**[理解 OpenStack 高可用（HA）（1）：OpenStack 高可用和灾备方案 [OpenStack HA and DR]](http://www.cnblogs.com/sammyliu/p/4741967.html)**

本系列会分析OpenStack 的高可用性（HA）概念和解决方案：

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

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

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

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

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

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

**1. 基础知识**

**1.1 高可用 （High Availability，简称 HA）**

    高可用性是指提供在本地系统单个组件故障情况下，能继续访问应用的能力，无论这个故障是业务流程、物理设施、IT软/硬件的故障。最好的可用性， 就是你的一台机器宕机了，但是使用你的服务的用户完全感觉不到。你的机器宕机了，在该机器上运行的服务肯定得做故障切换（failover），切换有两个维度的成本：RTO （Recovery Time Objective）和 RPO（Recovery Point Objective）。RTO 是服务恢复的时间，最佳的情况是 0，这意味着服务立即恢复；最坏是无穷大意味着服务永远恢复不了；RPO 是切换时向前恢复的数据的时间长度，0 意味着使用同步的数据，大于 0 意味着有数据丢失，比如 ” RPO = 1 天“ 意味着恢复时使用一天前的数据，那么一天之内的数据就丢失了。因此，恢复的最佳结果是 RTO = RPO = 0，但是这个太理想，或者要实现的话成本太高，全球估计 Visa 等少数几个公司能实现，或者几乎实现。

    对 HA 来说，往往使用共享存储，这样的话，RPO =0 ；同时往往使用 Active/Active （双活集群） HA 模式来使得 RTO 几乎0，如果使用 Active/Passive 模式的 HA 的话，则需要将 RTO 减少到最小限度。HA 的计算公式是[ 1 - (宕机时间)/（宕机时间 + 运行时间）]，我们常常用几个 9 表示可用性：

* 2 个9：99% = 1% \* 365 = 3.65 \* 24 小时/年 = 87.6 小时/年的宕机时间
* 4 个9: 99.99% = 0.01% \* 365 \* 24 \* 60 = 52.56 分钟/年
* 5 个9：99.999% = 0.001% \* 365 = 5.265 分钟/年的宕机时间，也就意味着每次停机时间在一到两分钟。
* 11 个 9：几乎就是几年才宕机几分钟。 据说 AWS S3 的设计高可用性就是 11 个 9。

**1.1.1 服务的分类**

HA 将服务分为两类：

* 有状态服务：后续对服务的请求依赖于之前对服务的请求。
* 无状态服务：对服务的请求之间没有依赖关系，是完全独立的。

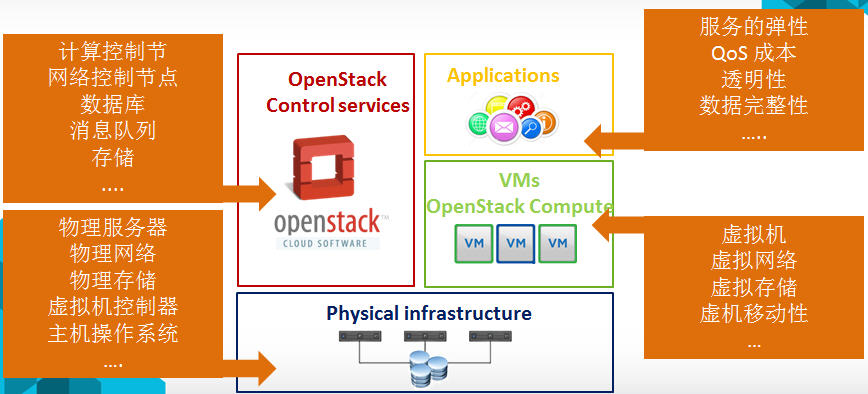
**1.1.2 HA 的种类**

HA 需要使用冗余的服务器组成集群来运行负载，包括应用和服务。这种冗余性也可以将 HA 分为两类：

* Active/Passive HA：集群只包括两个节点简称主备。在这种配置下，系统采用主和备用机器来提供服务，系统只在主设备上提供服务。在主设备故障时，备设备上的服务被启动来替代主设备提供的服务。典型地，可以采用 CRM 软件比如 Pacemaker 来控制主备设备之间的切换，并提供一个虚机 IP 来提供服务。
* Active/Active HA：集群只包括两个节点时简称双活，包括多节点时成为多主（Multi-master）。在这种配置下，系统在集群内所有服务器上运行同样的负载。以数据库为例，对一个实例的更新，会被同步到所有实例上。这种配置下往往采用负载均衡软件比如 HAProxy 来提供服务的虚拟 IP。

**1.1.3 云环境的 HA**

    云环境包括一个广泛的系统，包括硬件基础设施、IaaS层、虚机和应用。以 OpenStack 云为例：



云环境的 HA 将包括：

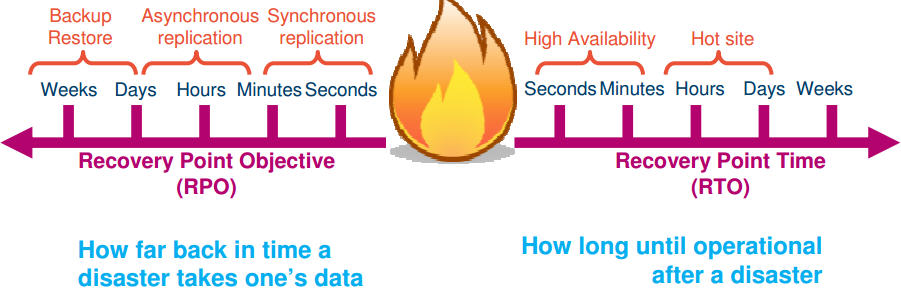
* 应用的 HA
* 虚机的 HA
* 云控制服务的 HA
* 物理IT层：包括网络设备比如交换机和路由器，存储设备等
* 基础设施，比如电力、空调和防火设施等

本文的重点是讨论 OpenStack 作为 IaaS 的 HA。

**1.2 灾难恢复 （Disaster Recovery）**

几个概念：

* 灾难（Disaster）是由于人为或自然的原因，造成一个数据中心内的信息系统运行严重故障或瘫痪，使信息系统支持的业务功能停顿或服务水平不可接受、达到特定的时间的突发性事件，通常导致信息系统需要切换到备用场地运行。
* 灾难恢复（Diaster Recovery）是指当灾难破坏生产中心时在不同地点的数据中心内恢复数据、应用或者业务的能力。
* 容灾是指，除了生产站点以外，用户另外建立的冗余站点，当灾难发生，生产站点受到破坏时，冗余站点可以接管用户正常的业务，达到业务不间断的目的。为了达到更高的可用性，许多用户甚至建立多个冗余站点。
* 衡量容灾系统有两个主要指标：RPO（Recovery Point Objective）和 RTO（Recovery Time Object），其中 RPO代表 了当灾难发生时允许丢失的数据量，而 RTO 则代表了系统恢复的时间。RPO 与 RTO 越小，系统的可用性就越高，当然用户需要的投资也越大。



    大体上讲，容灾可以分为3个级别：数据级别、应用级别以及业务级别。

|  |  |  |  |
| --- | --- | --- | --- |
| 级别 | 定义 | RTO | CTO |
| 数据级 | 指通过建立异地容灾中心，做数据的远程备份，在灾难发生之后要确保原有的数据不会丢失或者遭到破坏。但在数据级容灾这个级别，发生灾难时应用是会中断的。  在数据级容灾方式下，所建立的异地容灾中心可以简单地把它理解成一个远程的数据备份中心。数据级容灾的恢复时间比较长，但是相比其他容灾级别来讲它的费用比较低，而且构建实施也相对简单。  但是，“数据源是一切关键性业务系统的生命源泉”，因此数据级容灾必不可少。 | RTO 最长(若干天) ，因为灾难发生时，需要重新部署机器，利用备份数据恢复业务。 | 最低 |
| 应用级 | 在数据级容灾的基础之上，在备份站点同样构建一套相同的应用系统，通过同步或异步复制技术，这样可以保证关键应用在允许的时间范围内恢复运行，尽可能减少灾难带来的损失，让用户基本感受不到灾难的发生，这样就使系统所提供的服务是完整的、可靠的和安全的。 | RTO 中等（若干小时） | 中等。异地可以搭建一样的系统，或者小些的系统。 |
| 业务级 | 全业务的灾备，除了必要的 IT 相关技术，还要求具备全部的基础设施。其大部分内容是非IT系统（如电话、办公地点等），当大灾难发生后，原有的办公场所都会受到破坏，除了数据和应用的恢复，更需要一个备份的工作场所能够正常的开展业务。 | RTO 最小（若干分钟或者秒） | 最高 |

**1.3 HA 和 DR 的关系**

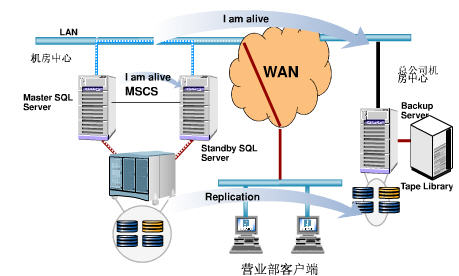
    两者相互关联，互相补充，互有交叉，同时又有显著的区别：

* HA 往往指本地的高可用系统，表示在多个服务器运行一个或多种应用的情况下，应确保任意服务器出现任何故障时，其运行的应用不能中断，应用程序和系统应能迅速切换到其它服务器上运行，即本地系统集群和热备份。HA 往往是用共享存储，因此往往不会有数据丢失（RPO = 0），更多的是切换时间长度考虑即 RTO。
* DR 是指异地（同城或者异地）的高可用系统，表示在灾害发生时，数据、应用以及业务的恢复能力。异地灾备的数据灾备部分是使用数据复制，根据使用的不同数据复制技术（同步、异步、Strectched Cluster 等），数据往往有损失导致 RPO >0；而异地的应用切换往往需要更长的时间，这样 RT0 >0。 因此，需要结合特定的业务需求，来定制所需要的 RTO 和 RPO，以实现最优的 CTO。

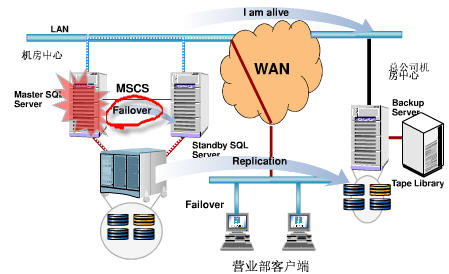
也可以从别的角度上看待两者的区别：

* 从故障角度，HA 主要处理单组件的故障导致负载在集群内的服务器之间的切换，DR 则是应对大规模的故障导致负载在数据中心之间做切换。
* 从网络角度，LAN 尺度的任务是 HA 的范畴，WAN 尺度的任务是 DR 的范围。
* 从云的角度看，HA 是一个云环境内保障业务持续性的机制，DR 是多个云环境间保障业务持续性的机制。
* 从目标角度，HA 主要是保证业务高可用，DR 是保证数据可靠的基础上的业务可用。

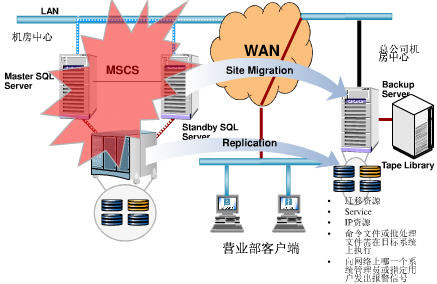
一个异地容灾系统，往往包括本地的 HA 集群和异地的 DR 数据中心。一个示例如下（来源：[百度文库](http://wenku.baidu.com/view/98871bbd960590c69ec376ae.html)）：



Master SQL Server 发生故障时，切换到 Standby SQL Server，继续提供数据库服务：



在主机房中心发生灾难时，切换到备份机房（总公司机房中心）上，恢复应用和服务：



**2. OpenStack HA**

OpenStack 部署环境中，各节点可以分为几类：

* Cloud Controller Node （云控制节点）：安装各种 API 服务和内部工作组件（worker process）。同时，往往将共享的 DB 和 MQ 安装在该节点上。
* Neutron Controller Node （网络控制节点）：安装 Neutron L3 Agent，L2 Agent，LBaas，VPNaas，FWaas，Metadata Agent 等 Neutron 组件。
* Storage Controller Node （存储控制节点）：安装 Cinder volume 以及 Swift 组件。
* Compute node （计算节点）：安装 Nova-compute 和 Neutron L2 Agent，在该节点上创建虚机。

要实现 OpenStack HA，一个最基本的要求是这些节点都是冗余的。根据每个节点上部署的软件特点和要求，每个节点可以采用不同的 HA 模式。但是，选择 HA 模式有个基本的原则：

* 能 A/A 尽量 A/A，不能的话则 A/P （RedHat 认为 A/P HA 是 No HA）
* 有原生（内在实现的）HA方案尽量选用原生方案，没有的话则使用额外的HA 软件比如 Pacemaker 等
* 需要考虑负载均衡
* 方案尽可能简单，不要太复杂

OpenStack 官方认为，在满足其 HA 要求的情况下，可以实现 IaaS 的 99.99% HA，但是，这不包括单个客户机的 HA。

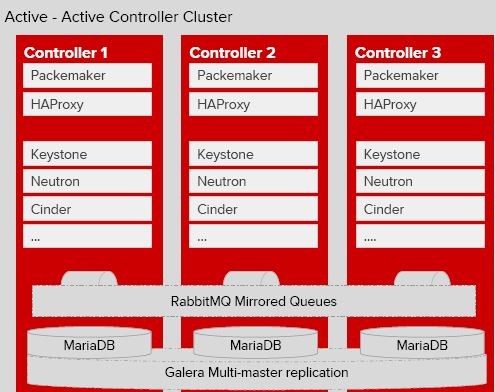
**2.1 云控制节点 HA**

    云控制节点上运行的服务中，API 服务和内部工作组件都是无状态的，因此很容易就可以实现 A/A HA；这样就要求 Mysql 和 RabbitMQ 也实现 A/A HA，而它们各自都有 A/A 方案。但是，Mysql Gelera 方案要求三台服务器。如果只想用两台服务器的话，则只能实现 A/P HA，或者引入一个 Arbiter 来做 A/A HA。

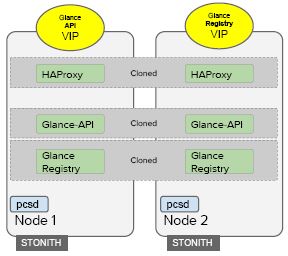
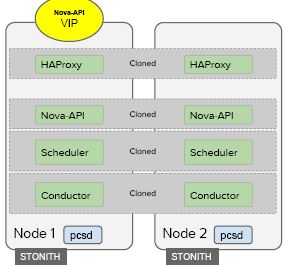
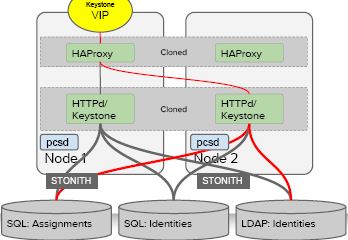
**2.1.1 云控制节点的 A/A HA 方案**

该方案至少需要三台服务器。以 RDO 提供的案例为例，它由三台机器搭建成一个 Pacemaker A/A集群，在该集群的每个节点上运行：

* API 服务：包括 \*-api, neutron-server，glance-registry, nova-novncproxy，keystone，httpd 等。由 HAProxy 提供负载均衡，将请求按照一定的算法转到某个节点上的 API 服务。由  Pacemaker 提供 VIP。
* 内部组件：包括 \*-scheduler，nova-conductor，nova-cert 等。它们都是无状态的，因此可以在多个节点上部署，它们会使用 HA 的 MQ 和 DB。
* RabbitMQ：跨三个节点部署 RabbitMQ 集群和镜像消息队列。可以使用 HAProxy 提供负载均衡，或者将 RabbitMQ host list 配置给 OpenStack 组件（使用 rabbit\_hosts 和 rabbit\_ha\_queues 配置项）。
* MariaDB：跨三个阶段部署 Gelera MariaDB 多主复制集群。由 HAProxy 提供负载均衡。
* HAProxy：向 API，RabbitMQ 和 MariaDB 多活服务提供负载均衡，其自身由 Pacemaker 实现 A/P HA，提供 VIP，某一时刻只由一个HAProxy提供服务。在部署中，也可以部署单独的 HAProxy 集群。
* Memcached：它原生支持 A/A，只需要在 OpenStack 中配置它所有节点的名称即可，比如，memcached\_servers = controller1:11211,controller2:11211。当 controller1:11211 失效时，OpenStack 组件会自动使用controller2:11211。



从每个 API 服务来看：

关于共享 DB 的几个说明 （主要来自 [这篇文章](http://www.joinfu.com/2015/01/understanding-reservations-concurrency-locking-in-nova/)）：

（1）根据该文章中的一个调查，被调查的 220 多个用户中，200 个在用 Mysql Galera，20 多个在用单 Mysql，只有一个用 PostgreSQL。

（2）以 Nova 为例，Mysql 使用 [***Write-intent locks***](http://en.wikipedia.org/wiki/Multiple_granularity_locking)机制来保证多个连接同时访问数据库中的同一条记录时的互斥。以给新建虚机分配 IP 地址为例，该锁机制保证了一个 IP 不会分给两个用户。

（3）使用 Mysql Galera 时，所有节点都是 Master 节点，都可以接受服务，但是这里有个问题，Mysql Galera 不会复制 Write-intent locks。两个用户可以在不同节点上获取到同一条记录，但是只有一个能够修改成功，另一个会得到一个 Deadlock 错误。对于这种情况，Nova 使用 retry\_on\_deadlock 机制来重试，比如@oslo\_db\_api.wrap\_db\_retry(max\_retries=5, retry\_on\_deadlock=True)。默认都是重试 5 次。但是，这种机制效率不高，文章作者提供了一种新的机制。

该 HA 方案具有以下优点：

* 多主，零切换，方便地实现负载均衡
* 将 API 服务和 DB, MQ 服务无缝整合在一起

由于这些优点，该方案被大量采用。具体配置请参考 [OpenStack High Availability Guide](http://docs.openstack.org/high-availability-guide/high-availability-guide.pdf)。

**2.1.2 云控制节点的 A/P HA方案**

    需要的话，可以使用 Pacemaker + Corosync 搭建两节点集群实现 A/P HA 方案。由主服务器实际提供服务，在其故障时由 Pacemaker 将服务切换到被服务器。OpenStack 给其组件提供了各种Pacemaker RA。对 Mysql 和 RabbitMQ 来说，可以使用 Pacemaker + Corosync + DRBD 实现 A/P HA。具体请参考 [理解 OpenStack 高可用（HA）（4）：RabbitMQ 和 Mysql HA](http://www.cnblogs.com/sammyliu/p/4730517.html)。具体配置请参考 [OpenStack High Availability Guide](http://docs.openstack.org/high-availability-guide/high-availability-guide.pdf)。

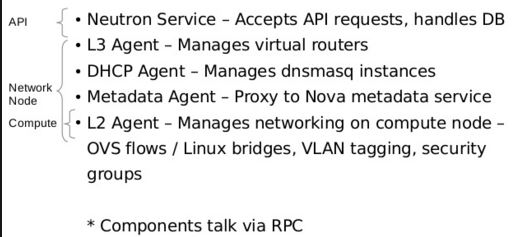
  该 HA 方案的问题是：

* 主备切换需要较长的时间
* 只有主提供服务，在使用两个节点的情况下不能做负载均衡
* DRBD 脑裂会导致数据丢失的风险。A/P 模式的 Mysql 的可靠性没有 Mysql Galera 高。

  因此，可以看到实际部署中，这种方案用得较少，只看到 Oracel 在使用这种方案。

**2.2 Neutron HA**

Neutron 包括很多的组件，比如 L3 Agent，L2 Agent，LBaas，VPNaas，FWaas，Metadata Agent 等 Neutron 组件，其中部分组件提供了原生的HA 支持。这些组件之间的联系和区别：



**2.2.1 原生 HA 方案**

Neutron 提供了多种原生的 HA 方案：

（1）L2 Agent HA： L2 agent 只在所在的网络或者计算节点上提供服务，因此它是不需要HA的。

（2）L3 Agent HA

    L3 Agent 比较特殊，因为它是所有 openstack （core）services 中唯一一个有状态的，因此，不能使用传统的在多个节点上部署多个实例使用LB来做HA。Neutron 本身的调度器（scheduler）支持在多个网络节点上部署多个L3 Agent，但是，由 L3 Agent 管理的 Virtual Router 自身需要有HA的实现。它的HA的Neutron 原生实现包括如下几种方式：

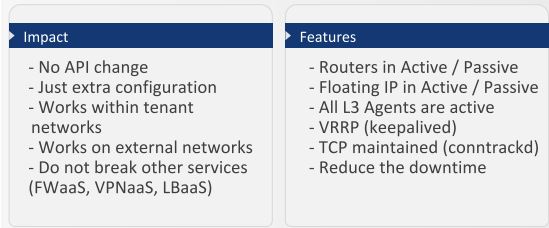
（a）Juno 中引入的 Automatic L3 Agent Failover （当 VR 所在的 L3 Agent 失效的时候，Neutron 自动将它 failover 到其它某个 L3 Agent 上）

    该方案增加了一个配置项 allow\_automatic\_l3agent\_failover。当它的值为 True 时，L3 plugin 去周期性地检查所有有管理 Virtual Router 的 L3 Agent 的状态。如果某 L3 Agent 死了，受它管理的 Router 会重新被 schedule 到别的 L3 Agent 上。 Neutron L3 Plugin 通过判断该 L3 Agent 是否在规定时间（agent\_down\_time）内有发回心跳消息来判断它是否活着。存在多种 L3 Agent 未能及时上报心跳但是 router 依然在转发网络包的可能。因此这种实现可能会存在 L3 Agent 被认为死了但是其 router namespace 依然在转发网络包和响应 ARP 请求而导致的问题。如果网络后端不阻止死掉了的 agent 使用 route 的 IP 地址，那新老 namespace 就可能存在冲突。这种冲突不会断掉 E-W 网络，因为新老 namespace 中的一个都可以承担无状态网络包的转发任务。然后，南-北网络可能会受影响，因为 NAT 只存在于一个router 上。而且，reschedule 后，浮动 IP 也会无法工作，因为它们与 router 的 外部端口的绑定关系不会被设置到新的router 上。

   这种方案要求使用多个网络控制节点，每个节点上运行一个 L3 Agent。在某个 Agent 死了时，Router 会被部署到别的 Agent 上。这种方案，除了上述的问题外，切换时间过长是其主要问题。

（b）Juno 中引入的 VRRP （Virtual Router Redundancy Protocol）方案 （由 VRRP/Keepalived 控制 VR 的 VIP 和 VMAC 的 failover）

    该方案使用多余一个的网络控制节点，提供 A/P HA。具体请参考我的另一篇文章 [理解 OpenStack 高可用（HA）（2）：Neutron L3 Agent HA 之 虚拟路由冗余协议（VRRP）](http://www.cnblogs.com/sammyliu/p/4692081.html)。其主要特点为：

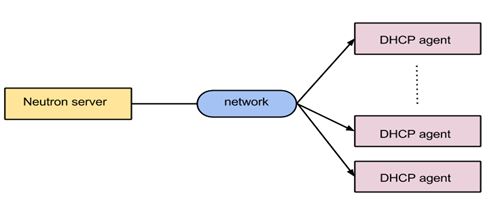


（c）Juno 引入的 DVR

    该方案将 NAT 和 L3 Agent 部署到虚机所在的计算节点，在网络控制节点上只部署 DHCP 和 SNAT。该方案解决了 L3 Agent 和 Metadata Agent 的 H/A 问题。目前，将 DHCP Agent 改成分布式，VPNaas 以及 FWaas 的修改工作已经在进行中。用户需要使用第三方软件提供 SNAT 的 HA 方案。可以参考 [理解 OpenStack 高可用（HA）（3）：Neutron 分布式虚拟路由（Neutron Distributed Virtual Routing）](http://www.cnblogs.com/sammyliu/p/4713562.html)。

（3）DHCP Agent 的 HA

    DHCP 协议自身就支持多个 DHCP 服务器，因此，只需要在多个网卡控制节点上，通过修改配置，为每个租户网络创建多个 DHCP Agent，就能实现 DHCP 的 HA 了。



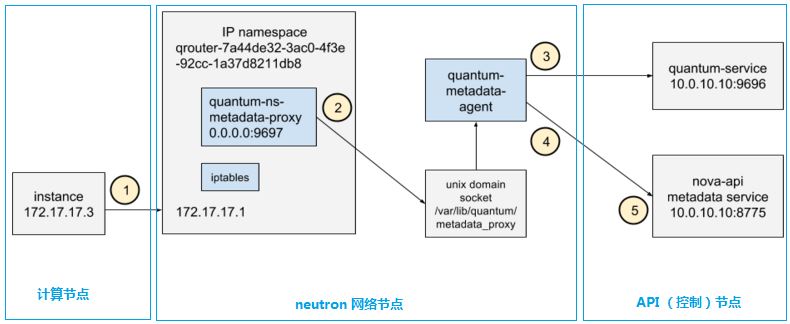
（4）Metadata agent 和 proxy 的 HA

跟 metadata service 相关的组件包括：

* neutron-ns-metadata-proxy：作为一个独立的进程运行在 master virtual router 的 network namespace 中。它接受由 qrouter 通过 iptables 控制转交的 instance 访问 metadata service 的 request。
* neutron-metadata-agent：Neutorn 的组件之一，运行在Neutorn 网络节点上，通过本地 socket 和 neutron-ns-metadata-proxy 进程通信，其配置文件是 /etc/neutron/metadata\_agent.ini；它会通过 http(s) 和 Nova metadata service 通信；它通过 RPC 和 neutron-server 通信。你还可以通过配置 metadata\_workers 的值来运行多个独立的进程。
* nova metadata api：这个和 nova api 类似，是 nova 的 API 的一部分，通常使用的端口是 8775。它接收neutron-metadata-agent 的request。

从 HA 角度来讲：

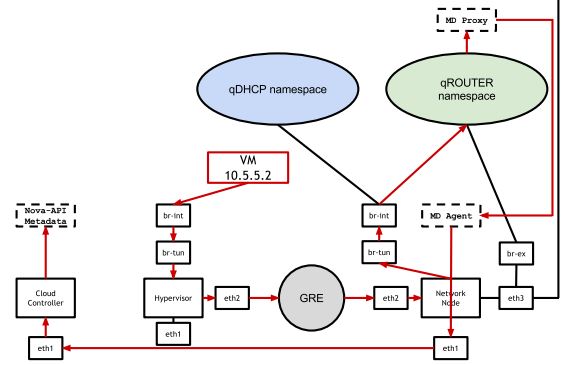
* neutron-ns-metadata-proxy 的 HA 不需要单独考虑，因为它受 Virtual router 控制。
* neutron-metadata-agent：需要和 neutron-ns-metadata-proxy 通过soket 通信，因此，简单地，可以在所有 neutron network 节点上都运行该 agent，只有 virtual router 所在的L3 Agent 上的 neutron-metadata-agent 才起作用，别的都standby。你可以在多个网络节点上启用该服务。
* nova metadata api：同 nova api 一样是无状态服务，可以部署在那个阶段上，使用 HAProxy 做 A/A HA。



（<http://techbackground.blogspot.com/2013/06/metadata-via-quantum-router.html>）

（注意，因为虚机在启动过程中需要访问 qrouter，这也就是说，要求虚机所在的子网必须已经添加到了一个 Virtual router 上，否则，它是无法通过 qrouter 走的，除非走 qdhcp）

或者更详细地看出完整的路径（图中红色线条，从VM开始，到 NOVA-API Metadata 结束）：



（<https://www.hastexo.com/system/files/neutron_packet_flows-notes-handout.pdf>）

（5）LBaas Agent HA

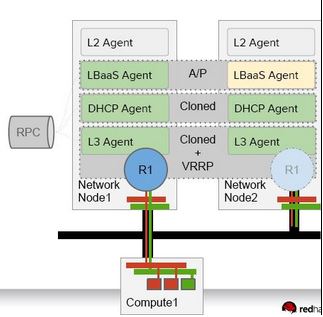
    目前 Neutron LBaaS 代理服务是无法通过其自带的 HAProxy 插件 实现高可用的。实现 HAProxy 高可用常见的方案是使用 VRRP (Virtual Router Redundancy Protocol ，虚拟路由冗余协议)，不过 LBaaS HAProxy 插件目前还不支持该协议。因此，只能使用 Pacemaker + 共享存储（放置 /var/lib/neutron/lbaas/ 目录） 的方式来部署 A/P 方式的 LBaas Agent HA，具体请参考 [这篇文章](http://blog.anynines.com/openstack-neutron-lbaas/) 中描述的方法。

**2.2.2 使用 Pacemaker 实现 A/P HA**

  使用 Pacemaker + Corosync 搭建两节点（或者多节点） A/P 集群。在主节点上，由 Pacemaker 启动 Neutron 的各种服务。

**2.2.3 小结**

    从上面可以看出，除了 DHCP Agent 天生就通过配置可以实现 A/A HA 以及 L3 HA 以外，其它的组件的 HA 都是 A/P 的，而且实现的技术可以是原生的，也可以使用 Pacemaker，也可以结合起来使用。比如 RDO 的方案：

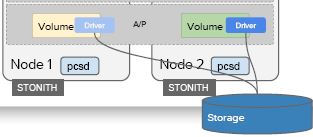


**2.3 存储控制节点 HA**

 这里只讨论 cinder-volume。

（1）在使用非共享存储时，cinder-volume 进程受 Pacemaker 监控，在其停止的时候重启。这种方案下，存储控制节点宕机的话，上面的所有卷都会损失掉。因此，在生产环境中，必须使用下一种方案。

（2）在使用共享存储时，考虑到目前代码中存在的资源竞争（参考[这里](https://bugzilla.redhat.com/show_bug.cgi?id=1193229)），该服务只能实现为 A/P HA 方式，也就是说在某个时刻，只有主节点上的 cinder-volume 在运行。RedHat 这个 [ticket](https://bugzilla.redhat.com/show_bug.cgi?id=1193229) 中有具体的分析。目前，cinder-volume 还没有内在的 HA 实现，只能借助第三方软件比如 Pacemaker。A/A 的实现在 Liberty 中正在进行，请 [参见](https://github.com/Akrog/test-cinder-atomic-states) 和 [这个](https://etherpad.openstack.org/p/cinder-kilo-stabilisation-work)。



**2.4 计算节点和虚机 HA**

    在测试环境中，我们常常将虚机创建在本地磁盘上，那么，在机器宕机的话，这些虚机将永远也回不来了。因此，在生产环境中，需要将虚机部署在 cinder-volume 或者共享的存储比如 RDB 或者 NFS 上。这样的话，在虚机损坏时，可以从共享存储上将其恢复（使用 nova evacuate 功能）。 使用 Pacemaker 部署 A/P 方案（类似 2.3 中 cinder-volume A/P HA）的话，生产环境中计算节点的数据往往远远超过 Corosync 集群中节点数目的限制。

  业界有几个解决方案：

（1）Controller 节点通过管理网 Ping 所有 Compute 节点，Controller 节点检查nova service-list，对出问题的节点 Evacuate

特征：太简单粗暴，容易引起误杀和数据损坏

（2）Pacemaker-remote： 突破Corosync的集群规模限制，

特征：启用多个心跳网时，处理策略单一，引起用户业务不必要的中断

（3）集中式检查



（4）分布式健康检查



（以上资料来自 基于Fuel的超融合一体机 by 周征晟 / 2015-06-27）

OpenStack 的各提供商中，就该需求，RadHat 使用的是上述的第二种方案，具体方案在 [计算节点HA 方案](http://www.redhat.com/archives/rdo-list/2015-April/msg00008.html)：

   部署方式如下：

* 使用 Pacemaker 集群作为控制平面
* 将计算节点做为 Partial members 加入到 Pacemaker 集群中，受其管理和监控。这时候，其数目不受 Corosync 集群内节点总数的限制。

   HA 实现细节：

* Pacemaker 通过 pacemaker\_remote 按照顺序（neutron-ovs-agent -> ceilometer-compute -> nova-compute) 来启动计算节点上的各种服务。前面的服务启动失败，后面的服务不会被启动。
* Pacemaker 监控和每个计算节点上的 pacemaker\_remote 的连接，来检查该节点是否处于活动状态。发现它不可以连接的话，启动恢复（recovery）过程。
* Pacemaker 监控每个服务的状态，如果状态失效，该服务会被重启。重启失败则触发防护行为（fencing action）；当所有服务都被启动后，虚机的网络会被恢复，因此，网络只会短时间受影响。

  当一个节点失效时，恢复（recovery）过程会被触发，Pacemaker 会依次：

1. 运行 'nova service-disable'
2. 将该节点关机
3. 等待 nova 发现该节点失效了
4. 将该节点开机
5. 如果节点启动成功，执行 'nova service-enable'
6. 如果节点启动失败，则执行 ‘nova evacuate’ 把该节点上的虚机移到别的可用计算节点上。

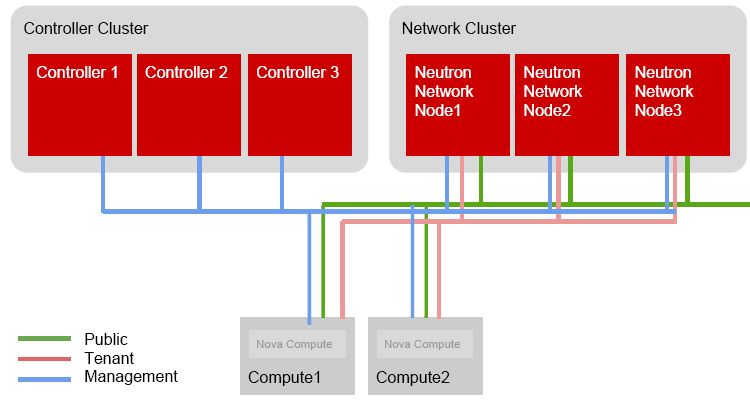
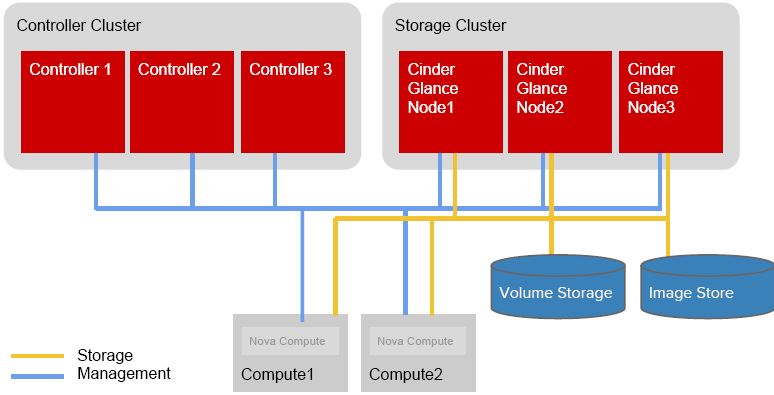
  其中：

* 步骤（1）和 （5）是可选的，其主要目的是防止 nova-scheduler 将新的虚机分配到该节点。
* 步骤（2）保证机器肯定会关机。
* 步骤（3）中目前 nova 需要等待一段较长的超时时间才能判断节点 down 了。Liberty 中有个 [Blueprint](https://blueprints.launchpad.net/python-novaclient/+spec/support-force-down-service) 来添加一个 Nova API 将节点状态直接设置为 down。

  其余一些前提条件：

* 虚机必须部署在 cinder-volume 或者共享的临时存储比如 RBD 或者 NFS 上，这样虚机 evaculation 将不会造成数据丢失。
* 如果虚机不使用共享存储，则必须周期性地创建虚机的快照并保存到 Glance 中。在虚机损坏后，可以从 Glance 快照上恢复。但是，这可能会导致状态或者数据丢失。
* 控制和计算节点需要安装 RHEL 7.1+
* 计算节点需要有防护机制，比如 IPMI，硬件狗 等

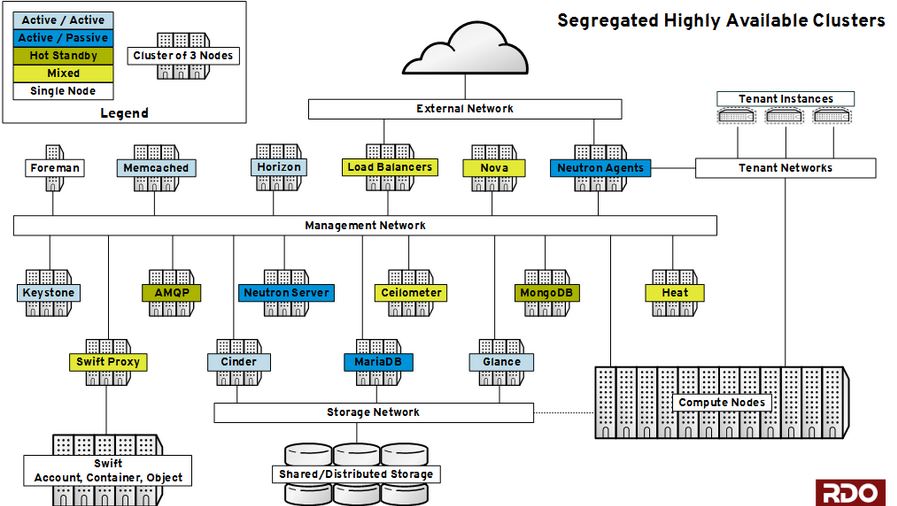
**小结： OpenStack 云/网络/存储 控制节点 HA 集群**



**3. 部分 OpenStack 方案提供者的 HA 方案**

**3.1 RDO HA**

[这里](https://github.com/beekhof/osp-ha-deploy) 有完整的 RDO HA 部署方案：



该配置最少需要五台机器：

1. 一台（物理或者虚拟）服务器部署 nfs server，dhcp，dns
2. 一台物理服务器来作为计算节点
3. 三台物理服务器组成 pacemaker 集群，创建多个虚机，安装各种应用

 特征：

* 每个集群使用三个节点，全部采用 A/A 模式，除了 cinder-volume 和 LBaas。RedHat 不认为 A/P 模式是真正的 HA。
* 提供使用 Pacemaker 或者 Keepalived 两套方案。
* 将 API 和内部无状态组件按功能组分布到各个专有集群，而不是放在一个集群上。
* Cinder 这里标识为 A/A HA，但是不包括 cinder-volume。
* 计算节点 HA 使用 2.4 部分描述的 HA 方式。

| **Service** | **Process** | **Mode** | **HA stragegy** |
| --- | --- | --- | --- |
| Support services | MariaDB - Galera | A/A | HAProxy / app cluster |
| Support services | RabbitMQ | A/A | App cluster / service config |
| Support services | HAProxy | A/A | Keepalived |
| Support services | MongoDB | A/A | App cluster （ceilometer 和 heat 会使用） |
| Support services | Memcached | A/A | Service configuration |
| Keystone | openstack-keystone | A/A | HAProxy |
| Glance | openstack-glance-api | A/A | HAProxy |
| Glance | openstack-glance-registry | A/A | HAProxy （向 glance-api 提供 REST API） |
| Nova | openstack-nova-api | A/A | HAProxy |
| Nova | openstack-nova-cert | A/A |  |
| Nova | openstack-nova-compute | A/A | 参见 2.4 部分描述 |
| Nova | openstack-nova-scheduler | A/A |  |
| Nova | openstack-nova-conductor | A/A |  |
| Nova | openstack-nova-novncproxy | A/A | HAProxy |
| Cinder | openstack-cinder-api | A/A | HAProxy |
| Cinder | openstack-cinder-scheduler | A/A |  |
| Cinder | openstack-cinder-volume | **A/P** | No HA （RH 不把 A/P HA 当作HA！）。参考[这里](https://bugzilla.redhat.com/show_bug.cgi?id=1193229) |
| Cinder | openstack-cinder-backup | A/A |  |
| Neutron | neutron-server | A/A | HAProxy |
| Neutron | neutron-dhcp-agent | A/A | Multiple DHCP agents |
| Neutron | neutron-l3-agent | A/A | L3 HA |
| Neutron | neutron-metadata-agent | A/A |  |
| Neutron | neutron-lbaas-agent | **A/P** | 目前的设计不允许 A/A |
| Neutron | neutron-openvswitch-agent | A/A |  |
| Neutron | neutron-metering-agent | A/A |  |
| Horizon | httpd | A/A | HAProxy |
| Ceilometer | openstack-ceilometer-api | A/A | HAProxy |
| Ceilometer | openstack-ceilometer-central | A/A | Workload partitioning: tooz + Redis |
| Ceilometer | openstack-ceilometer-compute | A/A |  |
| Ceilometer | openstack-ceilometer-alarm-notifier | A/A |  |
| Ceilometer | openstack-ceilometer-evaluator | A/A |  |
| Ceilometer | openstack-ceilometer-notification | A/A |  |
| Heat | openstack-heat-api | A/A | HAProxy |

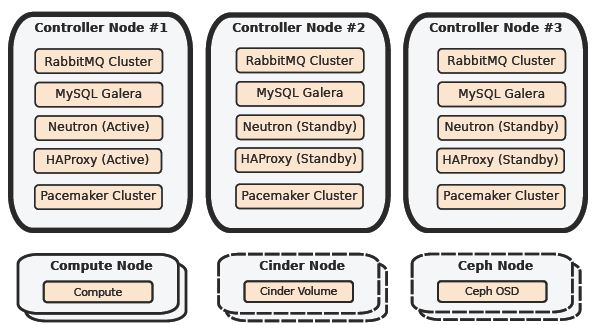
关于 MariaDB：

* 它是数据库管理系统 MySQL 的一个分支，主要由开源社区在维护，采用 GPL 授权许可。开发这个分支的原因之一是：甲骨文公司收购了 MySQL 后，有将 MySQL 闭源的潜在风险，因此社区采用分支的方式来避开这个风险。
* MariaDB 的目的是完全兼容MySQL，包括 API 和命令行，使之能轻松成为 MySQL 的代替品。除了作为一个Mysql的“向下替代品”，MariaDB包括的一些新特性使它优于MySQL。[这篇文章](http://www.linuxeden.com/html/news/20131014/144551.html) 有 Mysql 和 MariaDB 对比分析。

不由得赞一下 RDO 的文档！想起来之前去拜访一个 OpenStack 初创公司，CTO 说他们基本上是参考 RDO 做方案，看起来是很有道理的。

**3.2 Mirantis OpenStack 6.0 HA 方案：A/A 方案**

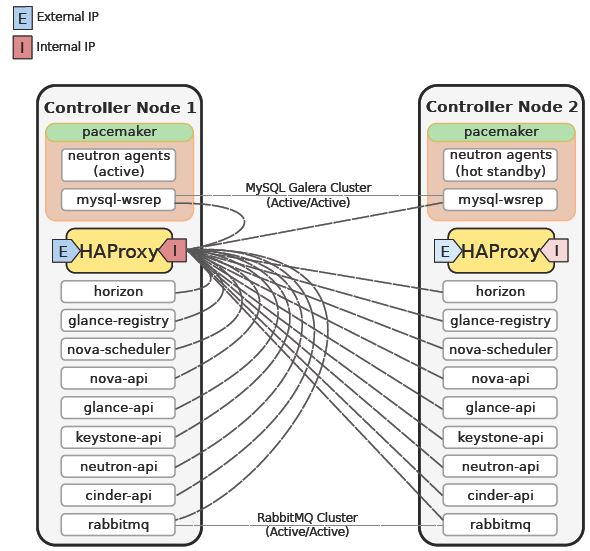
Mirantis 推荐在生产环境中使用带至少三个控制节点的 HA：



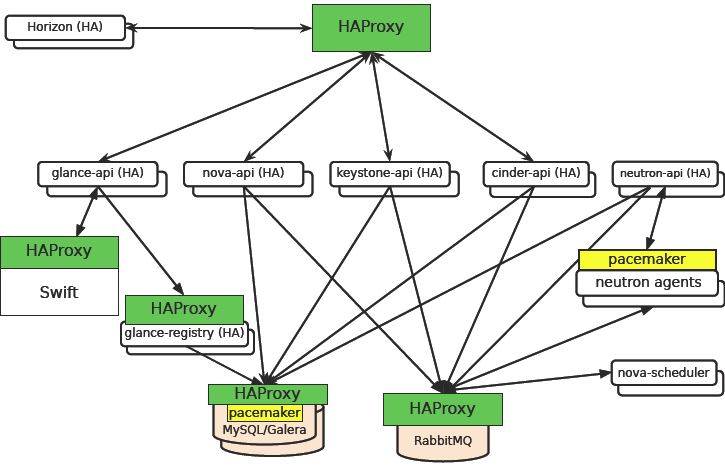
其中：

* 使用 Pacemaker 控制 Neutron Agent，实现 A/P HA
* API 服务运行在多个节点上，使用 HAProxy 实现负载均衡，提供 VIP
* RabbitMQ A/A
* Mysql A/A

**各 HA 组件之间的关系：**



各组件被调用的方式：

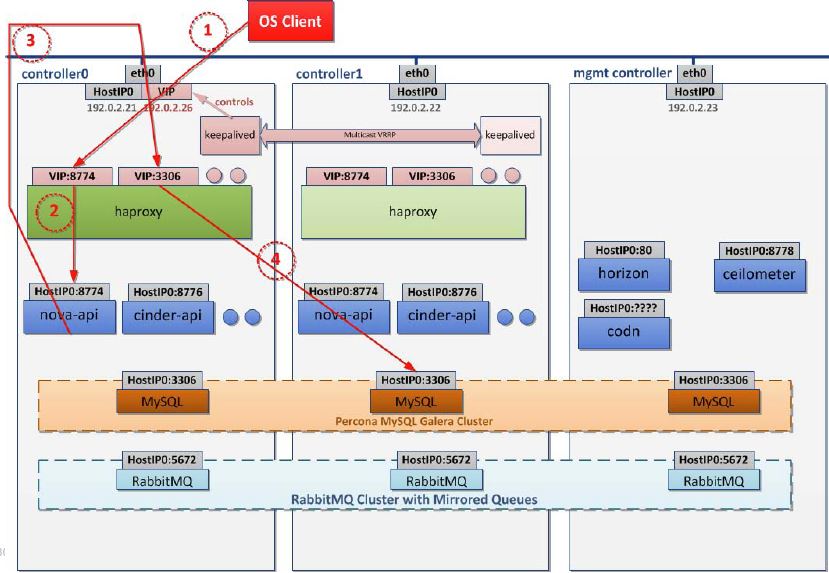


（来源：[Mirantis 官网](https://www.mirantis.com/products/mirantis-openstack-software/what-is-new-in-6-0/)）

点评：与 RDO 方案一样，该 HA 也是一个彻底的 HA 方案，消除了整个系统的 SPOF。但是，与 RDO 相比分散式控制节点相比，Mirantis 的集中式控制节点上运行的服务较多，可能会影响其性能，但是在小规模云环境中节省了硬件成本。

**3.3 HP Helion 的 HA 方案：A/A方案**

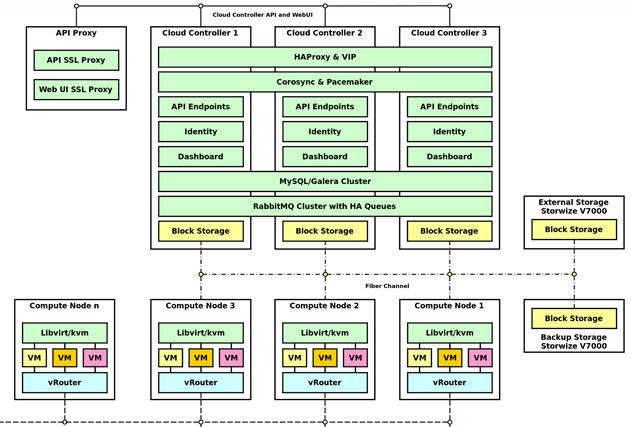
**系统组成：**（两节点 HAProxy + Keepalived 集群） +  第三个控制节点 +（RabbitMQ Cluster + Queue 镜像）+（Galera + Mysql）



1. OpenStack 客户端通过 VIP:8774 访问 nova-api
2. HAProxy 将请求转到 controller 0 上的 nova-api
3. nova-api 通过 VIP：3306 访问 Mysql DB
4. HAProxy 将请求转到 controller 1 上的 Mysql 实例

点评：HP 的 A/A 方案是不彻底的，甚至是有些怪异（为什么不三个控制节点做一个A/A 集群呢？），因为至少 Horizon、 Ceilomter 和 Neutron Agents 没有实现 HA，它只实现了 API，DB 和 MQ 的 HA。

**3.4 TCP Cloud OpenStack HA**

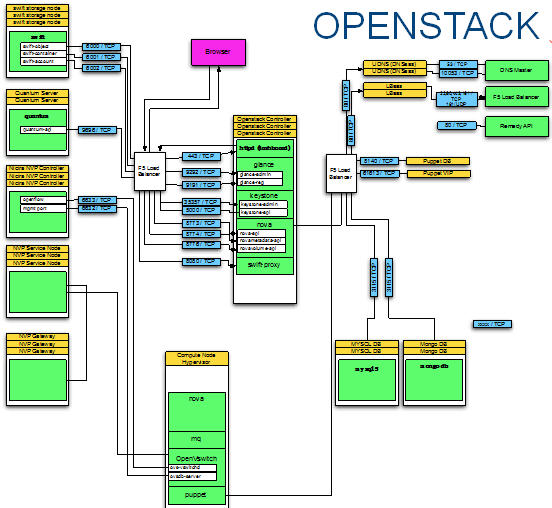


特征：

* 系统组成：Pacemaker, Corosync, HAProxy, Galera + IBM 硬件比如 Storwize V7000
* 使用三阶段集群 A/A 集群
* 提供 99.99% 的服务可靠性
* 没看到虚机 HA

来源：[TCP 官网](http://www.tcpcloud.eu/en/sluzby/virtual-private-cloud/)

**3.5 Paypal OpenStack 生产系统 HA**

（[来源及高清大图](https://www.openstack.org/summit/openstack-summit-hong-kong-2013/session-videos/presentation/openstack-high-availability-paypal)）

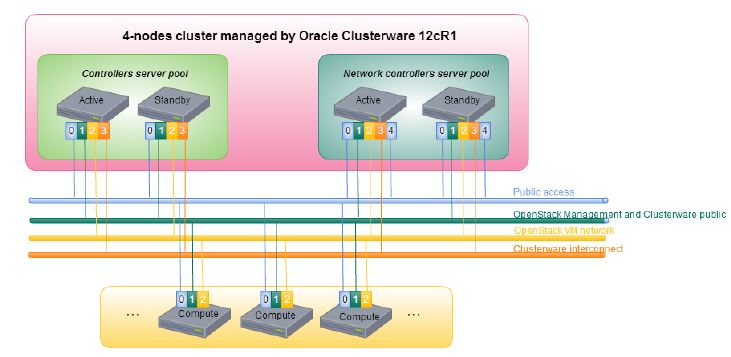
特征：

* 使用硬件负载均衡 F5
* 使用商业 SDN
* 使用 Monit 监控和重启各服务
* 使用 Pacemaker
* 用在生成系统，优化进行中

**3.6 Oracel OpenStack HA：A/P HA**

CRM：Oracel Clusterware（Oracle Grid Infrastructure Release 12c Release 1 or later）

组成：两个控制节点 + 两个网络节点组成的集群。除了网络节点上的组件外，别的组件都部署在控制节点上。

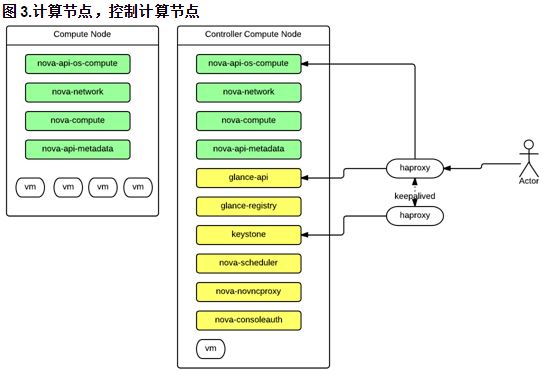
（[来源](http://www.oracle.com/technetwork/server-storage/openstack/linux/documentation/ha-guide-oracle-openstack-2296039.pdf)）

结论：该方案比不上前面几个公司的方案，因为：

* 只提供两节点 A/P 方案，可靠性和 CTO 比不上三节点方案
* 需要使用共享存储比如 NFS 来实现 A/P HA 模式的 DB 和 MQ，容易脑裂
* 不使用免费的 Pacemaker，部署成本增加。

**3.7 网易 OpenStack 云的 HA 方案**

好不容易找到一个国内公司的方案，来源在 [这里](https://www.ibm.com/developerworks/cn/cloud/library/1408_zhangxl_openstack/)：



特征：

* 使用 keepalived 管理的 HAProxy
* 控制节点应该是 A/A HA 方案
* 没有看到计算节点和网络控制节点的 HA 方案，似乎没有用 neutron，而是用 nova-network
* 高可用 RabbitMQ 集群和主备 MySQL，以及 memcache 集群是额外部署的

**3.5 小结**

* RDO > Mirantis > HP >> Oracel
* HA 是生产环境中的部署必须有的
* HA 模式方面，A/A HA 方案为主流
* 数据库方面，Mysql Galera 为主流
* MQ 方面，RabbitMQ 集群 + 镜像消息队列为主流
* CRM 方面，Pacemaker 三节点集群是主流
* 负载均衡方面，HAProxy 是主流
* 网络方面，Neutron 新的 HA 方案包括 VRRP 和 DVR 还未成熟，尚未真正进入生产环境 （2016/10： RedHat OpenStack platform 从Kilo版本就已经正式支持 VRRP，这意味着它已经成熟；但是DVR依然处于 tech preview 状态）
* 存储方面，OpenStack 需要解决 cinder-volume 的 A/A 实现
* 计算方面，OpenStack 需要原生的虚机 HA 实现

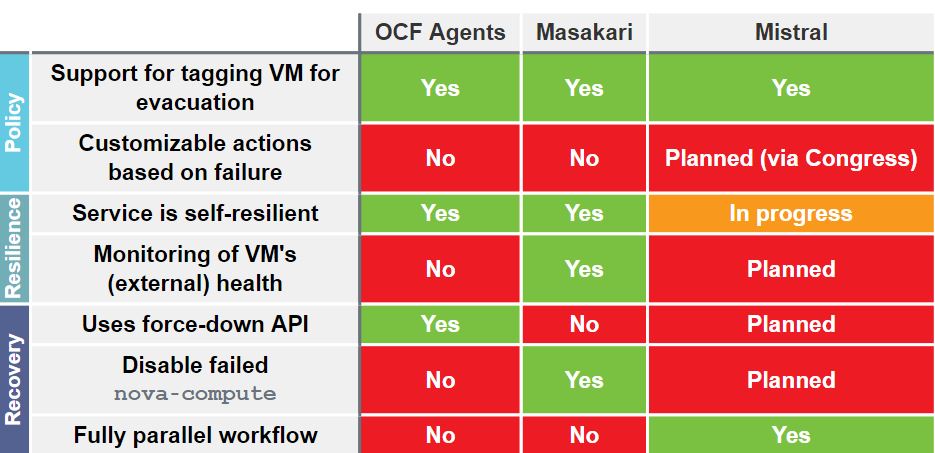
2016/10/23 一些更新：

1. 社区成立了HA Community：<http://docs.openstack.org/ha-guide/ha-community.html>
2. 社区已经将虚机的HA作为高优先级工作，相关实现会基于目前已有的各种实现而展开。具体可参考 <http://docs.openstack.org/ha-guide/instance-ha.html>
3. cinder-volume 的 A/A 模式的工作还在进行中，内容和进度可以参考 <https://blueprints.launchpad.net/cinder/+spec/cinder-volume-active-active-support>
4. 虚机 HA 有个总结：<http://aspiers.github.io/openstack-summit-2016-austin-compute-ha/>
5. 目前不同厂商有些可用虚机HA方案：[a mistral-based auto-recovery workflow](https://github.com/gryf/mistral-evacuate), by Intel,

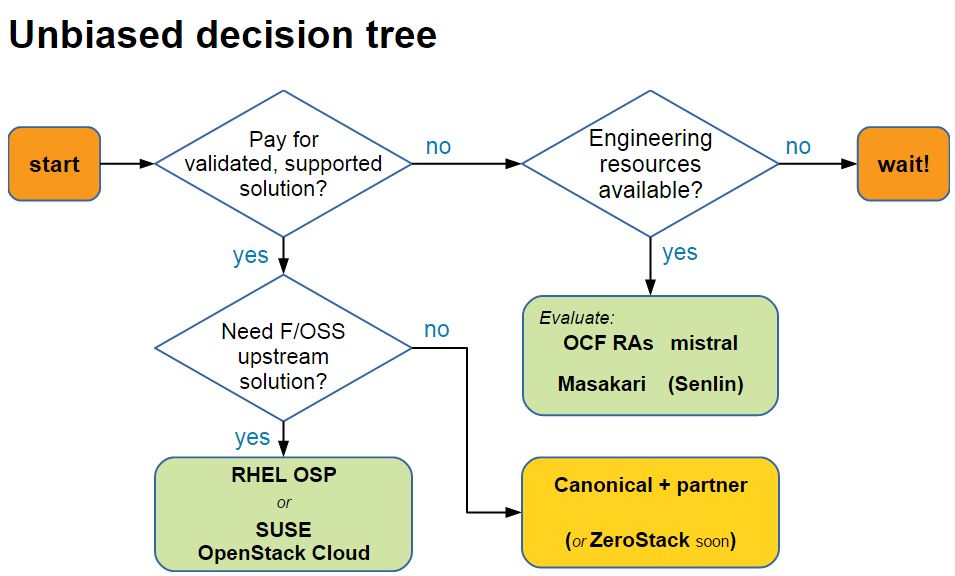
[masakari](https://launchpad.net/masakari), by NTT,

[OCF RAs](http://aspiers.github.io/openstack-summit-2016-austin-compute-ha/#/ocf-pros-cons), as used by Red Hat and SUSE

这些方案之间的对比：

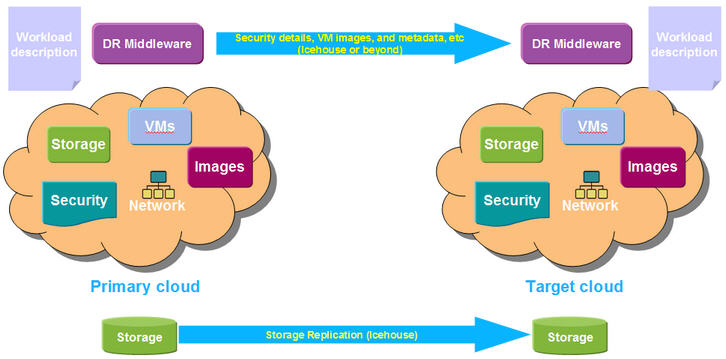


选择树：



**4. OpenStack DR**

目前，OpenStack 上没有实现 DR。 I[BM 和 RedHat 联合发起的 DRaas 提议](https://wiki.openstack.org/wiki/DisasterRecovery):



状态：

* 目前没有详细的方案，只有一个概要设计，还处于 Gap 识别和补齐阶段。
* 具体的实现主要集中在cinder 侧元数据（Juno IBM 实现了部分的 Volume Replication 功能）、业务数据同步相关，但是目前进展不乐观。

可以参考 RedHat 的更多文档，包括 [Preparing for the Worst Case Scenario: A Vision for OpenStack Disaster Recovery](http://redhatstackblog.redhat.com/2013/10/31/preparing-for-the-worst-case-scenario-a-vision-for-openstack-disaster-recovery/) 和 [Disaster Recovery Enablement in OpenStack](http://redhatstackblog.redhat.com/2013/11/26/disaster-recovery-enablement-in-openstack/)。

参考链接和文档：

* deepdiveintohighlyavailableopenstackarchitecture-openstacksummitvancouver2015-150520154718-lva1-app6891.pdf
* RDO 官网
* OpenStack 官网
* Mirantis-OpenStack-6.0-ReferenceArchitecture (1).pdf