目录

1、Openstack 环境架构

操作系统：centos 6.5 x86_64

针对计算存储一体的场景下，整个 openstack 平台由以下两种类型的节点组成。

controller-network-node:

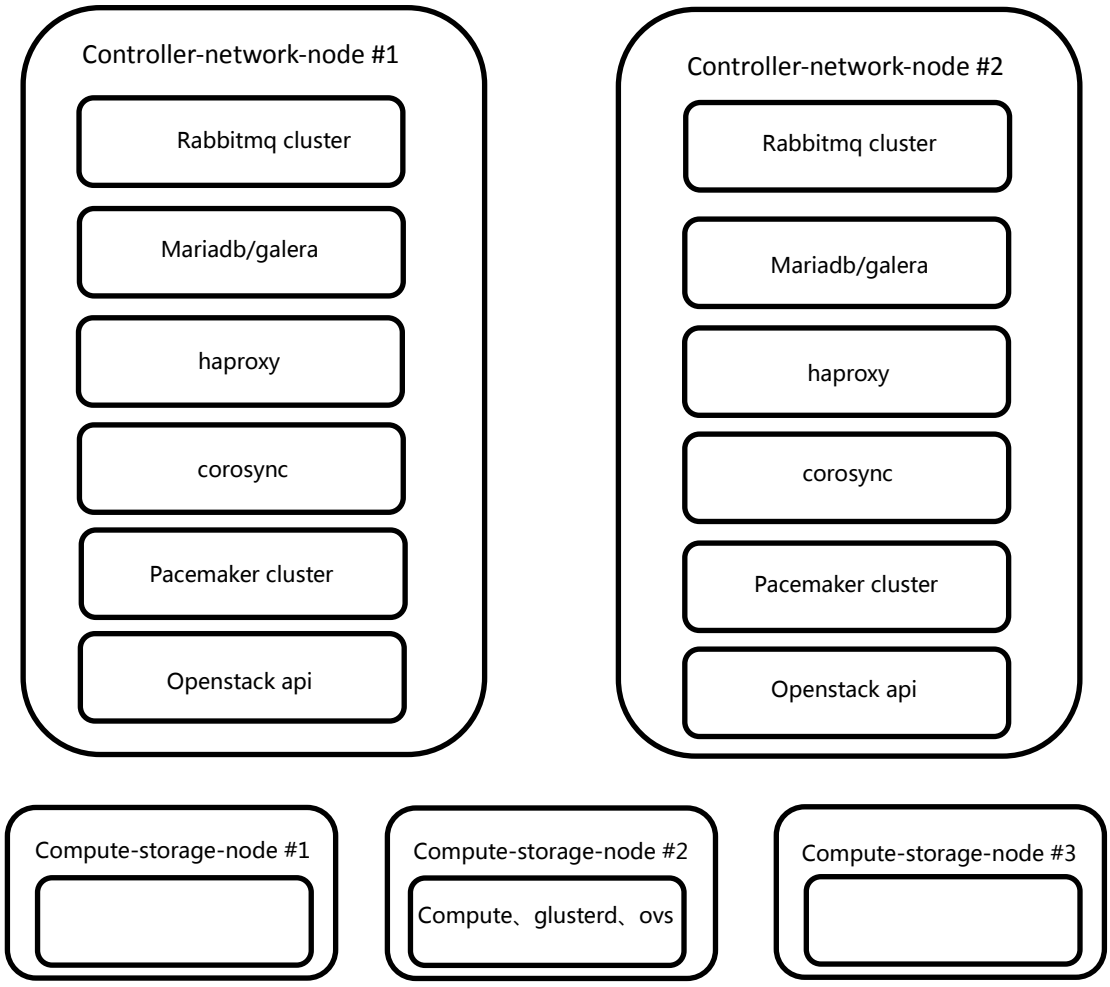
控制-网络-节点上运行着 nova、keystone、glance、neutron、cinder、ceilometer、horizon、rabbitmq、haproxy、mariadb/galera、corosync、pacemaker cluster 服务。

compute-storage-node:

计算-存储-节点上运行着 nova-compute、neutron-openvswitch-agent、glusterd、ceilometer-compute 服务。

2、双节点 HA 部署

Openstack 生产环境建议还是部署高可用。

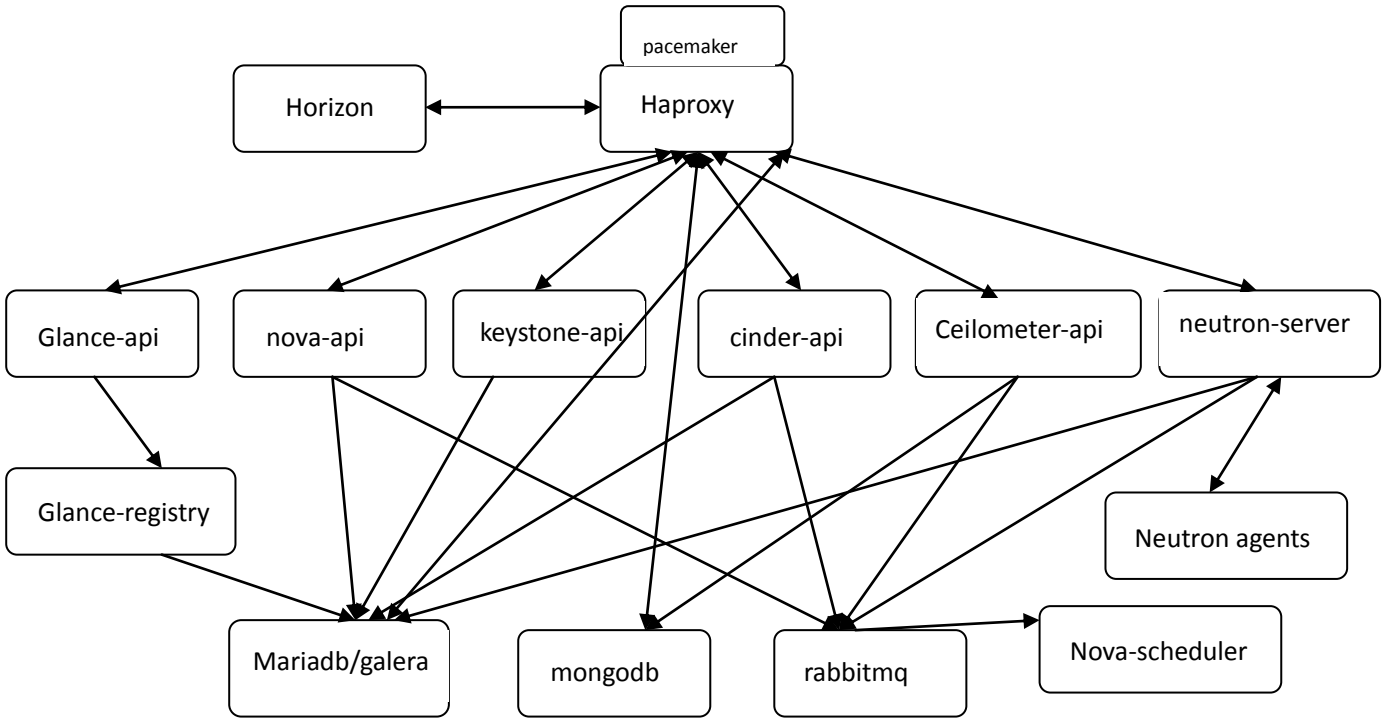


Compute、glusterd、ovs

Compute、glusterd、ovs

2.1、各服务组件流程

Openstack 服务是通过 restful api 和 rpc 消息互相通信的。Pacemaker 提供 vip，haproxy 提供 load balancing。

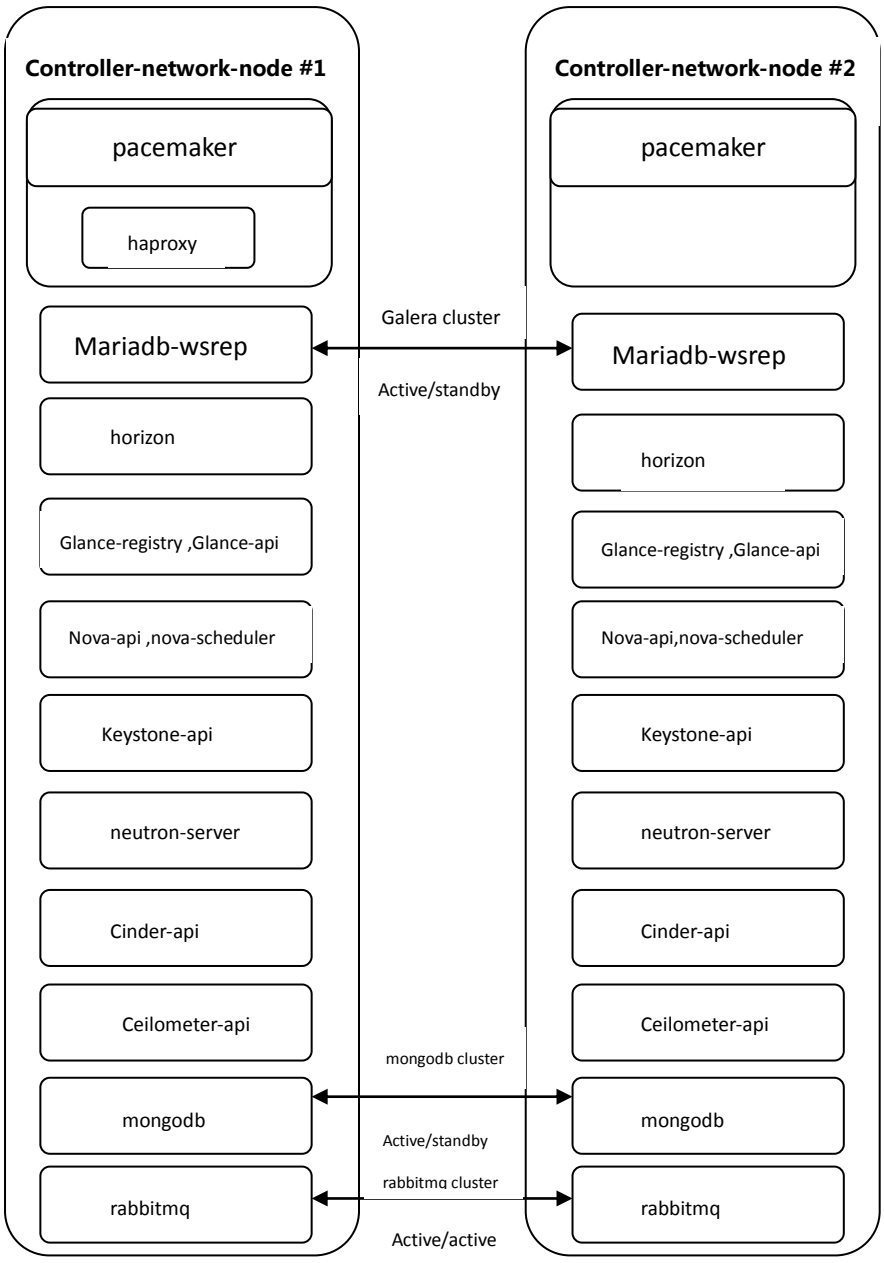


3、HA 建立逻辑

一个多节点的多节点 HA 环境要包括四种类型节点：controller node，compute node，network node，storage node。

3.1、Controller-network-nodes

Mariadb 数据库使用 galera 来实现高可用。Galera 可以实现 active/active，建议生产环境不要使用双主。Mongodb 本身就带高可用，采用副本集的方式可以实现高可用，但是副本集内部选举机制至少要三个节点。Mongodb 官方有主从方案，但是它推荐使用副本集实现高可用。



3.2、Compute-storage-nodes

针对云主机的高可用，官方目前还没有好的解决方案。业界倒是有一些做法：

1、 nova host-evacuate

让宕机的 host 下的所有 instance，在另外的 host 启动。弊端：宕机时间较长。

```
[root@controller001 ~(keystone_admin)]# nova help host-evacuate
usage: nova host-evacuate [--target_host <target_host>] [--on-shared-storage]
                          <host>

Evacuate all instances from failed host.

Positional arguments:
  <host>                      Name of host.

Optional arguments:
  --target_host <target_host>  Name of target host. If no host is specified
                               the scheduler will select a target.
  --on-shared-storage          Specifies whether all instances files are on
                               shared storage
```

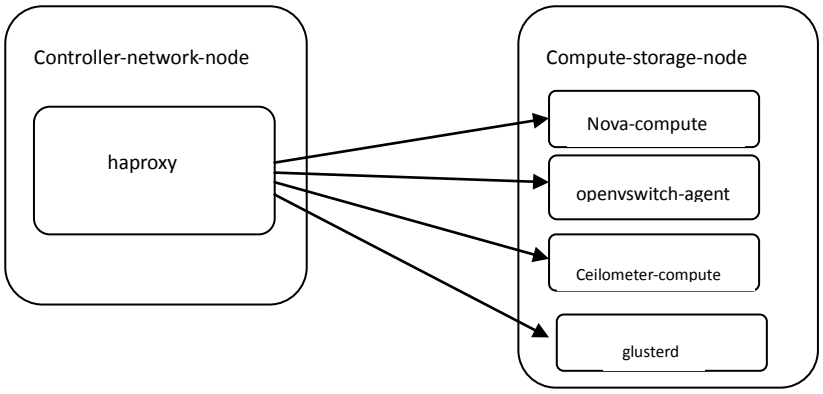
2、 neutron allowed_address_pairs 特性 + keepalived

使用 allowed_address_pairs 可以让一个 ip 对应两个 port mac，弊端：需要多创建一台虚拟机；又回到 neutron-l3-agent 高可用问题。

3、neutron lbaas

本质是在 namespace 里面启了 haproxy 程序，提供负载均衡服务。弊端：也需要多创建一台虚拟机；又是 neutron-l3-agent 高可用问题。

云主机后端存储采用 glusterfs 分布式文件系统，创建云主机的时候采用 boot from volume 方式(走 libgfapi)，性能好很多。



4、Corosync、pacemaker 建立 HA cluster

4.1、Corosync 简介:

Corosync 是 OpenAIS 发展到 Wilson 版本后衍生出来的开放性集群引擎工程。可以说 Corosync 是 OpenAIS 工程的一部分。OpenAIS 从 openais0.90 开始独立成两部分，一个是 Corosync；另一个是 AIS 标准接口 Wilson。Corosync 包含 OpenAIS 的核心框架用来对 Wilson 的标准接口的使用、管理。它为商用的或开源性的集群提供集群执行框架。Corosync 执行高可用应用程序的通信组系统，它有以下特征：

- 一个封闭的程序组（A closed process group communication model）通信模式，这个模式提供一种虚拟的同步方式来保证能够复制服务器的状态。
- 一个简单可用性管理组件（A simple availability manager），这个管理组件可以重新启动应用程序的进程当它失败后。
- 一个配置和内存数据的统计（A configuration and statistics in-memory database），内存数据能够被设置，回复，接受通知的更改信息。
- 一个定额的系统（A quorum system），定额完成或者丢失时通知应用程序。

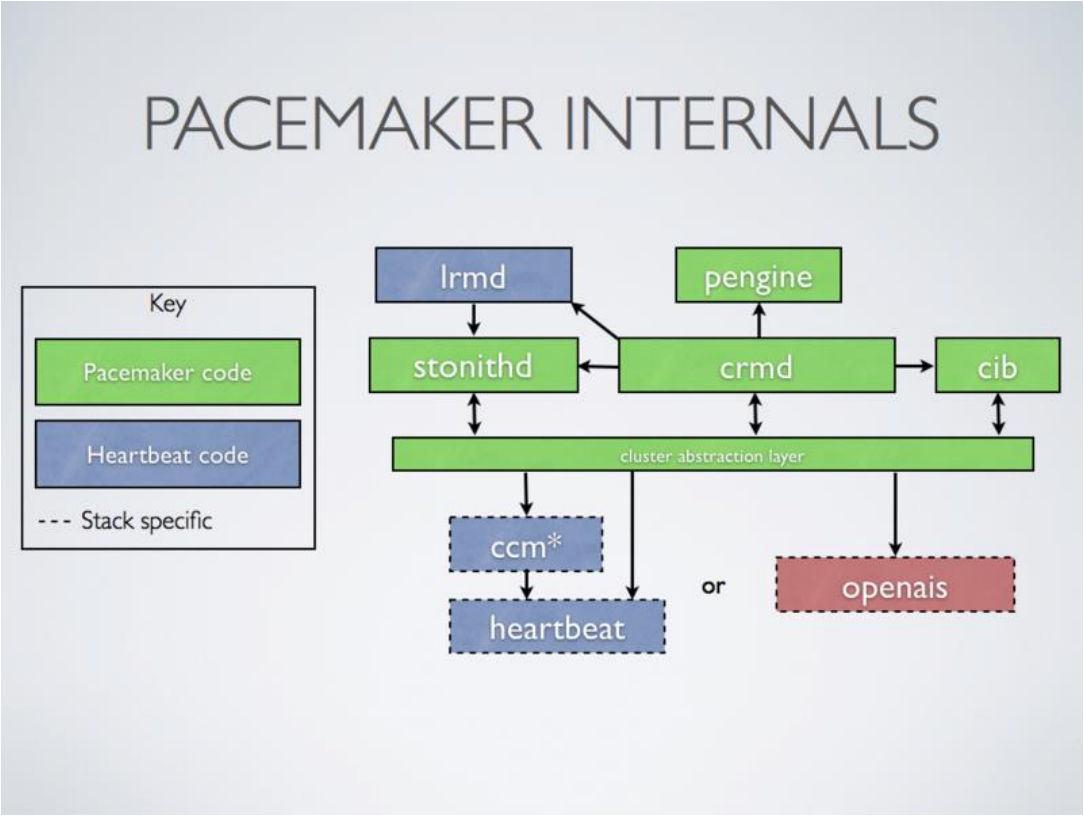
4.2、Pacemaker 简介:

pacemaker(直译：心脏起搏器)，是一个群集资源管理器。它实现最大可用性群集服务（亦称资源管理）的节点和资源级故障检测和恢复使用您的首选集群基础设施（OpenAIS 的或 Heaerbeat）提供的消息和成员能力。

它可以做乎任何规模的集群，并配备了一个强大的依赖模型，使管理员能够准确地表达群集资源之间的关系（包括顺序和位置）。几乎任何可以编写脚本，可以管理作为心脏起搏器集群的一部分。

我再次说明一下，pacemaker 是个资源管理器，不是提供心跳信息的，因为它似乎是一个普遍的误解，也是值得的。pacemaker 是一个延续的 CRM（亦称 Heartbeat V2 资源管理器），最初是为心跳，但已经成为独立的项目。

4.3、Pacemaker 内部结构



4.3.1、集群中组件介绍

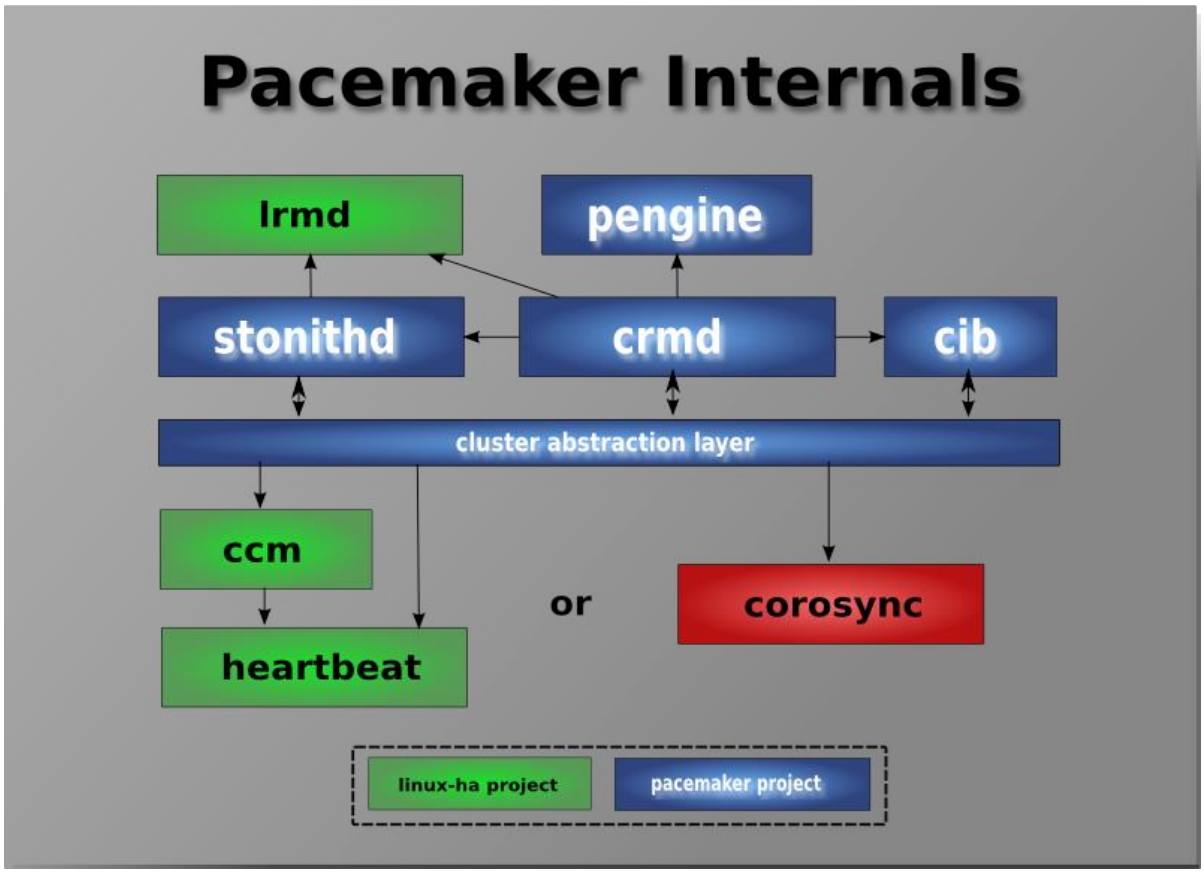
- stonithd：心跳系统。
- lrmd：本地资源管理守护进程。它提供了一个通用的接口支持的资源类型。直接调用资源代理（脚本）。
- pengine：政策引擎。根据当前状态和配置集群计算的下一个状态。产生一个过渡图，包含行动和依赖关系的列表。
- CIB：群集信息库。包含所有群集选项，节点，资源，他们彼此之间的关系和现状的定义。同步更新到所有群集节点。

- CRMD：集群资源管理守护进程。主要是消息代理的 PEngine 和 LRM，还选举一个领导者（DC）统筹活动（包括启动/停止资源）的集群。
- OpenAIS：OpenAIS 的消息和成员层。
- Heartbeat：心跳消息层，OpenAIS 的一种替代。
- CCM：共识群集成员，心跳成员层。

4.3.2、集群中组件功能说明

CIB 使用 XML 表示集群的集群中的所有资源的配置和当前状态。CIB 的内容会被自动在整个集群中同步，使用 PEngine 计算集群的理想状态，生成指令列表，然后输送到 DC（指定协调员）。Pacemaker 集群中所有节点选举的 DC 节点作为主决策节点。如果当选 DC 节点宕机，它会在所有的节点上， 迅速建立一个新的 DC。DC 将 PEngine 生成的策略，传递给其他节点上的 LRMd（本地资源管理守护程序）或 CRMD 通过集群消息传递基础结构。当集群中有节点宕机，PEngine 重新计算的理想策略。在某些情况下，可能有必要关闭节点，以保护共享数据或完整的资源回收。为此，Pacemaker 配备了 stonithd 设备。STONITH 可以将其它节点“爆头”，通常是实现与远程电源开关。Pacemaker 会将 STONITH 设备，配置为资源保存在 CIB 中，使他们可以更容易地监测资源失败或宕机。

4.4、Corosync 与 pacemaker 组合:



不管 heartbeat ,还是 corosync 都是高可用集群中的 Cluster Messaging Layer（集群信息层），是主要传递发集群信息与心跳信息的，并没有资源管理功能，资源管理还得依赖于上层的 crm(Cluster resource Manager，集群资源管理器)，最著名的资源管理器，就是 pacemaker，它是 heartbeat v3 分离出去的子项目。而现在 corosync+pacemaker 成了高可用集群中的最佳组合。

4.5、Corosync 配置:

4.5.1、修改 corosync.conf

corosync 主要配置文件/etc/corosync/corosync.conf

```
compatibility: whitetank
totem {
    version: 2
    token:                          3000
    token_retransmits_before_loss_const: 10
    join:                           60
    consensus:                      3600
    vsftype:                        none
    max_messages:                   20
    clear_node_high_bit:            yes
    rrp_mode:                       passive
    secauth:                        off
    threads: 0
    cluster_name: controller
    interface {
        ringnumber: 0
        bindnetaddr: 192.168.141.0  # 心跳网段
        mcastaddr: 224.1.1.2        # 组播地址
        mcastport: 5405
        ttl: 1
    }
    interface {
        ringnumber: 1
        bindnetaddr: 172.16.141.0 # 另一个心跳网段
        mcastaddr: 224.1.1.3
        mcastport: 5407
        ttl: 1
    }

}
logging {
    fileline: off
    to_stderr: no
    to_logfile: yes
    to_syslog: no
    logfile: /var/log/cluster/corosync.log  # 日志位置
    debug: off
    timestamp: on
    logger_subsys {
        subsys: AMF
        debug: off
    }
}
amf {
```

```
    mode: disabled
}

# corosync 接管 pacemaker
service {
    ver: 0
    name: pacemaker
}
aisexec {
    user: root
    group: root
}
```

注: man corosync.conf 可以查看所有选项的意思。

4.5.2、生成密钥

生成 key 文件默认会调用/dev/random 随机数设备 ,random 替换为 urandom 可以节省时间。

```
[root@node1 corosync]# mv /dev/{random,random.bak}
[root@node1 corosync]# ln -s /dev/urandom /dev/random
[root@node1 corosync]# corosync-keygen
Corosync Cluster Engine Authentication key generator.
Gathering 1024 bits for key from /dev/random.
Press keys on your keyboard to generate entropy.
Writing corosync key to /etc/corosync/authkey.
```

设定两个节点可以基于密钥进行 ssh 通信 , 密码要设为空 , 否则在节点切换的时候将会失败。拷贝 corosync.conf、authkey 到集群中每一个节点上 /etc/corosync/目录下

4.5.3、查看 corosync 启动日志

[root@node1 ~]# service corosync start

Starting Corosync Cluster Engine (corosync): [确定]

查看 corosync 引擎是否正常启动

[root@node1 ~]# egrep "Corosync Cluster Engine|configuration file" /var/log/cluster/corosync.log

Aug 13 14:20:15 corosync [MAIN] Corosync Cluster Engine ('1.4.1'): started and ready to provide service.

Aug 13 14:20:15 corosync [MAIN] Successfully read main configuration file '/etc/corosync/corosync.conf'.

Aug 13 17:08:51 corosync [MAIN] Corosync Cluster Engine ('1.4.1'): started and ready to provide service.

Aug 13 17:08:51 corosync [MAIN] Successfully read main configuration file '/etc/corosync/corosync.conf'.

Aug 13 17:08:51 corosync [MAIN] Corosync Cluster Engine exiting with status 18 at main.c:1794.

查看初始化集群成员节点的通知是否发出

```
[root@node1 ~]# grep –i TOTEM /var/log/cluster/corosync.log
Aug 13 14:20:15 corosync [TOTEM ] Initializing transport
(UDP/IP Multicast).
Aug 13 14:20:15 corosync [TOTEM ] Initializing transmit/receive security:
libtomcrypt SOBER128/SHA1HMAC (mode 0).
Aug 13 14:20:15 corosync [TOTEM ] The network interface
[192.168.18.201] is now up.
Aug 13 14:20:15 corosync [TOTEM ] A processor joined or left the
membership and a new membership was formed.
Aug 13 14:20:40 corosync [TOTEM ] A processor joined or left the
membership and a new membership was formed.
```

查看启动过程是否有错误

```
[root@node1 ~]# grep –i ERROR: /var/log/cluster/corosync.log
Aug 13 14:20:15 corosync [pcmk  ] ERROR: process_ais_conf: You have
configured a cluster using the Pacemaker plugin for Corosync. The
plugin is not supported in this environment and will be removed very
soon.
Aug 13 14:20:15 corosync [pcmk  ] ERROR: process_ais_conf:  Please see
Chapter 8 of 'Clusters from
```

Scratch' (http://www.clusterlabs.org/doc) for details on using Pacemaker with CMAN

查看 corosync 接管 pacemaker 是否成功

```
[root@node1 ~]# grep pcmk_startup /var/log/cluster/corosync.log
Aug 13 14:20:15 corosync [pcmk  ] info: pcmk_startup: CRM: Initialized
Aug 13 14:20:15 corosync [pcmk  ] Logging: Initialized pcmk_startup
Aug 13 14:20:15 corosync [pcmk  ] info: pcmk_startup: Maximum core file size is: 18446744073709551615
Aug 13 14:20:15 corosync [pcmk  ] info: pcmk_startup: Service: 9
Aug 13 14:20:15 corosync [pcmk  ] info: pcmk_startup: Local hostname: node1.test.com
```

查看集群状态

```
[root@node1 ~]# crm_mon
Last updated: Tue Aug 13 17:41:31 2013
Last change: Tue Aug 13 14:20:40 2013 via crmd on node1.test.com
Stack: classic openais (with plugin)
Current DC: node2.test.com - partition with quorum
Version: 1.1.8-7.el6-394e906
2 Nodes configured, 2 expected votes
```

0 Resources configured.

Online: [node1.yao.com node2.yao.com]

4.5.4、检验 corosync 安装

The first thing to check is if cluster communication is happy, for that we use **corosync-cfgtool**.

```
# corosync-cfgtool -s
Printing ring status.
Local node ID 1
RING ID 0
        id      = 192.168.122.101
        status  = ring 0 active with no faults
```

We can see here that everything appears normal with our fixed IP address, not a 127.0.0.x loopback address, listed as the **id** and **no faults** for the status.

If you see something different, you might want to start by checking the node's network, firewall and selinux configurations.

Next we check the membership and quorum APIs:

```
# corosync-cmapctl  | grep members
runtime.totem.pg.mrp.srp.members.1.ip (str) = r(0) ip(192.168.122.101)
runtime.totem.pg.mrp.srp.members.1.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.1.status (str) = joined
runtime.totem.pg.mrp.srp.members.2.ip (str) = r(0) ip(192.168.122.102)
runtime.totem.pg.mrp.srp.members.2.join_count (u32) = 1
runtime.totem.pg.mrp.srp.members.2.status (str) = joined

# pcs status corosync
Membership information
----------------------
    Nodeid      Votes Name
         1          1 pcmk-1 (local)
         2          1 pcmk-2
```

You should see both nodes have joined the cluster.

All good!

4.6、Pacemaker 配置

Pacemaker 官网：http://clusterlabs.org/

4.6.1、资源配置工具下载

Pacemaker 资源配置方式有两种：

一、 命令行配置

a、crmsh

b、pcs

二、 图形化配置

这里我们介绍 crmsh 配置资源，从 pacemaker 1.1.8 开始，crm sh 发展成一个独立项目，pacemaker 中不再提供，需要额外安装 crmsh 包。

crmsh 官网：

https://savannah.nongnu.org/forum/forum.php?forum_id=7672

crmsh 包下载地址：

http://download.opensuse.org/repositories/network:/ha-clustering:/Stable/

4.6.2、添加资源

在终端命令行敲入 crm 进入 crm shell 模式：

添加如下资源

```
node Openstack-bjctc21 \
        attributes standby="off"
node Openstack-bjctc22
primitive HAPROXY lsb:haproxy \
        op monitor interval="30s" \
        meta target-role="Started"
primitive NTP lsb:ntpd
primitive VIP ocf:heartbeat:IPaddr2 \
        params ip="192.168.2.23" nic="eth3" cidr_netmask="23" \
        meta target-role="Started" \
        op monitor interval="20s"
clone ntp-clone NTP \
        meta target-role="Started"
location cli-perfer-VIP VIP inf: Openstack-bjctc21
colocation haproxy-with-vip inf: VIP HAPROXY
order vip-before-haproxy Mandatory: VIP HAPROXY
property $id="cib-bootstrap-options" \
        dc-version="1.1.10-3.el6.2-368c726" \
        cluster-infrastructure="classic openais (with plugin)" \
        expected-quorum-votes="2" \
        stonith-enabled="false" \
        no-quorum-policy="ignore" \
        default-resource-stickiness="100"
```

注: 进入 crm shell 下，遇到不懂的命令，help 下。

这里贴几个链接：

Corosync [Link]

Pacemaker [Link]

crmsh [Link]

DRBD [Link]

GlusterFS [Link]

Wiki: High Availability [Link]

5、Mariadb 建立高可用 cluster

5.1、Galera 简介

Galera 本质是一个 wsrep 提供者（provider），运行依赖于 wsrep 的 API 接口。Wsrep API

定义了一系列应用回调和复制调用库 ,来实现事务数据库同步写集(writeset)复制以及相似应用。

目的在于从应用细节上实现抽象的，隔离的复制。虽然这个接口的主要目标是基于认证的多主复制，但同样适用于异步和同步的主从复制。

5.2、注意事项

1、使用 Galera 必须要给 MySQL-Server 打 wsrep 补丁。可以直接使用官方提供的已经打好补丁的 MySQL 安装包，如果服务器上已经安装了标准版 MYSQL，需要先卸载再重新安装。卸载前注意备份数据。

2、MySQL/Galera 集群只支持 InnoDB 存储引擎。如果你的数据表使用的 MyISAM，需要转换为 InnoDB，否则记录不会在多台复制。可以在备份老数据时，为 mysqldump 命令添加–skip-create-options 参数，这样会去掉表结构的声明信息，再导入集群时自动使用 InnoDB 引擎。不过这样会将 AUTO_INCREMENT 一并去掉，已有 AUTO_INCREMENT 列的表，必须在导入后重新定义。

3、MySQL 5.5 及以下的 InnoDB 引擎不支持全文索引（FULLTEXT indexes），如果之前使用 MyISAM 并建了全文索引字段的话，只能安装 MySQL 5.6 with wsrep patch。

4、所有数据表必须要有主键（PRIMARY）,如果没有主键可以建一条 AUTO_INCREMENT 列。

5、MySQL/Galera 集群不支持下面的查询：LOCK/UNLOCK TABLES，不支持下面的系统变量：character_set_server、utf16、utf32 及 ucs2。

6、数据库日志不支持保存到表，只能输出到文件（log_output = FILE），不能设置 binlog-do-db、binlog-ignore-db。

7、跟其他集群一样，为了避免节点出现脑裂而破坏数据，建议 Galera 集群最低添加 3 个节点。

8、在高并发的情况下，多主同时写入时可能会发生事务冲突，此时只有一个事务请求会成功，其他的全部失败。可以在写入/更新失败时，自动重试一次，再返回结果。

9、节点中每个节点的地位是平等的，没有主次，向任何一个节点读写效果都是一样的。实际可以配合 VIP/LVS 或 HA 使用，实现高可用性。

10、如果集群中的机器全部重启，如机房断电，第一台启动的服务器必须以空地址启动：Galera 的配置中第一台服务器的 wsrep_cluster_address 可以设置成“gcomm://”，而第二个节点的 wsrep_cluster_address 可以设置成“gcomm://第一个节点的 IP 地址”，第三个节点的 wsrep_cluster_address 可以设置成“gcomm://第二个节点的 IP 地址”，以此类推，但需要注意的是必须第 n 个节点先于第 n+1 个节点启动数据库，第 n+1 个数据库才能启动成功

5.3、建立 galera cluster

创建用于同步数据库的 SST 帐号

```
[root@mdb-01 ~]#  mysql -u root -p
mysql> GRANT USAGE ON *.* to sst_user@'%' IDENTIFIED BY 'sst_pass';
mysql> GRANT ALL PRIVILEGES on *.* to sst_user@'%';
mysql> FLUSH PRIVILEGES;
```

```
mysql> quit
```

第一个节点的 my.cnf 配置文件

```
[mysqld]
server_id=1
bind_address = controller1
datadir=/var/lib/mysql
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8

default-storage-engine = innodb
innodb_file_per_table = 1
innodb_buffer_pool_size = 512M
tmp_table_size = 256M
innodb_autoinc_lock_mode=2
innodb_doublewrite=1
innodb_locks_unsafe_for_binlog=1
innodb_flush_log_at_trx_commit=0
innodb_open_files = 500
innodb_write_io_threads = 4
innodb_read_io_threads = 4
innodb_thread_concurrency = 0
innodb_purge_threads = 1
innodb_log_buffer_size = 2M
innodb_log_file_size = 32M
innodb_log_files_in_group = 3
innodb_max_dirty_pages_pct = 90
innodb_lock_wait_timeout = 120

open_files_limit = 65535
max_allowed_packet = 4M
max_heap_table_size = 8M
tmp_table_size = 16M
read_buffer_size = 2M
read_rnd_buffer_size = 8M
sort_buffer_size = 8M
join_buffer_size = 8M
thread_cache_size = 64
query_cache_size = 0   # galera 不支持 cache
query_cache_type = 0   # galera 不支持 cache
interactive_timeout = 28800
```

```
wait_timeout = 28800

wsrep_provider=/usr/lib64/galera/libgalera_smm.so
wsrep_provider_options="pc.ignore_sb=false;
gmcast.listen_addr=tcp://controller1:4567;gcs.fc_limit = 256; gcs.fc_factor = 0.99;
gcs.fc_master_slave=yes"
#wsrep_cluster_address=gcomm://controller2
wsrep_cluster_address=gcomm://
wsrep_cluster_name='openstack-controller'
wsrep_node_address='controller1'
wsrep_node_name='mysql-galera-controller1'
wsrep_sst_method=xtrabackup  # sst 采用 Percona 提供的 xtrabackup(防止锁表，非阻塞)
wsrep_sst_auth=sst_user:sst_pass
wsrep_slave_threads=16

[mysqldump]
quick

max_allowed_packet = 16M
```

第二个节点的 my.cnf 配置文件

```
[mysqld]
server_id=2
bind_address = controller2
datadir=/var/lib/mysql
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8

default-storage-engine = innodb
innodb_file_per_table = 1
innodb_buffer_pool_size = 512M
tmp_table_size = 256M
innodb_autoinc_lock_mode=2
innodb_doublewrite=1
innodb_locks_unsafe_for_binlog=1
innodb_flush_log_at_trx_commit=0
innodb_open_files = 500
innodb_write_io_threads = 4
innodb_read_io_threads = 4
innodb_thread_concurrency = 0
innodb_purge_threads = 1
innodb_log_buffer_size = 2M
```

```
innodb_log_file_size = 32M
innodb_log_files_in_group = 3
innodb_max_dirty_pages_pct = 90
innodb_lock_wait_timeout = 120

open_files_limit = 65535
max_allowed_packet = 4M
max_heap_table_size = 8M
tmp_table_size = 16M
read_buffer_size = 2M
read_rnd_buffer_size = 8M
sort_buffer_size = 8M
join_buffer_size = 8M
thread_cache_size = 64
query_cache_size = 0   # galera 不支持 cache
query_cache_type = 0   # galera 不支持 cache
interactive_timeout = 28800
wait_timeout = 28800

wsrep_provider=/usr/lib64/galera/libgalera_smm.so
wsrep_provider_options="pc.ignore_sb=false;
gmcast.listen_addr=tcp://controller2:4567;gcs.fc_limit = 256; gcs.fc_factor = 0.99;
gcs.fc_master_slave=yes"
wsrep_cluster_address=gcomm://controller1
#wsrep_cluster_address=gcomm://
wsrep_cluster_name='openstack-controller'
wsrep_node_address='controller2'
wsrep_node_name='mysql-galera-controller2'
wsrep_sst_method=xtrabackup   # sst 采用 Percona 提供的 xtrabackup(防止锁表，非阻塞)
wsrep_sst_auth=sst_user:sst_pass
wsrep_slave_threads=16

[mysqldump]
quick

max_allowed_packet = 16M
```

5.4、添加仲裁节点

对于只有 2 个节点的 Galera Cluster 和其他集群软件一样，需要面对极端情况下的"脑裂"状态。

为了避免这种问题，Galera 引入了"arbitrator(仲裁人)"。

"仲裁人"节点上没有数据,它在集群中的作用就是在集群发生分裂时进行仲裁，集群中可以有多个"仲裁人"节点。

"仲裁人"节点加入集群的方法很简单,运行如下命令即可:

```
[root@garbd ~]# garbd -a gcomm://192.168.141.110:4567 -g my_wsrep_cluster -d
```

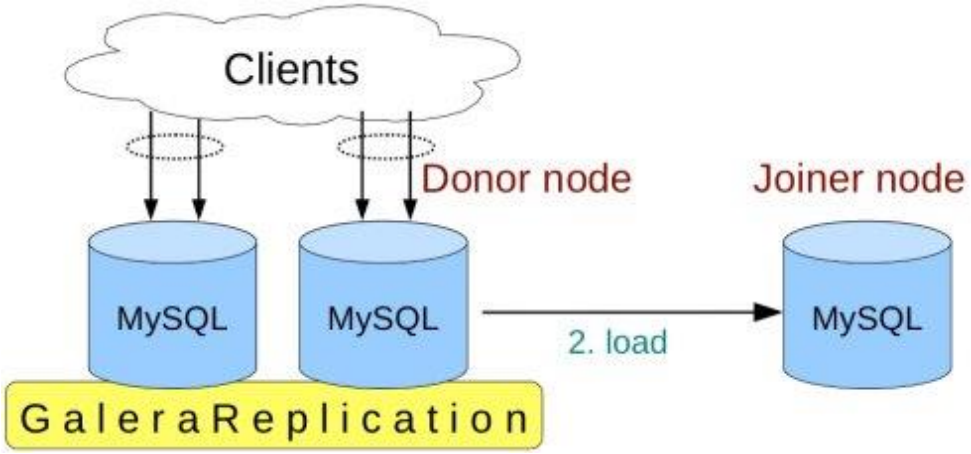
5.5、添加 haproxy 健康检查

HAProxy 可以通过 option mysql-check user dbuser 检查后端服务器数据库的运行情况

创建用于后端 MySQL 服务器健康检查的数据库帐号，这个检查会向后端服务器发送 2 个包,一个用于客户端认证，一个用于关闭连接。这个检查需要在每个节点上都需要创建一个无密码的 MySQL 帐号。

```
[root@controller001 ~]#  mysql -u root -p
Enter password:
mysql> INSERT INTO mysql.user (host,user) values ('%','haproxy');
mysql> FLUSH PRIVILEGES;
mysql> quit
```

5.6、galera 添加新节点



在 Galera Cluster 中，新接入的节点叫 Joiner,给 joiner 提供复制的节点叫 Donor。
在生产环境中，建议设置一个专用的 donor，这个专用的 donor 不执行任何来自客户端的 SQL 请求，这样做有以下几点好处：

1）数据的一致性：

因为 donor 本身不执行任何客户端 SQL，所以在这个节点上发生事务冲突的可能性最小，因此，如果发现集群有数据不一致时，donor 上的数据应该是整个集群中最准确的。

2）数据安全性：

因为专用 donor 本身不执行任何客户端 SQL，所以在这个节点上发生灾难事件的可能性最小，因此当整个集群宕掉的时候，该节点应该是恢复集群的最佳节点。

3）高可用性：

专用 donor 可以作为专门的 state snapshot donor。因为该节点不服务于客户端，因此当使用此节点进行 sst 的时候，不影响用户体验，并且前端的负载均衡设备也不需要重新配置。

6、Mongodb 建立高可用 cluster

6.1、Mongodb 简介

MongoDB 是一个高性能，开源，无模式的文档型数据库，是当前 NoSql 数据库中比较热门的一种。它在许多场景下可用于替代传统的关系型数据库或键/值存储方式。Mongo 使用 C++开发。Mongo 的官方网站地址是：http://www.mongodb.org/，读者可以在此获得更详细的信息。

特点:

高性能、易部署、易使用，存储数据非常方便。主要功能特性有：

- 面向集合存储，易存储对象类型的数据。
- 模式自由。
- 支持动态查询。
- 支持完全索引，包含内部对象。
- 支持查询。
- 支持复制和故障恢复。
- 使用高效的二进制数据存储，包括大型对象（如视频等）。

- 自动处理碎片，以支持云计算层次的扩展性
- 支持 Python，PHP，Ruby，Java，C，C#，Javascript，Perl 及 C++语言的驱动程序，社区中也提供了对 Erlang 及.NET 等平台的驱动程序。
- 文件存储格式为 BSON（一种 JSON 的扩展）。
- 可通过网络访问。

功能:

- **面向集合的存储：**适合存储对象及 JSON 形式的数据。
- **动态查询：**Mongo 支持丰富的查询表达式。查询指令使用 JSON 形式的标记，可轻易查询文档中内嵌的对象及数组。
- **完整的索引支持：**包括文档内嵌对象及数组。Mongo 的查询优化器会分析查询表达式，并生成一个高效的查询计划。
- **查询监视：**Mongo 包含一个监视工具用于分析数据库操作的性能。
- **复制及自动故障转移：**Mongo 数据库支持服务器之间的数据复制，支持主-从模式及服务器之间的相互复制。复制的主要目标是提供冗余及自动故障转移。
- **高效的传统存储方式：**支持二进制数据及大型对象（如照片或图片）
- **自动分片以支持云级别的伸缩性：**自动分片功能支持水平的数据库集群，可动态添加额外的机器。

适用场合:

- 网站数据：Mongo 非常适合实时的插入，更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。
- 缓存：由于性能很高，Mongo 也适合作为信息基础设施的缓存层。在系统重启之后，由 Mongo 搭建的持久化缓存层可以避免下层的数据源 过载。
- 大尺寸，低价值的数据：使用传统的关系型数据库存储一些数据时可能会比较昂贵，在此之前，很多时候程序员往往会选择传统的文件进行存储。
- 高伸缩性的场景：Mongo 非常适合由数十或数百台服务器组成的数据库。Mongo 的路线图中已经包含对 MapReduce 引擎的内置支持。
- 用于对象及 JSON 数据的存储：Mongo 的 BSON 数据格式非常适合文档化格式的存储及查询。

6.2、副本集 cluster

Mongodb 官方不建议使用主从模式建立高可用 cluster，主从模式存在一个很烦人的问题那就是当主节点挂了需要手动切换。

主从模式其实就是一个单副本的应用，没有很好的扩展性和容错性。而副本集具有多个副本保证了容错性，就算一个副本挂掉了还有很多副本存在，并且解决了上面第一个问题“主节点挂掉了，整个集群内会自动切换”。副本集高可用至少要三个节点才能做。原因参考下

面要讲的《副本集内部选举算法》。

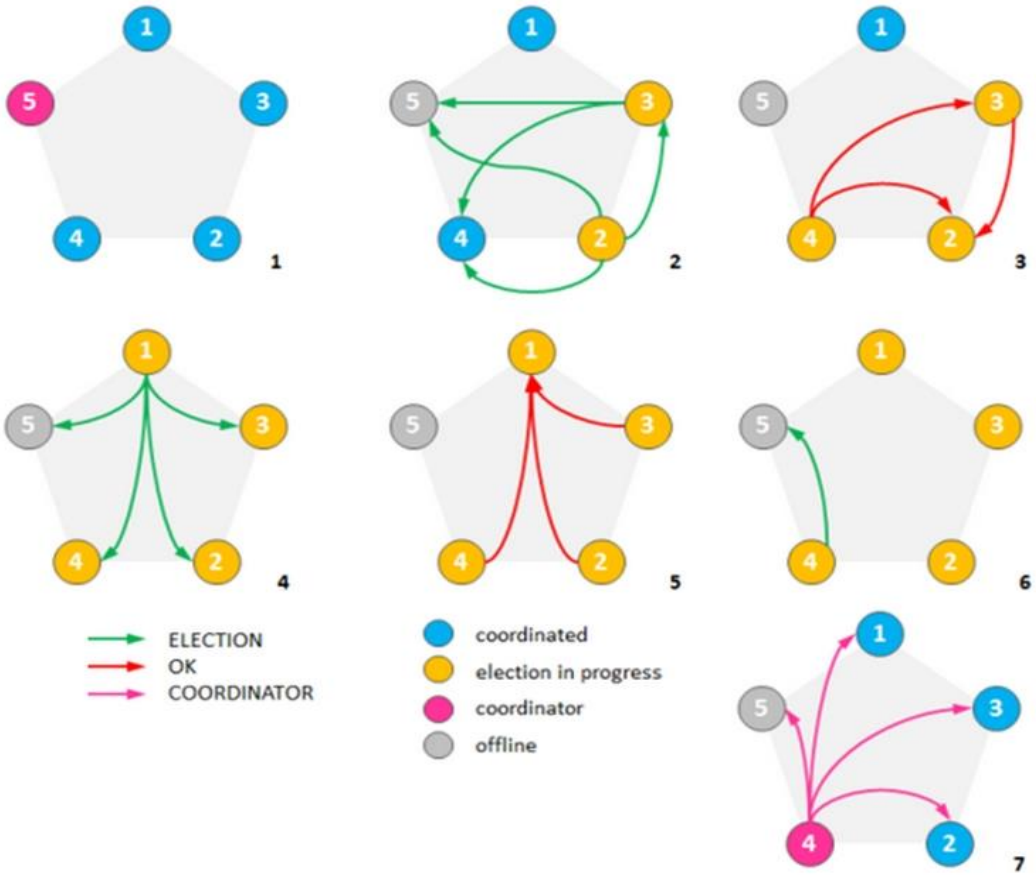
6.3、副本集内部选举算法

mongodb 副本集的选举机制采用 bully 算法，bully 算法是一种相对简单的协调者竞选算法，mongodb 用这个算法来选举副本集中的主节点。bully 算法主要思想是集群中的每个成员都可以声明它是主节点(协调者)并通知其他节点，别的节点可以选择接受这个声称或是拒绝并进入协调者竞争，被其他所有节点接受的节点才能成为协调者，节点按照一些属性来判断谁应该胜出，这个属性可以是一个静态 ID，也可以是更新的度量像最近一次事务 ID(最新的节点会胜出）。

这个网站有个 bully 算法例子，挺不错的 http://blog.nosqlfan.com/html/4139.html

下图的例子展示了 bully 算法的执行过程。使用静态 ID 作为度量，ID 值更大的节点会胜出：

1. 最初集群有 5 个节点，节点 5 是一个公认的协调者。
2. 假设节点 5 挂了，并且节点 2 和节点 3 同时发现了这一情况。两个节点开始竞选并发送竞选消息给 ID 更大的节点。
3. 节点 4 淘汰了节点 2 和 3，节点 3 淘汰了节点 2。
4. 这时候节点 1 察觉了节点 5 失效并向所有 ID 更大的节点发送了竞选信息。
5. 节点 2、3 和 4 都淘汰了节点 1。
6. 节点 4 发送竞选信息给节点 5。
7. 节点 5 没有响应，所以节点 4 宣布自己当选并向其他节点通告了这一消息。



这是某位大神画的图，看得真直观。我也尝试用 word 来画，画图果然是个细活啊！

协调者竞选前提条件：集群中有一半以上的节点参与了竞选。这确保了在网络隔离的情况下只有一部分节点能选出协调者（假设网络中网络会被分割成多块区域，之间互不联通，协调者竞选的结果必然会在节点数相对比较多的那个区域中选出协调者，当然前提是那个区域中的可用节点多于集群原有节点数的半数。如果集群被隔离成几个区块，而没有一个区块的节点数多于原有节点总数的一半，那就无法选举出协调者，当然这样的情况下也别指望集群能够继续提供服务了）。

选举触发条件

1　初始化一个副本集时

2 主节点和副本集断开连接(可能是网络问题)

3 主节点宕机

6.4、建立 mongodb cluster

Master 节点：

```
[root@controller001 ~]# vim /etc/mongodb.conf
logpath=/var/log/mongodb/mongod.log
logappend=true
fork=true
port=27017
dbpath=/var/lib/mongodb
pidfilepath=/var/run/mongodb/mongod.pid
bind_ip=controller001
replSet=s1                          # 副本集名字
```

slave 节点：

```
[root@controller002 ~]# vim /etc/mongodb.conf
logpath=/var/log/mongodb/mongod.log
logappend=true
fork=true
port=27017
dbpath=/var/lib/mongodb
pidfilepath=/var/run/mongodb/mongod.pid
bind_ip=controller002
replSet=s1
```

```
use admin
config =
{_id:'s1',members:[{_id:0,host:'controller1:27017',priority:3},{_id:1,host:'controller2:27017'}]}
rs.initiate(config)       # 初始化副本集
```

为了满足副本集内部选举算法的条件，还要添加一个仲裁节点

```
rs.addArb("controller3:27017")
```

6.5、动态修改节点 priority

```
rs0:PRIMARY> cfg=rs.conf()
rs0:PRIMARY> cfg.members[0].priority = 5##设置“id”为“0”的节点，priority为5##
5
rs0:PRIMARY> rs.reconfig(cfg)
rs0:SECONDARY> rs.conf()##查看配置文件，就会发现30001的priority变成5##
rs0:SECONDARY> rs.status();##过几分钟再看，就会发现30001变成了PRIMARY节点##
```

7、Rabbitmq 建立高可用 cluster

Rabbitmq 官方(www.rabbitmq.com)文档上搭建高可用集群的方式有两种：
对 rabbitmq 官方 cluster 文档的中文翻译 http://m.oschina.net/blog/93548

尽管 rabbitmq 本身支持 cluster，但是 cluster 并没有高可用。
一、 消息队列高可用(active/active)
二、 使用 pacemaker+drbd(active/standby)

这里我们介绍的是第一种高可用方式。

7.1、建立 cluster

Rabbitmq-server 启动的时候会产生一个随机 cookie，unix 系统下一般
在/var/lib/rabbitmq/.erlang.cookie。

集群中的每一个节点的.erlang.cookie 要一致（拷贝其中某一个节点的.erlang.cookie 到其它系节点上）。

正常方式启动 rabbitmq-server(这步好像是没必要的，可能你已经 service rabbitmq-server start 了)

```
rabbit1$ rabbitmq-server -detached
rabbit2$ rabbitmq-server -detached
```

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

```
[{nodes,[{disc,[rabbit@rabbit2]}]},{running_nodes,[rabbit@rabbit2]}]
...done.
```

```
rabbit2$ rabbitmqctl stop_app
Stopping node rabbit@rabbit2 ...done.
rabbit2$ rabbitmqctl join_cluster --ram rabbit@rabbit1
Clustering node rabbit@rabbit2 with [rabbit@rabbit1] ...done.
rabbit2$ rabbitmqctl start_app
Starting node rabbit@rabbit2 ...done.
```

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

7.2、改变节点类型

```
rabbit2$ rabbitmqctl stop_app
Stopping node rabbit@rabbit2 ...done.
rabbit2$ rabbitmqctl change_cluster_node_type disc
Turning rabbit@rabbit2 into a disc node ...
...done.
Starting node rabbit@rabbit2 ...done.
```

7.3、移除节点

从集群中移除节点有两种方式：

一、　　自身 reset

```
rabbit3$ rabbitmqctl stop_app
Stopping node rabbit@rabbit3 ...done.
```

```
rabbit3$ rabbitmqctl reset
Resetting node rabbit@rabbit3 ...done.
rabbit3$ rabbitmqctl start_app
Starting node rabbit@rabbit3 ...done.
```

这样 rabbit3 就变成一个独立的节点了。

二、 远程删除

```
rabbit1$ rabbitmqctl stop_app
Stopping node rabbit@rabbit1 ...done.
rabbit2$ rabbitmqctl forget_cluster_node rabbit@rabbit1
Removing node rabbit@rabbit1 from cluster ...
...done.
```

这样操作后，本地还是要 reset 下，不然它还是认为自己的是 cluster 的一部分。

```
rabbit1$ rabbitmqctl start_app
Starting node rabbit@rabbit1 ...
Error: inconsistent_cluster: Node rabbit@rabbit1 thinks it's clustered
with node rabbit@rabbit2, but rabbit@rabbit2 disagrees
rabbit1$ rabbitmqctl reset
Resetting node rabbit@rabbit1 ...done.
rabbit1$ rabbitmqctl start_app
Starting node rabbit@mcnulty ...
...done.
```

7.4、配置 HA queues

通过配置 policy 启用 mirror queues

rabbitmqctl set_policy ha-all “^ha.” ‘{“ha-mode”:”all”}’ # 所有名为 ha.开头的队列同步到 cluster 中的所有节点上

rabbitmqctl set_policy ha-all “^.” ‘{“ha-mode”:”all”}’ # 所有队列同步到 cluster 中的所有节点上

mirror queue 同步的仅是元数据，队列中的数据还是只存在某一个节点上。

注: cluster 中同步 queue 的节点数越多，整个 cluster 的性能越差。所以 mirror queues 要调优的东西很多，比如(flow control: limit the rate of message publication)。

7.5、启动管理插件

```
[root@controller001 ~(keystone_admin)]# find / -name rabbitmq-plugins

`find / -name rabbitmq-plugins`  enable rabbitmq_management   # 启用插件
```

默认 web ui url：http://server-name:15672
默认 user/pass: guest/guest

7.6、注意事项

一、 为了防止数据丢失的发生，在任何情况下都应该保证至少有一个 node 是采用磁盘 node 方式。RabbitMQ 在很多情况下会阻止创建仅有内存 node 的 cluster ，但是如果你通过手动将 cluster 中的全部磁盘 node 都停止掉或者强制 reset 所有的磁盘 node 的方式间接导致生成了仅有内存 node 的 cluster ，RabbitMQ 无法阻止你。你这么做本身是很不明智的，因为会导致你的数据非常容易丢失。

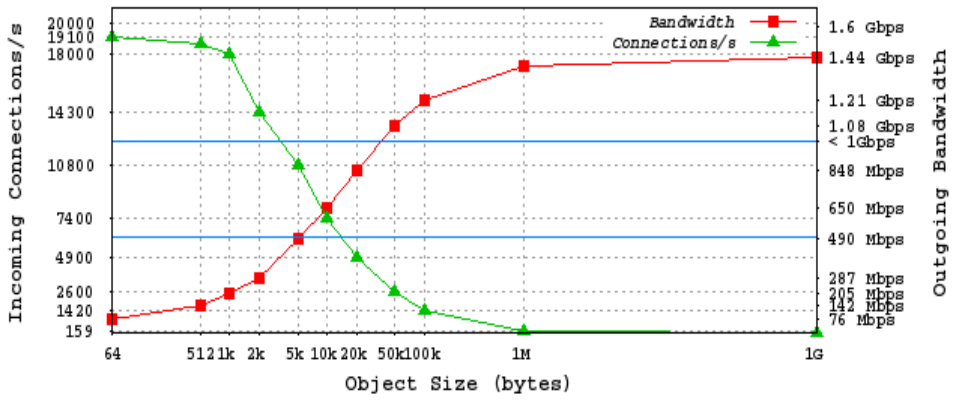
二、当整个 cluster 不能工作了，最后一个失效的 node 必须是第一个重新开始工作的那一个。如果这种情况得不到满足，所有 node 将会为最后一个磁盘 node 的恢复等待 30 秒。如果最后一个离线的 node 无法重新上线，我们可以通过命令 forget_cluster_node 将其从 cluster 中移除 - 具体参考 rabbitmqctl 的使用手册。

8、Haproxy 为 OpenStack 提供负载均衡

8.1、Haproxy 简介

HAProxy 是一款提供高可用性、负载均衡以及基于 TCP（第四层）和 HTTP（第七层）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。

- 免费开源，稳定性也是非常好，这个可通过一些项目可以看出来，单 Haproxy 也跑得不错，稳定性可以与硬件级的 F5 相媲美；
- 根据官方文档，HAProxy 可以跑满 10Gbps-New benchmark of HAProxy at 10 Gbps using Myricom's 10GbE NICs （Myri-10G PCI-Express），这个数值作为软件级负载均衡器是相当惊人的。官方测试的性能情况如下图，



- HAProxy 支持连接拒绝：因为维护一个连接的打开的开销是很低的，有时我们很需要限制攻击蠕虫（attack bots），也就是说限制它们的连接打开从而限制它们的危害。这个已经为一个陷于小型 DDoS 攻击的网站开发了而且已经拯救了很多站点，这个优点也是其它负载均衡器没有的。
- HAProxy 支持全透明代理（已具备硬件防火墙的典型特点）：可以用客户端 IP 地址或者任何其他地址来连接后端服务器. 这个特性仅在 Linux 2.4/2.6 内核打了 cttproxy

补丁后才可以使用. 这个特性也使得为某特殊服务器处理部分流量同时又不修改服务器的地址成为可能。

- HAProxy 现多于线上的 Mysql 集群环境，我们常用于它作为 MySQL（读）负载均衡。
- 自带强大的监控服务器状态的页面，实际环境中我们结合 Nagios 进行邮件或短信报警，这个也是我非常喜欢它的原因之一。
- HAProxy 支持虚拟主机。
- HAProxy 特别适用于那些负载特大的 web 站点， 这些站点通常又需要会话保持或七层处理。HAProxy 运行在当前的硬件上，完全可以支持数以万计的并发连接。并且它的运行模式使得它可以很简单安全的整合进您当前的架构中， 同时可以保护你的 web 服务器不被暴露到网络上。

8.2、Haproxy 配置文件解释

8.2.1、配置文件格式

主要有三类：
一、 最优先处理的命令行参数
二、 global 全局配置段，设置全局配置参数
三、 proxy 配置段，如 default、listen、frontend、backend

8.2.2、时间格式

默认以毫秒为单位，也可以使用其它的时间单位表示

us: 微秒(microseconds)，即 1/1000000 秒；

ms: 毫秒(milliseconds)，即 1/1000 秒；

s: 秒(seconds)；

m: 分钟(minutes)；

h：小时(hours)；

d: 天(days)；

8.2.3、配置选项说明

/etc/haproxy/haproxy.cfg 内容如下：

```
global
    log 127.0.0.1 local0  # log <address> <facility> [max level] 定义全局的 syslog 服务器，最多可以定义两个
    chroot      /var/lib/haproxy  # 修改 haproxy 的工作目录至指定的目录并在放弃权限之前执行 chroot()操作，可以提升 haproxy 的安全级别，不过需要注意的是要确保指定的目录为空目录且任何用户均不能有写权限

    pidfile     /var/run/haproxy.pid # 将所有进程的 pid 写入文件启动进程的用户必须有权限访问此文件
    maxconn     4096  # 最大连接数
    user        haproxy  # 用户
    group       haproxy  # 组
    daemon   # 创建一个进程进入 daemon 模式运行
    stats socket /var/lib/haproxy/stats  # unix socket 文件

defaults
    log global # 采用全局定义的日志
    mode tcp # {tcp|http|health} tcp 是 4 层，http 是 7 层，health 只会返回 ok
    option tcplog # 日志类别
    option dontlognull # 不记录健康检查的日志信息
    retries 3  # 3 次连接失败认为服务不可用，也可以后面再设置
    option redispatch # 对应的服务器宕机后，强制重定向到其它健康的服务器
    maxconn 2000 # 最大连接数
    contimeout 5000 # 连接超时
    clitimeout 50000 # 客户端连接超时
    srvtimeout 50000 # 服务器连接超时
```

```
listen galera-cluster
  bind controller:3306
  balance  source
  server controller1 controller1:3306 check port 4567 inter 2000 rise 2 fall 5
  server controller2 controller2:3306 check port 4567 inter 2000 rise 2 fall 5 backup


listen mongodb-cluster
  bind controller:27017
    balance  source
      server controller1 controller1:27017 check inter 2000 rise 2 fall 5
      server controller2 controller2:27017 check inter 2000 rise 2 fall 5 backup


listen keystone-admin
    bind controller:35357
    balance source
    option   tcpka
    option   httpchk
    option   tcplog
    server controller1 controller1:35357 check inter 10s
       server controller2 controller2:35357 check inter 10s
       server controller3 controller3:35357 check inter 10s

listen   keystone-public
    bind controller:5000
    balance  source
    option   tcpka
    option   httpchk
    option   tcplog
    server controller1 controller1:5000 check inter 10s
    server controller2 controller2:5000 check inter 10s
    server controller3 controller3:5000 check inter 10s


listen glance-registry
    bind controller:9191
    balance source
    option   tcpka
    option   tcplog
    server controller1 controller1:9191 check inter 10s
    server controller2 controller2:9191 check inter 10s
    server controller3 controller3:9191 check inter 10s
```

```
listen  glance-api
    bind controller:9292
    balance  source
    option   tcpka
    option   httpchk
    option   tcplog
    server controller1 controller1:9292 check inter 10s rise 2 fall 5
    server controller2 controller2:9292 check inter 10s rise 2 fall 5
    server controller3 controller3:9292 check inter 10s rise 2 fall 5

listen  cinder-api
    bind controller:8776
    balance  source
    option   tcpka
    option   httpchk
    option   tcplog
    server controller1 controller1:8776 check inter 10s rise 2 fall 5
    server controller2 controller2:8776 check inter 10s rise 2 fall 5
    server controller3 controller3:8776 check inter 10s rise 2 fall 5

#listen  ec2-api
#   bind controller:8773
#   balance  source
#   option   tcpka
#   option   tcplog
#   server controller1 controller1:8773 check inter 10s
#   server controller2 controller2:8773 check inter 10s

listen  nova-compute-api
    bind controller:8774
    balance  source
    option   tcpka
    option   httpchk
    option   tcplog
    server controller1 controller1:8774 check inter 10s
    server controller2 controller2:8774 check inter 10s
    server controller3 controller3:8774 check inter 10s

listen  nova-metadata
    bind controller:8775
    balance  source
    option   tcpka
    option   tcplog
    server controller1 controller1:8775 check inter 10s
```

```
    server controller2 controller2:8775 check inter 10s
    server controller3 controller3:8775 check inter 10s

listen ceilometer_api
    bind controller:8777
    balance source
    server controller1 controller1:8777 check inter 2000 rise 2 fall 5
    server controller2 controller2:8777 check inter 2000 rise 2 fall 5
    server controller3 controller3:8777 check inter 2000 rise 2 fall 5

listen  nova-spice
    bind controller:6082
    balance  source
    option   tcpka
    option   tcplog
    server controller1 controller1:6082 check inter 10s
    server controller2 controller2:6082 check inter 10s
    server controller3 controller3:6082 check inter 10s


listen neutron-server
    bind controller:9696
    balance  source
    option   tcpka
    option   httpchk
    option   tcplog
    server controller1 controller1:9696 check inter 10s
    server controller2 controller2:9696 check inter 10s
    server controller3 controller3:9696 check inter 10s

#listen  vnc
#    bind controller:6080
#    balance  source
#    server controller1 controller1:6080 check inter 10s
#    server controller2 controller2:6080 check inter 10s



listen animbus-dashboard
    bind controller:80
    balance  source
    option   httpchk
    option  tcplog
    server   controller1 controller1:80 check inter 10s
```

```
    server   controller2 controller2:80 check inter 10s
    server   controller3 controller3:80 check inter 10s

#listen rabbitmq
#    bind controller:5673
#    balance roundrobin
#    mode     tcp
#    server   controller1 controller1:5672 check inter 2000 rise 2 fall 3
#    server   controller2 controller2:5672 check inter 2000 rise 2 fall 3

listen memcache
    bind controller:11211
    balance source
    mode     tcp
    server   controller1 controller1:11211 check inter 2000 rise 2 fall 3
    server   controller2 controller2:11211 check inter 2000 rise 2 fall 3 backup
    server   controller3 controller3:11211 check inter 2000 rise 2 fall 3 backup

listen stats                                    # 启用 haproxy 监控功能
    mode         http
    bind         0.0.0.0:8080
    stats        enable
    stats        refresh 30
    stats        hide-version
    stats        uri      /haproxy_stats
    stats        realm        Haproxy\ Status
    stats        auth     admin:admin
    stats        admin        if TRUE
```

8.2.4、配置 haproxy 日志

修改 syslog 配置文件

```
[root@haproxy ~]# vim /etc/sysconfig/rsyslog
# Options for rsyslogd
# Syslogd options are deprecated since rsyslog v3.
# If you want to use them, switch to compatibility mode 2 by "-c 2"
# See rsyslogd(8) for more details
SYSLOGD_OPTIONS="-c 2 -r"
```

增加日志设置

```
[root@haproxy ~]# vim /etc/rsyslog.conf
#增加一行
```

```
Local0.*        /var/log/haproxy.log
```

重启日志服务生效

```
[root@haproxy ~]# service rsyslog restart
```

9、网络部署架构

网络部署采用 neutron flat vlan + dvr vxlan 两种网络模式并存。
目前测试下来，flat vlan 最稳定。
优势：

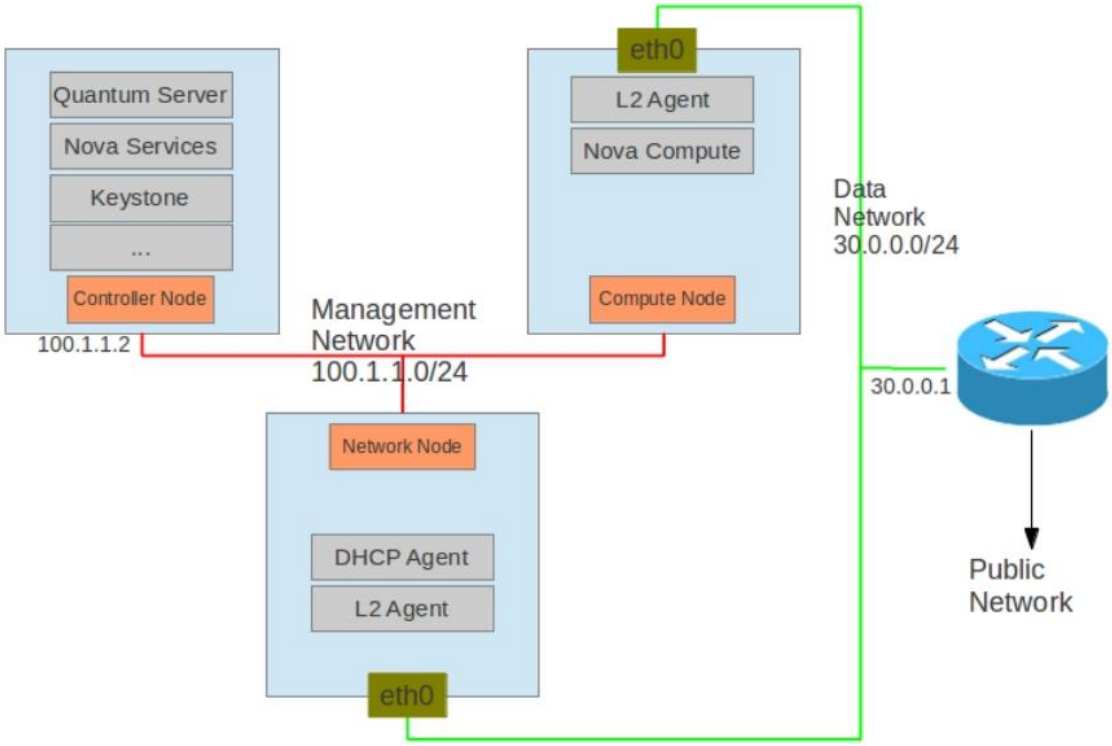
稳定，性能很好。有点类似分布式，虚拟机流量直接从宿主机 compute 节点出去。

劣势：

不用 l3-agent，floating ip 还有 neutron 那些依赖 namespace 的 feature 都不能用。

Juno 现在支持 router ha，本质上是用 keepalived 来实现，有待测试。L3-agent 的 ha 一直都是问题。

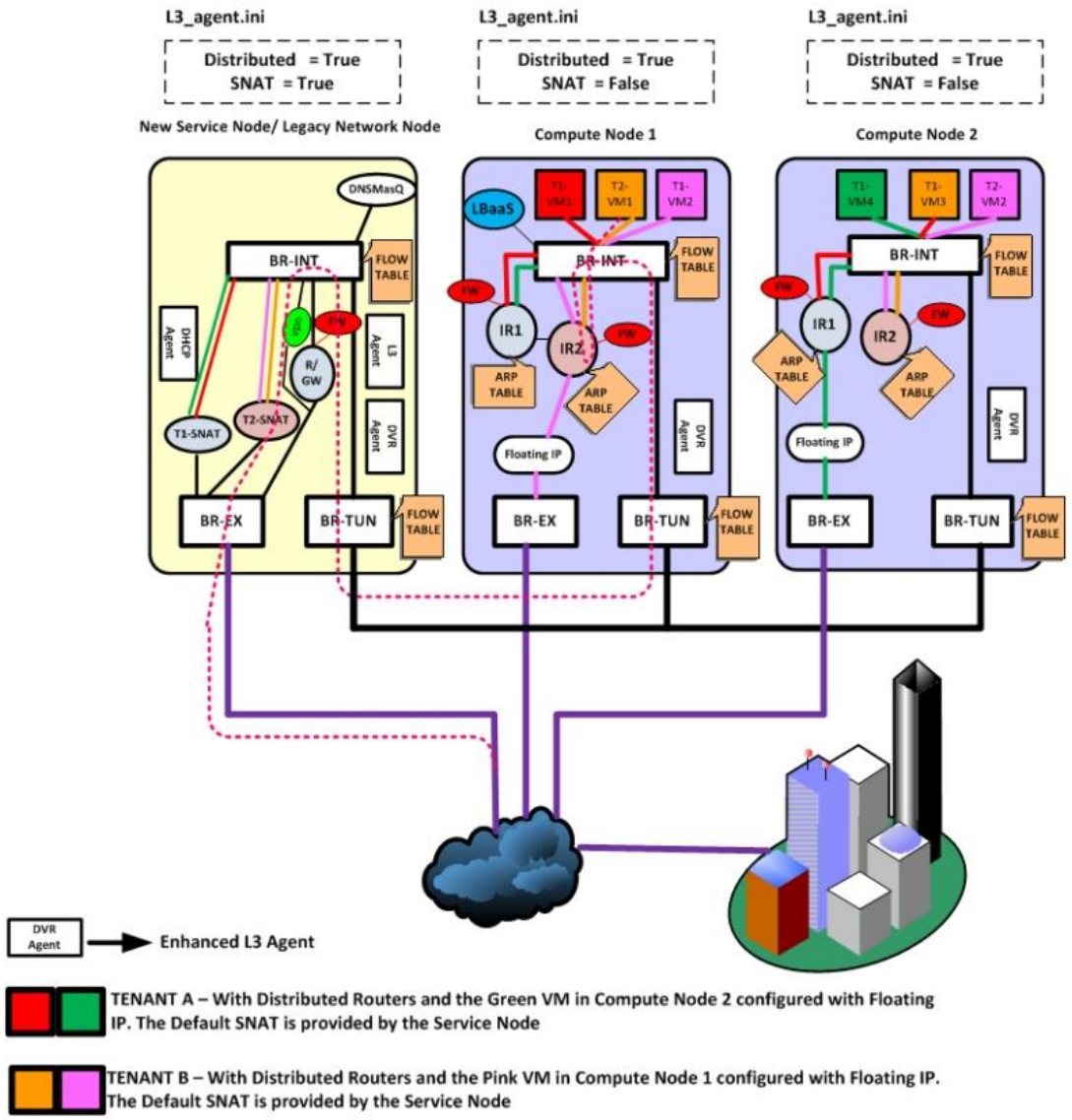
Flat vlan 架构图



Juno Neutron 的 dvr feature 现在还是在 testing 阶段，内部 openstack 平台测试了一段时间，floating ip 相当不稳定，官方声称 kilo 这个 feature 才能正式应用于生产环境。

这里简单介绍下 neutron dvr 下的流量转发

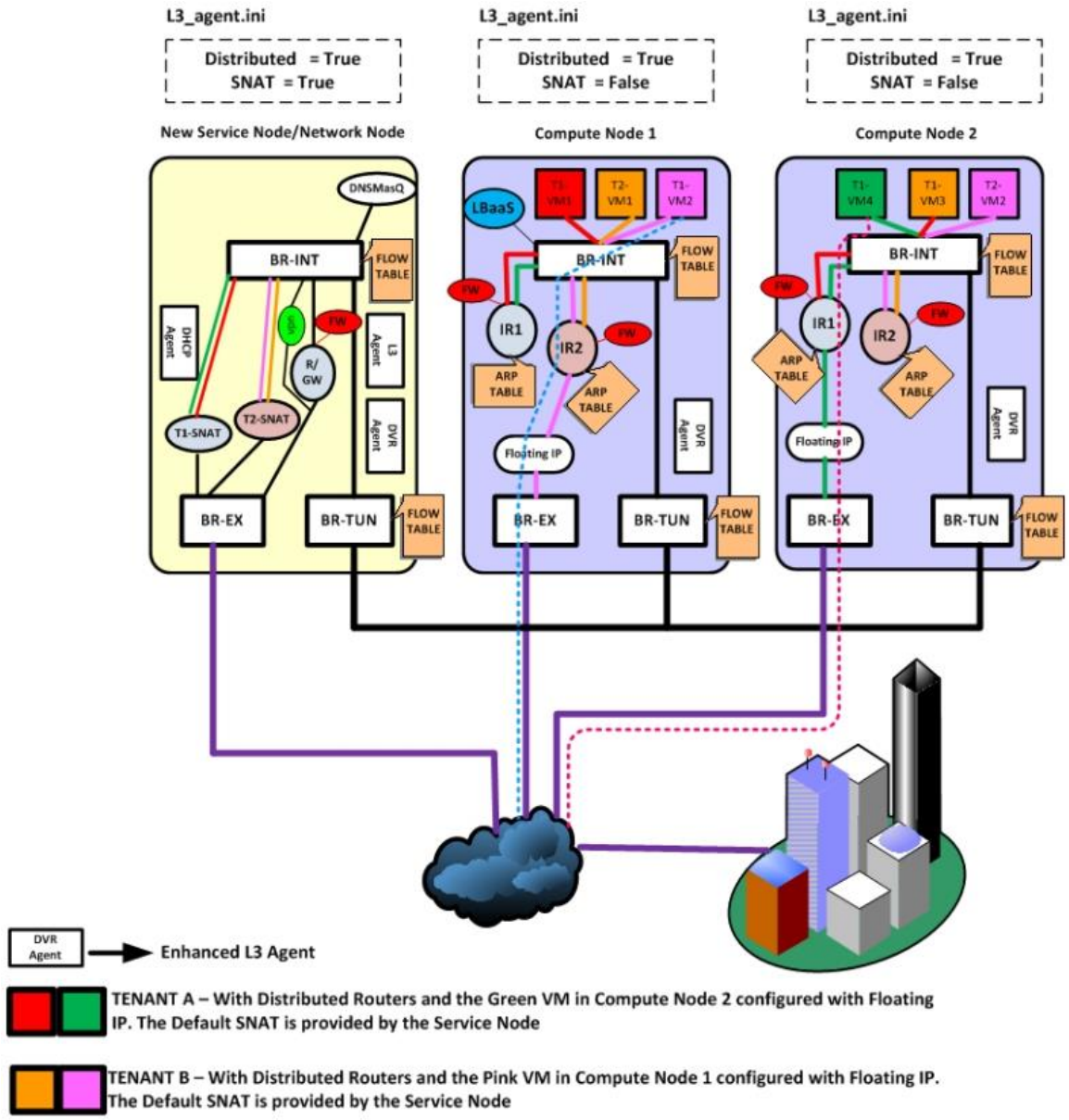
dvr 东西走向图



这是使用了dvr后 ,没有绑定floating ip的虚拟机上外网的流量走势(东西走向)，

最终出网是在网络节点

dvr 南北走向图



绑定 floating ip 的虚拟机也直接从它的宿主机 compute node 出网，不经过网络节点

Neutron l3_agent HA（python 脚本实现）

```
#!/usr/bin/env python
import sys
from neutronclient.v2_0 import client as neutronclient

TENANT_NAME="admin"
USERNAME="admin"
PASSWORD="admin"
AUTH_URL="http://192.168.1.150:35357/v2.0"

neutron = neutronclient.Client(auth_url=AUTH_URL,
                               username=USERNAME,
                               password=PASSWORD,
                               tenant_name=TENANT_NAME)

agents = neutron.list_agents()
alive_l3_agents = []
dead_l3_agents = []

for agent in agents['agents']:
    if agent['binary'] == 'neutron-l3-agent' and agent['alive'] == True:
        alive_l3_agents.append(agent)
    if agent['binary'] == 'neutron-l3-agent' and agent['alive'] != True:
        dead_l3_agents.append(agent)

if len(alive_l3_agents) == 0 :
    print "No active L3"
    sys.exit(1)

if len(dead_l3_agents) == 0 :
    print "No dead L3"
    sys.exit(1)

routers = neutron.list_routers()
dead_routers = []

for dead_l3_agent in dead_l3_agents:
    dead_routers = neutron.list_routers_on_l3_agent(dead_l3_agent['id'])
    for dead_router in dead_routers['routers']:
        neutron.remove_router_from_l3_agent(dead_l3_agent['id'], dead_router['id'])
#       print "remove_router_from_l3_agent : L3 id is %s, router id
is %s" %(dead_l3_agent['id'], dead_router['id'])
        # Currently, only add to the first alive agent
```

```
        neutron.add_router_to_l3_agent(alive_l3_agents[0]['id'],
{"router_id":dead_router['id']})
#                   print   "add_router_to_l3_agent   :   L3   id   is   %s,   router   id
is %s" %(alive_l3_agents[0]['id'], dead_router['id'])
```

Neutron l3_agent HA（shell 脚本实现）

```
#!/bin/bash

source /usr/lib/keystonerc

   alive_l3_agent_id=`neutron agent-list | grep "L3 agent" | grep ":-)" | awk '{print
$2}'| head -n 1`
   dead_l3_agent_id_list=`neutron agent-list | grep "L3 agent" | grep "xxx" | awk '{print
$2}'`
#echo $dead_l3_agent_id_list | wc -l
   if [ ! -z ${alive_l3_agent_id} ] && [ ! -z ${dead_l3_agent_id_list} ];then
        for dead_l3_agent_id in ${dead_l3_agent_id_list};do

              dead_router_id_list=`neutron router-list-on-l3-agent
${dead_l3_agent_id} | awk '{print $2}'| egrep -v '(id|^$)'`
              if [ ! -z `echo ${dead_router_id_list} | cut -d ' ' -f 1` ];then
                   for dead_router_id in ${dead_router_id_list};do
                       neutron l3-agent-router-remove ${dead_l3_agent_id}
${dead_router_id} &> /dev/null

                       neutron l3-agent-router-add ${alive_l3_agent_id}
${dead_router_id} &> /dev/null
                   done
              fi
        done
   fi
```