

RabbitMQ 消息服务用户手册

(UBP, Message Queue)

XXX

2016 年 7 月

1 基础知识

1.1 集群总体概述

Rabbitmq Broker 集群是多个 erlang 节点的逻辑组，每个节点运行 Rabbitmq 应用，他们之间共享用户、虚拟主机、队列、exchange、绑定和运行时参数。

1.2 集群复制信息

除了 message queue（存在一个节点，从其他节点都可见、访问该队列，要实现 queue 的复制就需要做 queue 的 HA）之外，任何一个 Rabbitmq broker 上的所有操作的 data 和 state 都会在所有的节点之间进行复制。

1.3 集群运行前提

- 1、集群所有节点必须运行相同的 erlang 及 Rabbitmq 版本。
- 2、hostname 解析，节点之间通过域名相互通信，本文为 3 个 node 的集群，采用配置 hosts 的形式。

1.4 集群互通方式

1、集群所有节点必须运行相同的 erlang 及 Rabbitmq 版本 hostname 解析，节点之间通过域名相互通信，本文为 3 个 node 的集群，采用配置 hosts 的形式。

1.5 端口及其用途

- 1、5672 客户端连接端口。
- 2、15672 web 管控台端口。
- 3、25672 集群通信端口。

1.6 集群配置方式

通过 rabbitmqctl 手工配置的方式。

1.7 集群故障处理

- 1、rabbitmq broker 集群允许个体节点宕机。
- 2、对应集群的网络分区问题（network partitions）集群推荐用于 LAN 环境，不适用 WAN 环境；要通过 WAN 连接 broker，Shovel or Federation 插件是最佳解决方案（**Shovel or Federation 不同于集群：注 Shovel 为中心服务远程异步复制机制，稍后会有介绍**）。

1.8 节点运行模式

为保证数据持久性，目前所有 node 节点跑在 disk 模式，如果今后压力大，需要提高性能，考虑采用 ram 模式。

1.9 集群认证方式

通过 Erlang Cookie，相当于共享秘钥的概念，长度任意，只要所有节点都一致即可。rabbitmq server 在启动的时候，erlang VM 会自动创建一个随机的 cookie 文件。cookie 文件的位置：
/var/lib/rabbitmq/.erlang.cookie 或者 /root/.erlang.cookie。我们的为保证 cookie 的完全一致，采用从一个节点 copy 的方式，实现各个节点的 cookie 文件一致。

2 集群搭建

2.1 集群节点安装

1、安装依赖包

PS:安装 rabbitmq 所需要的依赖包

```
yum install build-essential openssl openssl-devel unixODBC unixODBC-devel make gcc gcc-c++  
kernel-devel m4 ncurses-devel tk tc xz
```

2、下载安装包

```
wget www.rabbitmq.com/releases/erlang/erlang-18.3-1.el7.centos.x86\_64.rpm  
wget http://repo.iotti.biz/CentOS/7/x86\_64/socat-1.7.3.2-5.el7.linux.x86\_64.rpm  
wget www.rabbitmq.com/releases/rabbitmq-server/v3.6.5/rabbitmq-server-3.6.5-1.noarch.rpm
```

3、安装服务命令

```
rpm -ivh erlang-18.3-1.el7.centos.x86_64.rpm  
rpm -ivh socat-1.7.3.2-5.el7.linux.x86_64.rpm  
rpm -ivh rabbitmq-server-3.6.5-1.noarch.rpm
```

4、修改集群用户与连接心跳检测

注意修改 vim /usr/lib/rabbitmq/lib/rabbitmq_server-3.6.5/ebin/rabbit.app 文件
修改: loopback_users 中的 <<"guest">>, 只保留 guest
修改: heartbeat 为 1

5、安装管理插件

```
//首先启动服务  
/etc/init.d/rabbitmq-server start stop status restart  
//查看服务有没有启动: lsof -i:5672  
rabbitmq-plugins enable rabbitmq_management  
//可查看管理端口有没有启动: lsof -i:15672 或者 netstat -tnlp|grep 15672
```

6、服务指令

```
/etc/init.d/rabbitmq-server start stop status restart  
验证单个节点是否安装成功: http://192.168.11.71:15672/  
Ps: 以上操作三个节点 (71、72、73) 同时进行操作
```

2.2 文件同步步骤

PS:选择 76、77、78 任意一个节点为 Master (这里选择 76 为 Master), 也就是说我们需要把 76 的 Cookie 文件同步到 77、78 节点上去, 进入 /var/lib/rabbitmq 目录下, 把 /var/lib/rabbitmq/.erlang.cookie 文件的权限修改为 777, 原来是 400; 然后把 .erlang.cookie 文件 copy 到各个节点下; 最后把所有 cookie 文件权限还原为 400 即可。

```
/etc/init.d/rabbitmq-server stop
```

//进入目录修改权限；远程 copy77、78 节点，比如：

scp /var/lib/rabbitmq/.erlang.cookie 到 192.168.11.77 和 192.168.11.78 中

2.3 组成集群步骤

1、停止 MQ 服务

PS:我们首先停止 3 个节点的服务

```
rabbitmqctl stop
```

2、组成集群操作

PS:接下来我们就可以使用集群命令，配置 76、77、78 为集群模式，3 个节点（76、77、78）执行启动命令，后续启动集群使用此命令即可。

```
rabbitmq-server -detached
```

3、slave 加入集群操作（重新加入集群也是如此，以最开始的主节点为加入节点）

//注意做这个步骤的时候：需要配置/etc/hosts 必须相互能够寻址到

```
bhz77: rabbitmqctl stop_app
```

```
bhz77: rabbitmqctl join_cluster --ram rabbit@bhz76
```

```
bhz77: rabbitmqctl start_app
```

```
bhz78: rabbitmqctl stop_app
```

```
bhz78: rabbitmqctl join_cluster rabbit@bhz76
```

```
bhz78: rabbitmqctl start_app
```

//在另外其他节点上操作要移除的集群节点

```
rabbitmqctl forget_cluster_node rabbit@bhz24
```

4、修改集群名称

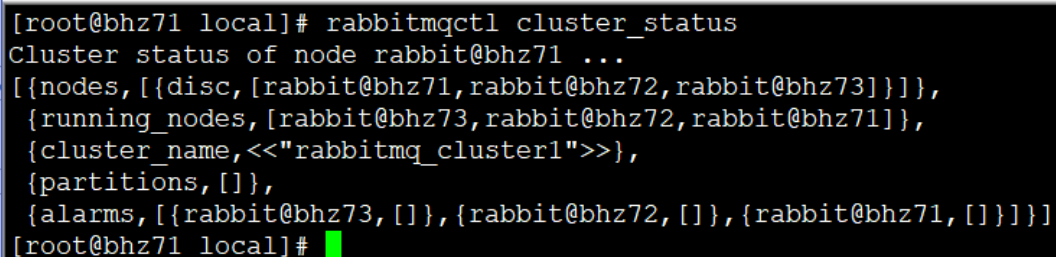
PS:修改集群名称（默认为第一个 node 名称）：

```
rabbitmqctl set_cluster_name rabbitmq_cluster1
```

5、查看集群状态

PS:最后在集群的任意一个节点执行命令：查看集群状态

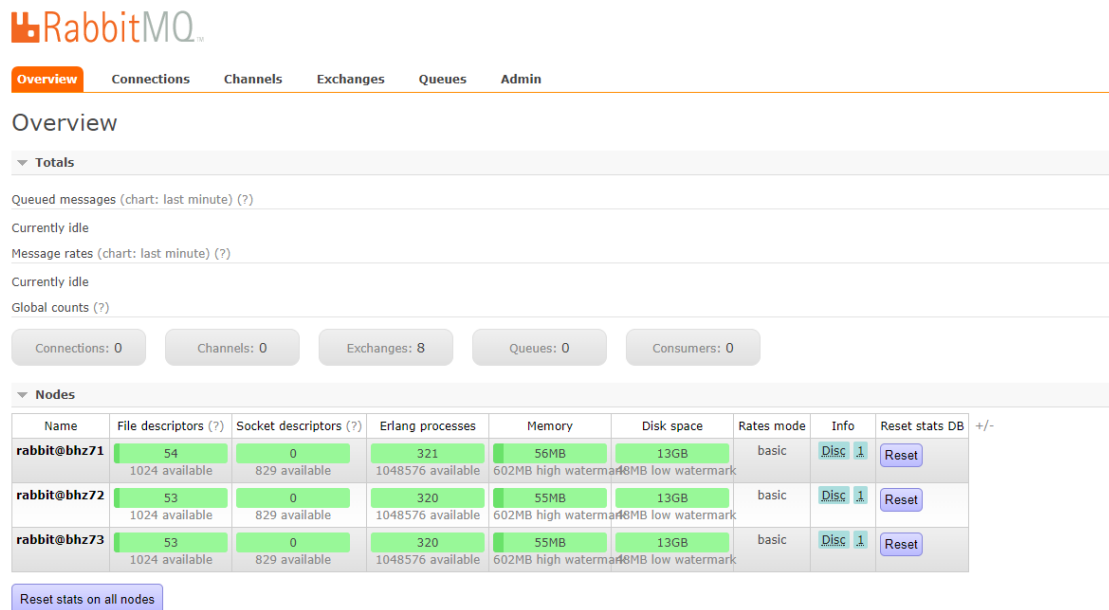
```
rabbitmqctl cluster_status
```



```
[root@bhz71 local]# rabbitmqctl cluster_status
Cluster status of node rabbit@bhz71 ...
[{nodes,[{disc,[rabbit@bhz71,rabbit@bhz72,rabbit@bhz73]}]},
{running_nodes,[rabbit@bhz73,rabbit@bhz72,rabbit@bhz71]},
{cluster_name,<<"rabbitmq_cluster1">>},
{partitions,[]},
{alarms,[{rabbit@bhz73,[]},{rabbit@bhz72,[]},{rabbit@bhz71,[]}}]}
[root@bhz71 local]#
```

6、管控台界面

PS: 访问任意一个管控台节点：<http://192.168.11.71:15672> 如图所示



2.4 配置镜像队列

PS: 设置镜像队列策略（在任意一个节点上执行）

```
rabbitmqctl set_policy ha-all "" '{"ha-mode":"all"}'
```

PS: 将所有队列设置为镜像队列, 即队列会被复制到各个节点, 各个节点状态一致, RabbitMQ 高可用集群就已经搭建好了, 我们可以重启服务, 查看其队列是否在从节点同步。

2.5 安装 Ha-Proxy

1、Haproxy 简介

HAProxy 是一款提供高可用性、负载均衡以及基于 TCP 和 HTTP 应用的代理软件, HAProxy 是完全免费的、借助 HAProxy 可以快速并且可靠的提供基于 TCP 和 HTTP 应用的代理解决方案。

HAProxy 适用于那些负载较大的 web 站点, 这些站点通常又需要会话保持或七层处理。

HAProxy 可以支持数以万计的并发连接, 并且 HAProxy 的运行模式使得它可以很简单安全的整合进架构中, 同时可以保护 web 服务器不被暴露到网络上。

2、Haproxy 安装

PS: 79、80 节点同时安装 Haproxy, 下面步骤统一

```
//下载依赖包
yum install gcc vim wget

//下载 haproxy
wget http://www.haproxy.org/download/1.6/src/haproxy-1.6.5.tar.gz

//解压
tar -zxvf haproxy-1.6.5.tar.gz -C /usr/local

//进入目录、进行编译、安装
cd /usr/local/haproxy-1.6.5
make TARGET=linux31 PREFIX=/usr/local/haproxy
make install PREFIX=/usr/local/haproxy
mkdir /etc/haproxy

//赋权
```

```
groupadd -r -g 149 haproxy
useradd -g haproxy -r -s /sbin/nologin -u 149 haproxy
//创建 haproxy 配置文件
touch /etc/haproxy/haproxy.cfg
```

3、Haproxy 配置

PS:haproxy 配置文件 haproxy.cfg 详解

```
vim /etc/haproxy/haproxy.cfg
```

```
#logging options
global
    log 127.0.0.1 local0 info
    maxconn 5120
    chroot /usr/local/haproxy
    uid 99
    gid 99
    daemon
    quiet
    nbproc 20
    pidfile /var/run/haproxy.pid

defaults
    log global
    #使用 4 层代理模式, " mode http" 为 7 层代理模式
    mode tcp
    #if you set mode to tcp, then you must change tcplog into httplog
    option tcplog
    option dontlognull
    retries 3
    option redispatch
    maxconn 2000
    timeout 5s
    ##客户端空闲超时时间为 60 秒 则 HA 发起重连机制
    clitimeout 60s
    ##服务器端链接超时时间为 15 秒 则 HA 发起重连机制
    srvtimeout 15s
    #front-end IP for consumers and producters

listen rabbitmq_cluster
    bind 0.0.0.0:5672
    #配置 TCP 模式
    mode tcp
```

```
#balance url_param userid
#balance url_param session_id check_post 64
#balance hdr(User-Agent)
#balance hdr(host)
#balance hdr(Host) use_domain_only
#balance rdp-cookie
#balance leastconn
#balance source //ip
#简单的轮询
balance roundrobin

#rabbitmq 集群节点配置 #inter 每隔五秒对 mq 集群做健康检查， 2 次正确证明服务器可用， 2 次
失败证明服务器不可用， 并且配置主备机制
    server bhz76 192.168.11.76:5672 check inter 5000 rise 2 fall 2
    server bhz77 192.168.11.77:5672 check inter 5000 rise 2 fall 2
    server bhz78 192.168.11.78:5672 check inter 5000 rise 2 fall 2

#配置 haproxy web 监控， 查看统计信息
listen stats
    bind 192.168.11.79:8100
    mode http
    option httplog
    stats enable

#设置 haproxy 监控地址为 http://localhost:8100/rabbitmq-stats
stats uri /rabbitmq-stats
stats refresh 5s
```

4、启动 haproxy

```
/usr/local/haproxy/sbin/haproxy -f /etc/haproxy/haproxy.cfg
//查看 haproxy 进程状态
ps -ef | grep haproxy
```

5、访问 haproxy

PS: 访问如下地址可以对 rmq 节点进行监控: <http://192.168.1.27:8100/rabbitmq-stats>

192.168.1.27:8100/rabbitmq-stats

HAProxy version 1.6.5, released 2016/05/10

Statistics Report for pid 1611

> General process information

pid = 1611 (process #20, nproc = 20)
 uptime = 0d 0h01m26s
 system limits: memmax = unlimited; ulimit-n = 10255
 maxsock = 10255; maxconn = 5120; maxpipes = 0
 current conns = 1; current pipes = 0/0; conn rate = 1/sec
 Running tasks: 1/7, idle = 100 %

active UP
 active UP, going down
 active DOWN, going up
 active or backup DOWN
 active or backup DOWN for maintenance (MAINT)
 active or backup SOFT STOPPED for maintenance
 Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

backup UP
 backup UP, going down
 backup DOWN, going up
 not checked

rabbitmq_cluster																				
	Queue			Session rate			Sessions					Bytes		Denied		Errors				
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	
Frontend	0	0	-	0	0	-	0	0	2 000	0	0	?	0	0	0	0	0	0	0	0
bhz24	0	0	-	0	0	-	0	0	-	0	0	?	0	0	0	0	0	0	0	
bhz25	0	0	-	0	0	-	0	0	-	0	0	?	0	0	0	0	0	0	0	
bhz26	0	0	-	0	0	-	0	0	-	0	0	?	0	0	0	0	0	0	0	
Backend	0	0	-	0	0	-	0	0	200	0	0	?	0	0	0	0	0	0	0	

stats																			
	Queue			Session rate			Sessions					Bytes		Denied		Errors			
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Res
Frontend	1	1	-	1	1	-	1	1	2 000	1	0	0s	0	0	0	0	0	0	0
Backend	0	0	-	0	0	-	0	0	200	0	0	0s	0	0	0	0	0	0	0

6、关闭 haproxy

```
killall haproxy
ps -ef | grep haproxy
```

2.6 安装 KeepAlived

1、Keepalived 简介

Keepalived，它是一个高性能的服务器高可用或热备解决方案，Keepalived 主要来防止服务器单点故障的发生问题，可以通过其与 Nginx、Haproxy 等反向代理的负载均衡服务器配合实现 web 服务端的高可用。Keepalived 以 VRRP 协议为实现基础，用 VRRP 协议来实现高可用性（HA）。VRRP（Virtual Router Redundancy Protocol）协议是用于实现路由器冗余的协议，VRRP 协议将两台或多台路由器设备虚拟成一个设备，对外提供虚拟路由器 IP（一个或多个）。

2、Keepalived 安装

PS: 下载地址: <http://www.keepalived.org/download.html>

```
//安装所需软件包
yum install -y openssl openssl-devel

//下载
wget http://www.keepalived.org/software/keepalived-1.2.18.tar.gz

//解压、编译、安装
tar -zxvf keepalived-1.2.18.tar.gz -C /usr/local/
cd keepalived-1.2.18/ && ./configure --prefix=/usr/local/keepalived
make && make install

//将 keepalived 安装成 Linux 系统服务，因为没有使用 keepalived 的默认安装路径（默认路径：/usr/local），
安装完成之后，需要做一些修改工作

//首先创建文件夹，将 keepalived 配置文件进行复制：
mkdir /etc/keepalived
cp /usr/local/keepalived/etc/keepalived/keepalived.conf /etc/keepalived/

//然后复制 keepalived 脚本文件：
```



```
cp /usr/local/keepalived/etc/rc.d/init.d/keepalived /etc/init.d/  
cp /usr/local/keepalived/etc/sysconfig/keepalived /etc/sysconfig/  
ln -s /usr/local/sbin/keepalived /usr/sbin/  
ln -s /usr/local/keepalived/sbin/keepalived /sbin/  
//可以设置开机启动: chkconfig keepalived on, 到此我们安装完毕!  
chkconfig keepalived on
```

3、Keepalived 配置

PS: 修改 keepalived.conf 配置文件

```
vim /etc/keepalived/keepalived.conf
```

PS: 79 节点 (Master) 配置如下

```
! Configuration File for keepalived  
  
global_defs {  
    router_id bhz79  ##标识节点的字符串, 通常为 hostname  
}  
  
vrrp_script chk_haproxy {  
    script "/etc/keepalived/haproxy_check.sh"  ##执行脚本位置  
    interval 2  ##检测时间间隔  
    weight -20  ##如果条件成立则权重减 20  
}  
  
vrrp_instance VI_1 {  
    state MASTER  ## 主节点为 MASTER, 备份节点为 BACKUP  
    interface eth0  ## 绑定虚拟 IP 的网络接口 (网卡), 与本机 IP 地址所在的网络接口相同 (我这里是 eth0)  
    virtual_router_id 79  ## 虚拟路由 ID 号 (主备节点一定要相同)  
    mcast_src_ip 192.168.11.79  ## 本机 ip 地址  
    priority 100  ##优先级配置 (0-254 的值)  
    nopreempt  
    advert_int 1  ## 组播信息发送间隔, 两个节点必须配置一致, 默认 1s
```

```

authentication { ## 认证匹配
    auth_type PASS
    auth_pass bhz
}

track_script {
    chk_haproxy
}

virtual_ipaddress {
    192.168.11.70 ## 虚拟 ip, 可以指定多个
}
}

```

PS: 80 节点 (backup) 配置如下

```

! Configuration File for keepalived

global_defs {
    router_id bhz80 ##标识节点的字符串, 通常为 hostname
}

vrrp_script chk_haproxy {
    script "/etc/keepalived/haproxy_check.sh" ##执行脚本位置
    interval 2 ##检测时间间隔
    weight -20 ##如果条件成立则权重减 20
}

vrrp_instance VI_1 {
    state BACKUP ## 主节点为 MASTER, 备份节点为 BACKUP
    interface eno16777736 ## 绑定虚拟 IP 的网络接口 (网卡), 与本机 IP 地址所在的网络接口相同 (我
    这里是 eno16777736)
    virtual_router_id 79 ## 虚拟路由 ID 号 (主备节点一定要相同)
    mcast_src_ip 192.168.11.80 ## 本机 ip 地址
    priority 90 ##优先级配置 (0-254 的值)
    nopreempt
}

```

```

advert_int 1  ## 组播信息发送间隔，两个节点必须配置一致，默认 1s
authentication {  ## 认证匹配
    auth_type PASS
    auth_pass bhz
}

track_script {
    chk_haproxy
}

virtual_ipaddress {
    192.168.1.70  ## 虚拟 ip，可以指定多个
}
}

```

4、执行脚本编写

PS:添加文件位置为/etc/keepalived/haproxy_check.sh（79、80 两个节点文件内容一致即可）

```

#!/bin/bash
COUNT=`ps -C haproxy --no-header |wc -l`
if [ $COUNT -eq 0 ];then
    /usr/local/haproxy/sbin/haproxy -f /etc/haproxy/haproxy.cfg
    sleep 2
    if [ `ps -C haproxy --no-header |wc -l` -eq 0 ];then
        killall keepalived
    fi
fi

```

5、执行脚本赋权

PS:haproxy_check.sh 脚本授权, 赋予可执行权限.

```

chmod +x /etc/keepalived/haproxy_check.sh

```

6、启动 keepalived

PS:当我们启动俩个 haproxy 节点以后，我们可以启动 keepalived 服务程序：

```

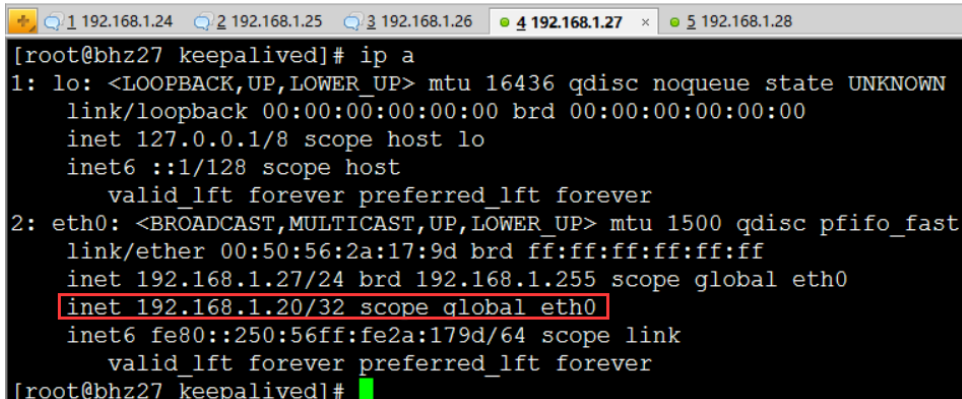
//启动两台机器的 keepalived
service keepalived start | stop | status | restart
//查看状态
ps -ef | grep haproxy

```

```
ps -ef | grep keepalived
```

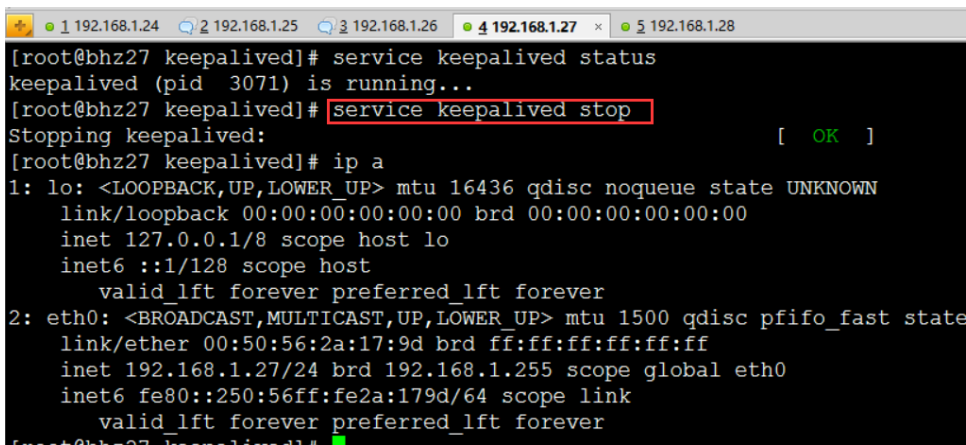
7、高可用测试

PS:vip 在 27 节点上



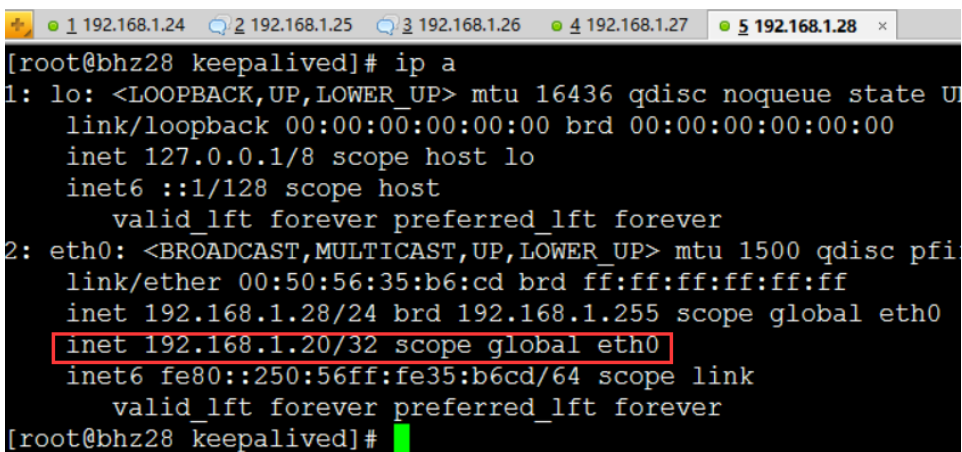
```
[root@bhaz27 keepalived]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
    link/ether 00:50:56:2a:17:9d brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.27/24 brd 192.168.1.255 scope global eth0
    inet 192.168.1.20/32 scope global eth0
    inet6 fe80::250:56ff:fe2a:179d/64 scope link
        valid_lft forever preferred_lft forever
[root@bhaz27 keepalived]#
```

PS:27 节点宕机测试：停掉 27 的 keepalived 服务即可。



```
[root@bhaz27 keepalived]# service keepalived status
keepalived (pid 3071) is running...
[root@bhaz27 keepalived]# service keepalived stop
Stopping keepalived: [ OK ]
[root@bhaz27 keepalived]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
    link/ether 00:50:56:2a:17:9d brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.27/24 brd 192.168.1.255 scope global eth0
    inet6 fe80::250:56ff:fe2a:179d/64 scope link
        valid_lft forever preferred_lft forever
[root@bhaz27 keepalived]#
```

PS:查看 28 节点状态：我们发现 VIP 漂移到了 28 节点上，那么 28 节点的 haproxy 可以继续对外提供服务！



```
[root@bhaz28 keepalived]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UP
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfi
    link/ether 00:50:56:35:b6:cd brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.28/24 brd 192.168.1.255 scope global eth0
    inet 192.168.1.20/32 scope global eth0
    inet6 fe80::250:56ff:fe35:b6cd/64 scope link
        valid_lft forever preferred_lft forever
[root@bhaz28 keepalived]#
```

2.7 集群配置文件

创建如下配置文件位于：/etc/rabbitmq 目录下（这个目录需要自己创建）

环境变量配置文件：rabbitmq-env.conf

配置信息配置文件: `rabbitmq.config` (可以不创建和配置, 修改)

`rabbitmq-env.conf` 配置文件:

```
-----关键参数配置-----
RABBITMQ_NODE_IP_ADDRESS=本机 IP 地址
RABBITMQ_NODE_PORT=5672
RABBITMQ_LOG_BASE=/var/lib/rabbitmq/log
RABBITMQ_MNESIA_BASE=/var/lib/rabbitmq/mnesia
```

配置参考参数如下:

`RABBITMQ_NODENAME=FZTEC-240088` 节点名称

`RABBITMQ_NODE_IP_ADDRESS=127.0.0.1` 监听 IP

`RABBITMQ_NODE_PORT=5672` 监听端口

`RABBITMQ_LOG_BASE=/data/rabbitmq/log` 日志目录

`RABBITMQ_PLUGINS_DIR=/data/rabbitmq/plugins` 插件目录

`RABBITMQ_MNESIA_BASE=/data/rabbitmq/mnesia` 后端存储目录

更详细的配置参见: <http://www.rabbitmq.com/configure.html#configuration-file>

配置文件信息修改:

`/usr/lib/rabbitmq/lib/rabbitmq_server-3.6.4/ebin/rabbit.app` 和 `rabbitmq.config` 配置文件配置任意一个即可, 我们进行配置如下:

`vim /usr/lib/rabbitmq/lib/rabbitmq_server-3.6.4/ebin/rabbit.app`

```
-----关键参数配置-----
tcp_listeners 设置 rabbitmq 的监听端口, 默认为[5672]。
disk_free_limit 磁盘低水位线, 若磁盘容量低于指定值则停止接收数据, 默认值为{mem_relative, 1.0},
即与内存相关联 1: 1, 也可定制为多少 byte。
vm_memory_high_watermark, 设置内存低水位线, 若低于该水位线, 则开启流控机制, 默认值是 0.4, 即内存
总量的 40%。
hipe_compile 将部分 rabbitmq 代码用 High Performance Erlang compiler 编译, 可提升性能, 该参数是实
验性, 若出现 erlang vm segfaults, 应关掉。
force_fine_statistics, 该参数属于 rabbitmq_management, 若为 true 则进行精细化的统计, 但会影响性能
```

更详细的配置参见: <http://www.rabbitmq.com/configure.html>

3 Stream 调研

3.1 Stream 简介

Spring Cloud Stream 是创建消息驱动微服务应用的框架。Spring Cloud Stream 是基于 spring boot 创建，用来建立单独的 / 工业级 spring 应用，使用 spring integration 提供与消息代理之间的连接。本文提供不同代理中的中间件配置，介绍了持久化发布订阅机制，以及消费组以及分割的概念。

将注解@EnableBinding 加到应用上就可以实现与消息代理的连接，@StreamListener 注解加到方法上，使之可以接收处理流的事件。

3.2 官方参考文档

原版：

http://docs.spring.io/spring-cloud-stream/docs/current-SNAPSHOT/reference/htmlsingle/#_main_concepts

翻译：

<http://blog.csdn.net/phyllisy/article/details/51352868>

3.3 API 操作手册

3.3.1 生产者示例

PS:生产者 yml 配置

```
spring:
  cloud:
    stream:
      instanceCount: 3
      bindings:
        output_channel:      #输出 生产者
        group: queue-1      #指定相同的exchange-1和不同的queue 表示广播模式 #指定相同的
                             exchange和相同的queue表示集群负载均衡模式
        destination: exchange-1 # kafka:发布订阅模型里面的topic rabbitmq:
                             exchange的概念（但是exchange的类型那里设置呢？）
      binder: rabbit_cluster
      binders:
        rabbit_cluster:
          type: rabbit
          environment:
            spring:
              rabbitmq:
                host: 192.168.1.27
                port: 5672
                username: guest
                password: guest
                virtual-host: /
```

PS: Barista 接口为自定义管道

```

package bhz.spring.cloud.stream;

import org.springframework.cloud.stream.annotation.Input;
import org.springframework.cloud.stream.annotation.Output;
import org.springframework.messaging.MessageChannel;
import org.springframework.messaging.SubscribableChannel;

/**
 * <B>中文类名: </B><BR>
 * <B>概要说明: </B><BR>
 * 这里的 Barista 接口是定义来作为后面类的参数，这一接口定义来通道类型和通道名称。
 * 通道名称是作为配置用，通道类型则决定了 app 会使用这一通道进行发送消息还是从中接收消息。
 * @author bhz (Alienware)
 * @since 2015 年 11 月 22 日
 */
public interface Barista {

    String INPUT_CHANNEL = "input_channel";
    String OUTPUT_CHANNEL = "output_channel";

    //注解@Input 声明了它是一个输入类型的通道，名字是 Barista.INPUT_CHANNEL，也就是 position3
    //的 input_channel。这一名字与上述配置 app2 的配置文件中 position1 应该一致，表明注入了一个名字叫
    //做 input_channel 的通道，它的类型是 input，订阅的主题是 position2 处声明的 mydest 这个主题
    @Input(Barista.INPUT_CHANNEL)
    SubscribableChannel loginput();

    //注解@Output 声明了它是一个输出类型的通道，名字是 output_channel。这一名字与 app1 中通道名
    //一致，表明注入了一个名字为 output_channel 的通道，类型是 output，发布的主题名为 mydest。
    @Output(Barista.OUTPUT_CHANNEL)
    MessageChannel logoutput();
}

```

PS: 生产者消息投递

```

package bhz.spring.cloud.stream;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.support.MessageBuilder;
import org.springframework.stereotype.Service;

@Service
public class RabbitmqSender {

```

```

@Autowired
private Barista source;

// 发送消息
public String sendMessage(Object message) {
    try {
        source.logout().send(MessageBuilder.withPayload(message).build());
        System.out.println("发送数据: " + message);
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}
}

```

PS: Spring Boot 应用入口

```

package bhz.spring.cloud.stream;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.stream.annotation.EnableBinding;

@SpringBootApplication
@EnableBinding(Barista.class)
public class ProducerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ProducerApplication.class, args);
    }
}

```

3.3.2 消费者示例

PS: 消费者 yml 配置

```

spring:
  cloud:
    stream:
      instanceCount: 3
      bindings:
        input_channel:      #输出 生产者
          destination: exchange-1 # kafka:发布订阅模型里面的topic rabbitmq:
exchange的概念（但是exchange的类型那里设置呢？）
          group: queue-1 #指定相同的exchange-1和不同的queue 表示广播模式 #指定相同的
exchange和相同的queue表示集群负载均衡模式

```



```

    binder: rabbit_cluster
    consumer:
      concurrency: 1
  rabbit:
    bindings:
      input_channel:
        consumer:
          transacted: true
          txSize: 10
          acknowledgeMode: MANUAL
          durableSubscription: true
          maxConcurrency: 20
          recoveryInterval: 3000
  binders:
    rabbit_cluster:
      type: rabbit
      environment:
        spring:
          rabbitmq:
            host: 192.168.1.27
            port: 5672
            username: guest
            password: guest
            virtual-host: /

```

PS: Barista 接口为自定义管道

```

package bhz.spring.cloud.stream;

import org.springframework.cloud.stream.annotation.Input;
import org.springframework.cloud.stream.annotation.Output;
import org.springframework.messaging.MessageChannel;
import org.springframework.messaging.SubscribableChannel;

/**
 * <B>中文类名: </B><BR>
 * <B>概要说明: </B><BR>
 * 这里的 Barista 接口是定义来作为后面类的参数, 这一接口定义来通道类型和通道名称。
 * 通道名称是作为配置用, 通道类型则决定了 app 会使用这一通道进行发送消息还是从中接收消息。
 * @author bhz (Alienware)
 * @since 2015 年 11 月 22 日
 */
public interface Barista {

```

```
String INPUT_CHANNEL = "input_channel";
String OUTPUT_CHANNEL = "output_channel";

//注解@Input 声明了它是一个输入类型的通道，名字是 Barista.INPUT_CHANNEL，也就是 position3
的 input_channel。这一名字与上述配置 app2 的配置文件中 position1 应该一致，表明注入了一个名字叫
做 input_channel 的通道，它的类型是 input，订阅的主题是 position2 处声明的 mydest 这个主题
@Input(Barista.INPUT_CHANNEL)
SubscribableChannel loginput();

//注解@Output 声明了它是一个输出类型的通道，名字是 output_channel。这一名字与 app1 中通道名
一致，表明注入了一个名字为 output_channel 的通道，类型是 output，发布的主题名为 mydest。
@Output(Barista.OUTPUT_CHANNEL)
MessageChannel logoutput();

}
```

PS：消费者消息获取

```
package bhz.spring.cloud.stream;

import java.io.IOException;

import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.amqp.rabbit.support.CorrelationData;
import org.springframework.amqp.support.AmqpHeaders;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.cloud.stream.annotation.EnableBinding;
import org.springframework.cloud.stream.annotation.StreamListener;
import org.springframework.cloud.stream.binding.ChannelBindingService;
import org.springframework.cloud.stream.config.ChannelBindingServiceConfiguration;
import org.springframework.cloud.stream.endpoint.ChannelsEndpoint;
import org.springframework.integration.channel.PublishSubscribeChannel;
import org.springframework.integration.channel.RendezvousChannel;
import org.springframework.messaging.Message;
import org.springframework.messaging.MessageChannel;
import org.springframework.messaging.SubscribableChannel;
import org.springframework.messaging.core.MessageReceivingOperations;
import org.springframework.messaging.core.MessageRequestReplyOperations;
import org.springframework.messaging.support.ChannelInterceptor;
import org.springframework.stereotype.Service;

import com.rabbitmq.client.Channel;
```

```

@EnableBinding(Barista.class)
@Service
public class RabbitmqReceiver {

    @Autowired
    private Barista source;

    @StreamListener(Barista.INPUT_CHANNEL)
    public void receiver( Message message) {

        //广播通道
        //PublishSubscribeChannel psc = new PublishSubscribeChannel();
        //确认通道
        //RendezvousChannel rc = new RendezvousChannel();
        Channel channel = (com.rabbitmq.client.Channel) message.getHeaders().get(AmqpHeaders.CHANNEL);
        Long deliveryTag = (Long) message.getHeaders().get(AmqpHeaders.DELIVERY_TAG);
        System.out.println("Input Stream 1 接受数据: " + message);
        try {
            channel.basicAck(deliveryTag, false);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

PS: Spring Boot 应用入口

```

package bhz.spring.cloud.stream;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.stream.annotation.EnableBinding;
import org.springframework.transaction.annotation.EnableTransactionManagement;

@SpringBootApplication
@EnableBinding(Barista.class)
@EnableTransactionManagement
public class ConsumerApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConsumerApplication.class, args);
    }
}

```

4 制定扩展

4.1 延迟队列插件

#step1: upload the 'rabbitmq_delayed_message_exchange-0.0.1.ez' file:

<https://github.com/rabbitmq/rabbitmq-delayed-message-exchange>

<http://www.rabbitmq.com/community-plugins.html>

https://bintray.com/rabbitmq/community-plugins/rabbitmq_delayed_message_exchange/v3.6.x#files/

#step2: PUT Directory:

/usr/lib/rabbitmq/lib/rabbitmq_server-3.6.4/plugins

#step3: Then run the following command:

Start the rabbitmq cluster for command ## rabbitmq-server -detached

rabbitmq-plugins enable rabbitmq_delayed_message_exchange

```
[root@bh22 plugins]# rabbitmq-plugins enable rabbitmq_delayed_message_exchange
The following plugins have been enabled:
  rabbitmq_delayed_message_exchange

Applying plugin configuration to rabbit@bh22... started 1 plugin.
[root@bh22 plugins]#
```

访问地址: <http://192.168.1.21:15672/#/exchanges>, 添加延迟队列

▼ Add a new exchange

Name: delay-exchange *

Type: x-delayed-message ▼

Durability: Durable ▼

Auto delete: (?) No ▼

Internal: (?) No ▼

Arguments: x-delayed-type = topic String ▼

String ▼

Add Alternate exchange (?)

Add exchange