# nova 兼容Netease Block Service插件实现

## 现状说明

本文基于havana版本nova来进行代码分析和原型实现。

实现一套新的volume插件的原因是，nbs不提供兼容cinder的api，无法使用现有的cinder插件，并且libvirt volume driver也没有nbs的实现，需要重新编写代码来完成这部分工作。

已有的实现是比较不好的一种方式，也即通过配置参数在api层区分是nbs 的volume还是cinder的volume，如果是nbs的就走新增加的挂卷、卸卷等流程（nbs支持扩容卷、修改卷QoS功能），这样就要维护很多冗余代码，包括从compute/api->compute/rpc\_api->compute/manager->libvirt driver一整条代码链都要自己实现，并且绝大部分代码都是直接拷贝的cinder的实现流程，维护起来也很困难，一旦要进行大版本升级就带来很大的工作量，比如nova从F版本升级H版本的时候就花费了很长时间进行代码移植和测试工作。此外还有很多设计到卷的代码都要考虑兼容性问题，很多周边流程需要处理，导致很多与volume相关的功能在没有修改前都没办法使用。到目前为止还有很多功能都没能提供，比如unshelve、在线迁移、从volume启动虚拟机等等，我们仅仅实现了已有的基本功能（创建、删除、开机、关机、resize、cold migration等）。

## 改进目标

基于上述原因，以及更重要的一点，为后续新功能开发做准备，我们想要实现一套新的的流程，目标是最大化的复用nova已有的代码，来完成挂卷、卸卷以及更多的涉及到卷操作的各种虚拟机生命周期管理功能。

也即达到如下目标：

1. 最大化的与nova当前的cinder流程保持兼容，可以不经改动支持现有的各种涉及到volume的操作
2. 尽量不增加新的配置项（多使用已有的配置项，但配置项的功能有少许差异），减少SA的运维工作
3. 减少私有代码量，尽量重用nova中已有实现，以减少代码维护工作量
4. 为后续涉及到volume的新功能开发做准备，比如从volume启动虚拟机，在线迁移虚拟机等功能
5. 使用更多的nova流程也可以利用社区的力量来帮我们完成一定的开发测试工作，我们也可以把自己的代码贡献到社区

## 当前实现

直接拿代码来说明，这里是F版本移植H版本首次提交gerrit记录：[https://scm.service.163.org/#/c/2049/18](https://scm.service.163.org/%23/c/2049/18)，下面截取部分代码段进行说明：

nova/api/openstack/compute/contrib/volumes.py:

@wsgi.serializers(xml=VolumeAttachmentTemplate)

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

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

# Go to our owned process if we are attaching a nbs volume

if CONF.ebs\_backend == 'nbs':

return self.\_attach\_nbs\_volume(req, server\_id, body)

nova/compute/api.py:

@check\_instance\_lock

@check\_instance\_state(vm\_state=[vm\_states.ACTIVE, vm\_states.PAUSED,

vm\_states.SUSPENDED, vm\_states.STOPPED,

vm\_states.RESIZED, vm\_states.SOFT\_DELETED],

task\_state=None)

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

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

# 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.

# Raise exception if the instance is forbidden to attach volume.

allow\_attach = self.check\_allow\_attach(context, instance)

if not allow\_attach:

raise exception.NbsAttachForbidden()

# Check volume exists and is available to attach

# FIXME(wangpan): we just deal with single attachment status now

try:

volume = self.nbs\_api.get(context, volume\_id)['volumes'][0]

except (IndexError, KeyError, TypeError):

raise exception.VolumeNotFound(volume\_id=volume\_id)

nova/compute/manager.py:

@reverts\_task\_state

@wrap\_instance\_fault

def attach\_nbs\_volume(self, context, volume\_id, device, instance):

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

# TODO(wangpan): if this host is forbidden to attach nbs volume,

# an exception needs to be raised.

try:

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

device, instance)

except Exception:

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

capi = self.conductor\_api

capi.block\_device\_mapping\_destroy\_by\_instance\_and\_volume(

context, instance, volume\_id)

nova/virt/libvirt/driver.py:

def attach\_nbs\_volume(self, instance\_name, device, host\_dev,

qos\_info, volume\_id):

"""

Attach a nbs volume to instance, and check the device or slot is

in-use, return retry if in-use, if need retry, the used device is

returned, too.

"""

target\_dev = device['mountpoint'].rpartition("/")[2]

conf = vconfig.LibvirtConfigGuestDisk()

conf.source\_type = "block"

conf.driver\_name = libvirt\_utils.pick\_disk\_driver\_name(

self.get\_hypervisor\_version(), is\_block\_dev=True)

conf.driver\_format = "raw"

nova/compute/manager.py:

def \_finish\_resize():

……

nbs = (CONF.ebs\_backend == 'nbs')

block\_device\_info = self.\_get\_instance\_volume\_block\_device\_info(

context, instance, refresh\_conn\_info=True, is\_nbs=nbs)

# re-attach nbs volumes if needed.

if nbs:

bdms = block\_device\_info.get('block\_device\_mapping', [])

else:

bdms = []

same\_host = (migration['source\_compute'] == migration['dest\_compute'])

# call nbs to re-attach volumes to this host if it is not resize to

# same host.

if nbs and bdms and not same\_host:

host\_ip = utils.get\_host\_ip\_by\_ifname(CONF.host\_ip\_ifname)

for bdm in bdms:

## 改进实现

还是直接拿代码来说明，这里的提交是原型试验代码，有很多细节问题没有处理，但已经可以正常工作：[https://scm.service.163.org/#/c/3090/](https://scm.service.163.org/%23/c/3090/)。

这次的实现是增加了类似cinder的插件，主要增加了nova/volume/nbs.py模块，它的功能是模仿同一目录下的cinder.py来实现的，这样nova的其他代码只要把默认的volume api从cinder改为nbs，就可以直接调用，它主要调用了nova/volume/nbs\_client.py，这个文件是之前已经实现了的，主要用来调用nbs的api，跟nbs服务打交道，实现查询卷信息、挂载卷到宿主机、从宿主机上卸载卷等各种需要nbs完成的操作；另外还在nova/virt/libvirt/volume.py模块里面增加了LibvirtNBSVolumeDriver，这个类主要是为libvirt生成挂盘所需要的xml文件，由于实际底层挂卷到宿主机是nbs的agent来负责的，所以这部分功能也不用加到这里了（其他volume服务有些是需要的）。

简单来说就是主要增加nbs交互的前端模块（与nbs api交互）和半个后端模块（与libvirt交互，缺少了nbs自己维护的agent那部分功能）。

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

\_volume\_opts = [

oslo.config.cfg.StrOpt('ebs\_backend',

default='cinder',

help='The backend type of ebs service, '

'should be nbs or cinder'),

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

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

help='The full class name of the '

'volume API class to use'),

]

"""

Handles all requests relating to volumes + nbs.

"""

import datetime

from nova.db import base

from nova import exception

from nova.openstack.common.gettextutils import \_

from nova.openstack.common import jsonutils

from nova.openstack.common import log as logging

from nova.volume import nbs\_client

LOG = logging.getLogger(\_\_name\_\_)

NBS\_CLIENT = None

def nbsclient():

global NBS\_CLIENT

if NBS\_CLIENT is None:

NBS\_CLIENT = nbs\_client.API()

return NBS\_CLIENT

def \_untranslate\_volume\_summary\_view(context, vol):

"""Maps keys for volumes summary view."""

d = {}

d['id'] = vol['volumeId']

d['status'] = vol['status']

d['size'] = vol['size']

d['availability\_zone'] = vol['availabilityZone']

created\_at = long(vol['createTime']) / 1000

created\_at = datetime.datetime.utcfromtimestamp(created\_at)

created\_at = created\_at.strftime("%Y-%m-%d %H:%M:%S")

d['created\_at'] = created\_at

d['attach\_time'] = ""

d['mountpoint'] = ""

if vol['attachments']:

att = vol['attachments'][0]

d['attach\_status'] = att['status']

d['instance\_uuid'] = att['instanceId']

d['mountpoint'] = att['device']

d['attach\_time'] = att['attachTime']

else:

d['attach\_status'] = 'detached'

d['display\_name'] = vol['volumeName']

d['display\_description'] = vol['volumeName']

# FIXME(wangpan): all nbs volumes are 'share' type, so we fix here to 0

d['volume\_type\_id'] = 0

d['snapshot\_id'] = vol['snapshotId']

# NOTE(wangpan): nbs volumes don't have metadata attribute

d['volume\_metadata'] = {}

# NOTE(wangpan): nbs volumes don't have image metadata attribute now

return d

class API(base.Base):

"""API for interacting with the volume manager."""

def get(self, context, volume\_id):

item = nbsclient().get(context, volume\_id)['volumes'][0]

return \_untranslate\_volume\_summary\_view(context, item)

def get\_all(self, context, search\_opts={}):

items = nbsclient().get(context)['volumes']

rval = []

for item in items:

rval.append(\_untranslate\_volume\_summary\_view(context, item))

return rval

def check\_attached(self, context, volume):

"""Raise exception if volume not in use."""

if volume['status'] != "in-use":

msg = \_("status must be 'in-use'")

raise exception.InvalidVolume(reason=msg)

def check\_attach(self, context, volume, instance=None):

# TODO(vish): abstract status checking?

if volume['status'] != "available":

msg = \_("status must be 'available'")

raise exception.InvalidVolume(reason=msg)

if volume['attach\_status'] in ("attached", "attachedInVM"):

msg = \_("already attached")

raise exception.InvalidVolume(reason=msg)

def check\_detach(self, context, volume):

# TODO(vish): abstract status checking?

if volume['status'] == "available":

msg = \_("already detached")

raise exception.InvalidVolume(reason=msg)

if volume['attach\_status'] not in ("attached", "attachedInVM"):

msg = \_("volume not attached")

raise exception.InvalidVolume(reason=msg)

def reserve\_volume(self, context, volume\_id):

"""We do not need to reserve nbs volume now."""

pass

def unreserve\_volume(self, context, volume\_id):

"""We do not need to unreserve nbs volume now."""

pass

def begin\_detaching(self, context, volume\_id):

"""We do not need to notify nbs begin detaching volume now."""

pass

def roll\_detaching(self, context, volume\_id):

"""We do not need to roll detaching nbs volume now."""

pass

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

"""We do not need to change volume state now.

We implement this operation in volume driver.

"""

pass

def post\_attach(self, context, volume\_id, instance\_uuid,

mountpoint, host\_ip):

"""Tell NBS manager attachment success."""

device = jsonutils.loads(mountpoint)

return nbsclient().notify\_nbs\_libvirt\_result(context, volume\_id,

'attach', True, device=device['real\_path'],

host\_ip=host\_ip, instance\_uuid=instance\_uuid)

def detach(self, context, volume\_id):

"""We do not need to change volume state now.

We implement this operation in volume driver.

"""

pass

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

"""We do attachment of nbs volume to host in the method."""

instance\_uuid = connector['instance\_uuid']

host\_ip = connector['ip']

device = jsonutils.loads(connector['device'])

real\_path = device['real\_path']

result = nbsclient().attach(context, volume\_id, instance\_uuid,

host\_ip, real\_path)

if result is None:

raise exception.NbsException()

# check volume status, wait for nbs attaching finish

succ = nbsclient().wait\_for\_attached(context, volume\_id,

instance\_uuid)

if not succ:

raise exception.NbsTimeout()

# get host dev path and QoS params from nbs

return nbsclient().get\_host\_dev\_and\_qos\_info(

context, volume\_id, host\_ip)

def terminate\_connection(self, context, volume\_id, connector):

"""We do detachment of nbs volume from host in the method."""

host\_ip = connector['ip']

return nbsclient().detach(context, volume\_id, host\_ip)

def migrate\_volume\_completion(self, context, old\_volume\_id, new\_volume\_id,

error=False):

raise NotImplementedError()

def create(self, context, size, name, description, snapshot=None,

image\_id=None, volume\_type=None, metadata=None,

availability\_zone=None):

"""We do not support create nbs volume now."""

raise NotImplementedError()

def delete(self, context, volume\_id):

"""We do not support delete nbs volume now."""

raise NotImplementedError()

def update(self, context, volume\_id, fields):

raise NotImplementedError()

def get\_snapshot(self, context, snapshot\_id):

"""We do not support nbs volume snapshot now."""

raise NotImplementedError()

def get\_all\_snapshots(self, context):

"""We do not support nbs volume snapshot now."""

raise NotImplementedError()

def create\_snapshot(self, context, volume\_id, name, description):

"""We do not support nbs volume snapshot now."""

raise NotImplementedError()

def create\_snapshot\_force(self, context, volume\_id, name, description):

"""We do not support nbs volume snapshot now."""

raise NotImplementedError()

def delete\_snapshot(self, context, snapshot\_id):

"""We do not support nbs volume snapshot now."""

raise NotImplementedError()

def get\_volume\_encryption\_metadata(self, context, volume\_id):

"""We do not support for encrypting nbs volume snapshot now."""

return {}

def get\_volume\_metadata(self, context, volume\_id):

raise NotImplementedError()

def delete\_volume\_metadata(self, context, volume\_id, key):

raise NotImplementedError()

def update\_volume\_metadata(self, context, volume\_id,

metadata, delete=False):

raise NotImplementedError()

def get\_volume\_metadata\_value(self, volume\_id, key):

raise NotImplementedError()

def update\_snapshot\_status(self, context, snapshot\_id, status):

"""We do not support nbs volume snapshot now."""

raise NotImplementedError()

nova/virt/libvirt.py:

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

default=[

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

'iser=nova.virt.libvirt.volume.LibvirtISERVolumeDriver',

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

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

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

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

'nfs=nova.virt.libvirt.volume.LibvirtNFSVolumeDriver',

'aoe=nova.virt.libvirt.volume.LibvirtAOEVolumeDriver',

'glusterfs='nova.virt.libvirt.volume.LibvirtGlusterfsVolumeDriver',

'fibre\_channel=nova.virt.libvirt.volume.LibvirtFibreChannelVolumeDriver',

'scality=nova.virt.libvirt.volume.LibvirtScalityVolumeDriver',

'nbs=nova.virt.libvirt.volume.LibvirtNBSVolumeDriver',

],

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

nova/virt/libvirt/volume.py:

class LibvirtNBSVolumeDriver(LibvirtBaseVolumeDriver):

"""Driver to attach NetEase Block Service volume to libvirt."""

def \_\_init\_\_(self, connection):

"""Create back-end to NBS."""

super(LibvirtNBSVolumeDriver,

self).\_\_init\_\_(connection, is\_block\_dev=False)

def connect\_volume(self, connection\_info, disk\_info):

"""Returns xml for libvirt."""

import ipdb;ipdb.set\_trace()

conf = super(LibvirtNBSVolumeDriver,

self).connect\_volume(connection\_info,

disk\_info)

conf.source\_type = 'block'

conf.source\_path = connection\_info['host\_dev']

conf.slot = disk\_info['device']['slot']

return conf

def disconnect\_volume(self, connection\_info, disk\_dev):

"""Disconnect the volume."""

pass

## 剩余工作

剩下的工作主要包括：

1. 完善nbs插件代码：主要是要确认不同操作的时候试验代码能否满足要求，比如卸载nbs卷操作是否能用试验代码来完成
2. 清理旧的实现：包括挂载卷、卸载卷两个主要功能，以及较多的保证兼容性的冗余代码
3. 细节完善：包括为nbs新增的部分功能（带slot号挂载卷、支持卷的QoS设置、相关通知操作等），以及补充相关bug修复代码到nova的各种卷操作流程（如防止频繁挂卸载卷等）
4. 外围功能验证：支持带nbs情况下resize、离线迁移、强制重启、查询虚拟机详细信息可显示挂载的卷等功能
5. 兼容性验证及完善：支持带nbs情况下的其他功能如shelve、resume等等，以及已有nbs卷的兼容性问题（要支持虚拟机上挂载已有卷的情况下，使用修改后的代码完成各种生命周期操作，以及卸载卷操作等）