NetApp Replication功能分析

# 对接NetApp Replication功能

NetApp的Replication功能基于NetApp原生的SnapMirror的异步镜像功能实现。SnapMirror会在源SVM和镜像SVM之间建立对等关系，所有对源SVM的操作会被同步到到镜像SVM中。一旦源SVM出现故障，Cinder可以通过failover-host命令切换到镜像SVM，保证系统可以继续工作。

## NetApp对接步骤

1. 创建镜像SVM

略

1. 配置Cinder-Volume的cinder.conf

[DEFAULT]

enabled\_backends = NetAppIscsiBackend

[NetAppIscsiBackend]

netapp\_vserver = <source\_svm>

replication\_device = backend\_id:NetAppIscsiBackend\_Replication

netapp\_replication\_aggregate\_map = backend\_id:NetAppIscsiBackend\_Replication,aggr1\_NetApp9000\_01:aggr1\_NetApp9000\_02

# 其他配置项

[NetAppIscsiBackend\_Replication]

netapp\_vserver = <target\_svm>

# 其他配置项

上述配置，[NetAppIscsiBackend]中配置源SVM，[NetAppIscsiBackend\_Replication]中配置镜像SVM，当Cinder启动服务时只会启动[NetAppIscsiBackend]中的配置。

配置项replication\_device指定了[NetAppIscsiBackend]对应的镜像配置是[NetAppIscsiBackend\_Replication]，此处每一个源可以配置对个镜像设备。

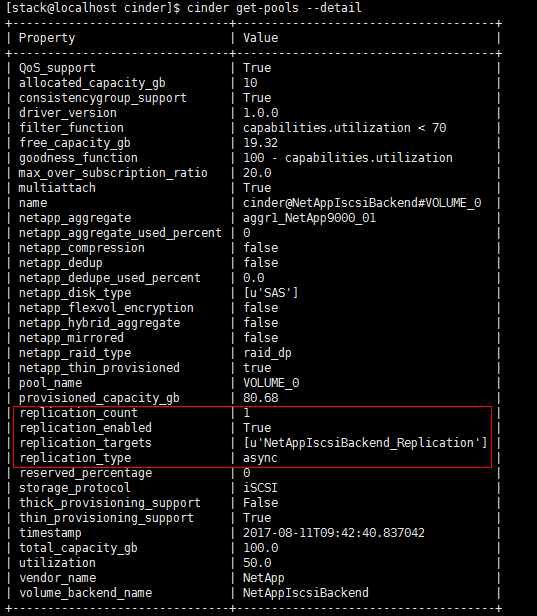
netapp\_replication\_aggregate\_map配置指示了源SVM和镜像SVM中VOLUME的聚合对应关系，按照上述配置，假设源SVM中存在VOLUME\_0，为位于aggr1\_NetApp9000\_01聚合中，那么他的镜像在将会创建在aggr1\_NetApp9000\_02聚合中。

注意：源Backend和镜像Backend的驱动（volume\_driver）必须一致，且其他配置项也应该一致。

1. 重启Cinder\_Volume服务

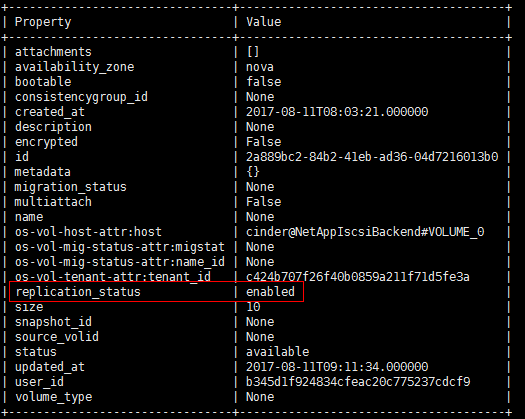
重启服务执行get-pool查看Backend状态，看到Capabilities中replication\_enabled=True，且包含其他relication信息。

如下图：



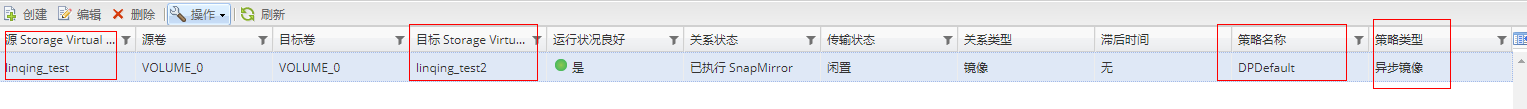
1. 创建卷

创建卷发现卷的replication\_status状态为enabled

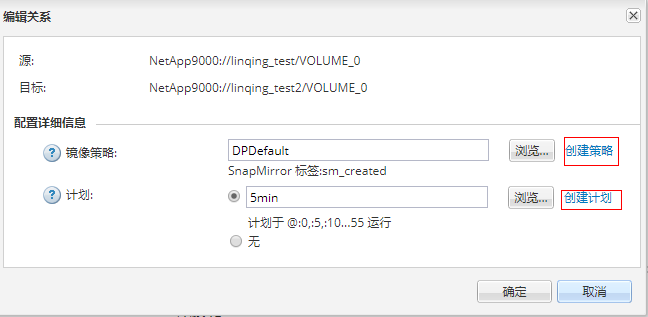


1. 登录NetApp的WEB页面查看Snapmirror信息

点击“保护 -> 关系”，可以看到源SVM和镜像SVM中建立了对等关系，对等关系是以SVM中VOLUME为单位的，源SVM中有几个VOLUME，netapp的驱动就会建立个对等关系。



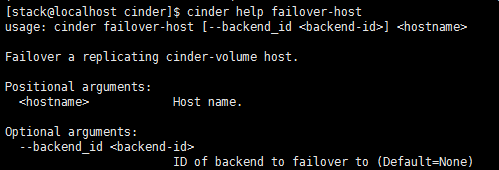
编辑对等关系可以修改SnapMirror策略，当前Cinder使用的关系类型是异步镜像（其他还有存储、镜像存储），计划制定了镜像的同步周期（NetApp的Cinder驱动默认1小时同步一次）。



比较源SVM和镜像SVM可以发现，两个SVM的内容完全一样！

# 故障切换

当源设备发生故障时可以通过failover-host命令切换到镜像



以上参数中：

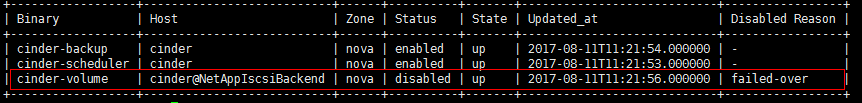
backend\_id：指定了要切换的镜像目标（即replication\_device参数指定的那个）；

hostname：指定发生故障需要切换的backend（通过service-list命令可以获取）；

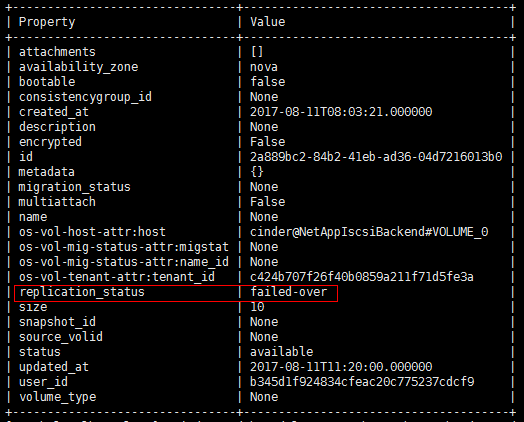
使用以下命名切换到备机：

cinder failover-host --backend\_id NetAppIscsiBackend\_Replication cinder@NetAppIscsiBackend

查看当前服务状态，发现cinder@NetAppIscsiBackend的对应服务被禁用，且get-pools也已经无法获取相应的Pool信息。



查看cinder@NetAppIscsiBackend下的卷信息，replication\_status状态变为failed-over，其他状态正常，卷可以挂载使用。但是此时无法新建卷（原因是服务被禁用了，无法调度到这个目录），但是可以删除卷。



# 从故障恢复

按照failover-host的代码逻辑，从当backend\_id指定为default时，backend应该从存储备机切换回主机，且清除所有的failed-over状态。

从目前NetApp的驱动代码分析，NetApp驱动似乎没有实现恢复的逻辑，执行

cinder failover-host --backend\_id default cinder@NetAppIscsiBackend

直接抛出异常

# 代码分析

NetApp驱动和Replication相关的逻辑在下面的python文件中。

/cinder/volume/drivers/netapp/dataontap/utils/data\_motion.py

## 配置Relication时的启动过程分析

cinder-volume启动时会调用netapp驱动代码初始化NetAppBlockStorageCmodeLibrary，该类包含了集群模式下NetApp的所有功能逻辑：

def \_\_init\_\_(self, driver\_name, driver\_protocol, \*\*kwargs):

super(NetAppBlockStorageCmodeLibrary, self).\_\_init\_\_(driver\_name,driver\_protocol,\*\*kwargs)

# 读取netapp驱动本身规定的附加配置项

self.configuration.append\_config\_values(na\_opts.netapp\_cluster\_opts)

# 集群模式

self.driver\_mode = 'cluster'

# active\_backend\_id参数被存放在Service表中，正常情形下值为None，

# 当backend执行过failover操作时，该值被设定为备机的backend\_id!

self.failed\_over\_backend\_name = kwargs.get('active\_backend\_id')

self.failed\_over = self.failed\_over\_backend\_name is not None

# 判断设备是否启用Replication功能，

# get\_replication\_backend\_names读取配置文件的replication\_device配置

self.replication\_enabled = (

True if self.get\_replication\_backend\_names(

self.configuration) else False)

初始化成功NetAppBlockStorageCmodeLibrary后，会紧接着调用do\_setup方法，创建一个到Target连接。根据backend的failover状态，do\_setup会读取配置文件中的不同section。

def do\_setup(self, context):

super(NetAppBlockStorageCmodeLibrary, self).do\_setup(context)

na\_utils.check\_flags(self.REQUIRED\_CMODE\_FLAGS, self.configuration)

# get\_client\_for\_backend接口会根据传入的backend\_id创建一个客户端连接

# 如果当前failed\_over\_backend\_name非空，即backend状态时failover的，

# 那么初始化时，创建备机的SVM客户端

self.zapi\_client = dot\_utils.get\_client\_for\_backend(

self.failed\_over\_backend\_name or self.backend\_name)

self.vserver = self.zapi\_client.vserver

......

\_add\_looping\_tasks方法会在驱动初始化过程中被check\_for\_setup\_error调用，该方法会注册若干周期任务，其中\_handle\_housekeeping\_tasks和Relication功能相关，执行周期是10分钟，并且立即执行。

handle\_housekeeping\_tasks会定时检查Relication的状态，如果replication\_enabled = True，并且backend未曾failover，那么会调用ensure\_snapmirrors打开NetApp的snapmirrors功能，Replication正是基于snapmirrors。

def \_add\_looping\_tasks(self):

# \_update\_ssc() ： 扫描当前SVM中的Volume是否发生了变动，周期是一小时？

self.loopingcalls.add\_task(self.\_update\_ssc,

loopingcalls.ONE\_HOUR,

loopingcalls.ONE\_HOUR)

# remove\_unused\_qos\_policy\_groups : Qos的软删除接口

self.loopingcalls.add\_task(

self.zapi\_client.remove\_unused\_qos\_policy\_groups,

loopingcalls.ONE\_MINUTE,

loopingcalls.ONE\_MINUTE)

# \_handle\_housekeeping\_tasks ： Replication功能接口，执行周期是10min，并且立即执行

self.loopingcalls.add\_task(

self.\_handle\_housekeeping\_tasks,

loopingcalls.TEN\_MINUTES,

0)

# 启动定时任务

super(NetAppBlockStorageCmodeLibrary, self).\_add\_looping\_tasks()

ensure\_snapmirrors为每一源Volume在Relication设备（SVM）上创建一个镜像Volume。镜像Volume和源Volume完全同名，大小一致。镜像Volume位于哪个聚合取决于在netapp\_replication\_aggregate\_map中定义的聚合映射关系。如果Volume已经存在不会重新创建，而是在原有的基础上建立，镜像和源的对等关系（SnapMirror relationship）。

def ensure\_snapmirrors(self, config, src\_backend\_name, src\_flexvol\_names):

"""Ensure all the SnapMirrors needed for whole-backend replication."""

backend\_names = self.get\_replication\_backend\_names(config)

for dest\_backend\_name in backend\_names:

for src\_flexvol\_name in src\_flexvol\_names:

dest\_flexvol\_name = src\_flexvol\_name

self.create\_snapmirror(src\_backend\_name,

dest\_backend\_name,

src\_flexvol\_name,

dest\_flexvol\_name)

create\_snapmirror完成初始化Snapmirror的功能，包括下面的流程。

流程中比较重要的包括，初次创建镜像Volume时的initialize操作，以及后续的resync和resume。其中，initialize用来启动基线拷贝，resume操作使后续的镜像同步可用，resync命令开始一次异步镜像同步。

1.检查镜像VOLUME是否存在 ：

volume.drivers.netapp.dataontap.client.client\_cmode.Client:flexvol\_exist

2.镜像卷不存在创建FlexVolume ：

volume.drivers.netapp.dataontap.utils.data\_motion.DataMotionMixin:create\_destination\_flexvol

3.查看镜像和源之间是否有SnapMirror relationship：

cinder.volume.drivers.netapp.dataontap.client.client\_cmode.Client : get\_snapmirrors

4.如果SnapMirror relationship不存在创建SnapMirror relationship，并且初始化 ，流程结束：

cinder.volume.drivers.netapp.dataontap.client.client\_cmode.Client : create\_snapmirror，initialize\_snapmirror

等价于命令：

snapmirror create -source-vserver <src\_vserver\_name>

-source-volume <volume\_name>

-destination-vserver <tgt\_vserver\_name>

-destination-volume <volume\_name>

-schedule hourly

-type DP

snapmirror initialize -source-vserver <src\_vserver\_name>

-source-volume <volume\_name>

-destination-vserver <tgt\_vserver\_name>

-destination-volume <volume\_name>

5.检查SnapMirror relationship的状态，如果状态是snapmirrored（说明上一次同步发生的错误）流程直接结束

6.执行resume操作

cinder.volume.drivers.netapp.dataontap.client.client\_cmode.Client : resume\_snapmirror

等价于命令

snapmirror resume -source-vserver <src\_vserver\_name>

-source-volume <volume\_name>

-destination-vserver <tgt\_vserver\_name>

-destination-volume <volume\_name>

7.执行resync操作

cinder.volume.drivers.netapp.dataontap.client.client\_cmode.Client : resync\_snapmirror

等价于命令

snapmirror resync -source-vserver <src\_vserver\_name>

-source-volume <volume\_name>

-destination-vserver <tgt\_vserver\_name>

-destination-volume <volume\_name>

## failover-host命令逻辑分析

首先，c-api 调用 \_replication\_db\_change 修改Service的relication状态（对应的字段是replication\_status），从enabled、failed-over到 failing-over。

之后c-vol调用failover接口，通知后端进行切换。

def failover(self, context, secondary\_backend\_id=None):

updates = {}

repl\_status = fields.ReplicationStatus

svc\_host = vol\_utils.extract\_host(self.host, 'backend')

service = objects.Service.get\_by\_args(context, svc\_host,constants.VOLUME\_BINARY)

# 获取当前backend下的所有卷信息

volumes = self.\_get\_my\_volumes(context)

exception\_encountered = True

try:

# 调用driver的failover方法，该方法返回Replication设备的backend\_id,

# 并且会修改Volume的状态。他的返回值中，active\_backend\_id表示切换到的backend，

# volume\_update\_list表示切换过程中对volume信息的修改

failover = getattr(self.driver,'failover' if service.is\_clustered

else 'failover\_host')

active\_backend\_id, volume\_update\_list = failover(

context,

volumes,

secondary\_id=secondary\_backend\_id)

exception\_encountered = False

except exception.UnableToFailOver:

LOG.exception(\_LE("Failed to perform replication failover"))

updates['replication\_status'] = repl\_status.FAILOVER\_ERROR

except exception.InvalidReplicationTarget:

# Relication 设备不合法，切换失败。

# 如果指定的backend\_id是None，那么切换Service的replication\_status状态为ENABLED，否者设为FAILED\_OVER

LOG.exception(\_LE("Invalid replication target specified for failover"))

if service.active\_backend\_id:

updates['replication\_status'] = repl\_status.FAILED\_OVER

else:

updates['replication\_status'] = repl\_status.ENABLED

except exception.VolumeDriverException:

# 如果驱动报异常错误，禁用该backend

LOG.error(\_LE("Driver reported error during replication failover."))

updates.update(disabled=True,

replication\_status=repl\_status.FAILOVER\_ERROR)

if exception\_encountered:

# 持久化数据库

LOG.error(\_LE("Error encountered during failover on host: "

"%(host)s invalid target ID %(backend\_id)s"),

{'host': self.host, 'backend\_id':

secondary\_backend\_id})

self.finish\_failover(context, service, updates)

return

# 当backend\_id = default时，被Cinder视为对Replication进行failback，

# 此时会重置replication\_status的状态。否者禁用backend，并且记录active\_backend\_id

if secondary\_backend\_id == "default":

updates['replication\_status'] = repl\_status.ENABLED

updates['active\_backend\_id'] = ''

updates['disabled'] = service.frozen

updates['disabled\_reason'] = 'frozen' if service.frozen else ''

else:

updates['replication\_status'] = repl\_status.FAILED\_OVER

updates['active\_backend\_id'] = active\_backend\_id

updates['disabled'] = True

updates['disabled\_reason'] = 'failed-over'

self.finish\_failover(context, service, updates)

# 更新卷的信息

for update in volume\_update\_list:

if not update.get('volume\_id'):

raise exception.UnableToFailOver(

reason=\_("Update list, doesn't include volume\_id"))

vobj = objects.Volume.get\_by\_id(context, update['volume\_id'])

vobj.update(update.get('updates', {}))

vobj.save()

LOG.info(\_LI("Failed over to replication target successfully."))

NetApp驱动中实际进行failover-host的逻辑是：

volume.drivers.netapp.dataontap.utils.data\_motion.DataMotionMixin: \_failover\_host

要求传入的backend\_id必须在replication\_device参数中定义，并且不能和主backend\_id重名。之后\_failover\_host调用，\_complete\_failover接口切断镜像和源的联系，并且更新Volume状态。当\_complete\_failover接口返回，会切换当前的SVM连接到Replication设备，并且更新驱动的failed\_over 为True，failed\_over\_backend\_name 为Replication设备的backend\_id。

\_complete\_failover接口包含以下操作：

1. 主动发起一次同步，尽量挽救数据。等价的命令为：snapmirror update
2. 断开镜像和源的Snapmirror连接。
3. 修改volume的replication\_status装态为failed-over

def \_complete\_failover(self, source\_backend\_name, replication\_targets,

flexvols, volumes, failover\_target=None):

"""Failover a backend to a secondary replication target."""

volume\_updates = []

# 虽然可以自动选择，但似乎这个逻辑没有什么用

active\_backend\_name = failover\_target or self.\_choose\_failover\_target(

source\_backend\_name, flexvols, replication\_targets)

if active\_backend\_name is None:

msg = \_("No suitable host was found to failover.")

raise exception.NetAppDriverException(msg)

source\_backend\_config = config\_utils.get\_backend\_configuration(

source\_backend\_name)

# 1. 主动发起一次同步，尽量挽救数据。等价的命令为：snapmirror update

self.update\_snapmirrors(source\_backend\_config, source\_backend\_name,

flexvols)

# 2. 断开镜像和源的Snapmirror连接。包括以下步骤：

# snapmirror quiesce ：禁用之后的同步，并且等待当前同步完成，等待时间取决于netapp\_snapmirror\_quiesce\_timeout(默认3600)，

# 该过程如果出现错误会执行snapmirror abort

# snapmirror abort ： 终止当前正在进行的同步

# snapmirror break ：断开连接

# volume mount ： 挂载镜像Volume到命令空间，创建一个接合路径，主要针对NAS协议

failed\_to\_break = self.break\_snapmirrors(source\_backend\_config,

source\_backend\_name,

flexvols, active\_backend\_name)

# 3. 修改volume的replication\_status装态为failed-over

for volume in volumes:

replication\_status = fields.ReplicationStatus.FAILED\_OVER

volume\_pool = volume\_utils.extract\_host(volume['host'],

level='pool')

if volume\_pool in failed\_to\_break:

replication\_status = 'error'

volume\_update = {

'volume\_id': volume['id'],

'updates': {

'replication\_status': replication\_status,

},

}

volume\_updates.append(volume\_update)

return active\_backend\_name, volume\_updates

# 为NetApp驱动添加Failback逻辑，使他支持切换后回滚

## 背景

当后端设备执行failover-host切换到Replication设备后，如果主设备恢复正常，那么通过下面的命令可以将后端设备从Replication设备切换回主设备。

cinder failover --backend\_id default <backend>

netapp驱动没有实现当backend\_id指定为default时，对后端设备进行回滚的逻辑需要修改代码手工实现。

## 实现方案

修改cinder/volume/drivers/netapp/dataontap/utils/data\_motion.py文件中，DataMotionMixin类的\_failover\_host接口。该接口判断传入的backend\_id是否为default，如果不是执行原来逻辑，如果不是调用\_complete\_failback接口回滚replication设备。

\_complete\_failback接口执行回退的过程包括以下步骤：

1. 建立备机到主机的同步关系，将主机宕机过程中备机上的修改和入主机，并且等待传输完成！
2. 一次同步完成后，删除1中建立的同步
3. 重建主机到备机的同步关系
4. 修改卷的replication\_status状态

# NetApp SnapMirror功能相关命令

## 列出卷的SnapMirror relationships

snapmirror list-destinations -source-vserver <src-vserver> -source-volume <src-volume>

## 释放卷的SnapMirror relationships

snapmirror release -source-vserver <src-vserver> -source-volume <src-volume> -destination-vserver <target-vserver> -destination-volume <target-volume>

# 问题

## 切换Replication时对in-use状态的卷是否有影响

failover命令只会切换Cinder连接的SVM，并且修改卷的replication\_status状态。对于in-use状态的卷已经建立了iscsi连接不会产生影响。Relication机制不会让Nova切换正在使用的磁盘。

# 参考

<https://docs.openstack.org/cinder/ocata/devref/replication.html>

<http://geek.csdn.net/news/detail/76325>

<http://specs.openstack.org/openstack/cinder-specs/specs/pike/replication-group.html>

<http://specs.openstack.org/openstack/cinder-specs/specs/ocata/ha-aa-replication.html>

<http://specs.openstack.org/openstack/cinder-specs/specs/liberty/replication_v2.html>

http://specs.openstack.org/openstack/cinder-specs/specs/juno/volume-replication.html