[**理解 OpenStack 高可用（HA）（5）：RabbitMQ HA**](http://www.cnblogs.com/sammyliu/p/4730517.html)

本系列会分析OpenStack 的高可用性（HA）概念和解决方案：

（1）[OpenStack 高可用方案概述](http://www.cnblogs.com/sammyliu/p/4741967.html%20)

（2）[Neutron L3 Agent HA - VRRP （虚拟路由冗余协议）](http://www.cnblogs.com/sammyliu/p/4692081.html%20)

（3）[Neutron L3 Agent HA - DVR （分布式虚机路由器）](http://www.cnblogs.com/sammyliu/p/4713562.html%20)

（4）[Pacemaker 和 OpenStack Resource Agent （RA）](http://www.cnblogs.com/sammyliu/p/5025362.html%20)

（5）[RabbitMQ HA](http://www.cnblogs.com/sammyliu/p/4730517.html)

（6）[MySQL HA](http://www.cnblogs.com/sammyliu/p/4995868.html%20)

**1. RabbitMQ 集群**

    你可以使用若干个RabbitMQ 节点组成一个 RabbitMQ 集群。集群解决的是扩展性问题。所有的数据和状态都会在集群内所有的节点上被复制，只有queue是例外。默认的情况下，消息只会存在于它被创建的节点上，但是它们在所有节点上可见和可访问。

    对于Queue 来说，消息实体只存在于其中一个节点，A、B两个节点仅有相同的元数据，即队列结构。当消息进入A节点的Queue中后，consumer从B节点拉取时，RabbitMQ会临时在A、B间进行消息传输，把A中的消息实体取出并经过B发送给consumer。所以consumer应尽量连接每一个节点，从中取消息。即对于同一个逻辑队列，要在多个节点建立物理Queue。否则无论consumer连A或B，出口总在A，会产生瓶颈。该模式存在一个问题就是当A节点故障后，B节点无法取到A节点中还未消费的消息实体。如果做了消息持久化，那么得等A节点恢复，然后才可被消费；如果没有持久化的话，然后就没有然后了。

   因此，在集群环境中,队列只有元数据会在集群的所有节点同步，但队列中的数据只会存在于一个节点，数据没有冗余且容易丢，甚至在durable的情况下,如果所在的服务器节点宕机，就要等待节点恢复才能继续提供消息服务。那既然有这种问题，为什么依然有这个选项呢？官方的说法是：

   （1）存储空间：如果集群的每个节点都有每个queue的一个拷贝，那么增加节点将无法增加存储容量。比如，如果一个节点可以存放 1GB 的消息，增加另外两个节点只会增加另外两个拷贝而已。

  （2）性能：发布消息，将会将它拷贝其它节点上。如果是持久性消息，那么每个节点上都会触发磁盘操作。你的网络和磁盘负载在每次增加节点时都会增加。

  可见，RabbitMQ Clustering 技术不能完全解决HA 问题。单纯的集群只适合于在不需要HA的场景中使用。

**2. Active/Passive HA 方案**

    RabbitMQ A/P HA [官方方案](https://www.rabbitmq.com/pacemaker.html) 是采用 Pacemaker + （DRBD 或者其它可靠的共享 NAS/SNA 存储） + （CoroSync 或者 Heartbeat 或者 OpenAIS）组合来实现的。需要注意的是 CoroSync 需要使用多播，而多播在某些云环境中是被禁用的，这时候可以改为使用 Heartbeat，它采用单播。其架构为：

实现 HA 的原理：

1. RabbitMQ 将数据库文件和持久性消息保存在 DRBD 挂载点上。注意，在 failover 时，非持久性消息会丢失。
2. DRBD 要求在某一时刻，只能有一个 node （primary）能够访问其共享存储，因此，不会有多个node 同时写一块数据的风险。这要求必须依赖 Pacemaker 来控制使用 DRBD 共享存储的 node。Pacemaker 使得在某一时刻，只有 active node 来访问 DRBD 共享存储。在 failover 时，Pacemaker 会卸载当前 active node 上的 DRBD 共享存储，并在新的 active node （原 secondary node）上挂载 DRBD 共享存储。
3. 在 node 启动时，不会自动启动 RabbitMQ。Pacemaker 会在 active node 启动 RabbitMQ。
4. RabbitMQ HA 只会提供一个访问 RabbitMQ 的 虚拟IP 地址。该方案依赖 Pacemaker 来保证 VIP 的切换。

**2.1 基于 Pacemaker + DRBD + CoroSync 的 RabbitMQ HA 方案配置**

    RabbitMQ 官方的 [这篇文章](https://www.rabbitmq.com/pacemaker.html)介绍了基于 Pacemaker 的 RabbitMQ HA 方案。它同时指出，这是传统的 RabbitMQ HA 方案，并且建议使用 RabbitMQ 集群 + 镜像 Message Queue 的方式来实现 A/A HA。使用 Pacemaker 实现 HA 的方案主要步骤包括：

1. 为 RabbitMQ 配置一个 DRBD 设备
2. 配置 RabbitMQ 使用建立在 DRBD 设备之上的数据目录
3. 选择并绑定一个可以在各集群节点之间迁移的虚拟 IP 地址 （即 VIP ）
4. 配置 RabbitMQ 监听该 IP 地址
5. 使用 Pacemaker 管理上述所有资源，包括 RabbitMQ 守护进程本身

**2.1.1 安装 Corosync 和 Pacemaker**

本配置使用两个节点。

首先，在两个节点上安装软件包：apt-get install pacemaker crmsh corosync cluster-glue resource-agents，并配置 Corosync，设它为 boot 自启动：vi /etc/default/corosync，START=yes。

两个节点上，

（1）修改配置文件：vim /etc/corosync/corosync.conf

[复制代码](javascript:void(0);)

totem {

#...

interface {

ringnumber: 0

bindnetaddr: #节点1上，192.168.1.15；节点2上，192.168.1.32

broadcast: yes

mcastport: 5405

}

transport: udpu

}

nodelist {

node {

ring0\_addr: 192.168.1.15

nodeid: 1

}

node {

ring0\_addr: 192.168.1.32

nodeid: 2

}

}

[复制代码](javascript:void(0);)

（2）启动服务 service corosync start

（3）查看状态

[复制代码](javascript:void(0);)

root@compute2:/home/s1# corosync-cfgtool -s

Printing ring status.  
Local node ID 2  
RING ID 0  
id = 192.168.1.32  
status = ring 0 active with no faults

root@compute2:/home/s1# corosync-cmapctl runtime.totem.pg.mrp.srp.members #需要确保两个节点都加入了组  
runtime.totem.pg.mrp.srp.members.1.config\_version (u64) = 0  
runtime.totem.pg.mrp.srp.members.1.ip (str) = r(0) ip(192.168.1.15)  
runtime.totem.pg.mrp.srp.members.1.join\_count (u32) = 1  
runtime.totem.pg.mrp.srp.members.1.status (str) = joined  
runtime.totem.pg.mrp.srp.members.2.config\_version (u64) = 0  
runtime.totem.pg.mrp.srp.members.2.ip (str) = r(0) ip(192.168.1.32)  
runtime.totem.pg.mrp.srp.members.2.join\_count (u32) = 1  
runtime.totem.pg.mrp.srp.members.2.status (str) = joined

[复制代码](javascript:void(0);)

（4）启动 pacemaker：service pacemaker start

（5）查看 pacemaker 状态：crm\_mon

[复制代码](javascript:void(0);)

Last updated: Sun Aug 16 15:59:10 2015

Last change: Sun Aug 16 15:58:59 2015 via crmd on compute1

Stack: corosync

Current DC: compute1 (1) - partition with quorum

Version: 1.1.10-42f2063

3 Nodes configured

0 Resources configured

Node compute1 (1084752143): UNCLEAN (offline)

Online: [ compute1 compute2 ]

[复制代码](javascript:void(0);)

（6）配置 pacemaker

[复制代码](javascript:void(0);)

root@compute2:/home/s1# crm

crm(live)# configure

crm(live)configure# property no-quorum-policy="ignore" \ #因为这里只有两个节点，因此需要设置为 ‘ignore’。两个节点以上，不可以设置为 ‘ignore’

> pe-warn-series-max="1000" \

> pe-input-series-max="1000" \

> pe-error-series-max="1000" \

> cluster-recheck-interval="2min"

crm(live)configure# quit

There are changes pending. Do you want to commit them? yes

bye

[复制代码](javascript:void(0);)

**2.1.2 为 RabbitMQ 配置一个 DRBD 设备**

 在 node1 和 node2 上，依次进行：

（0）安装 DRBD，并设置机器启动时 DRBD 服务不自动启动。该服务的启停升降级等都需要由 Pacemaker 控制。

（1）添加一个 1G 的 hard disk，fdiskl -l 可见 /dev/sdb

（2）创建 LVM：pvcreate /dev/sdb，vgcreate vg\_rabbit /dev/sdb，lvcreate -L1000 -n rabbit vg\_rabbit。创建好的 LVM 为 /dev/mapper/vg\_rabbit-rabbit。

（3）定义 rabbitmq 资源：vi /etc/drbd.d/rabbitmq.res

根据 /etc/drbd.conf，资源的配置文件需要放置在 /etc/drbd.d/ 目录中，并以 res 为文件类型。DRBD 会自动读取这些配置文件并创建相应的资源。

[复制代码](javascript:void(0);)

rource rabbitmq {

device minor 1; # DRBD device 是 /dev/drbd1，相当于 device /dev/drbd1

disk "/dev/mapper/vg\_rabbit-rabbit"; # Raw device, 本例中是 LVM path。也可以使用不同名字的 Raw device，这样的话需要放到 host 定义部分。

meta-disk internal; # 这是默认值，更多信息可以参考[这个](http://www.drbd.org/users-guide-emb/ch-internals.html#s-metadata)

on compute1 { # 节点1，名称需要与 uname -n 的输出相同。

address ipv4 192.168.1.15:7701; #7701 是 DRBD 通信的 TCP 端口，需要确保该端口可以被访问

}

on compute2 { # 节点2

address ipv4 192.168.1.32:7701;

}

}

[复制代码](javascript:void(0);)

 （4）drbdadm create-md rabbitmq

root@compute2:/home/s1# drbdadm create-md rabbitmq

Writing meta data...

initializing activity log

NOT initializing bitmap

New drbd meta data block successfully created.

（5）drbdadm up rabbitmq  
  
在 compute 2 上，  
（1）drbdadm -- --force primary rabbitmq  
（2）mkfs -t xfs /dev/drbd1 （如果提示 mkfs.xsf 找不到，则 sudo apt-get install xfsprogs）

[复制代码](javascript:void(0);)

meta-data=/dev/drbd1 isize=256 agcount=4, agsize=63996 blks

= sectsz=512 attr=2, projid32bit=0

data = bsize=4096 blocks=255983, imaxpct=25

= sunit=0 swidth=0 blks

naming =version 2 bsize=4096 ascii-ci=0

log =internal log bsize=4096 blocks=1200, version=2

= sectsz=512 sunit=0 blks, lazy-count=1

realtime =none extsz=4096 blocks=0, rtextents=0

[复制代码](javascript:void(0);)

（3）mount /dev/drbd1 /var/lib/rabbitmq

（4）umount /var/lib/rabbitmq

（3）drbdadm secondary rabbitmq

在 compute 1 上：

1. drbdadm -- --force primary rabbitmq
2. mkfs -t xfs /dev/drbd1
3. mount /dev/drbd1 /var/lib/rabbitmq

至此，DRBD 设置完成，目前 compute 1 上的 DRBD 是主，compute 2 上的是备。查看其状态：

[复制代码](javascript:void(0);)

root@compute1:/home/s1# /etc/init.d/drbd status

drbd driver loaded OK; device status:

version: 8.4.3 (api:1/proto:86-101)

srcversion: 6551AD2C98F533733BE558C

m:res cs ro ds p mounted fstype

1:rabbitmq Connected Primary/Secondary UpToDate/UpToDate C /var/lib/rabbitmq xfs

root@compute2:/home/s1# /etc/init.d/drbd status

drbd driver loaded OK; device status:

version: 8.4.3 (api:1/proto:86-101)

srcversion: 6551AD2C98F533733BE558C

m:res cs ro ds p mounted fstype

1:rabbitmq Connected Secondary/Primary UpToDate/UpToDate C

[复制代码](javascript:void(0);)

调试时遇到一个脑裂问题，就是两个node上 drbd 的状态都是 StandAlone，然后 DRBD 服务在 compute 2 上能起来，在 compute 1 上起不来。

[复制代码](javascript:void(0);)

root@compute1:/home/s1# /etc/init.d/drbd status

drbd driver loaded OK; device status:

version: 8.4.3 (api:1/proto:86-101)

srcversion: 6551AD2C98F533733BE558C

m:res cs ro ds p mounted fstype

1:rabbitmq StandAlone Secondary/Unknown UpToDate/DUnknown r-----

DRBD's startup script waits for the peer node(s) to appear.

- In case this node was already a degraded cluster before the

reboot the timeout is 0 seconds. [degr-wfc-timeout]

- If the peer was available before the reboot the timeout will

expire after 0 seconds. [wfc-timeout]

(These values are for resource 'rabbitmq'; 0 sec -> wait forever)

To abort waiting enter 'yes' [ 148]:

[复制代码](javascript:void(0);)

[这篇文章](http://www.gossamer-threads.com/lists/drbd/users/13667) 和 [这篇文章](https://drbd.linbit.com/users-guide/s-resolve-split-brain.html) 给出了这个问题的解决办法：

* 在起不来 DRBD 服务的节点（compute1）上，运行 drbdadm disconnect rabbitmq，drbdadm secondary rabbitmq，和 drbdadm --discard-my-data connect rabbitmq
* 在能起来 DRBD 服务的节点（compute2）上，运行 drbdadm connect rabbitmq
* 然后 DRBD 状态恢复成 Connected 了。

SUSE 有关于 DRBD 安装、配置和测试的 [不错的文档](https://www.suse.com/documentation/sle_ha/book_sleha/data/sec_ha_drbd_test.html) 。

**2.1.3 安装和配置 RabbitMQ**

（1）在两个节点上安装 RabbitMQ。rabbitmq-server 进程会在 rabbitmq 组内的 rabbitmq 用户下运行。

（2）确保 rabbitmq 组和用户在两个节点上具有相同的 ID。运行 cat /etc/passwd | grep rabbitmq 和 cat /etc/group | grep rabbitmq 查看 ID，必要时进行修改。

（3）确保 rabbitmq 用户对 /var/lib/rabbitmq/ 有读写权限。 运行 chmod a+w /var/lib/rabbitmq/。

（4）确保各节点都使用相同的 Erlang cookie。需要将 compute 1 上的 RabbitMQ cookie 拷贝到 compute 2 上：

（1）compute 1：scp /var/lib/rabbitmq/.erlang.cookie s1@compute2:/home/s1  
（2）compute 2：mv .erlang.cookie /var/lib/rabbitmq/  
（3）compute 2：chown rabbitmq: /var/lib/rabbitmq/.erlang.cookie  
（4）compute 2：chmod 400 /var/lib/rabbitmq/.erlang.cookie  
（5）再执行相同的步骤，将该文件拷贝到 DRBD 共享存储文件系统中

（5）确保计算机启动时不自动启动 rabbitmq。修改 /etc/init.d/rabbitmq-server 文件，在文件头注释后面添加 exit 0.

**2.1.4 在 Pacemaker 中添加 RabbitMQ 资源**

在添加前，在 compute 1 节点上，运行 crm configure，输入：

[复制代码](javascript:void(0);)

primitive p\_ip\_rabbitmq ocf:heartbeat:IPaddr2 \ #定义访问 RabbitMQ 的 VIP。使用 RA ocf:heartbeat:IPaddr2。对该 VIP 的监控间隔为 10s。

params ip="192.168.1.222" cidr\_netmask="24" \

op monitor interval="10s"

primitive p\_drbd\_rabbitmq ocf:linbit:drbd \ #定义RabbitMQ 的 DRBD 资源，pacemaker 会对它 start，stop，promote，demote 和 monitor。

params drbd\_resource="rabbitmq" \

op start timeout="90s" \ #启动的超时时间

op stop timeout="180s" \

op promote timeout="180s" \

op demote timeout="180s" \

op monitor interval="30s" role="Slave" \ #对 Slave drbd 的监控间隔定义为 30s

op monitor interval="29s" role="Master" #对 master drbd 的监控间隔定义为 29s

primitive p\_fs\_rabbitmq ocf:heartbeat:Filesystem \ # 在 /dev/drbd1 上创建 xfs 文件系统，并 mount 到 /var/lib/rabbitmq

params device="/dev/drbd1" directory="/var/lib/rabbitmq" fstype="xfs" options="relatime" \

op start timeout="60s" op stop timeout="180s" op monitor interval="60s" timeout="60s"

primitive p\_rabbitmq ocf:rabbitmq:rabbitmq-server \ #定义 RabbitMQ 资源

params nodename="rabbit@localhost" \

mnesia\_base="/var/lib/rabbitmq" \

op monitor interval="20s" timeout="10s"

group g\_rabbitmq p\_ip\_rabbitmq p\_fs\_rabbitmq p\_rabbitmq #group 指定一组资源需要在同一个node上 ip -> fs -> rabbitmq，顺序启动，逆序停止

ms ms\_drbd\_rabbitmq p\_drbd\_rabbitmq meta notify="true" master-max="1" clone-max="2" # 定义一个资源集合，往往用于 master-slave 资源集群。其中，clone-max 定义集合内的 drbd 节点数目，master-max 指定 master 最多只能有一个；notify = “true” 表示启用notification。

colocation c\_rabbitmq\_on\_drbd inf: g\_rabbitmq ms\_drbd\_rabbitmq:Master #colocation 定义在同一个节点上启动的资源。这里定义了一个约束，rabbitmq 只能在 DRBD Master 节点上启动

order o\_drbd\_before\_rabbitmq inf: ms\_drbd\_rabbitmq:promote g\_rabbitmq:start #oder 指定服务启动顺序。这个定义要求先 DRBD 升级为 Master 然后依次执行 ip，fs 和 rabbitmq 启动。

[复制代码](javascript:void(0);)

更多的 Pacemaker 配置细节，请参考 [这个文档](http://clusterlabs.org/doc/en-US/Pacemaker/1.1/html-single/Pacemaker_Explained)。

Pacemaker 一些有用的命令：

* crm status：显示集群状态
* crm configure show：显示集群配置
* crm configure edit：修改集群配置
* crm node standby：将当前节点设为备节点
* crm node online：将当前节点设为主节点
* crm ra meta ocf:heartbeat:IPaddr2：查看 RA 的 metadata

配置完成后，RabbitMQ 起不来。一堆问题，挨个来说：

（1）Corosync 调试：设置日志文件

[复制代码](javascript:void(0);)

logging {

fileline: off

to\_stderr: yes

to\_logfile: yes

to\_syslog: yes

logfile: /var/log/cluster/corosync.log

syslog\_facility: daemon

debug: off

timestamp: on

}

[复制代码](javascript:void(0);)

（2）pacemaker 调试：pacemaker 的日志在 corosync 的 log file 中。

（3）运行 crm status，如果出现  Could not establish cib\_ro connection: Connection refused (111) ，原因是 pacemaker 服务没有启动。运行 service pacemaker start 即可。

（4）rabbitmq 启动的错误日志可以在 /var/log/rabbitmq 目录下找到。可能的错误之一是，rabbit 用户没有目录 /var/lib/rabbitmq 的写权限。

成功后，是这样的效果：

[复制代码](javascript:void(0);)

root@compute2:/var/lib/rabbitmq/rabbit@localhost# crm status

Last updated: Mon Aug 17 00:16:11 2015

Last change: Sun Aug 16 22:54:27 2015 via cibadmin on compute1

Stack: corosync

Current DC: compute1 (1) - partition with quorum

Version: 1.1.10-42f2063

3 Nodes configured

5 Resources configured

Online: [ compute1 compute2 ] # pacemaker 集群中有两个节点

OFFLINE: [ compute1 ] # compute 1 是备节点

Resource Group: g\_rabbitmq

p\_ip\_rabbitmq (ocf::heartbeat:IPaddr2): Started compute2 #在 compute 2 上启动 VIP，成功

p\_rabbitmq (ocf::rabbitmq:rabbitmq-server): Started compute2 #在 compute 2 上启动 rabbitmq，成功

Master/Slave Set: ms\_drbd\_rabbitmq [p\_drbd\_rabbitmq]

Masters: [ compute2 ]

Slaves: [ compute1 ]

p\_fs\_rabbitmq (ocf::heartbeat:Filesystem): Started compute2 # 在 compute 2 上设置 drbd 文件系统，成功

[复制代码](javascript:void(0);)

可见，根据配置，pacemaker 会（1）根据 corosync 确定集群中的节点和状态 （2）启动 VIP （3）启动指定的服务 （4）设置 DRBD 文件系统。每一步都可能失败，失败则需要调试。

（5）将主节点停机，rabbitmq 服务顺利地被切换到备节点上。

**2. 基于集群+镜像队列的A/A 方案**

    从 3.6.0 版本开始，RabbitMQ 支持镜像队列功能，官方文档在[这里](http://www.rabbitmq.com/clustering.html)。。与普通集群相比，其实质和普通模式不同之处在于，消息实体会主动在镜像节点间同步，而不是在 consumer 取数据时临时拉取。该模式带来的副作用也很明显，除了降低系统性能外，如果镜像队列数量过多，加之大量的消息进入，集群内部的网络带宽将会被这种同步通讯大大消耗掉。所以在对可靠性要求较高的场合中适用。

    Mirrorred queue 是 RabbitMQ 高可用的一种方案，相对于普通的集群方案来讲，queue中的消息每个节点都会存在一份 copy, 这个在单个节点失效的情况下，整个集群仍旧可以提供服务。但是由于数据需要在多个节点复制，在增加可用性的同时，系统的吞吐量会有所下降。

    选举机制：mirror queue 内部实现了一套选举算法，有一个 master 和多个slave，queue 中的消息以 master 为主。镜像队列有主从之分，一个主节点(master)，0个或多个从节点(slave)。当master宕掉后，会在 slave 中选举新的master，其选举算法为最早启动的节点。 若master节点失效，则 mirror queue 会自动选举出一个节点（slave中消息队列最长者）作为master，作为消息消费的基准参考; 在这种情况下可能存在ack消息未同步到所有节点的情况(默认异步)，若 slave 节点失效，mirror queue 集群中其他节点的状态无需改变。所以，看起来可以使用两节点的镜像队列集群。

    使用：对于publish，可以选择任意一个节点进行连接，rabbitmq内部若该节点不是master，则转发给master，master向其他slave节点发送该消息，后进行消息本地化处理，并组播复制消息到其他节点存储；对于consumer，可以选择任意一个节点进行连接，消费的请求会转发给master,为保证消息的可靠性，consumer需要进行ack确认，master收到ack后，才会删除消息，ack消息会同步(默认异步)到其他各个节点，进行slave节点删除消息。

   考虑到性能问题，复制可以在可控范围内进行，包括集群内全节点复制的镜像队列和集群内局部节点复制的镜像队列两种模式。

**2.1 配置**

    多个单独的 RabbitMQ 服务，可以加入到一个集群中，也可以从集群中退出。集群中的 RabbitMQ 服务，使用同样的 Erlang cookie（unix 系统上默认为 /var/lib/rabbitmq/.erlang.cookie）。所有在一个 RabbitMQ 服务中被操作的数据和状态（date/state）都会被同步到集群中的其它节点。

   镜像队列的配置通过添加 policy 完成，policy 添加的命令为：

[复制代码](javascript:void(0);)

* rabbitmqctl set\_policy [-p Vhost] Name Pattern Definition [Priority]
* -p Vhost: 可选参数，针对指定vhost下的queue进行设置
* Name: policy的名称
* Pattern: queue的匹配模式（正则表达式）
* Definition: 镜像定义，包括三个部分 ha-mode，ha-params，ha-sync-mode
  + ha-mode: 指明镜像队列的模式，有效值为 all/exactly/nodes
    - all表示在集群所有的节点上进行镜像
    - exactly 表示在指定个数的节点上进行镜像，节点的个数由ha-params指定
    - nodes 表示在指定的节点上进行镜像，节点名称通过ha-params指定
  + ha-params: ha-mode模式需要用到的参数
  + ha-sync-mode: 镜像队列中消息的同步方式，有效值为automatic，manually
* Priority: 可选参数， policy的优先级

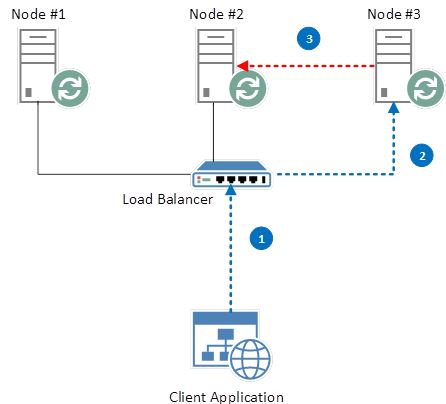
[复制代码](javascript:void(0);)

例如，对队列名称以 ’hello‘ 开头的所有队列进行镜像，并在集群的两个节点上完成镜像，policy的设置命令为：

rabbitmqctl set\_policy hello-ha "^hello" '{"ha-mode":"exactly","ha-params":2,"ha-sync-mode":"automatic"}'

**2.2 HAProxy 和 RabbitMQ A/A 集群**

RabbitMQ 实现镜像队列的方式比较特别。[这篇文章](http://insidethecpu.com/2014/11/17/load-balancing-a-rabbitmq-cluster/)进行了深入的阐述。假设有如下的配置：



创建 queue 的过程：

1. LB 将 client request 分发到 node 2，client 创建队列 “NewQueue”，然后开始向其中放入 message。
2. 最终，后端服务会对 node 2 上的 “NewQueue” 创建一个快照，并在一段时间内将其拷贝到node 1 和 3 上。这时候，node2 上的队列是 master Queue，node 1 和 3 上的队列是 slave queue。

假如现在 node2 宕机了：

* node 2 不再响应心跳，它会被认为已经被从集群中移出了
* node 2 上的 master queue 不再可用
* RabbitMQ 将 node 1 或者 3 上的 salve instance 升级为 master instance

假设 master queue 还在 node 2 上，客户端通过 LB 访问该队列：

1. 客户端连接到集群，要访问 “NewQueue” 队列
2. LB 根据配置的轮询算法将请求分发到一个节点上
3. 假设客户端请求被转到 node 3 上
4. RabbitMQ 发现 “NewQueue” master node 是 node 2
5. RabbitMQ 将消息转到 node 2 上
6. 最终客户端成功连接到 node 2 上的 master 队列

可见，这种配置下，2/3 的客户端请求需要重定向，这会造成大概率的访问延迟，但是终究访问还是会成功的。要优化的话，总共有两种方式：

* 直接连到 master queue 所在的节点，这样就不需要重定向了。但是对这种方式，需要提前计算，然后告诉客户端哪个节点上有 master queue。
* 尽可能地在所有节点间平均分布队列，减少重定向概率

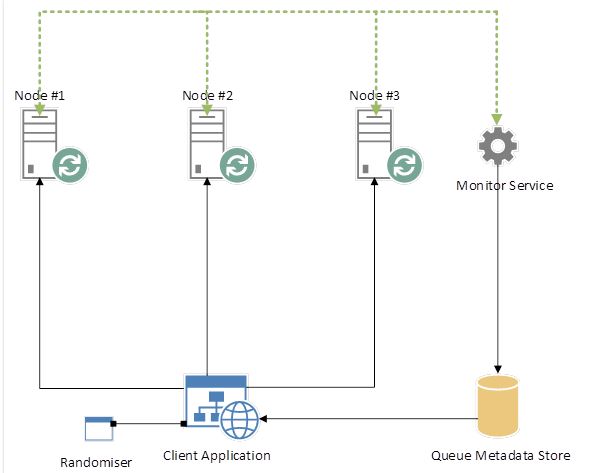
**2.3 镜像队列的负载均衡**

     使用镜像队列的 RabbitMQ 不支持负载均衡，这是由其镜像队列的实现机制决定的。如前面所述，假设一个集群里有两个实例，记作 rabbitA 和 rabbitB。如果某个队列在rabbitA 上创建，随后在 rabbitB 上镜像备份，那么 rabbitA 上的队列称为该队列的主队列（master queue），其它备份均为从队列。接下来，无论client 访问rabbitA 或 rabbitB，最终消费的队列都是主队列。换句话说，即使在连接时主动连接rabbitB，RabbitMQ的 cluster 会自动把连接转向 rabbitA。当且仅当rabbitA服务down掉以后，在剩余的从队列中再选举一个作为继任的主队列。

    如果这种机制是真的（需要看代码最最终确认），那么负载均衡就不能简单地随机化连接就能做到了。要实现轮询，需要满足下面的条件：

* 队列本身的建立需要随机化，即将队列分布于各个服务器
* client 访问需要知道每个队列的主队列保存在哪个服务器
* 如果某个服务器down了，需要知道哪个从队列被选择成为继任的主队列。

   要实现这种方案，[这篇文章](http://insidethecpu.com/2014/11/17/load-balancing-a-rabbitmq-cluster/) 给出了一个方案：



首先，在建立一个新队列的时候，Randomiser 会随机选择一个服务器，这样能够保证队列均匀分散在各个服务器（这里暂且不考虑负载）。建立队列后需要在Meta data 里记录这个队列对应的服务器；另外，Monitor Service是关键，它用于处理某个服务器down掉的情况。一旦发生down机，它需要为之前主队列在该服务器的队列重新建立起与服务器的映射关系。

这里会遇到一个问题，即怎么判断某个队列的主队列呢？一个方法是通过rabbitmqctl，如下面的例子：

|  |  |
| --- | --- |
| 1  2  3 | ./rabbitmqctl -p production list\_queues pid slave\_pids  registration-email-queue        <rabbit@mq01.2.1076.0>       [<rabbit@mq00.1.285.0>]  registration-sms-queue  <rabbit@mq01.2.1067.0>       [<rabbit@mq00.1.281.0>] |

 可以看到pid和slave\_pids分别对应主队列所在的服务器和从服务器（可能有多个）。利用这个命令就可以了解每个队列所在的主服务器了。

**3. OpenStack 中RabbitMQ集群的用法和配置**

**3.1 配置**

    OpenStack 官方建议至少使用三节点 RabbitMQ 集群，而且推荐配置是使用镜像队列。对于测试和演示环境，使用两节点也是可以。以下OpenStack 服务都支持这种 A/A 形式的 RabbitMQ：

* 计算服务
* 块设备存储服务
* 网络服务
* Telemetry

    OpenStack 支持如下的 RabbitMQ 配置：

* rabbit\_hosts=rabbit1:5672,rabbit2:5672,rabbit3:5672：所有RabbitMQ 服务列表
* rabbit\_retry\_interval=1: 连接失败时候的重试间隔
* rabbit\_retry\_backoff=2： How long to back-off for between retries when connecting to RabbitMQ。不太明白其含义。
* rabbit\_max\_retries=0：最大重试次数。0 表示一直重试
* rabbit\_durable\_queues=true：true 的话表示使用持久性队列，Kilo 中默认为 false。
* rabbit\_ha\_queues=true: 设置为 true 的话则使用镜像队列，并设置 x-ha-policy 为 all；但是 Kilo 中其默认值为 false。

具体的配置可以参考 [这篇文章](http://www.lnmpy.com/rabbitmq-ha-lb/) 以及 OpenStack 官网，以及 [RabbitMQ 官网](https://www.rabbitmq.com/clustering.html)。

**3.2 RabbitMQ 节点选择**

OpenStack 的 oslo\_messaging 的 RabbitMQ driver 的实现代码在 [这里](https://github.com/openstack/oslo.messaging/blob/master/oslo_messaging/_drivers/impl_rabbit.py)。它采用的逻辑正是上面所提到的第二种优化方法“尽可能地在所有节点间平均分布队列，减少重定向概率”。这表现在：

（1）将 rabbit\_hosts 打乱，然后构造新的包含多host的 url：

[复制代码](javascript:void(0);)

if len(self.rabbit\_hosts) > 1:

random.shuffle(self.rabbit\_hosts) #将多个 hosts 打乱

for adr in self.rabbit\_hosts:

hostname, port = netutils.parse\_host\_port(

adr, default\_port=self.rabbit\_port)

self.\_url += '%samqp://%s:%s@%s:%s/%s' % ( #将多个url合并，以分号分割，kombu 会按照指定策略在连接失败时重新选择别的host重连

";" if self.\_url else '',

parse.quote(self.rabbit\_userid, ''),

parse.quote(self.rabbit\_password, ''),

self.\_parse\_url\_hostname(hostname), port,

virtual\_host)

[复制代码](javascript:void(0);)

（2）在建立 Connection 的时候，设置 failover\_strategy 为 “shuffle”，这样，在连接不成功需要重连的时候，kombu 会 “随机”地选择一个新的 host:port 重新连接（kombu 库支持 轮询和随机两种策略，[代码在这里](https://github.com/celery/kombu/blob/master/kombu/connection.py)）

[复制代码](javascript:void(0);)

self.connection = kombu.connection.Connection(

self.\_url, ssl=self.\_fetch\_ssl\_params(),

login\_method=self.login\_method,

failover\_strategy="shuffle",

heartbeat=self.heartbeat\_timeout\_threshold,

transport\_options={

'confirm\_publish': True,

'on\_blocked': self.\_on\_connection\_blocked,

'on\_unblocked': self.\_on\_connection\_unblocked,

},

)

[复制代码](javascript:void(0);)

**3.3 性能配置**

OpenStack 支持如下几种 RPC 性能配置：

* rpc\_conn\_pool\_size = 30 (IntOpt) （RPC 连接池的大小）
* rpc\_response\_timeout = 60 (IntOpt) （等待返回的超时时间，单位是秒）
* rpc\_thread\_pool\_size = 64 (IntOpt) （RPC 线程池的大小）

 这些参数在做RPC 性能调试的时候往往需要考虑到。

参考链接：

* <http://leejia.blog.51cto.com/4356849/841084>
* <http://drbd.linbit.com/>
* <http://kafecho.github.io/presentations/introduction-to-pacemaker/#/8>
* <http://chuansong.me/n/412792>
* <http://www.gpfeng.com/?p=603>
* <http://fengchj.com/?p=2273>
* <http://onlychoice.github.io/blog/2013/11/12/rabbitmq-ha/>
* <http://insidethecpu.com/2014/11/17/load-balancing-a-rabbitmq-cluster/>
* <http://my.oschina.net/hncscwc/blog/186350>
* <http://www.rabbitmq.com/ha.html>