Openstack 高可用指导书

1.Openstack 高可用介绍

高可用系统至少追求两方面：

- 系统故障----当面向用户的服务在超过指定最大时间后还未响应时发生

- 数据丢失----意外删除或数据损坏

大部分高可用系统可以在单一故障发生的时候提供保护机制来阻止系统故障和数据丢失。然而，他们也期望提供保护来阻止由单一故障恶化为一系列的级联故障。

高可用最重要的方面就是消除单点故障(SPOFs)。SPOF 是一个单独的设备或软件，一旦出错，会导致系统宕机或数据丢失，为了消除 SPOFs，检查下列冗余机制：

- 网络组件，例如交换机和路由器
- 应用程序和自动服务迁移
- 存储组件
- 设备服务，例如电力，通风，消防条件

大部分高可用系统在多个独立故障（非级联）发生的时候会失效。在这种情况下，大部分系统会保护数据而非保持可用性。

高可用系统可以使正常运行时间达到 99.99%以上，大概可以达到相当于每年少于一小时的累计宕机时间的效果。为了达到这个要求，高可用系统应当在一到两分钟时间的故障后保持恢复时间，有时候故障时间更短。

Openstack 目前在自己的基础设施服务中符合这样的可用性要求，这意味着 99.99%的正常运行时间对 Openstack 基础设施是适当可行的。然而对于个别用户而言，Openstack 不能保证 99.99%的可用性。

预防单点故障可以依靠检测是否有服务是无状态的。

无状态与有状态服务

无状态服务是对您的请求作出回应后，不会做进一步关注。为了使无状态服务高可用，您需要提供冗余的实例和负载均衡。Openstack 中的无状态服务包括 nova-api,nova-conductor, glance-api, keystone-api, neutron-api and nova-scheduler。

一个有状态服务就是后续的请求依赖于第一个请求的结果，有状态服务管理起来比较困难，因为一个单一的操作涉及多个请求，因此，简单提供额外的实例和负载均衡不能解决问题。例如，如果 Horizon 用户界面在你每次去到新的一页时自动重置，这不是很有用的。Openstack 中的有状态服务包括数据库和消息队列。

使有状态的服务高可用依赖于你选择主动/被动还是主动/主动配置。

主动/被动

在主动/被动配置中，系统被设置把额外的资源放到环境中去代替已经产生故障的。例如，Openstack 在写主数据库的同时维护着一个灾难复原数据库，当主数据库发生故障时灾难恢复数据库会代替它继续工作。

通常情况下，安装一个主动/被动式的无状态服务需要维护一个在需要时能被放到环境中继续运行的冗余实例。用一个虚拟 IP 和负载均衡器来保持请求平衡，例如 HAProxy。

安装一个主动/被动式的有状态服务维护着一个在需要时能被放到环境中继续运行的替代资源。有个应用（例如 Pacemaker，Corosync）去分开监视这些服务，必要时将后备资源投入到线上使用。

主动/主动

在主动主动配置中，系统也使用备份但是会同时管理主系统和冗余系统。这种方式下，如果发生了一个用户可能没有注意到的故障，备份系统仍然在运行，并且会采取增加负载当主系统恢复并继续启用。

通常情况下，安装一个主动/主动式的无状态服务需要维护一个冗余实例，用

一个虚拟 IP 和负载均衡器来保持请求平衡，例如 HAProxy。

一个典型的主动/主动式的有状态服务将包含所有实例拥有同一状态的冗余服务。例如，更新一个数据库的实例也会更新其他所有的实例，这样的话对一个实例的请求也同样会对其他实例请求。一个负载均衡器管理着这些系统的流量，确保运行的系统总是在处理这些请求。

这都是一些实现高可用架构的常用方法，单不意味着这是一些唯一的方法。重要的事是确保你的服务是多余的，可用的，怎么实现取决于你自己。本文将介绍高可用系统常用的一些选项。

2.主动/被动式 HA

Pacemaker 集群堆栈

Openstack 基础架构高可用性依赖于 Pacemaker 集群堆栈，它是 Linux 平台下最先进的高可用和负载均衡堆栈。Pacemaker 并不是 Openstack 特定有的，并且与它的存储和应用无关。

Pacemaker 可靠的集群通信依赖于 Corosync 消息层，Corosync 实现单环顺序和成员协议，并且向 Pacemaker 提供以消息传送，消息仲裁和集群成员为基础的 UDP 和无限带宽技术。

Pacemaker 通过资源代理（RAs）与应用程序进行交互，其中它本身支持超过 70%，Pacemaker 也可以使用第三方的 RAs。Openstack 高可用配置使用现存的本地的 Pacemaker RAs（例如 MySQL 数据库管理和虚拟 IP 地址管理），现存的第三方 RAs（例如 RabbitMQ）和本地的 Openstack RAs（例如 Openstack 的身份认证和镜像服务管理）。

安装包

任何 Pacemaker 集群中的主机，首先必须要通过 Corosync 消息层去建立集群通信。包括安装下列包（都是独立的包）：

- Pacemaker
- Corosync
- Cluster-glue
- Fence-agents（只有 fedora 需要安装，其他系统中 Cluster-glue 包括 Fence-agents）
- Resource-agents

设置 Corosync

除了安装 corosync 包外，还需要修改配置文件（/etc/corosync/corosync.conf），大部分系统会在 corosync 安装包中将实例配置文件（corosync.conf.example）作为文档的一部分，下面是一个事例：

Corosync configuration file(corosync.conf)

```
totem {
        version: 2
        # Time (in ms) to wait for a token
        token: 10000
        # How many token retransmits before forming a new
        # configuration
        token_retransmits_before_loss_const: 10
        # Turn off the virtual synchrony filter
        vsftype: none
        # Enable encryption
        secauth: on
        # How many threads to use for encryption/decryption
        threads: 0
        # This specifies the redundant ring protocol, which may be
        # none, active, or passive.
        rrp_mode: active
        # The following is a two-ring multicast configuration.
        interface {
                ringnumber: 0
                bindnetaddr: 192.168.42.0
                mcastaddr: 239.255.42.1
                mcastport: 5405
        }
        interface {
                ringnumber: 1
                bindnetaddr: 10.0.42.0
                mcastaddr: 239.255.42.2
                mcastport: 5405
        }
}
```

```
amf {
        mode: disabled
}
service {
        # Load the Pacemaker Cluster Resource Manager
        ver:       1
        name:      pacemaker
}
aisexec {
        user:   root
        group:  root
}
logging {
        fileline: off
        to_stderr: yes
        to_logfile: no
        to_syslog: yes
        syslog_facility: daemon
        debug: off
        timestamp: on
        logger_subsys {
                subsys: AMF
                debug: off
                tags: enter|leave|trace1|trace2|trace3|trace4|trace6
        }
}
```

启动 Corosync

Corosync 作为一个常用的系统服务被启动，根据你的发行版，可能会附带一个 LSB 初始化脚本，一个特殊的工作，或者一个 systemd 单元文件。无论哪种

方式，这个服务都被命名为 corosync：

- /etc/init.d/corosync start (LSB)
- service corosync start (LSB, alternate)
- start corosync (upstart)
- systemctl start corosync (systemd)

您现在可以用两个工具检查 Corosync 的连通性。

Corosync-cfgtool 工具，加上-s 调用参数，可以给出通信环的概要信息：

```
# corosync-cfgtool -s
Printing ring status.
Local node ID 435324542
RING ID 0
        id      = 192.168.42.82
        status  = ring 0 active with no faults
RING ID 1
        id      = 10.0.42.100
        status  = ring 1 active with no faults
```

Corosync-objctl 可以被用来存储 Corosync 集群成员列表：

```
# corosync-objctl runtime.totem.pg.mrp.srp.members
runtime.totem.pg.mrp.srp.435324542.ip=r(0) ip(192.168.42.82) r(1) ip(10.0.42.
100)
runtime.totem.pg.mrp.srp.435324542.join_count=1
runtime.totem.pg.mrp.srp.435324542.status=joined
runtime.totem.pg.mrp.srp.983895584.ip=r(0) ip(192.168.42.87) r(1) ip(10.0.42.
254)
runtime.totem.pg.mrp.srp.983895584.join_count=1
runtime.totem.pg.mrp.srp.983895584.status=joined
```

你应该可以看到 status=joined 的构成集群的每个节点。

启动 Pacemaker

一旦 Corosync 服务启动，并且你已经建立了可以正常通信的集群后，可以安

全启动 Pacemaker 的主控制进程----pacemakerd：

- /etc/init.d/pacemaker start (LSB)
- service pacemaker start (LSB, alternate)
- start pacemaker (upstart)
- systemctl start pacemaker (systemd)

一旦 Pacemaker 服务启动，Pacemaker 将会创建一个默认的没有配置任何资源的空集群。你可以通过 crm_mon 观察 Pacemaker 的状态：

```
============
Last updated: Sun Oct  7 21:07:52 2012
Last change: Sun Oct  7 20:46:00 2012 via cibadmin on node2
Stack: openais
Current DC: node2 - partition with quorum
Version: 1.1.6-9971ebba4494012a93c03b40a2c58ec0eb60f50c
2 Nodes configured, 2 expected votes
0 Resources configured.
============
Online: [ node2 node1 ]
```

设置基本的集群性能参数

Pacemaker 集群建立后，建议去设置一些基础集群属性。启动 crm 脚本，键入 configure 切换到配置菜单。或者，你可以通过 crm configure 直接跳到 Pacemaker 配置界面，然后设置下列属性：

```
property no-quorum-policy="ignore" \ #  ①
    pe-warn-series-max="1000" \             #  ②
    pe-input-series-max="1000" \
    pe-error-series-max="1000" \
    cluster-recheck-interval="5min"       #  ③
```

①设置 no-quorum-policy="ignore"需要在两节点的 Pacemaker 集群，原因如下：如果两个节点中有一个产生故障，强制执行仲裁时，剩下的节点不能得到大多数的仲裁投票去运行服务，因此不能接管任何资源。合理的变通方案是忽略集群中

的仲裁，在两个节点的集群中是安全和必要的。在多余两个节点的集群中不要设置此属性。

②把 pe-warn-series-max， pe-input-series-max， pe-error-series-max 设置为 1000 以保持策略引擎产生的输入日志，错误和警告较长时间，如果集群发生故障时这些历史信息是很有用的。

③Pacemaker 使用事件驱动接近集群状态，然而，它发生在一个可配置的间隔内，cluster-recheck-interval，默认是 15 分钟，一般被削减到 3-5 分钟。

做完这些事后，你可以保存这个更新完的配置。

云控制集群堆栈

云管理器包括在网络管理中，并且与其他所有的服务进行通信。

高可用 MySQL

MySQL 被多个 Openstack 服务作为默认的数据库使用，使得 MySQL 服务高可用包括：

- 为 MysQL 配置一个 DRBD 设备
- 配置 MySQL 使用一个属于 DRBD 设备的数据目录
- 选择和分配一个可以在集群节点之间浮动的虚拟 IP
- 配置 MySQL 监听这个 IP
- 管理所有的资源，包括 MySQL 自身

配置 DRBD

Pacemaker 以 MySQL 服务为基础，需要挂载在/var/lib/mysql 目录下的一个 DRBD 资源，在这个例子中，这个 DRBD 设备被简单命名为 mysql：

mysql DRBD resource configuration (/etc/drbd.d/mysql.res).

```
resource mysql {
   device       minor 0;
   disk         "/dev/data/mysql";
   meta-disk internal;
   on node1 {
      address ipv4 10.0.42.100:7700;
```

```
    }
    on node2 {
        address ipv4 10.0.42.254:7700;
    }
}
```

这个资源在集群的所有节点上都使用被命名为/dev/data/mysql 的本地磁盘，一般情况下，这是为了这个目的而专门留出的一个 LVM 逻辑卷。DRBD 中 meta-disk 为 internal，意味着 DRBD 指定的元数据被存储在 disk 设备末端。这个设备被配置在 10.0.42.100 到 10.0.42.254 之间通信，使用 tcp 端口为 7700。一旦启用，它会映射到本地小设备号为 0 的 DRBD 块设备上，也就是/dev/drbd0。

如何启用一个 DRBD 资源在 DRBD 用户指导书中会详细介绍，简而言之，命令的固有顺序为：

```
drbdadm create-md mysql
drbdadm up mysql
drbdadm -- --force primary mysql
```

创建一个文件系统

DRBD 资源运行并充当主要角色后（可能仍然在运行初始设备同步的过程中），你可以继续为 MySQL 数据创建文件系统，XFS 是一般被推荐的文件系统：

```
mkfs -t xfs /dev/drbd1
```

你还可以使用 DRBD 设备的备用设备路径，它可能更容易被记住，因为它包含了可以不言自明的资源名称：

```
mkfs -t xfs /dev/drbd/by-res/mysql
```

完成后，你可以安全地把设备放回次要角色，任何在进行的设备同步会在后台继续运行：

```
drbdadm secondary mysql
```

为 Pacemaker 高可用性准备 MySQL 数据库

为了使 Pacemaker 监视正常工作，你必须确保 MySQL 的数据库文件属于 DRBD 设备，如果你总是存在一个 MySQL 数据库，最简单的方法就是把/var/lib/mysql 目录下的内容移到新创建的 DRBD 设备上的文件系统。

注意：当数据库停止服务后，你必须完成以下操作

```
node1:# mount /dev/drbd/by-res/mysql /mnt
node1:# mv /var/lib/mysql/* /mnt
node1:# umount /mnt
```

对于一个新安装的 MySQL 数据库，你需要运行 mysql_install_db 命令：

```
node1:# mount /dev/drbd/by-res/mysql /mnt
node1:# mysql_install_db --datadir=/mnt
node1:# umount /mnt
```

不管如何，在一个单节点的集群中必须完成上面的指令。

为 Pacemaker 添加 MySQL 数据库资源

现在需要继续为 MySQL 资源添加 Pacemaker 配置，用 crm configure 连接到 Pacemaker 集群上，并且添加下列集群配置：

```
primitive p_ip_mysql ocf:heartbeat:IPaddr2 \
    params ip="192.168.42.101" cidr_netmask="24" \
    op monitor interval="30s"
primitive p_drbd_mysql ocf:linbit:drbd \
    params drbd_resource="mysql" \
    op start timeout="90s" \
    op stop timeout="180s" \
    op promote timeout="180s" \
    op demote timeout="180s" \
    op monitor interval="30s" role="Slave" \
    op monitor interval="29s" role="Master"
primitive p_fs_mysql ocf:heartbeat:Filesystem \
    params device="/dev/drbd/by-res/mysql" \
        directory="/var/lib/mysql" \
        fstype="xfs" \
        options="relatime" \
    op start timeout="60s" \
    op stop timeout="180s" \
```

```
    op monitor interval="60s" timeout="60s"
primitive p_mysql ocf:heartbeat:mysql \
    params additional_parameters="--bind-address=50.56.179.138"
        config="/etc/mysql/my.cnf" \
        pid="/var/run/mysqld/mysqld.pid" \
        socket="/var/run/mysqld/mysqld.sock" \
        log="/var/log/mysql/mysqld.log" \
    op monitor interval="20s" timeout="10s" \
    op start timeout="120s" \
    op stop timeout="120s"
group g_mysql p_ip_mysql p_fs_mysql p_mysql
ms ms_drbd_mysql p_drbd_mysql \
meta notify="true" clone-max="2"
colocation c_mysql_on_drbd inf: g_mysql ms_drbd_mysql:Master
order o_drbd_before_mysql inf: ms_drbd_mysql:promote g_mysql:start
```

- p_ip_mysql，MySQL 使用的一个虚拟 IP 地址（192.168.42.101）
- p_fs_mysql，一个 Pacemaker 管理的文件系统，位于挂载到当前每个运行 MySQL 服务的节点的/var/lib/mysql 目录下
- ms_drbd_mysql，mysql DRBD 资源的主仆管理模式
- Group，order，colocation 限制服务确保资源以正确的顺序运行在正确的节点上

Crm configure 支持批量输入，所以你可以将上述配置拷贝到 Pacemaker 配置中，根据需要修改一些参数。比如，你需要修改 p_ip_mysql 为自己设置的虚拟 ip 地址。

完成后，在 crm configure 菜单中键入 commit 来提交修改，Pacemaker 在其中一个节点上启动 mysql 服务和它的依赖项。

为高可用 MySQL 配置 Openstack 服务

你的 Openstack 服务现在一定将 MySQL 配置为高可用，将虚拟的集群 ip 地址，而不是 MySQL 物理 ip 地址作为你的意愿。

假如你的 MySQL 服务 ip 地址是之前配置文件中的 192.168.42.101，你现在需要将下面指令加入到 Openstack 镜像配置文件（glance-registry.conf）中：

```
sql_connection = mysql://glancedbadmin:<password>@192.168.42.101/glance
```

以你的 Openstack 配置，不需要做其他改变，如果目前托管你数据库的节点遇到问题迫使服务故障转移，你的 Openstack 服务会遇到短暂的 MySQL 中断，虽然发生在网络拥堵的时候，它们会继续正常运行。

高可用 RabbitMQ

RabbitMQ 是被很多 Openstack 服务默认使用的 AMQP 服务，使 RabbitMQ 服务高可用包括：

- 配置 DRBD 设备
- 配置 RabbitMQ 使用一个属于 DRBD 设备的数据目录
- 选择和分配一个可以在集群节点之间浮动的虚拟 IP
- 配置 RabbitMQ 监听这个 IP
- 管理所有的资源，包括 RabbitMQ 自身

配置 DRBD

Pacemaker 以 RabbitMQ 服务为基础，需要挂载在/var/lib/rabbitmq 目录下的一个 DRBD 资源，在这个例子中，这个 DRBD 设备被简单命名为 rabbitmq：

rabbitmq DRBD resource configuration (/etc/drbd.d/rabbitmq.res).

```
resource rabbitmq {
    device      minor 1;
    disk        "/dev/data/rabbitmq";
    meta-disk internal;
    on node1 {
        address ipv4 10.0.42.100:7701;
    }
    on node2 {
        address ipv4 10.0.42.254:7701;
    }
}
```

这个资源在集群的所有节点上都使用被命名为/dev/data/rabbitmq 的本地磁盘，一般情况下，这是为了这个目的而专门留出的一个 LVM 逻辑卷。DRBD 中 meta-disk 为 internal，意味着 DRBD 指定的元数据被存储在 disk 设备末端。这个设备被配置在 10.0.42.100 到 10.0.42.254 之间通信，使用 tcp 端口为 7701。一旦启用，它会映射到本地小设备号为 1 的 DRBD 块设备上，也就是/dev/drbd1。

如何启用一个 DRBD 资源在 DRBD 用户指导书中会详细介绍，简而言之，命令的固有顺序为：

```
drbdadm create-md rabbitmq ①
drbdadm up rabbitmq ②
drbdadm -- --force primary rabbitmq③
```

创建一个文件系统

DRBD 资源运行并充当主要角色后（可能仍然在运行初始设备同步的过程中），你可以继续为 RabbitMQ 数据创建文件系统，XFS 是一般被推荐的文件系统：

```
mkfs -t xfs /dev/drbd1
```

你还可以使用 DRBD 设备的备用设备路径，它可能更容易被记住，因为它包含了可以不言自明的资源名称：

```
mkfs -t xfs /dev/drbd/by-res/rabbitmq
```

完成后，你可以安全地把设备放回次要角色，任何在进行的设备同步会在后台继续运行：

```
drbdadm secondary rabbitmq
```

为 Pacemaker 高可用性准备 RabbitMQ

为了使 Pacemaker 监视正常工作，不管 DRBD 是否被挂载，你都要确保 RabbitMQ 的.erlang.cookie 文件在所有节点上都是完全相同的。最简单的方法就是从一个节点中将.erlang.cookie 拷贝到其他节点的 RabbitMQ 数据目录下，并且拷贝到 DRBD 后备文件系统。

```
node1:# scp -a /var/lib/rabbitmq/.erlang.cookie node2:/var/lib/rabbitmq/
node1:# mount /dev/drbd/by-res/rabbitmq /mnt
node1:# cp -a /var/lib/rabbitmq/.erlang.cookie /mnt
node1:# umount /mnt
```

向 Pacemaker 添加 RabbitMQ 资源

现在需要继续为 RabbitMQ 资源添加 Pacemaker 配置，用 crm configure 连接到 Pacemaker 集群上，并且添加下列集群配置：

```
primitive p_ip_rabbitmp ocf:heartbeat:IPaddr2 \
   params ip="192.168.42.100" cidr_netmask="24" \
   op monitor interval="10s"
primitive p_drbd_rabbitmq ocf:linbit:drbd \
   params drbd_resource="rabbitmq" \
   op start timeout="90s" \
   op stop timeout="180s" \
   op promote timeout="180s" \
   op demote timeout="180s" \
   op monitor interval="30s" role="Slave" \
   op monitor interval="29s" role="Master"
primitive p_fs_rabbitmq ocf:heartbeat:Filesystem \
   params device="/dev/drbd/by-res/rabbitmq" \
      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 \
   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
ms ms_drbd_rabbitmq p_drbd_rabbitmq \
   meta notify="true" master-max="1" clone-max="2"
colocation c_rabbitmq_on_drbd inf: g_rabbitmq ms_drbd_rabbitmq:Master
order o_drbd_before_rabbitmq inf: ms_drbd_rabbitmq:promote g_rabbitmq:start
```

- p_ip_rabbitmq，RabbitMQ 使用的一个虚拟 IP 地址（192.168.42.100）
- p_fs_rabbitmq，一个 Pacemaker 管理的文件系统，位于挂载到当前每个运行 RabbitMQ 服务的节点的/var/lib/rabbitmq 目录下
- ms_drbd_rabbitmq，rabbitmq DRBD 资源的主仆管理模式
- Group，order，colocation 限制服务确保资源以正确的顺序运行在正确的节点上

Crm configure 支持批量输入，所以你可以将上述配置拷贝到 Pacemaker 配置中，根据需要修改一些参数。比如，你需要修改 p_ip_rabbitmq 为自己设置的虚拟 ip 地址。

完成后，在 crm configure 菜单中键入 commit 来提交修改，Pacemaker 在其中一个节点上启动 rabbitmq 服务和它的依赖项。

为高可用 MySQL 配置 Openstack 服务

你的 Openstack 服务现在一定将 RabbitMQ 配置为高可用，将虚拟的集群 ip 地址，而不是 RabbitMQ 物理 ip 地址作为你的意愿。

假如你的 RabbitMQ 服务 ip 地址是之前配置文件中的 192.168.42.100，你现在需要将下面指令加入到 Openstack 镜像配置文件（glance-api.conf）中：

```
rabbit_host = 192.168.42.100
```

以你的 Openstack 配置，不需要做其他改变，如果目前托管你 RabbitMQ 的节点遇到问题迫使服务故障转移，你的 Openstack 服务会遇到短暂的 RabbitMQ 中断，虽然发生在网络拥堵的时候，它们会继续正常运行。

集群堆栈 API 节点

API 节点将 Openstack API 端点暴露在外部网络上（Internet），它需要与云控制器的管理网络相互通信。

配置虚拟 IP

首先，需要选择和分配可以在各个集群节点之间相互使用的虚拟 ip 地址；

配置文件创建了一个供 API 节点使用的虚拟 ip 地址（192.168.42.103）

```
primitive p_api-ip ocf:heartbeat:IPaddr2 \
        params ip="192.168.42.103" cidr_netmask="24" \
```

```
        op monitor interval="30s"
```

高可用 Openstack Identity

Openstack Identity 是 Openstack 的身份认证，并且被很多服务使用，使得 Openstack 身份认证在主动/被动模式中高可用包括：

- 配置 Openstack Identity 去监听虚拟 ip
- 用 Pacemaker 集群管理 Openstack Identity 守护进程
- 配置 Openstack 服务使用这个 ip

向 Pacemaker 中添加 Openstack Identity 资源

首先，需要给你的系统下载资源代理：

```
cd /usr/lib/ocf/resource.d
mkdir openstack
cd openstack
wget https://raw.github.com/madkiss/openstack-resource-agents/master/ocf/keystone
chmod a+rx *
```

你现在需要继续向 Openstack Identity 资源中添加 Pacemaker 配置，用“crm configure”连接 Pacemaker 集群，并且加入下列集群设置：

```
primitive p_keystone ocf:openstack:keystone \
   params config="/etc/keystone/keystone.conf" os_password="secret"
 os_username="admin" os_tenant_name="admin" os_auth_url="http://192.168.42.103:5000/v2.0/" \
   op monitor interval="30s" timeout="30s"
```

其中创建了 p_keystone，一个管理 Openstack Identity 服务的资源。

Crm configure 支持批量输入，所以你可以将上述配置拷贝到 Pacemaker 配置中，根据需要修改一些参数。比如，你需要修改 p_ip_keystone 为自己设置的虚拟 ip 地址。

完成后，在 crm configure 菜单中键入 commit 来提交修改，Pacemaker 在其中一个节点上启动 Openstack Identity 服务和它的依赖项。

配置 Openstack Identity 服务

你需要修改 Openstack Identity 配置文件（keystone.conf）中的参数：

```
bind_host = 192.168.42.103
```

为了确信所有的数据高可用，你需要确保你将所有数据存在 MySQL 数据库（也是高可用）中：

```
[catalog]
driver = keystone.catalog.backends.sql.Catalog
...
[identity]
driver = keystone.identity.backends.sql.Identity
...
```

配置 Openstack 服务去使用高可用 Openstack Identity

你的 Openstack 服务现在一定将 Openstack Identity 配置为高可用，将虚拟的集群 ip 地址，而不是 Openstack Identity 物理 ip 地址作为你的意愿。

例如，在 Openstack 计算节点中，如果 Openstack Identity 服务 ip 地址为 192.168.42.103，你需要在 API 配置文件（api-paste.ini）中加入下行：

```
auth_host = 192.168.42.103
```

你还需要用这个 ip 创建一个 Openstack Identity 端点

如果你同时使用私有和共有 IP 地址，你应该创建两个虚拟 ip 地址并且如下定义你的端点：

```
keystone endpoint-create --region $KEYSTONE_REGION --service-id $service-id
--publicurl 'http://PUBLIC_VIP:5000/v2.0' --adminurl 'http://192.168.42.
103:35357/v2.0' --internalurl 'http://192.168.42.103:5000/v2.0'
```

如果你使用 Horizon Dashboard，你需要修改 local_settings.py 文件：

```
OPENSTACK_HOST = 192.168.42.103
```

高可用 Openstack Image API

Openstack 镜像服务提供发现，注册，恢复虚拟机镜像操作，使 Openstack 镜像服务在主动/被动模式下高可用包括：

- 配置 Openstack Image 监听虚拟 ip 地址
- 用 Pacemaker 集群管理 Openstack Image API 守护进程

- 配置 Openstack 服务使用这个 IP 地址

向 Pacemaker 添加 Openstack Image API 资源

首先，需要在你的系统中下载资源代理：

```
cd /usr/lib/ocf/resource.d/openstack
wget https://raw.github.com/madkiss/openstack-resource-agents/master/ocf/
glance-api
chmod a+rx *
```

你需要继续向 Openstack Image API 资源添加 Pacemaker 配置，用“crm configure”命令连接 Pacemaker 集群，并且添加下列内容：

```
primitive p_glance-api ocf:openstack:glance-api \
      params config="/etc/glance/glance-api.conf" os_password="secrete"
 os_username="admin" os_tenant_name="admin" os_auth_url="http://192.168.42.
103:5000/v2.0/" \
      op monitor interval="30s" timeout="30s"
```

- p_glance-api，管理 Openstack Image API 服务的资源

Crm configure 支持批量输入，所以你可以将上述配置拷贝到 Pacemaker 配置中，根据需要修改一些参数。比如，你需要修改 p_ip_glance-api 为自己设置的虚拟 ip 地址。

完成后，在 crm configure 菜单中键入 commit 来提交修改，Pacemaker 在其中一个节点上启动 Openstack Image API 服务和它的依赖项。

配置 Openstack Image API 服务

编辑 /etc/glance/glance-api.conf：

```
# We have to use MySQL connection to store datas :
sql_connection=mysql://glance:password@192.168.42.101/glance
# We bind OpenStack Image API to the VIP :
bind_host = 192.168.42.103
# Connect to OpenStack Image Registry service :
registry_host = 192.168.42.103
# We send notifications to High Available RabbitMQ :
notifier_strategy = rabbit
```

```
rabbit_host = 192.168.42.102
```

配置 Openstack 服务使用高可用 Openstack Image API

你的 Openstack 服务现在一定将 Openstack Image API 配置为高可用，将虚拟的集群 ip 地址，而不是 Openstack Image API 物理 ip 地址作为你的意愿。

例如，在 Openstack 计算节点中，如果 Openstack Image API 服务 ip 地址为 192.168.42.104，你需要在 nova.conf 文件中加入下行：

```
glance_api_servers = 192.168.42.103
```

你还需要用这个 ip 创建一个 Openstack Image API 端点

如果你同时使用私有和共有 IP 地址，你应该创建两个虚拟 ip 地址并且如下定义你的端点：

```
keystone endpoint-create --region $KEYSTONE_REGION --service-id $service-id --publicurl 'http://PUBLIC_VIP:9292' --adminurl 'http://192.168.42.103:9292' --internalurl 'http://192.168.42.103:9292'
```

高可用 Cinder API

Cinder 是 Openstack 中的块存储，使得 Cinder API 服务在主动/被动模式下高可用包括：

- 配置 Cinder 监听一个虚拟 IP 地址
- 用 Pacemaker 集群管理器管理 Cinder API 守护进程
- 配置 Openstack 服务使用这个 IP 地址

向 Pacemaker 中添加 Cinder API 资源

首先，需要在你的系统中下载资源代理：

```
cd /usr/lib/ocf/resource.d/openstack
wget https://raw.github.com/madkiss/openstack-resource-agents/master/ocf/cinder-api
chmod a+rx *
```

你需要继续为 Openstack Cinder API 资源添加 Pacemaker 配置，用“crm configure”命令连接 Pacemaker 集群，并且添加下列内容：

```
primitive p_cinder-api ocf:openstack:cinder-api \
params config="/etc/cinder/cinder.conf" os_password="secrete" os_username=
```

```
"admin" \
    os_tenant_name="admin" keystone_get_token_url="http://192.168.42.103:5000/
v2.0/tokens" \
    op monitor interval="30s" timeout="30s"
```

- p_cinder-api，管理 Openstack Cinder API 服务的资源

Crm configure 支持批量输入，所以你可以将上述配置拷贝到 Pacemaker 配置中，根据需要修改一些参数。比如，你需要修改 p_ip_cinder-api 为自己设置的虚拟 ip 地址。

完成后，在 crm configure 菜单中键入 commit 来提交修改，Pacemaker 在其中一个节点上启动 Openstack Cinder API 服务和它的依赖项。

配置 Cinder API 服务

编辑 /etc/cinder/cinder.conf：

```
# We have to use MySQL connection to store datas :
sql_connection=mysql://cinder:password@192.168.42.101/cinder
# We bind Cinder API to the VIP :
osapi_volume_listen = 192.168.42.103
# We send notifications to High Available RabbitMQ :
notifier_strategy = rabbit
rabbit_host = 192.168.42.102
```

配置 Openstack 服务使用高可用 Cinder API

你的 Openstack 服务现在一定将 Openstack Cinder API 配置为高可用，将虚拟的集群 ip 地址，而不是 Openstack Cinder API 物理 ip 地址作为你的意愿。

你还需要用这个 ip 创建一个 Openstack Cinder API 端点

如果你同时使用私有和共有 IP 地址，你应该创建两个虚拟 ip 地址并且如下定义你的端点：

```
    keystone endpoint-create --region $KEYSTONE_REGION --service-id
$service-id --
    publicurl 'http://PUBLIC_VIP:8776/v1/%(tenant_id)s' --adminurl 'http://192.
    168.42.103:8776/v1/%(tenant_id)s' --internalurl 'http://192.168.42.103:8776/
    v1/%(tenant_id)s'
```

高可用 Openstack Networking Server

Openstack Networking 是 Openstack 中的网络连接服务，使得 Openstack 网络服务在主动/被动模式下高可用包括：

- 配置 Openstack Networking 监听一个虚拟 IP 地址
- 用 Pacemaker 集群管理器管理 Openstack Networking API 守护进程
- 配置 Openstack 服务使用这个 IP 地址

向 Pacemaker 中添加 Openstack Networking 资源

首先，需要在你的系统中下载资源代理：

```
cd /usr/lib/ocf/resource.d/openstack
wget https://raw.github.com/madkiss/openstack-resource-agents/master/ocf/
neutron-server
chmod a+rx *
```

你需要继续为 Openstack Networking Server 资源添加 Pacemaker 配置，用“crm configure”命令连接 Pacemaker 集群，并且添加下列内容：

```
primitive p_neutron-server ocf:openstack:neutron-server \
    params os_password="secrete" os_username="admin" os_tenant_name="admin" \
     keystone_get_token_url="http://192.168.42.103:5000/v2.0/tokens" \
     op monitor interval="30s" timeout="30s"
```

配置中声明的 p_neutron-server 是 Openstack Networking 服务的一个资源。

Crm configure 支持批量输入，所以你可以将上述配置拷贝到 Pacemaker 配置中，根据需要修改一些参数。比如，你需要修改 p_neutron-server 为自己设置的虚拟 ip 地址。

完成后，在 crm configure 菜单中键入 commit 来提交修改，Pacemaker 在其中一个节点上启动 Openstack Networking Server 服务和它的依赖项。

配置 Openstack Networking 服务

编辑 /etc/neutron/neutron.conf :

```
# We bind the service to the VIP :
bind_host = 192.168.42.103
# We bind OpenStack Networking Server to the VIP :
```

```
bind_host = 192.168.42.103
# We send notifications to Highly available RabbitMQ :
notifier_strategy = rabbit
rabbit_host = 192.168.42.102
[database]
# We have to use MySQL connection to store datas :
connection = mysql://neutron:password@192.168.42.101/neutron
```

配置 Openstack 服务使用高可用 Openstack Networking 服务

你的 Openstack 服务现在一定将 Openstack Networking Server 配置为高可用，将虚拟的集群 ip 地址，而不是 Openstack Networking Server 物理 ip 地址作为你的意愿。

例如，在 Openstack 计算节点中，你需要在 nova.conf 文件中加入下行：

```
neutron_url = http://192.168.42.103:9696
```

你还需要用这个 ip 创建一个 Openstack Networking Server 端点

如果你同时使用私有和共有 IP 地址，你应该创建两个虚拟 ip 地址并且如下定义你的端点：

```
keystone endpoint-create --region $KEYSTONE_REGION --service-id $service-id --
publicurl 'http://PUBLIC_VIP:9696/' --adminurl 'http://192.168.42.103:9696/'
 --internalurl 'http://192.168.42.103:9696/'
```

高可用 Ceilometer Central Agent

Ceilometer 是 Openstack 中的测量服务，主要的对资源利用率的代理测验不依赖于实例或者计算节点。

使得 Ceilometer Central Agent 服务在主动/被动模式下高可用包括 Pacemaker 集群管理器对守护进程的管理。

向 Pacemaker 中添加 Ceilometer Central Agent 资源

首先，需要在你的系统中下载资源代理：

```
cd /usr/lib/ocf/resource.d/openstack
wget https://raw.github.com/madkiss/openstack-resource-agents/master/ocf/
ceilometer-agent-central
```

```
chmod a+rx *
```

你需要继续为 Ceilometer Central Agent 资源添加 Pacemaker 配置，用“crm configure”命令连接 Pacemaker 集群，并且添加下列内容：

```
primitive p_ceilometer-agent-central ocf:openstack:ceilometer-agent-central \
   params config="/etc/ceilometer/ceilometer.conf" \
      op monitor interval="30s" timeout="30s"
```

- p_ceilometer-agent-central，一个 Ceilometer Central Agent 管理资源

Crm configure 支持批量输入，所以你可以将上述配置拷贝到 Pacemaker 配置中，根据需要修改一些参数。

完成后，在 crm configure 菜单中键入 commit 来提交修改，Pacemaker 在其中一个节点上启动 Ceilometer Central Agent 服务和它的依赖项。

配置 Ceilometer Central Agent 服务

编辑 /etc/ceilometer/ceilometer.conf：

```
# We use API VIP for Identity Service connection :
os_auth_url=http://192.168.42.103:5000/v2.0
# We send notifications to High Available RabbitMQ :
notifier_strategy = rabbit
rabbit_host = 192.168.42.102
[database]
# We have to use MySQL connection to store datas :
sql_connection=mysql://ceilometer:password@192.168.42.101/ceilometer
```

配置 Pacemaker 组

最后，我们需要创建一个服务组去确保虚拟 ip 连接到 API 服务资源上：

```
group g_services_api p_api-ip p_keystone p_glance-api p_cinder-api \
    p_neutron-server p_glance-registry p_ceilometer-agent-central
```

集群堆栈网络控制器

网络控制器是管理和数据网络中的一员，如果一个虚拟机需要访问它，需要将其连接到网络中。

高可用 Neutron 三层代理

Neutron 三层代理提供 L3/NAT 转发，确保租户网络上的虚拟机能访问外网，采用 Pacemaker 可以实现 L3 代理的高可用。

向 Pacemaker 中添加 Neutron 三层代理

首先，需要在你的系统中下载资源代理：

```
cd /usr/lib/ocf/resource.d/openstack
wget https://raw.github.com/madkiss/openstack-resource-agents/master/ocf/neutron-l3-agent
chmod a+rx neutron-l3-agent
```

你需要继续为 Neutron 三层代理资源添加 Pacemaker 配置，用“crm configure”命令连接 Pacemaker 集群，并且添加下列内容：

```
primitive p_neutron-l3-agent ocf:openstack:neutron-l3-agent \
    params config="/etc/neutron/neutron.conf" \
    plugin_config="/etc/neutron/l3_agent.ini" \
    op monitor interval="30s" timeout="30s"
```

- p_neutron-l3-agent，一个 Neutron L3 Agent 管理资源

Crm configure 支持批量输入，所以你可以将上述配置拷贝到 Pacemaker 配置中，根据需要修改一些参数。

完成后，在 crm configure 菜单中键入 commit 来提交修改，Pacemaker 在其中一个节点上启动 Neutron L3 Agent 服务和它的依赖项。

高可用 Neutron DHCP 代理

Neutron DHCP 代理用 dnsmasq 将 IP 地址分配给各个虚拟机，采用 Pacemaker 可以使 Neutron DHCP 达到高可用。

向 Pacemaker 中添加 Neutron DHCP 代理资源

首先，需要在你的系统中下载资源代理：

```
cd /usr/lib/ocf/resource.d/openstack
wget https://raw.github.com/madkiss/openstack-resource-agents/master/ocf/neutron-dhcp-agent
chmod a+rx neutron-dhcp-agent
```

你需要继续为 Neutron DHCP Agent 资源添加 Pacemaker 配置，用“crm configure”命令连接 Pacemaker 集群，并且添加下列内容：

```
primitive p_neutron-dhcp-agent ocf:openstack:neutron-dhcp-agent \
    params config="/etc/neutron/neutron.conf" \
    plugin_config="/etc/neutron/dhcp_agent.ini" \
    op monitor interval="30s" timeout="30s"
```

- p_neutron-dhcp-agent，一个 Neutron DHCP Agent 管理资源

Crm configure 支持批量输入，所以你可以将上述配置拷贝到 Pacemaker 配置中，根据需要修改一些参数。

完成后，在 crm configure 菜单中键入 commit 来提交修改，Pacemaker 在其中一个节点上启动 Neutron DHCP Agent 服务和它的依赖项。

高可用 Neutron 元数据代理

Neutron Metadata 代理允许 Nova API Metadata 对租户网络上的虚拟机来说是可达的，采用 Pacemaker 可使得 Metadata Agent 是高可用的。

向 Pacemaker 中添加 Neutron Metadata 代理资源

首先，需要在你的系统中下载资源代理：

```
cd /usr/lib/ocf/resource.d/openstack
wget https://raw.github.com/madkiss/openstack-resource-agents/master/ocf/
neutron-metadata-agent
chmod a+rx neutron-metadata-agent
```

你需要继续为 Neutron Metadata Agent 资源添加 Pacemaker 配置，用“crm configure”命令连接 Pacemaker 集群，并且添加下列内容：

```
primitive p_neutron-metadata-agent ocf:openstack:neutron-metadata-agent \
    params config="/etc/neutron/neutron.conf" \
    plugin_config="/etc/neutron/metadata_agent.ini" \
    op monitor interval="30s" timeout="30s"
```

- p_neutron-metadata-agent，一个 Neutron Metadata Agent 管理资源

Crm configure 支持批量输入，所以你可以将上述配置拷贝到 Pacemaker 配置中，根据需要修改一些参数。

完成后，在 crm configure 菜单中键入 commit 来提交修改，Pacemaker 在其中一个节点上启动 Neutron Metadata Agent 服务和它的依赖项。

管理网络资源

你现在需要为管理所有网络资源的一个组添加 Pacemaker 配置，用“crm configure”命令连接 Pacemaker 集群，并且添加下列内容：

```
group g_services_network p_neutron-l3-agent p_neutron-dhcp-agent \
    p_neutron-metadata_agent
```

3.主动/主动式 HA

数据库

第一步就是安装作为集群心脏的数据库，我们讨论高可用，然而，我们谈论的不只是一个数据库，而是多个（用于备份）并且让它们同步的方法。在在这种情况下，我们选择 MySQL 数据库，并且用 Galera 同步多主机，选择它不是唯一的方法，你可以不选择 MySQL。但是，在 Openstack 中选择 MySQL 是很普遍的方法，这里我们选择它。

MySQL Galera

除了安装最新版本的 MySQL，并且将 Galera 加入进去外，你还需要安装 MySQL 的一个补丁 https://launchpad.net/codership-mysql/0.7。安装环境比较敏感，确保阅读 Readme 保证没有错过每一步。

下一步，下载 Galera https://launchpad.net/galera/+download。安装*.rpm 和 *.debs，注意系统中安装的依赖。

完成安装后，需要进行一些配置：

系统级的 my.conf 文件中，确保 mysqld 不是 127.0.0.1，并检查 /etc/mysql/conf.d。你也可以在/etc/my.cnf 中找到这些：

```
[mysqld]
...
!includedir /etc/mysql/conf.d/
...
```

```
#bind-address = 127.0.0.1
```

当添加新节点时，需要用 MySQL 账户进行配置，确保能从其他节点上请求快照，指定 /etc/mysql/conf.d/wsrep.cnf 中的账户信息：

```
wsrep_sst_auth=wsrep_sst:wspass
```

用 root 连接并设置用户权限：

```
$ mysql -e "SET wsrep_on=OFF; GRANT ALL ON *.* TO wsrep_sst@'%' IDENTIFIED BY 'wspass'";
```

你还需要移除用户无 usernames 的账户，防止出现问题:

```
$ mysql -e "SET wsrep_on=OFF; DELETE FROM mysql.user WHERE user='';"
```

设置 MySQL 的一些特定命令选项：

```
query_cache_size=0
binlog_format=ROW
default_storage_engine=innodb
innodb_autoinc_lock_mode=2
innodb_doublewrite=1
```

最后，确保节点可以通过防火墙，检测 iptables:

```
# iptables --insert RH-Firewall-1-INPUT 1 --proto tcp --source <my IP>/24 --destination <my IP>/32 --dport 3306 -j ACCEPT
# iptables --insert RH-Firewall-1-INPUT 1 --proto tcp --source <my IP>/24 --destination <my IP>/32 --dport 4567 -j ACCEPT
```

现在你需要去创建集群。

创建集群

创建集群前，首先启动一个实例，剩下的 MySQL 实例连接到这个集群上，例如，在 10.0.0.10 上输入以下命令：

```
# service mysql start wsrep_cluster_address=gcomm://
```

在其他节点上通过引用这个 IP 地址连接到集群中：

```
# service mysql start wsrep_cluster_address=gcomm://10.0.0.10
```

你需要设置 /etc/mysql/conf.d/wsrep.cnf 中的 wsrep_cluster_address 选项，或者通过命令来设置，例如，连接 MySQL 去设置各个参数：

```
mysql> SET GLOBAL wsrep_cluster_address='<cluster address string>';
mysql> SHOW STATUS LIKE 'wsrep%';
```

检测以下状态：

```
mysql> show status like 'wsrep%';
+----------------------------+--------------------------------------+
| Variable_name              | Value                                |
+----------------------------+--------------------------------------+
| wsrep_local_state_uuid     | 111fc28b-1b05-11e1-0800-e00ec5a7c930 |
| wsrep_protocol_version     | 1                                    |
| wsrep_last_committed       | 0                                    |
| wsrep_replicated           | 0                                    |
| wsrep_replicated_bytes     | 0                                    |
| wsrep_received             | 2                                    |
| wsrep_received_bytes       | 134                                  |
| wsrep_local_commits        | 0                                    |
| wsrep_local_cert_failures  | 0                                    |
| wsrep_local_bf_aborts      | 0                                    |
| wsrep_local_replays        | 0                                    |
| wsrep_local_send_queue     | 0                                    |
| wsrep_local_send_queue_avg | 0.000000                             |
| wsrep_local_recv_queue     | 0                                    |
| wsrep_local_recv_queue_avg | 0.000000                             |
| wsrep_flow_control_paused  | 0.000000                             |
| wsrep_flow_control_sent    | 0                                    |
| wsrep_flow_control_recv    | 0                                    |
| wsrep_cert_deps_distance   | 0.000000                             |
| wsrep_apply_oooe           | 0.000000                             |
| wsrep_apply_oool           | 0.000000                             |
| wsrep_apply_window         | 0.000000                             |
| wsrep_commit_oooe          | 0.000000                             |
```

```
| wsrep_commit_oool                | 0.000000                                 |
| wsrep_commit_window              | 0.000000                                 |
| wsrep_local_state                | 4                                        |
| wsrep_local_state_comment        | Synced (6)                               |
| wsrep_cert_index_size            | 0                                        |
| wsrep_cluster_conf_id            | 1                                        |
| wsrep_cluster_size               | 1                                        |
| wsrep_cluster_state_uuid         | 111fc28b-1b05-11e1-0800-e00ec5a7c930 |
| wsrep_cluster_status             | Primary                                  |
| wsrep_connected                  | ON                                       |
| wsrep_local_index                | 0                                        |
| wsrep_provider_name              | Galera                                   |
| wsrep_provider_vendor            | Codership Oy                             |
| wsrep_provider_version           | 21.1.0(r86)                              |
| wsrep_ready                      | ON                                       |
+--------------------------+------------------------------------+
38 rows in set (0.01 sec)
```

RabbitMQ

RabbitMQ 是被很多 Openstack 服务采用的 AMQP 服务，使得 RabbitMQ 服务高可用包括：

- 安装 RabbitMQ
- 为 HA 队列配置 RabbitMQ
- 配置 Openstack 服务去使用 HA 队列

安装 RabbitMQ

安装 RabbitMQ 2.7.1

Ubuntu/Debian

```
apt-get install rabbitmq-server rabbitmq-plugins
```

Fedora/RHEL

```
yum install erlang
```

配置 RabbitMQ

我们这里考虑建立两个 RabbitMQ 服务器，确保所有节点有相同的 erlang cookie 文件。可以这样做，停掉 RabbitMQ，将一个节点中的 cookie 拷贝到其他所有节点中：

```
scp /var/lib/rabbitmq/.erlang.cookie \
root@rabbit2:/var/lib/rabbitmq/.erlang.cookie
```

然后启动 RabbitMQ，现在，我们创建 HA 集群，第二个 rabbit 上，执行下列命令：

```
rabbitmqctl stop_app
rabbitmqctl cluster rabbit@rabbit1
rabbitmqctl start_app
```

查看集群状态命令：

```
rabbitmqctl cluster_status
Cluster status of node rabbit@rabbit2 ...
[{nodes,[{disc,[rabbit@rabbit1]},{ram,[rabbit@rabbit2]}]},{running_nodes,
[rabbit@rabbit2,rabbit@rabbit1]}]
```

如果集群创建成功，你可以为队列继续创建用户和密码。

3.0 版本 Rabbit 注意：

队列备份不再由 x-ha-policy 控制，我们需要确保所有的节点上的队列是交叉运行：

```
rabbitmqctl set_policy HA '^(?!amq\.).*' '{"ha-mode": "all"}'
```

配置 Openstack 服务使用 RabbitMQ

自从 Grizzly 发布，大部分拥有队列的 Openstack 组件支持这个特性，使用最少两个 RabbitMQ 服务器时需要配置它。

RabbitMQ HA 集群 host:port 对：

```
rabbit_hosts=rabbit1:5672,rabbit2:5672
```

频繁连接 RabbitMQ:

```
rabbit_retry_interval=1
```

重连 RabbitMQ 间隔：

```
rabbit_retry_backoff=2
```

最大连接 RabbitMQ 次数：

```
rabbit_max_retries=0
```

在 RabbitMQ 中使用持久队列：

```
rabbit_durable_queues=false
```

在 RabbitMQ 中使用 H/A 队列：

```
rabbit_ha_queues=true
```

如果你修改了配置并且没有使用 HA 队列，应该重启服务：

```
rabbitmqctl stop_app
rabbitmqctl reset
rabbitmqctl start_app
```

使用 HA 队列的服务： OpenStack Compute, Cinder, OpenStack Networking, Ceilometer。

HAproxy 节点

HAproxy 是提供高可用，负载均衡，TCP 和 HTTP 基础应用的可靠并且快的解决方法，尤其适合做网站建设。

在你的节点上安装 HAproxy，你需要参考官方文档，为了防止单点故障，你需要至少两个节点运行 HAproxy。

下列是 HAproxy 配置文件事例：

```
global
    chroot  /var/lib/haproxy
    daemon
    group  haproxy
    maxconn  4000
    pidfile  /var/run/haproxy.pid
    user  haproxy
defaults
```

```
    log    global
    maxconn    8000
    option    redispatch
    retries    3
    timeout    http-request 10s
    timeout    queue 1m
    timeout    connect 10s
    timeout    client 1m
    timeout    server 1m
    timeout    check 10s
listen dashboard_cluster
    bind <Virtual IP>:443
    balance    source
    option    tcpka
    option    httpchk
    option    tcplog
    server controller1 10.0.0.1:443 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:443 check inter 2000 rise 2 fall 5
listen galera_cluster
    bind <Virtual IP>:3306
    balance    source
    option    httpchk
    server controller1 10.0.0.4:3306 check port 9200 inter 2000 rise 2 fall 5
    server controller2 10.0.0.5:3306 check port 9200 inter 2000 rise 2 fall 5
    server controller3 10.0.0.6:3306 check port 9200 inter 2000 rise 2 fall 5
listen glance_api_cluster
    bind <Virtual IP>:9292
    balance    source
    option    tcpka
    option    httpchk
```

```
    option    tcplog
    server controller1 10.0.0.1:9292 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:9292 check inter 2000 rise 2 fall 5
listen glance_registry_cluster
    bind <Virtual IP>:9191
    balance    source
    option    tcpka
    option    tcplog
    server controller1 10.0.0.1:9191 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:9191 check inter 2000 rise 2 fall 5
listen keystone_admin_cluster
    bind <Virtual IP>:35357
    balance    source
    option    tcpka
    option    httpchk
    option    tcplog
    server controller1 10.0.0.1:35357 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2.42:35357 check inter 2000 rise 2 fall 5
listen keystone_public_internal_cluster
    bind <Virtual IP>:5000
    balance    source
    option    tcpka
    option    httpchk
    option    tcplog
    server controller1 10.0.0.1:5000 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:5000 check inter 2000 rise 2 fall 5
listen nova_ec2_api_cluster
    bind <Virtual IP>:8773
    balance    source
    option    tcpka
```

```
    option    tcplog
    server controller1 10.0.0.1:8773 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:8773 check inter 2000 rise 2 fall 5
listen nova_compute_api_cluster
    bind <Virtual IP>:8774
    balance    source
    option    tcpka
    option    httpchk
    option    tcplog
    server controller1 10.0.0.1:8774 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:8774 check inter 2000 rise 2 fall 5
listen nova_metadata_api_cluster
    bind <Virtual IP>:8775
    balance    source
    option    tcpka
    option    tcplog
    server controller1 10.0.0.1:8775 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:8775 check inter 2000 rise 2 fall 5
listen cinder_api_cluster
    bind <Virtual IP>:8776
    balance    source
    option    tcpka
    option    httpchk
    option    tcplog
    server controller1 10.0.0.1:8776 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:8776 check inter 2000 rise 2 fall 5
listen ceilometer_api_cluster
    bind <Virtual IP>:8777
    balance    source
    option    tcpka
```

```
    option    httpchk
    option    tcplog
    server controller1 10.0.0.1:8774 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:8774 check inter 2000 rise 2 fall 5
listen spice_cluster
    bind <Virtual IP>:6082
    balance    source
    option    tcpka
    option    tcplog
    server controller1 10.0.0.1:6080 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:6080 check inter 2000 rise 2 fall 5
listen neutron_api_cluster
    bind <Virtual IP>:9696
    balance    source
    option    tcpka
    option    httpchk
    option    tcplog
    server controller1 10.0.0.1:9696 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:9696 check inter 2000 rise 2 fall 5
listen swift_proxy_cluster
    bind <Virtual IP>:8080
    balance    source
    option    tcplog
    option    tcpka
    server controller1 10.0.0.1:8080 check inter 2000 rise 2 fall 5
    server controller2 10.0.0.2:8080 check inter 2000 rise 2 fall 5
```

都配置后，你需要重启 HAproxy

Openstack 控制节点

Openstack 控制节点包括：

- 所有 OpenstackAPI 服务
- 所有 Openstack 调度
- Memcached 服务

运行 Openstack API&调度

API 服务

所有的 Openstack 工程都有一个管理它所有云上面的资源的 API 服务。在主动/主动模式下，大部分安装规模为两个节点，并且使用负载均衡和虚拟 ip。

配置高可用云和大规模 API 服务，我们需要确保：

- 配置 Openstack 认证端点时使用虚拟 ip
- 所有的配置依靠虚拟 ip

监视器检查很简单，因为它只是建立一个 TCP 连接的 API port。与使用 Corosync 和资源代理的主动/被动模式相比较，我们不检查该服务是否实际运行。这就是为什么所有的 OpenStack API 应该被另一个工具（例如 Nagios 的），其目的是检测故障的云基础结构进行监测。

调度

Openstack 调度者被用来决定怎么分发计算，网络，卷请求，大部分将 RabbitMQ 作为一个消息系统的步骤已经作为指导文档，这些服务被连接后去管理后端，并且可以扩展：

- nova-scheduler
- nova-conductor
- cinder-scheduler
- neutron-server
- ceilometer-collector
- heat-engine

请参考 RabbitMQ 部分去配置拥有多个消息服务器的服务。

Memcached

大部分 Openstack 服务用一个应用程序去提供数据的短暂和长久的保存（比如 tokens），Memcached 就是其中一个不需要特殊操作的可扩展的方法。

安装配置 Memcached 请参考官方文档。

内存管理由 Oslo-incubator 管理，在所有项目中使用相同的多个内存管理服务器。比如两个主机：

```
memcached_servers = controller1:11211,controller2:11211
```

默认情况下，controller1 会处理缓存服务，但是如果产生故障，controller2 会替代它运行，更多细节见 Openstack 计算节点安装手册。

Openstack 网络节点

Openstack 网络节点包括：

- Neutron DHCP Agent
- Neutron L2 Agent
- Neutron L3 Agent
- Neutron Metadata Agent
- Neutron LBaaS Agent

运行 Neutron DHCP Agent

自从 Grizzly 发布后，Openstack 网络服务就有一个可以在节点上交叉运行多个代理的调度者。并且，DHCP 代理在本地高可用，更多细节见 OpenStack Configuration Reference.

运行 Neutron L3 Agent

自从 Grizzly 发布后，Neutron L3 Agent 变为可扩展的服务，由于调度者可以使多个节点之间的虚拟路由交叉分配。但是它不是让路由高可用的根本特性，现在，主动/被动模式使用 Pacemaker 支持 Neutron L3 Agent 的故障恢复，见主动/被动章节。

运行 Neutron Metadata Agent

没有原有特性使得 Metadata 服务高可用，现在，主动/被动模式使用 Pacemaker 支持 Neutron Metadata Agent 的故障恢复，见主动/被动章节。