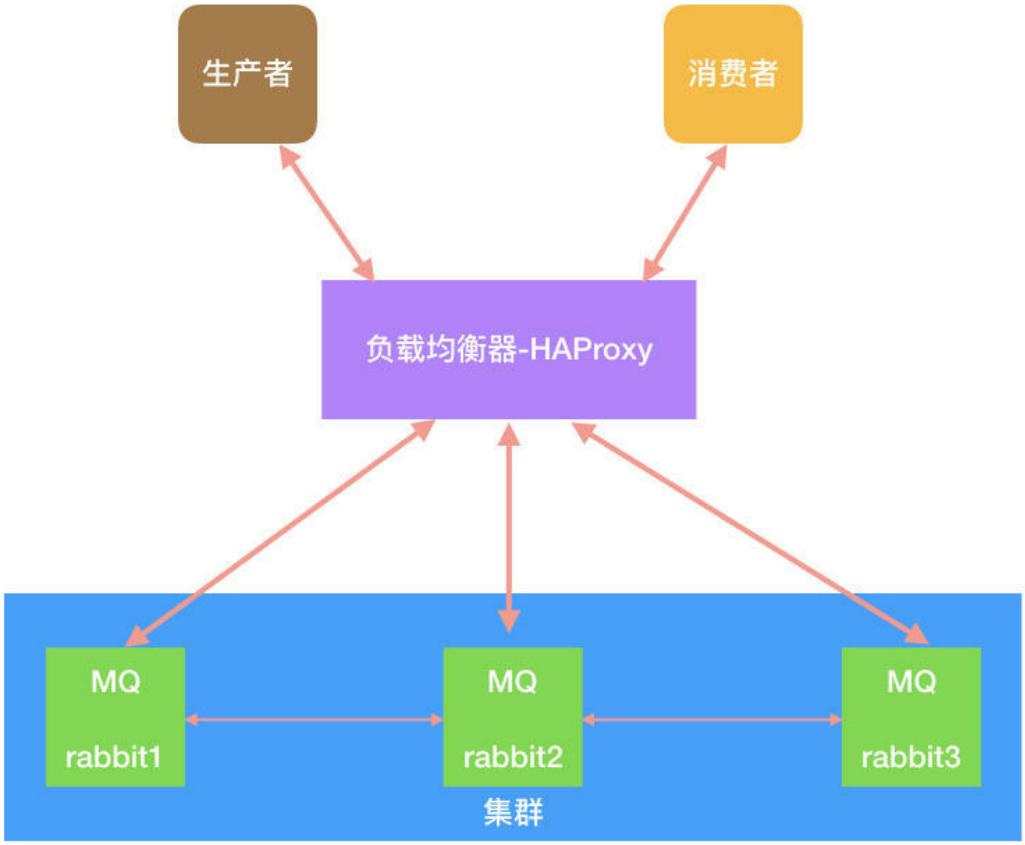
RabbitMQ 高可用集群

整体架构



部署步骤

基于 Docker

1. **基本概念**
   - 内存节点

     只保存状态到内存，例外情况是：持久的 `queue` 的内容将被保存到磁盘。

   - 磁盘节点

> 保存状态到内存和磁盘。

- 内存节点由于不进行磁盘读写，它的性能比磁盘节点高。
- 集群中可以存在多个磁盘节点，磁盘节点越多整个集群可用性越好，但是集群整体性能不会线性增加，需要权衡考虑。
- 如果集群中只有内存节点，那么不能停止它们，否则所有状态和消息都会丢失。

2. **安装 RabbitMQ**

   - 准备四台主机，我是用 Docker 来创建的，命令如下：

```
➜  ~ docker run -i -d --name rabbitmq1 --hostname rabbitmq1 registry.docke
r-cn.com/library/erlang:latest
0823993f94d732b9e501ff62920ed0149cb33bcea1992e73f054b79fad5747c0
➜  ~ docker run -i -d --name rabbitmq2 --hostname rabbitmq2 registry.docke
r-cn.com/library/erlang:latest
961e45d0a3219e43887866eea39b3d68aeb6ea81c78cf5bf6006be5483930e68
➜  ~ docker run -i -d --name rabbitmq3 --hostname rabbitmq3 registry.docke
r-cn.com/library/erlang:latest
ef08f580066489465f5fd6201455a4680954cfa3e0591e36346a6e610c7cd82c
➜  ~ docker run -i -d --name haproxy --hostname haproxy -p 8100:8100 -p 81
01:8101 -p 8102:8102 registry.docker-cn.com/library/ubuntu:14.04
f05a01380908ca639004e11bef0ce3bb38c0a6485d2ffb16f30ad62dedca9647
➜  ~ docker ps
CONTAINER ID        IMAGE                                               COMMAND
             CREATED              STATUS              PORTS
               NAMES
f05a01380908        registry.docker-cn.com/library/ubuntu:14.04         "/bin/b
ash"         About a minute ago   Up About a minute   0.0.0.0:8100-8102->8
100-8102/tcp   haproxy
ef08f5800664        registry.docker-cn.com/library/erlang:latest        "erl"
             2 minutes ago        Up 2 minutes
               rabbitmq3
961e45d0a321        registry.docker-cn.com/library/erlang:latest        "erl"
             2 minutes ago        Up 2 minutes
               rabbitmq2
0823993f94d7        registry.docker-cn.com/library/erlang:latest        "erl"
             3 minutes ago        Up 2 minutes
               rabbitmq1
➜  ~ docker inspect --format='{{.NetworkSettings.IPAddress}}' rabbitmq1
172.17.0.2
➜  ~ docker inspect --format='{{.NetworkSettings.IPAddress}}' rabbitmq2
172.17.0.3
➜  ~ docker inspect --format='{{.NetworkSettings.IPAddress}}' rabbitmq3
172.17.0.4
➜  ~ docker inspect --format='{{.NetworkSettings.IPAddress}}' haproxy
172.17.0.5
```

haproxy 主机开了三个端口，后面讲到 HAProxy 负载均衡时再解释这三个端口的作用。

- 安装 RabbitMQ
  - 选择一个合适的版本下载， 下载地址：Downloading and Installing RabbitMQ
  - 下面是每一步的命令：

```
root@rabbitmq1:~# wget https://dl.bintray.com/rabbitmq/binaries/rabbitm
q-server-generic-unix-3.6.14.tar.xz
root@rabbitmq1:~# xz -d rabbitmq-server-generic-unix-3.6.14.tar.xz
root@rabbitmq1:~# tar -xvf rabbitmq-server-generic-unix-3.6.14.tar
root@rabbitmq1:~# cd rabbitmq_server-3.6.14/
root@rabbitmq1:~/rabbitmq_server-3.6.14# ls -l
total 248
-rw-r--r-- 1 1023 1023    93 Nov  7 07:46 INSTALL
-rw-r--r-- 1 1023 1023 28247 Nov  7 07:46 LICENSE
-rw-r--r-- 1 1023 1023 11425 Nov  7 07:46 LICENSE-APACHE2
-rw-r--r-- 1 1023 1023 11358 Nov  7 07:46 LICENSE-APACHE2-ExplorerCanva
s
-rw-r--r-- 1 1023 1023 11358 Nov  7 07:46 LICENSE-APACHE2-excanvas
-rw-r--r-- 1 1023 1023 10175 Nov  7 07:46 LICENSE-APL2-Rebar
-rw-r--r-- 1 1023 1023 10851 Nov  7 07:46 LICENSE-APL2-Stomp-Websocket
-rw-r--r-- 1 1023 1023  1206 Nov  7 07:46 LICENSE-BSD-base64js
-rw-r--r-- 1 1023 1023  1304 Nov  7 07:46 LICENSE-BSD-glMatrix
-rw-r--r-- 1 1023 1023  1469 Nov  7 07:46 LICENSE-BSD-recon
-rw-r--r-- 1 1023 1023 14041 Nov  7 07:46 LICENSE-EPL-OTP
-rw-r--r-- 1 1023 1023   757 Nov  7 07:46 LICENSE-ISC-cowboy
-rw-r--r-- 1 1023 1023  1084 Nov  7 07:46 LICENSE-MIT-EJS
-rw-r--r-- 1 1023 1023  1087 Nov  7 07:46 LICENSE-MIT-EJS10
-rw-r--r-- 1 1023 1023  1057 Nov  7 07:46 LICENSE-MIT-Erlware-Commons
-rw-r--r-- 1 1023 1023  1069 Nov  7 07:46 LICENSE-MIT-Flot
-rw-r--r-- 1 1023 1023  1087 Nov  7 07:46 LICENSE-MIT-Mochi
-rw-r--r-- 1 1023 1023  1087 Nov  7 07:46 LICENSE-MIT-Mochiweb
-rw-r--r-- 1 1023 1023  1073 Nov  7 07:46 LICENSE-MIT-Sammy
-rw-r--r-- 1 1023 1023  1076 Nov  7 07:46 LICENSE-MIT-Sammy060
-rw-r--r-- 1 1023 1023  1056 Nov  7 07:46 LICENSE-MIT-SockJS
-rw-r--r-- 1 1023 1023  1074 Nov  7 07:46 LICENSE-MIT-jQuery
-rw-r--r-- 1 1023 1023  1075 Nov  7 07:46 LICENSE-MIT-jQuery164
-rw-r--r-- 1 1023 1023 24897 Nov  7 07:46 LICENSE-MPL-RabbitMQ
-rw-r--r-- 1 1023 1023 16726 Nov  7 07:46 LICENSE-MPL2
drwxr-xr-x 2 1023 1023 12288 Nov  7 07:46 ebin
drwxr-xr-x 3 1023 1023  4096 Nov  7 07:46 etc
drwxr-xr-x 2 1023 1023  4096 Nov  7 07:46 include
drwxr-xr-x 2 1023 1023  4096 Nov  7 07:46 plugins
drwxr-xr-x 2 1023 1023  4096 Nov  7 07:46 sbin
drwxr-xr-x 3 1023 1023  4096 Nov  7 07:46 share
```

在 rabbitmq2 和 rabbitmq3 同样执行上面的步骤即可。

- 运行 Rabbitmq

```
root@rabbitmq1:~/rabbitmq_server-3.6.14# cd sbin/
root@rabbitmq1:~/rabbitmq_server-3.6.14/sbin# ls -l
total 36
-rwxr-xr-x 1 1023 1023  1885 Nov  7 07:46 rabbitmq-defaults
-rwxr-xr-x 1 1023 1023 12095 Nov  7 07:46 rabbitmq-env
-rwxr-xr-x 1 1023 1023  1362 Nov  7 07:46 rabbitmq-plugins
-rwxr-xr-x 1 1023 1023 10971 Nov  7 07:46 rabbitmq-server
-rwxr-xr-x 1 1023 1023  1480 Nov  7 07:46 rabbitmqctl
root@rabbitmq1:~/rabbitmq_server-3.6.14/sbin# ./rabbitmq-server

              RabbitMQ 3.6.14. Copyright (C) 2007-2017 Pivotal Software
, Inc.
  ##  ##      Licensed under the MPL.  See http://www.rabbitmq.com/
  ##  ##
  ##########  Logs: /root/rabbitmq_server-3.6.14/var/log/rabbitmq/rabbi
t@rabbitmq1.log
  ######  ##        /root/rabbitmq_server-3.6.14/var/log/rabbitmq/rabbi
t@rabbitmq1-sasl.log
  ##########
              Starting broker...
 completed with 0 plugins.
```

现在 RabbitMQ 就已经启动起来了，但是有一个问题，RabbitMQ 一直占据着终端，我们得让它在后台运行。Ctrl+C 终止进程，然后执行命令

```
root@rabbitmq1:~/rabbitmq_server-3.6.14/sbin# ./rabbitmq-server -detach
ed
Warning: PID file not written; -detached was passed.
```

可以使用命令查看运行状态：

```
root@rabbitmq1:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl status
```

下面开始将这三个独立的 RabbitMQ 组建成一个集群。

3. **Rabbit 集群**

   - 集群模式

     集群中的节点有两种，一种是内存节点，一种是磁盘节点；内存节点由于没有磁盘读写，性能比磁盘节点要好，磁盘节点可以将状态持久化到磁盘，可用性比内存节点要好，需要权衡考虑。
     本文的模式是一台磁盘节点，作为数据备份，两台内存节点，用于提高性能。

   - 配置 hosts

     在安装好 RabbitMQ 的三台机器上分别修改 /etc/hosts 文件。

rabbit@rabbitmq1 上：

```
root@rabbitmq1:~# cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2  rabbitmq1
172.17.0.3      rabbitmq2
172.17.0.4      rabbitmq3
```

rabbit@rabbitmq2 上：

```
root@rabbitmq2:~# cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      rabbitmq1
172.17.0.3  rabbitmq2
172.17.0.4      rabbitmq3
```

rabbit@rabbitmq3 上：

```
root@rabbitmq2:~# cat /etc/hosts
127.0.0.1   localhost
::1 localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2      rabbitmq1
172.17.0.3  rabbitmq2
172.17.0.4      rabbitmq3
```

- 配置 Cookie

```
Erlang 节点间通过认证 Erlang cookie 的方式允许互相通信。因为 rabbitmqctl 使用 Erl
ang OTP 通信机制来和 Rabbit 节点通信，运行 rabbitmqctl 的机器和所要连接的 Rabbit
节点必须使用相同的 Erlang cookie 。否则你会得到一个错误。
```

为了保证这三台机器的 Erlang cookie 相同，我们将 rabbitmq1 上面的 Erlang cookie 文件复制到另

外两台机器上面。在我的机器上面， Erlang cookie 文件的路径是 `/root/.erlang.cookie`，下面是将这个 cookie 文件复制到另外两台机器上的步骤：(在 Docker 宿主机上执行)

```
➜  ~ docker cp rabbitmq1:/root/.erlang.cookie .
➜  ~ docker cp .erlang.cookie rabbitmq2:/root
➜  ~ docker cp .erlang.cookie rabbitmq3:/root
```

步骤解释：现将 rabbitmq1 上的 .erlang.cookie 复制到宿主机的当前目录，然后在分别复制到 rabbitmq2 和 rabbitmq3 上的 root 目录下。

OK，现在三台机器上具有相同的 Erlang cookie 了。下面开始组建集群。

- 组建集群
  - 查看集群状态

    ```
    root@rabbitmq1:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl cluster_sta
    tus
    Cluster status of node rabbit@rabbitmq1
    [{nodes,[{disc,[rabbit@rabbitmq1]}]},
     {running_nodes,[rabbit@rabbitmq1]},
     {cluster_name,<<"rabbit@rabbitmq1">>},
     {partitions,[]},
     {alarms,[{rabbit@rabbitmq1,[]}]}]
    ```

    可以看到 `{cluster_name,<<"rabbit@rabbitmq1">>}`，这是集群名字，其他节点可以 `join` 到这个集群中。

    其实，所有节点都是平等的，你可以加入到任意一个集群中，最终这些节点都会在同一个集群中。

  - 将 RabbitMQ 停止运行

    rabbitmq2:

    ```
    root@rabbitmq2:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl stop_app
    Stopping rabbit application on node rabbit@rabbitmq2
    ```

    rabbitmq3:

    ```
    root@rabbitmq3:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl stop_app
    Stopping rabbit application on node rabbit@rabbitmq3
    ```

  - 执行加入集群命令

    rabbitmq2:

```
root@rabbitmq2:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl join_cluste
r rabbit@rabbitmq1
Clustering node rabbit@rabbitmq2 with rabbit@rabbitmq1
root@rabbitmq2:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl start_app
Starting node rabbit@rabbitmq2
```

rabbitmq3:

```
root@rabbitmq3:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl join_cluste
r rabbit@rabbitmq1
Clustering node rabbit@rabbitmq3 with rabbit@rabbitmq1
root@rabbitmq3:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl start_app
Starting node rabbit@rabbitmq3
```

在任意节点查看集群状态：

```
root@rabbitmq1:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl cluster_sta
tus
Cluster status of node rabbit@rabbitmq1
[{nodes,[{disc,[rabbit@rabbitmq1,rabbit@rabbitmq2,rabbit@rabbitmq3]}]},
 {running_nodes,[rabbit@rabbitmq3,rabbit@rabbitmq2,rabbit@rabbitmq1]},
 {cluster_name,<<"rabbit@rabbitmq1">>},
 {partitions,[]},
 {alarms,[{rabbit@rabbitmq3,[]},{rabbit@rabbitmq2,[]},{rabbit@rabbitmq1
,[]}]}]
```

可以看到三个节点已经构建成了一个集群，但是有一个小问题，`{disc,[rabbit@rabbitmq1,rabbit@rabbitmq2,rabbit@rabbitmq3]}`，所有的节点都是磁盘节点，和刚开始构想的不太一样，我们只需要一个磁盘节点，另外两个都是内存节点，这样我们在保证可用性的同时，还能提高集群的整体性能。

所以，我们需要将两台磁盘节点改成内存节点。

- 将磁盘节点修改为内存节点
  - 将节点从集群中移除

rabbitmq2:

```
root@rabbitmq2:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl stop_app
Stopping rabbit application on node rabbit@rabbitmq2
root@rabbitmq2:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl reset
Resetting node rabbit@rabbitmq2
```

rabbitmq3:

```
root@rabbitmq3:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl stop_app
Stopping rabbit application on node rabbit@rabbitmq3
root@rabbitmq3:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl reset
Resetting node rabbit@rabbitmq3
```

这里关键的命令是 `./rabbitmqctl reset` 。reset 命令在节点为单机状态和是集群的一部分时行为有点不太一样。

节点单机状态时，reset 命令将清空节点的状态，并将其恢复到空白状态。当节点是集群的一部分时，该命令也会和集群中的磁盘节点通信，告诉他们该节点正在离开集群。

这很重要，不然，集群会认为该节点出了故障，并期望其最终能够恢复回来，在该节点回来之前，集群禁止新的节点加入。

- 将节点重新加入集群

  rabbitmq2:

  ```
  root@rabbitmq2:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl join_cl
  uster --ram rabbit@rabbitmq1
  Clustering node rabbit@rabbitmq2 with rabbit@rabbitmq1
  root@rabbitmq2:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl start_a
  pp
  Starting node rabbit@rabbitmq2
  ```

  rabbitmq3:

  ```
  root@rabbitmq3:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl join_cl
  uster --ram rabbit@rabbitmq1
  Clustering node rabbit@rabbitmq3 with rabbit@rabbitmq1
  root@rabbitmq3:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl start_a
  pp
  Starting node rabbit@rabbitmq3
  ```

  然后查看集群状态：

```
root@rabbitmq1:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl cluster
_status
Cluster status of node rabbit@rabbitmq1
[{nodes,[{disc,[rabbit@rabbitmq1]},{ram,[rabbit@rabbitmq3,rabbit@ra
bbitmq2]}]},
 {running_nodes,[rabbit@rabbitmq3,rabbit@rabbitmq2,rabbit@rabbitmq1
]},
 {cluster_name,<<"rabbit@rabbitmq1">>},
 {partitions,[]},
 {alarms,[{rabbit@rabbitmq3,[]},{rabbit@rabbitmq2,[]},{rabbit@rabbi
tmq1,[]}]}]
```

现在集群已经大体符合开始时我们的构想了，一个磁盘节点，两个内存节点。但是，我们的集群还是有点小问题，我们这个集群更适合非持久化队列，只有该队列是非持久的，客户端磁能重新连接到集群里的其他节点，并创建队列。加入该队列是持久化的，那么我们将集群恢复的唯一办法是将故障节点恢复起来。所以，我们需要对我们的集群进行改造，稍后我们会看到 Rabbit 团队给我们带来的内建的双活冗余选项：镜像队列。

4. **使用 HAProxy 做负载均衡**

   - 下载 HAProxy 源码

     Git 仓库地址：Git: haproxy

     下载源码：

     ```
     root@haproxy:~# git clone https://github.com/haproxy/haproxy.git
     ```

   - 编译&安装

     更详细的编译参数请参考文档，本文通用的编译参数。

     编译 & 安装：

     ```
     root@haproxy:~/haproxy# make TARGET=generic
     root@haproxy:~/haproxy# make install
     ```

     现在你可以运行 `haproxy --help` 来看看它的配置选项。现在需要对 HAProxy 进行配置，这样它就可以 RabbitMQ 做负载均衡了。

   - 配置 HAProxy

     HAProxy 使用单一配置文件来定义所有属性，包括从前端 IP 到后端服务器。下列清单展示了用于本地 Rabbit 集群负载均衡的配置。

     配置文件参数解释：

```
###########全局配置#########
global
    log 127.0.0.1 local0 #[日志输出配置，所有日志都记录在本机，通过local0输出]
    log 127.0.0.1 local1 notice #定义haproxy 日志级别[error warringinfo debug
]
    daemon #以后台形式运行harpoxy
    nbproc 1 #设置进程数量
    maxconn 4096 #默认最大连接数,需考虑ulimit-n限制
    #user haproxy #运行haproxy的用户
    #group haproxy #运行haproxy的用户所在的组
    #pidfile /var/run/haproxy.pid #haproxy 进程PID文件
    #ulimit-n 819200 #ulimit 的数量限制
    #chroot /usr/share/haproxy #chroot运行路径
    #debug #haproxy 调试级别，建议只在开启单进程的时候调试
    #quiet

########默认配置############
defaults
    log global
    mode http #默认的模式mode { tcp|http|health }，tcp是4层，http是7层，health
只会返回OK
    option httplog #日志类别,采用httplog
    option dontlognull #不记录健康检查日志信息
    retries 2 #两次连接失败就认为是服务器不可用，也可以通过后面设置
    #option forwardfor #如果后端服务器需要获得客户端真实ip需要配置的参数，可以从Http
 Header中获得客户端ip
    option httpclose #每次请求完毕后主动关闭http通道,haproxy不支持keep-alive,只能
模拟这种模式的实现
    #option redispatch #当serverId对应的服务器挂掉后，强制定向到其他健康的服务器，以
后将不支持
    option abortonclose #当服务器负载很高的时候，自动结束掉当前队列处理比较久的链接
    maxconn 4096 #默认的最大连接数
    timeout connect 5000ms #连接超时
    timeout client 30000ms #客户端超时
    timeout server 30000ms #服务器超时
    #timeout check 2000 #心跳检测超时
    #timeout http-keep-alive10s #默认持久连接超时时间
    #timeout http-request 10s #默认http请求超时时间
    #timeout queue 1m #默认队列超时时间
    balance roundrobin #设置默认负载均衡方式，轮询方式
    #balance source #设置默认负载均衡方式，类似于nginx的ip_hash
    #balnace leastconn #设置默认负载均衡方式，最小连接数

########统计页面配置########
listen stats
    bind 0.0.0.0:1080 #设置Frontend和Backend的组合体，监控组的名称，按需要自定义名
称
    mode http #http的7层模式
```

```
    option httplog #采用http日志格式
    #log 127.0.0.1 local0 err #错误日志记录
    maxconn 10 #默认的最大连接数
    stats refresh 30s #统计页面自动刷新时间
    stats uri /stats #统计页面url
    stats realm XingCloud\ Haproxy #统计页面密码框上提示文本
    stats auth admin:admin #设置监控页面的用户和密码:admin,可以设置多个用户名
    stats auth Frank:Frank #设置监控页面的用户和密码：Frank
    stats hide-version #隐藏统计页面上HAProxy的版本信息
    stats admin if TRUE #设置手工启动/禁用，后端服务器(haproxy-1.4.9以后版本)

########设置haproxy 错误页面#####
#errorfile 403 /home/haproxy/haproxy/errorfiles/403.http
#errorfile 500 /home/haproxy/haproxy/errorfiles/500.http
#errorfile 502 /home/haproxy/haproxy/errorfiles/502.http
#errorfile 503 /home/haproxy/haproxy/errorfiles/503.http
#errorfile 504 /home/haproxy/haproxy/errorfiles/504.http

########frontend前端配置##############
frontend main
    bind *:80 #这里建议使用bind *:80的方式，要不然做集群高可用的时候有问题，vip切换到其他机器就不能访问了。
    acl web hdr(host) -i www.abc.com  #acl后面是规则名称，-i为忽略大小写，后面跟的是要访问的域名，如果访问www.abc.com这个域名，就触发web规则，。
    acl img hdr(host) -i img.abc.com  #如果访问img.abc.com这个域名，就触发img规则。
    use_backend webserver if web   #如果上面定义的web规则被触发，即访问www.abc.com，就将请求分发到webserver这个作用域。
    use_backend imgserver if img   #如果上面定义的img规则被触发，即访问img.abc.com，就将请求分发到imgserver这个作用域。
    default_backend dynamic #不满足则响应backend的默认页面

########backend后端配置##############
backend webserver #webserver作用域
    mode http
    balance roundrobin #balance roundrobin 负载轮询，balance source 保存session值，支持static-rr，leastconn，first，uri等参数
    option httpchk /index.html HTTP/1.0 #健康检查，检测文件，如果分发到后台index.html访问不到就不再分发给它
    server web1 10.16.0.9:8085 cookie 1 weight 5 check inter 2000 rise 2 fall 3
    server web2 10.16.0.10:8085 cookie 2 weight 3 check inter 2000 rise 2 fall 3
    #cookie 1表示serverid为1，check inter 1500 是检测心跳频率
    #rise 2是2次正确认为服务器可用，fall 3是3次失败认为服务器不可用，weight代表权重

backend imgserver
    mode http
```

```
    option httpchk /index.php
    balance roundrobin
    server img01 192.168.137.101:80 check inter 2000 fall 3
    server img02 192.168.137.102:80 check inter 2000 fall 3

backend dynamic
    balance roundrobin
    server test1 192.168.1.23:80 check maxconn 2000
    server test2 192.168.1.24:80 check maxconn 2000

listen tcptest
    bind 0.0.0.0:5222
    mode tcp
    option tcplog #采用tcp日志格式
    balance source
    #log 127.0.0.1 local0 debug
    server s1 192.168.100.204:7222 weight 1
    server s2 192.168.100.208:7222 weight 1
```

我们的配置文件（**/root/haproxy/haproxy.cnf**）内容如下：

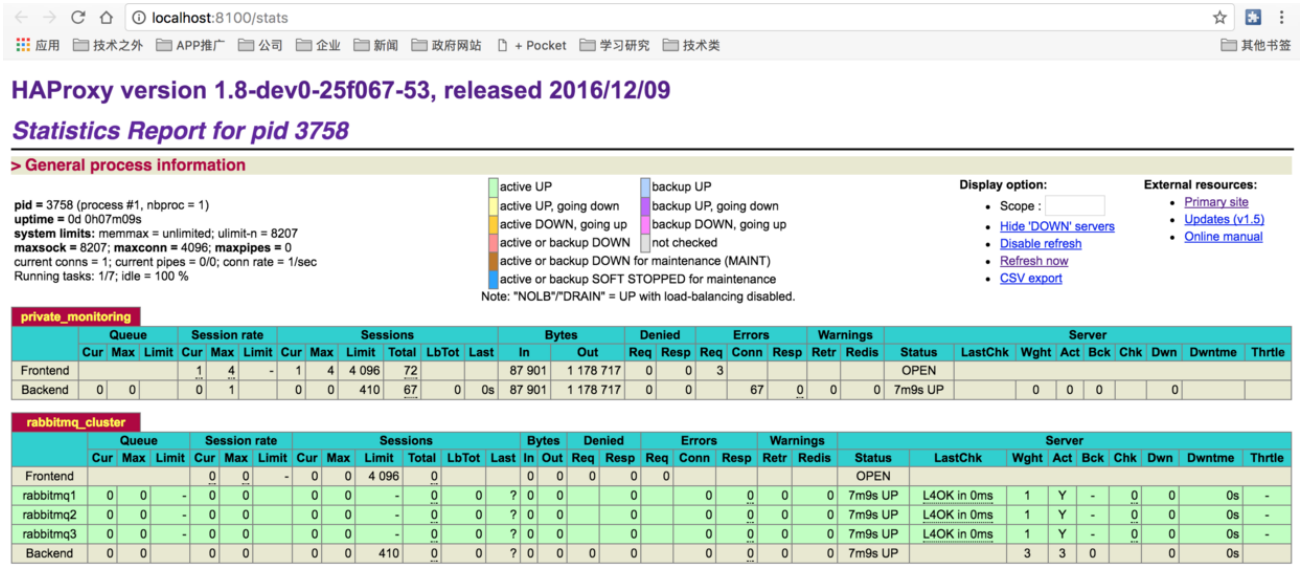
```
root@haproxy:~/haproxy# cat haproxy.cnf
global
    log     127.0.0.1  local0 info
    log     127.0.0.1  local1 notice
    daemon
    maxconn 4096

defaults
    log     global
    mode    tcp
    option  tcplog
    option  dontlognull
    retries 3
    option  abortonclose
    maxconn 4096
    timeout connect  5000ms
    timeout client  3000ms
    timeout server  3000ms
    balance roundrobin

listen private_monitoring
    bind    0.0.0.0:8100
    mode    http
    option  httplog
    stats   refresh  5s
    stats   uri  /stats
    stats   realm   Haproxy
    stats   auth  admin:admin

listen rabbitmq_admin
    bind    0.0.0.0:8102
    server  rabbitmq1 172.17.0.2:15672
    server      rabbitmq2 172.17.0.3:15672
    server      rabbitmq3 172.17.0.4:15672

listen rabbitmq_cluster
    bind    0.0.0.0:8101
    mode    tcp
    option  tcplog
    balance roundrobin
    server  rabbitmq1  172.17.0.2:5672  check  inter  5000  rise  2  fall
 3
    server  rabbitmq2  172.17.0.3:5672  check  inter  5000  rise  2  fall
 3
    server  rabbitmq3  172.17.0.4:5672  check  inter  5000  rise  2  fall
 3
```

- 运行 HAProxy

让我们用新的配置启动 HAProxy 并确保它能工作：

```
root@haproxy:~/haproxy# haproxy -f haproxy.cnf
```

如果一切正常的话，你现在可以顺利的加载 `http://localhost:8100/stats` 页面，如果配置文件中配置了 `auth` 的话，你需要进行登录：



这有个小缺陷，虽然我们在配置文件中做了 management 的反向代理，但是我们仍然不能访问 `http://localhost:8102/` 。这是因为我们没有开启 rabbit management 插件，下面我们来开启这个插件。

5. **开启 Rabbit Management 插件，从 WEB 端管理 RabbitMQ**

插件是一种扩展作者未能预见到的服务器的行为方式。RabbitMQ 插件是由 Erlang 语言编写的，并且和服务器一同运行在同一个 Erlang VM 中。我们先开启 Management 插件。执行如下命令：

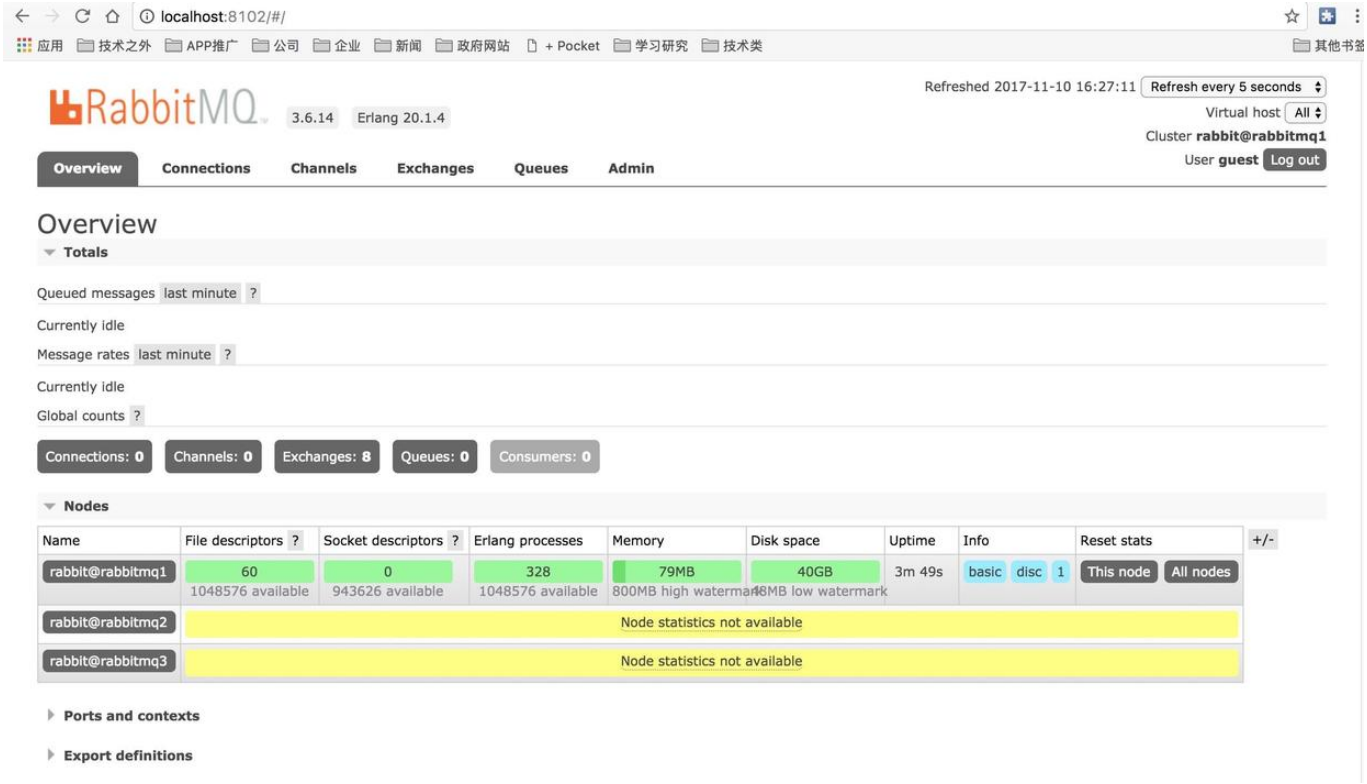
```
root@rabbitmq1:~/rabbitmq_server-3.6.14/sbin# ./rabbitmq-plugins enable rabbit
mq_management
The following plugins have been enabled:
  amqp_client
  cowlib
  cowboy
  rabbitmq_web_dispatch
  rabbitmq_management_agent
  rabbitmq_management

Applying plugin configuration to rabbit@rabbitmq1... started 6 plugins.
root@rabbitmq1:~/rabbitmq_server-3.6.14/sbin# ./rabbitmqctl stop
Stopping and halting node rabbit@rabbitmq1
root@rabbitmq1:~/rabbitmq_server-3.6.14/sbin# ./rabbitmq-server -detached
Warning: PID file not written; -detached was passed.
```

然后重新访问 `http://localhost:8102/`，页面提示我们需要登录，RabbitMQ 有一个默认的账号 guest，密码是 guest。我们用这个账号登陆会发现，我们被拒绝了，页面提示 `User can only log in via localhost`，这是因为rabbitmq从3.3.0开始禁止使用guest/guest权限通过除localhost外的访问。如果想使用 guest 通过远程机器访问，需要在 RabbitMQ 配置文件中设置 loopback_users 为 []，我们的配置文件内容如下：

/root/rabbitmq_server-3.6.14/etc/rabbitmq/rabbitmq.config：

```
root@rabbitmq1:~/rabbitmq_server-3.6.14/etc/rabbitmq# cat rabbitmq.config
r
```

刷新页面，然后重新登陆，现在我们能看到如下的界面：



下面黄色部分是因为另外两个节点还未开启 Management 插件，开启即可。

6. **使用 RabbitMQ 镜像功能（待补充）**

7. 升级集群节点

    1. 关闭生产者
    2. 备份当前配置
    3. 排空队列 - 观察队列状态，直到所有队列都为空
    4. 关闭节点
    5. 解压新版本 RabbitMQ 到现有的安装目录
    6. 启动磁盘节点
    7. 启动内存节点