try:**

# 这里返回的是initiator信息，下面有分析

connector **=** self**.**driver**.**get\_volume\_connector**(**instance**)**

# 云硬盘需要实现的方法，下面有cinder的具体实现

connection\_info **=** self**.**volume\_api**.**initialize\_connection**(**context**,**

volume**,**

connector**)**

**except** Exception**:** # pylint: disable=W0702

**with** excutils**.**save\_and\_reraise\_exception**():**

msg **=** \_**(**"Failed to connect to volume %(volume\_id)s "

"while attaching at %(mountpoint)s"**)**

LOG**.**exception**(**msg **%** locals**(),** context**=**context**,**

instance**=**instance**)**

# 这个方法也要实现

self**.**volume\_api**.**unreserve\_volume**(**context**,** volume**)**

**if** 'serial' **not** **in** connection\_info**:**

connection\_info**[**'serial'**]** **=** volume\_id

**try:**

self**.**driver**.**attach\_volume**(**connection\_info**,**

instance**[**'name'**],**

mountpoint**)**

**except** Exception**:** # pylint: disable=W0702

**with** excutils**.**save\_and\_reraise\_exception**():**

msg **=** \_**(**"Failed to attach volume %(volume\_id)s "

"at %(mountpoint)s"**)**

LOG**.**exception**(**msg **%** locals**(),** context**=**context**,**

instance**=**instance**)**

self**.**volume\_api**.**terminate\_connection**(**context**,**

volume**,**

connector**)**

# 这个方法也要实现，作用是更新cinder数据库中的卷的状态

self**.**volume\_api**.**attach**(**context**,**

volume**,**

instance**[**'uuid'**],**

mountpoint**)**

values **=** **{**

'instance\_uuid'**:** instance**[**'uuid'**],**

'connection\_info'**:** jsonutils**.**dumps**(**connection\_info**),**

'device\_name'**:** mountpoint**,**

'delete\_on\_termination'**:** **False,**

'virtual\_name'**:** **None,**

'snapshot\_id'**:** **None,**

'volume\_id'**:** volume\_id**,**

'volume\_size'**:** **None,**

'no\_device'**:** **None}**

self**.**db**.**block\_device\_mapping\_update\_or\_create**(**context**,** values**)**

# nova\virt\libvirt\driver.py:LibvirtDriver.get\_volume\_connector()

**def** get\_volume\_connector**(**self**,** instance**):**

**if** **not** self**.**\_initiator**:**

self**.**\_initiator **=** libvirt\_utils**.**get\_iscsi\_initiator**()**

**if** **not** self**.**\_initiator**:**

LOG**.**warn**(**\_**(**'Could not determine iscsi initiator name'**),**

instance**=**instance**)**

**return** **{**

'ip'**:** FLAGS**.**my\_ip**,** #宿主机IP地址

'initiator'**:** self**.**\_initiator**,**

'host'**:** FLAGS**.**host #宿主机名

**}**

# nova\virt\libvirt\utils.py:get\_iscsi\_initiator()

**def** get\_iscsi\_initiator**():**

"""Get iscsi initiator name for this machine"""

# NOTE(vish) openiscsi stores initiator name in a file that

# needs root permission to read.

contents **=** utils**.**read\_file\_as\_root**(**'/etc/iscsi/initiatorname.iscsi'**)**

**for** l **in** contents**.**split**(**'\n'**):**

**if** l**.**startswith**(**'InitiatorName='**):**

**return** l**[**l**.**index**(**'='**)** **+** 1**:].**strip**()**

nova中cinder API封装实现：

# nova\volume\cinder.py:API.initialize\_connection():

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

**return** cinderclient**(**context**).**\

volumes**.**initialize\_connection**(**volume**[**'id'**],** connector**)**

调用的是cinder中的initialize\_connection，iscsi driver的实现如下：

# cinder\volume\iscsi.py:LioAdm.initialize\_connection()

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

volume\_iqn **=** volume**[**'provider\_location'**].**split**(**' '**)[**1**]**

**(**auth\_method**,** auth\_user**,** auth\_pass**)** **=** \

volume**[**'provider\_auth'**].**split**(**' '**,** 3**)**

# Add initiator iqns to target ACL

**try:**

self**.**\_execute**(**'rtstool'**,** 'add-initiator'**,**

volume\_iqn**,**

auth\_user**,**

auth\_pass**,**

connector**[**'initiator'**],**

run\_as\_root**=True)**

**except** exception**.**ProcessExecutionError **as** e**:**

LOG**.**error**(**\_**(**"Failed to add initiator iqn %s to target"**)** **%**

connector**[**'initiator'**])**

**raise** exception**.**ISCSITargetAttachFailed**(**volume\_id**=**volume**[**'id'**])**

# nova\virt\libvirt\driver.py:LibvirtDriver.attach\_volume()

@exception.wrap\_exception**()**

**def** attach\_volume**(**self**,** connection\_info**,** instance\_name**,** mountpoint**):**

virt\_dom **=** self**.**\_lookup\_by\_name**(**instance\_name**)**

mount\_device **=** mountpoint**.**rpartition**(**"/"**)[**2**]**

# 可能需要改动，下面会分析这个方法

conf **=** self**.**volume\_driver\_method**(**'connect\_volume'**,**

connection\_info**,**

mount\_device**)**

**if** FLAGS**.**libvirt\_type **==** 'lxc'**:**

self**.**\_attach\_lxc\_volume**(**conf**.**to\_xml**(),** virt\_dom**,** instance\_name**)**

**else:**

**try:**

# 挂载到虚拟机上

virt\_dom**.**attachDevice**(**conf**.**to\_xml**())**

**except** Exception**,** ex**:**

**if** isinstance**(**ex**,** libvirt**.**libvirtError**):**

errcode **=** ex**.**get\_error\_code**()**

**if** errcode **==** libvirt**.**VIR\_ERR\_OPERATION\_FAILED**:**

self**.**volume\_driver\_method**(**'disconnect\_volume'**,**

connection\_info**,**

mount\_device**)**

**raise** exception**.**DeviceIsBusy**(**device**=**mount\_device**)**

**with** excutils**.**save\_and\_reraise\_exception**():**

self**.**volume\_driver\_method**(**'disconnect\_volume'**,**

connection\_info**,**

mount\_device**)**

# TODO(danms) once libvirt has support for LXC hotplug,

# replace this re-define with use of the

# VIR\_DOMAIN\_AFFECT\_LIVE & VIR\_DOMAIN\_AFFECT\_CONFIG flags with

# attachDevice()

# 重新define一下，以间接实现持久化的挂载

domxml **=** virt\_dom**.**XMLDesc**(**libvirt**.**VIR\_DOMAIN\_XML\_SECURE**)**

self**.**\_conn**.**defineXML**(**domxml**)**

# nova\virt\libvirt\driver.py:LibvirtDriver.volume\_driver\_method()

**def** volume\_driver\_method**(**self**,** method\_name**,** connection\_info**,**

**\***args**,** **\*\***kwargs**):**

driver\_type **=** connection\_info**.**get**(**'driver\_volume\_type'**)**

**if** **not** driver\_type **in** self**.**volume\_drivers**:**

**raise** exception**.**VolumeDriverNotFound**(**driver\_type**=**driver\_type**)**

driver **=** self**.**volume\_drivers**[**driver\_type**]**

method **=** getattr**(**driver**,** method\_name**)**

**return** method**(**connection\_info**,** **\***args**,** **\*\***kwargs**)**

**def** \_\_init\_\_**():**

……

self**.**volume\_drivers **=** **{}**

**for** driver\_str **in** FLAGS**.**libvirt\_volume\_drivers**:**

driver\_type**,** \_sep**,** driver **=** driver\_str**.**partition**(**'='**)**

driver\_class **=** importutils**.**import\_class**(**driver**)**

self**.**volume\_drivers**[**driver\_type**]** **=** driver\_class**(**self**)**

volume\_drivers是由配置项libvirt\_volume\_drivers决定的，默认配置是：

cfg**.**ListOpt**(**'libvirt\_volume\_drivers'**,**

default**=[**

'iscsi=nova.virt.libvirt.volume.LibvirtISCSIVolumeDriver'**,**

'local=nova.virt.libvirt.volume.LibvirtVolumeDriver'**,**

'fake=nova.virt.libvirt.volume.LibvirtFakeVolumeDriver'**,**

'rbd=nova.virt.libvirt.volume.LibvirtNetVolumeDriver'**,**

'sheepdog=nova.virt.libvirt.volume.LibvirtNetVolumeDriver'

**],**

help**=**'Libvirt handlers for remote volumes.'**),**

云硬盘可以使用已有的iscsi driver，也可以参考iscsi实现自己的driver，iscsi driver的内容为：

# nova\virt\libvirt\volume.py:LibvirtISCSIVolumeDriver:

**class** **LibvirtISCSIVolumeDriver(**LibvirtVolumeDriver**):**

"""Driver to attach Network volumes to libvirt."""

**def** \_run\_iscsiadm**(**self**,** iscsi\_properties**,** iscsi\_command**,** **\*\***kwargs**):**

check\_exit\_code **=** kwargs**.**pop**(**'check\_exit\_code'**,** 0**)**

**(**out**,** err**)** **=** utils**.**execute**(**'iscsiadm'**,** '-m'**,** 'node'**,** '-T'**,**

iscsi\_properties**[**'target\_iqn'**],**

'-p'**,** iscsi\_properties**[**'target\_portal'**],**

**\***iscsi\_command**,** run\_as\_root**=True,**

check\_exit\_code**=**check\_exit\_code**)**

LOG**.**debug**(**"iscsiadm %s: stdout=%s stderr=%s" **%**

**(**iscsi\_command**,** out**,** err**))**

**return** **(**out**,** err**)**

**def** \_iscsiadm\_update**(**self**,** iscsi\_properties**,** property\_key**,** property\_value**,**

**\*\***kwargs**):**

iscsi\_command **=** **(**'--op'**,** 'update'**,** '-n'**,** property\_key**,**

'-v'**,** property\_value**)**

**return** self**.**\_run\_iscsiadm**(**iscsi\_properties**,** iscsi\_command**,** **\*\***kwargs**)**

@utils.synchronized**(**'connect\_volume'**)**

**def** connect\_volume**(**self**,** connection\_info**,** mount\_device**):**

"""Attach the volume to instance\_name"""

iscsi\_properties **=** connection\_info**[**'data'**]**

# NOTE(vish): If we are on the same host as nova volume, the

# discovery makes the target so we don't need to

# run --op new. Therefore, we check to see if the

# target exists, and if we get 255 (Not Found), then

# we run --op new. This will also happen if another

# volume is using the same target.

**try:**

self**.**\_run\_iscsiadm**(**iscsi\_properties**,** **())**

**except** exception**.**ProcessExecutionError **as** exc**:**

# iscsiadm returns 21 for "No records found" after version 2.0-871

**if** exc**.**exit\_code **in** **[**21**,** 255**]:**

self**.**\_run\_iscsiadm**(**iscsi\_properties**,** **(**'--op'**,** 'new'**))**

**else:**

**raise**

**if** iscsi\_properties**.**get**(**'auth\_method'**):**

self**.**\_iscsiadm\_update**(**iscsi\_properties**,**

"node.session.auth.authmethod"**,**

iscsi\_properties**[**'auth\_method'**])**

self**.**\_iscsiadm\_update**(**iscsi\_properties**,**

"node.session.auth.username"**,**

iscsi\_properties**[**'auth\_username'**])**

self**.**\_iscsiadm\_update**(**iscsi\_properties**,**

"node.session.auth.password"**,**

iscsi\_properties**[**'auth\_password'**])**

# NOTE(vish): If we have another lun on the same target, we may

# have a duplicate login

self**.**\_run\_iscsiadm**(**iscsi\_properties**,** **(**"--login"**,),**

check\_exit\_code**=[**0**,** 255**])**

self**.**\_iscsiadm\_update**(**iscsi\_properties**,** "node.startup"**,** "automatic"**)**

host\_device **=** **(**"/dev/disk/by-path/ip-%s-iscsi-%s-lun-%s" **%**

**(**iscsi\_properties**[**'target\_portal'**],**

iscsi\_properties**[**'target\_iqn'**],**

iscsi\_properties**.**get**(**'target\_lun'**,** 0**)))**

# The /dev/disk/by-path/... node is not always present immediately

# TODO(justinsb): This retry-with-delay is a pattern, move to utils?

tries **=** 0

**while** **not** os**.**path**.**exists**(**host\_device**):**

**if** tries **>=** FLAGS**.**num\_iscsi\_scan\_tries**:**

**raise** exception**.**NovaException**(**\_**(**"iSCSI device not found at %s"**)**

**%** **(**host\_device**))**

LOG**.**warn**(**\_**(**"ISCSI volume not yet found at: %(mount\_device)s. "

"Will rescan & retry. Try number: %(tries)s"**)** **%**

locals**())**

# The rescan isn't documented as being necessary(?), but it helps

self**.**\_run\_iscsiadm**(**iscsi\_properties**,** **(**"--rescan"**,))**

tries **=** tries **+** 1

**if** **not** os**.**path**.**exists**(**host\_device**):**

time**.**sleep**(**tries **\*\*** 2**)**

**if** tries **!=** 0**:**

LOG**.**debug**(**\_**(**"Found iSCSI node %(mount\_device)s "

"(after %(tries)s rescans)"**)** **%**

locals**())**

connection\_info**[**'data'**][**'device\_path'**]** **=** host\_device

sup **=** super**(**LibvirtISCSIVolumeDriver**,** self**)**

**return** sup**.**connect\_volume**(**connection\_info**,** mount\_device**)**

@utils.synchronized**(**'connect\_volume'**)**

**def** disconnect\_volume**(**self**,** connection\_info**,** mount\_device**):**

"""Detach the volume from instance\_name"""

sup **=** super**(**LibvirtISCSIVolumeDriver**,** self**)**

sup**.**disconnect\_volume**(**connection\_info**,** mount\_device**)**

iscsi\_properties **=** connection\_info**[**'data'**]**

# NOTE(vish): Only disconnect from the target if no luns from the

# target are in use.

device\_prefix **=** **(**"/dev/disk/by-path/ip-%s-iscsi-%s-lun-" **%**

**(**iscsi\_properties**[**'target\_portal'**],**

iscsi\_properties**[**'target\_iqn'**]))**

devices **=** self**.**connection**.**get\_all\_block\_devices**()**

devices **=** **[**dev **for** dev **in** devices **if** dev**.**startswith**(**device\_prefix**)]**

**if** **not** devices**:**

self**.**\_iscsiadm\_update**(**iscsi\_properties**,** "node.startup"**,** "manual"**,**

check\_exit\_code**=[**0**,** 255**])**

self**.**\_run\_iscsiadm**(**iscsi\_properties**,** **(**"--logout"**,),**

check\_exit\_code**=[**0**,** 255**])**

self**.**\_run\_iscsiadm**(**iscsi\_properties**,** **(**'--op'**,** 'delete'**),**

check\_exit\_code**=[**0**,** 21**,** 255**])**

也即主要实现了卷挂载到宿主机和从宿主机卸载两个方法。

## 相关代码源文件

nova\volume\cinder**.**py源文件（云硬盘需要实现的方法或者要封装的API都在这里面）： <https://github.com/openstack/nova/blob/stable/folsom/nova/volume/cinder.py>

nova\virt\libvirt\volume**.**py源文件（云硬盘需要实现的driver可以参考这个文件）： <https://github.com/openstack/nova/blob/stable/folsom/nova/virt/libvirt/volume.py>

# 默认的driver映射关系，可以看出iscsi卷使用的是LibvirtISCSIVolumeDriver

cfg**.**ListOpt**(**'libvirt\_volume\_drivers'**,**

default**=[**

'iscsi=nova.virt.libvirt.volume.LibvirtISCSIVolumeDriver'**,**

'local=nova.virt.libvirt.volume.LibvirtVolumeDriver'**,** 'fake=nova.virt.libvirt.volume.LibvirtFakeVolumeDriver'**,**

'rbd=nova.virt.libvirt.volume.LibvirtNetVolumeDriver'**,**

'sheepdog=nova.virt.libvirt.volume.LibvirtNetVolumeDriver'

**],**

help**=**'Libvirt handlers for remote volumes.'**),**

cinder处理各种API请求的抽象类源文件： <https://github.com/openstack/cinder/blob/master/cinder/volume/manager.py>

上述抽象类会调用不同的driver去执行实际的动作，完成API的请求，其中iSCSI driver源文件为：

# 默认的volume driver是cinder.volume.drivers.lvm.LVMISCSIDriver

cfg**.**StrOpt**(**'volume\_driver'**,**

default**=**'cinder.volume.drivers.lvm.LVMISCSIDriver'**,**

help**=**'Driver to use for volume creation'**),**

**]** [https://github.com/openstack/cinder/blob/master/cinder/volume/drivers/lvm.py#L304](https://github.com/openstack/cinder/blob/master/cinder/volume/drivers/lvm.py%23L304)

它继承了LVMVolumeDriver**,** driver**.**ISCSIDriver两个类，其中后一个类所在的源文件为： [https://github.com/openstack/cinder/blob/master/cinder/volume/driver.py#L199](https://github.com/openstack/cinder/blob/master/cinder/volume/driver.py%23L199) <https://github.com/openstack/cinder/blob/master/cinder/volume/driver.py#L339>这里的self.tgtadm是在 <https://github.com/openstack/cinder/blob/master/cinder/volume/drivers/lvm.py#L321>这里初始化的，调用的是 <https://github.com/openstack/cinder/blob/master/cinder/volume/iscsi.py#L460>这里的方法。

iscsi\_helper默认使用的是tgtadm：

cfg**.**StrOpt**(**'iscsi\_helper'**,**

default**=**'tgtadm'**,**

help**=**'iscsi target user-land tool to use'**),**

# 需要新增的API

1. 扩容云硬盘的API（或者直接调用云硬盘已有的API，但是推荐nova新增一个，这样云硬盘就不必对外暴露任何API了，都可以经过nova来转发处理。）

# 需要注意的问题

1. 之前云硬盘agent实现的一下错误恢复、异常处理逻辑需要在nova里面实现
2. 挂载点在云主机内外看到的不一致问题（因为nova挂载动作是异步的，所以返回给用户的是libvirt看到的挂载点，不是实际的虚拟机内部的挂载点，目前考虑通过查询卷信息接口返回最终的挂载点）
3. 用户及认证问题（之前云硬盘应该用的是管理平台的用户认证逻辑，如果改为使用nova接口，需要使用keystone的用户认证，不知道可否在管理平台那一层转换一下）

总的来说云硬盘所需要做的改动应该不大，工作重点在于封装已有的API，提供client即可（参考<https://github.com/openstack/nova/blob/stable/folsom/nova/volume/cinder.py>），另外driver（参考<https://github.com/openstack/nova/blob/stable/folsom/nova/virt/libvirt/volume.py>）里面要实现扩容逻辑，应该可以重用agent中现有的代码。