RabbitMQ消息服务用户手册

**（UBP, Message Queue）**

**XXX**

**2016年7月**

# 基础知识

## 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。

## 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。

## 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。

## 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。

## 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。

## 集群配置方式

通过rabbitmqctl手工配置的方式。

## 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）。

## 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。

## 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。

# 集群搭建

## 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](http://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.lux.x86\_64.rpm  wget www.rabbitmq.com/releases/rabbitmq-server/v3.6.5/[**rabbitmq-server-3.6.5-1.noarch.rpm**](https://github.com/rabbitmq/rabbitmq-server/releases/download/rabbitmq_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.lux.x86\_64.rpm  rpm -ivh [**rabbitmq-server-3.6.5-1.noarch.rpm**](https://github.com/rabbitmq/rabbitmq-server/releases/download/rabbitmq_v3_6_5/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/>   * 1. Ps：以上操作三个节点（71、72、73）同时进行操作 |

## 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 |

## 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 |

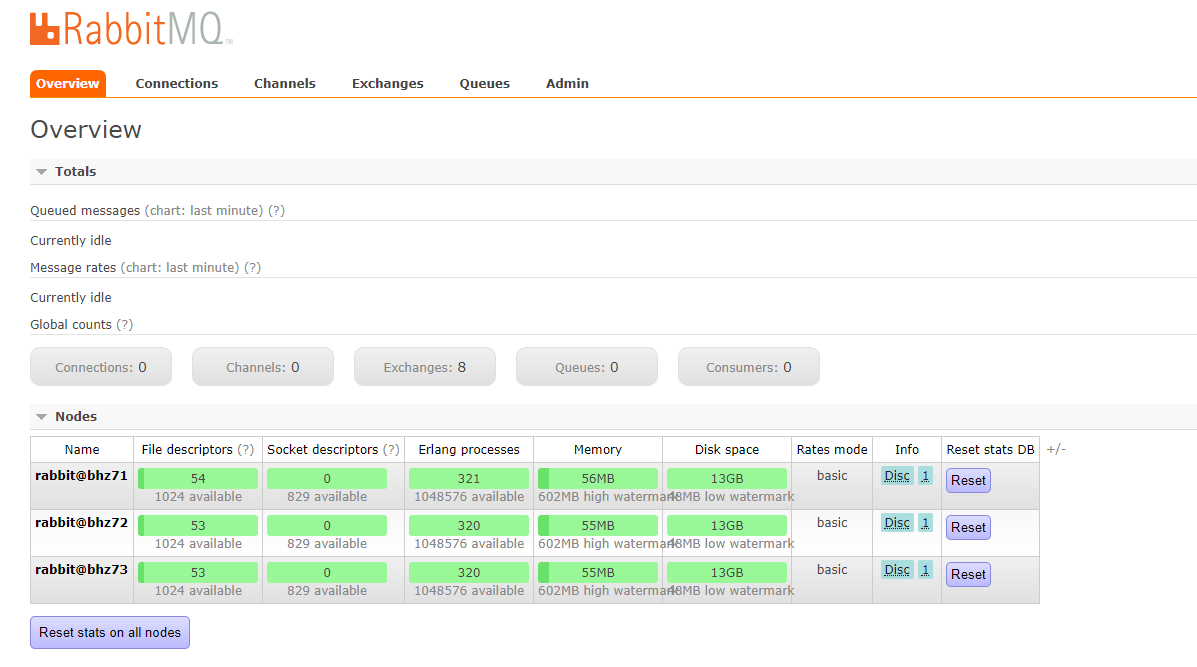
5、查看集群状态

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 |
|  |

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



## 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。

## 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](http://www.linuxea.com/tag/haproxy/)  wget http://www.[haproxy](http://www.linuxea.com/tag/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 |

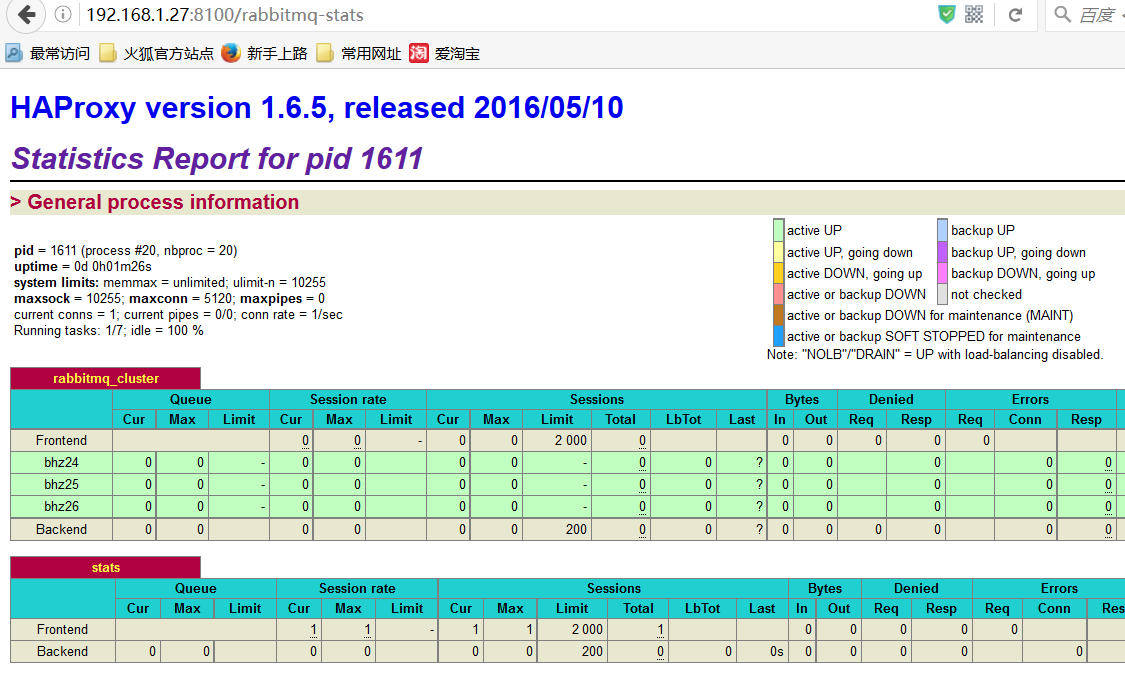
|  |
| --- |
| #logging options  global  log 127.0.0.1 local0 info  maxconn 5120  chroot /usr/local/haproxy  uid 99  gid 99  daemon  quiet  nbproc 20  pidfile /var/run/haproxy.pid  defaults  log global  #使用4层代理模式，”mode http”为7层代理模式  mode tcp  #if you set mode to tcp,then you nust change tcplog into httplog  option tcplog  option dontlognull  retries 3  option redispatch  maxconn 2000  contimeout 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>



6、关闭haproxy

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

## 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，俩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，俩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 |

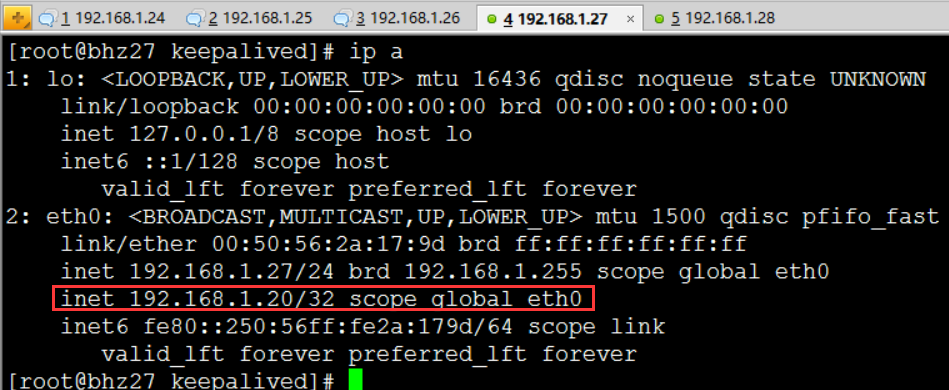
6、启动keepalived

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

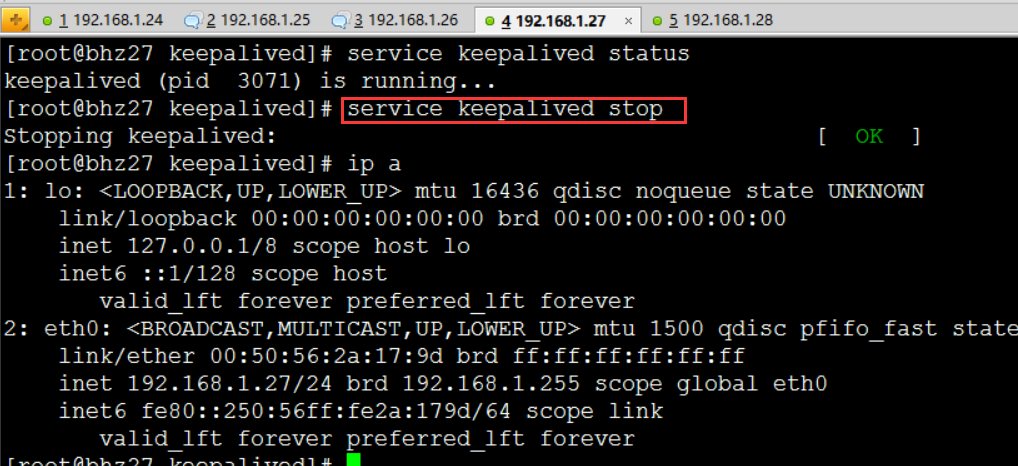
|  |
| --- |
| //启动两台机器的keepalived  service keepalived start | stop | status | restart  //查看状态  ps -ef | grep haproxy  ps -ef | grep keepalived |

7、高可用测试

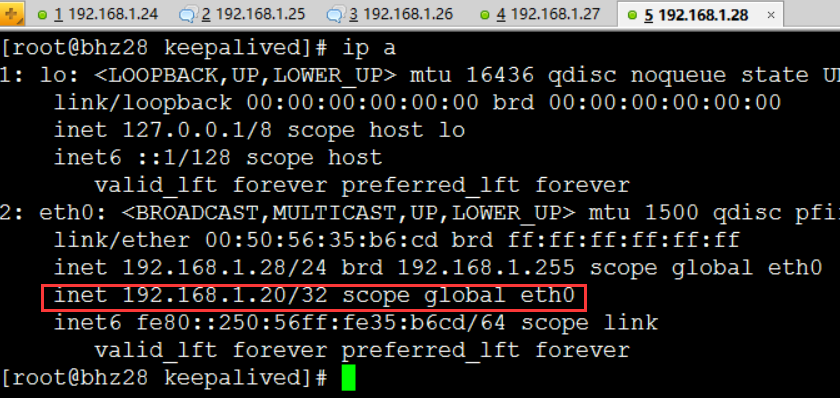
PS:vip在27节点上



PS:27节点宕机测试：停掉27的keepalived服务即可。



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！



## 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\_listerners 设置rabbimq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rabbimq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rabbimq\_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

# Stream调研

## 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。

## 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>

## API操作手册

### 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.logoutput().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);  }  } |

### 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);  }  } |

# 制定扩展

## 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/>

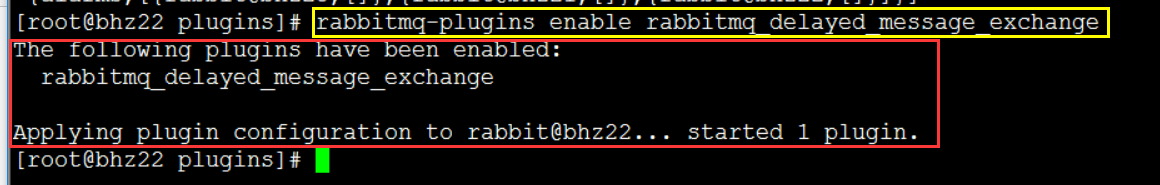
#step2：PUT Directory：

/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



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

