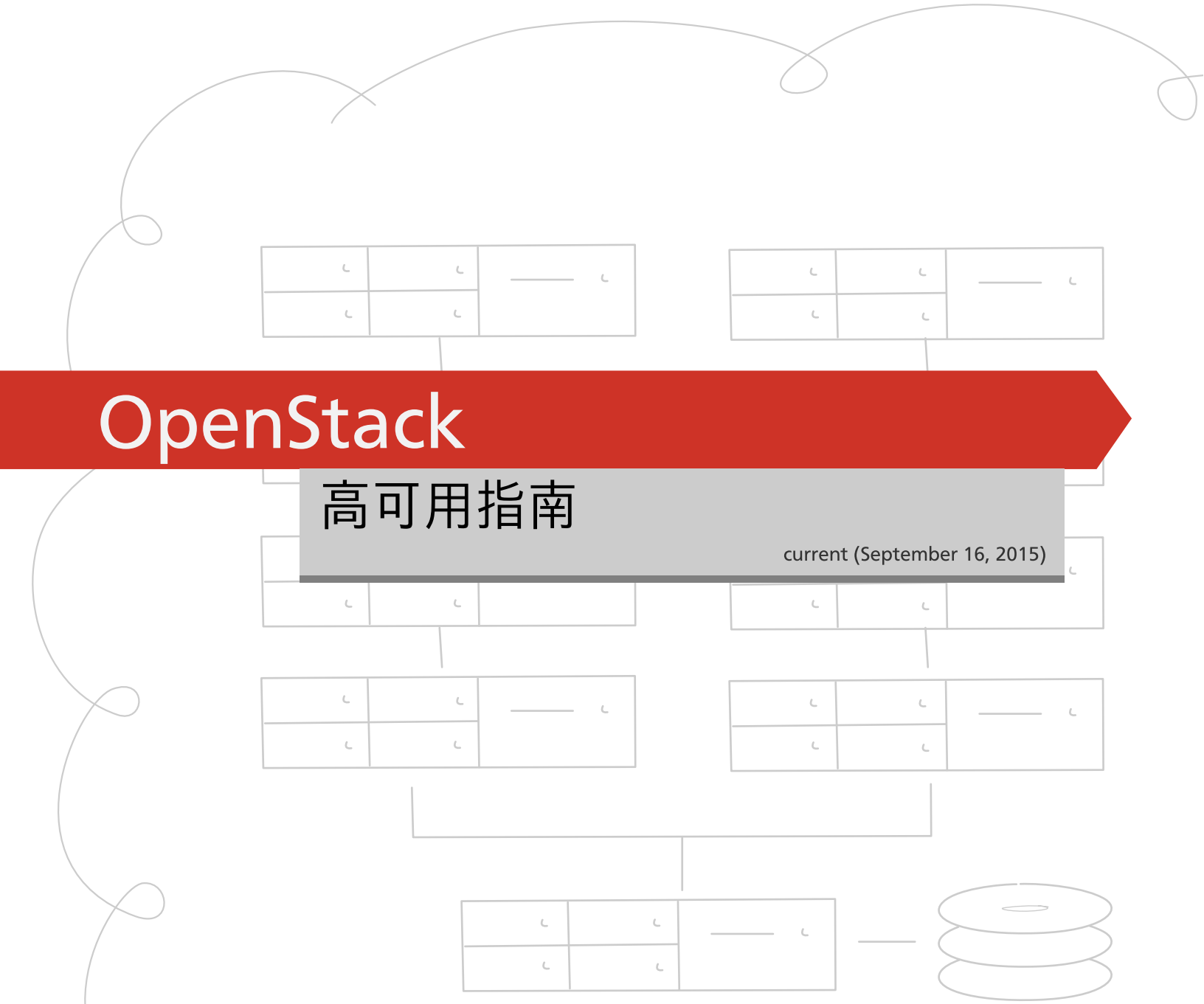

OpenStack

高可用指南

current (September 16, 2015)

docs.openstack.org

OpenStack高可用指南

current (2015-09-16)

摘要

本手册将对如何实现 OpenStack 各服务的高可用进行说明。

目录

Preface

Conventions

The OpenStack documentation uses several typesetting conventions.

Notices

Notices take these forms:

注意

A handy tip or reminder.

重要

Something you must be aware of before proceeding.

警告

Critical information about the risk of data loss or security issues.

Command prompts

$ prompt — Any user, including the root user, can run commands that are prefixed with the $ prompt.

prompt — The root user must run commands that are prefixed with the # prompt. You can also prefix these commands with the sudo command, if available, to run them.

Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

Revision Date	Summary of Changes
April 30, 2015	・本教程针对Kilo版本有多项更新，比如添加MariaDB，更新MySQL信息，corosync和网络更新。
October 17, 2014	・本手册根据 OpenStack 文档规范进行了修订，改正了不少细微的错误。
May 16, 2014	・转换为DocBook格式。
April 17, 2014	・清理一些拼写错误，相对于 Icehouse 版本没有大的改动。
January 16, 2012	・本指南将介绍如何安装控制节点和计算节点
May 24, 2012	・开始主干指定。

第 1 章 OpenStack高可用介绍

目录

实现系统高可用是为了减少以下 2 种异常情况：

系统停机　面向客户的服务无法正常工作的时间超出服务承诺的上限。

数据丢失　意外发生的数据删除和数据损坏。

大多数的高可用系统只能在发生单一故障的情况下为降低停机时间和避免数据丢失提供保障。但是用户也期望高可用系统同样能够处理由单一故障演变为一系列连锁故障的情况。

在高可用系统中，最基本的原则是排除单点故障。所谓单点故障，是指系统中的某一单独部件（硬件设备或者软件组件），当它发生故障时会导致系统停机或者数据丢失。可以通过检查下列系统组成部分中是否包含冗余机制来消除单点故障：

- 网络设备，如交换机、路由器；
- 应用程序以及服务自动迁移工具；
- 存储设备；
- 辅助设施，如电源、空调、防火等；

当某个组件崩溃后， 备用系统必须把该组件的工作负载承接过来。绝大多数的高可用系统会尽快的把崩溃的组件替换掉用以保证系统的必要冗余安全性。该处理方案中系统处在保护降级状态中的时间要尽可能的少

大多数高可用系统都无法应对发生一连串不相关故障的情况，此时保护数据优先于保证系统的高可用性。

通常，高可用系统能够保证 99.99% 的在线时间，相当于一年之中发生系统故障的累积时间不超过 1 个小时。要达到这一目标，高可用系统应将故障恢复时间控制在 1 ～ 2 分钟之内甚至更短。

OpenStack 的基础服务，在合理配置的情况下，能够满足上述 99.99% 在线时间的高可用性要求。但是 OpenStack 不能保证单个虚拟机实例的 99.99% 在线时间。

避免单点故障的方法根据该服务是否属于无状态类型而有所不同。

无状态和有状态服务

无状态服务是指，当该服务对一个请求作出响应之后，不会再有任何相关操作。实现无状态服务的高可用，只需要同时运行该服务的多个实例，并保证这些实例的负载均衡即

可。OpenStack 中无状态的服务包括： nova-api, nova-conductor, glance-api, keystone-api, neutron-api and nova-scheduler 。

有状态服务，是指客户端发送的后续请求依赖于之前相关请求的处理结果。由于单独一项操作可能涉及若干相关请求，有状态服务相对难于管理，只是通过多个实例和负载均衡无法实现高可用。例如，如果每次访问 Horizon 时都是打开一个全新的页面（之前的操作都消失了），对于用户来说是毫无意义的。OpenStack 中有状态服务包括 OpenStack 数据库和消息队列。

实现有状态服务高可用的方案有“主/从”和“主/主” 2 种模式。

主/从

在“主/从”模式中，当系统中的资源失效时，新的资源会被激活，替代失效部份继续提供服务。例如，在 OpenStack 集群中，可以在主数据库之外维护一套灾备数据库，当主数据库发生故障时，激活灾备数据库可以保证集群继续正常运行。

通常情况下，针对无状态服务实现“主/从”模式的高可用是维护该服务的一个冗余实例，在必要时，这一实例会被激活。客户端的请求统一发送到一个虚拟的 IP 地址（该地址指向实际的后端服务），这样当发生切换时，后端服务和客户端几乎不需要进行任何改动。

有状态服务的“主/从”模式高可用则是维护一套额外的备份资源，当故障发生时，可以直接替代失效部份继续工作。单独的应用程序（如 Pacemaker 、Corosync 等）负责监控各项服务，并在发生故障时激活备份资源。

主/主

在“主/主”模式中，服务的冗余实例和主实例会同时工作。这样主实例发生故障，不会对用户产生影响，因为冗余实例一直处于在线状态，后续客户端的请求直接由冗余实例处理，而主实例的故障恢复可以同步进行。

通常，无状态服务“主/主”模式的高可用会维护冗余的服务实例，同时通过虚拟 IP 地址以及负载调度程序（如 HAProxy ）对客户端的请求进行负载均衡。

而有状态服务的“主/主”模式高可用则是维护多个完全相同的冗余实例。例如，更新其中一个数据库实例时，其它所有实例都会被更新。这样客户端发送给其中一个实例的请求相当于发给了所有实例。负载调度程序管理客户端和这些实例之间的连接，确保请求发送到正常运行的服务实例。

上面提到的是较为常见的高可用实现方案，但是并非只有这些方案可以实现系统的高可用。基本原则只是保证服务冗余和可用，具体如何实现则是视需求而定的。本文档会提供如何实现高可用系统的一些通用建议。

部分 I. 主/从模式高可用集群

目录

第 2 章 Packmaker 集群

目录

OpenStack infrastructure high availability relies on the Pacemaker cluster stack, the state-of-the-art high availability and load balancing stack for the Linux platform. Pacemaker is storage and application-agnostic, and is in no way specific to OpenStack.

Pacemaker relies on the Corosync messaging layer for reliable cluster communications. Corosync implements the Totem single-ring ordering and membership protocol. It also provides UDP and InfiniBand based messaging, quorum, and cluster membership to Pacemaker.

Pacemaker 通过资源代理程序(RAs)（默认提供了 70 多种）和应用程序进行交互，在 Pacemaker 集群中应用第三方资源代理程序(RAs)也非常容易。OpenStack 高可用配置中使用了 Pacemaker 自带的资源代理程序RAs（如 MySQL 数据库服务、虚拟 IP 地址等）、已有的第三方资源代理程序(如 RabbitMQ 服务)以及 OpenStack 资源代理程序RAs(如 OpenStack 身份认证服务、磁盘镜像服务)。

安装软件包

Pacemaker 中的节点服务器之间必须通过 Corosync 建立集群通信，需要安装以下软件包(以及它们的依赖软件包，通常软件包管理器将自动所有依赖软件包)：

- pacemaker （说明：crm 命令行工具需要另外单独下载。）
- crmsh
- corosync
- cluster-glue
- fence-agents （说明：只针对 Fedora 发行版；其它 Linux 发行版都使用 cluster-glue 软件包中的 fence 资源代理）
- resource-agents

Corosync 基本配置

除了安装 Corosync 之外，还需要创建一个配置文件 /etc/corosync/corosync.conf 。Corosync 可以使用组播或者单播 IP 地址进行集群心跳通信。

配置 Corosync 使用组播

大多数 Linux 发行版都会中在 Corosync 软件包中附带一份配置示例（corosync.conf.example）。Corosync示例配置文件如下：

Corosync 配置文件（corosync.conf）.

```
totem {
    version: 2

    # Time (in ms) to wait for a token ❶
    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 ❷
    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. ❸
    rrp_mode: active

    # The following is a two-ring multicast configuration. ❹
    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 ❺
    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
    }}
```

❶ token 是时间，单位为毫秒，在该配置项指定的时间内， Corosync 令牌应该完成在回环网络中的传输。如果令牌传输超时就会被丢弃，而一台节点服务器连续出现 token_retransmits_before_loss_const 令牌失效，将会被认为是无效节点。也就是说，一台节点服务器最长的无响应时间不能超对 token × token_retransmits_before_loss_const 的乘积（单位毫秒），否则会被认为是无效节点。token 的默认值是 1000 （即 1 秒），同时默认的重试次数为 4 。默认配置的目标是尽量缩短故障恢复时间，但是可能出现较多的 “false alarm” 提醒，发生短期的网络故障时也有可能导致失效切换。本处示例中的配置参数更安全一些，但是失效切换的时间会长一些。

❷ 当启用 secauth 时，Corosync 节点之间通信时会使用一个 128 位的密钥进行双向认证。密钥存放在 /etc/corosync/authkey 文件中，可以通过 corosync-keygen 命令生成。启用 secauth 后，集群通信数据也会进行加密。

❸ Cororsync 可以使用冗余的心跳网络（即多个 interface 配置），但是必须同时将 RRP 模式设置为除 none之外的其它值，建议使用 active 模式。

❹ 在推荐的网络接口配置中有几件事需要注意：

- 所有心跳网络的 ringnumber 配置不能重复，最小值为 0 。
- bindnetaddr 是心跳网卡 IP 地址对应的网络地址。示例中使用了两个子网掩码为 /24 的 IPv4 网段。
- Multicast groups (mcastaddr) must not be reused across cluster boundaries. In other words, no two distinct clusters should ever use the same multicast group. Be sure to select multicast addresses compliant with RFC 2365, "Administratively Scoped IP Multicast".
- Corosync 通信使用 UDP 协议，端口为 mcastport （接收数据）和 mcastport - 1 （发送数据）。配置防火墙时需要打开这两个端口。

❺ pacemaker 对应的 service 配置段，可以放在 corosync.conf ，也可以单独作为一个配置文件 /etc/corosync/service.d/pacemaker 。

注意

如果是在 Ubuntu 14.04 系统中运行 Corosync 2，那么应该将 stanza 对应的 service 配置段删除或者全部注释，以确保 Pacemaker 可以启动。

corosync.conf （以及 authkey ，如果 secauth 启用）一旦创建，则必须在各节点服务器之间保持同步。

配置 Corosync 使用单播

某些环境中可能不支持组播。这时应该配置 Corosync 使用单播，下面是使用单播的 Corosync 配置文件的一部分：

Corosync 配置文件片断： （ corosync.conf ）.

```
totem {
    #...
    interface {
        ringnumber: 0
        bindnetaddr: 192.168.42.0
        broadcast: yes ❶
        mcastport: 5405
    }
    interface {
        ringnumber: 1
        bindnetaddr: 10.0.42.0
        broadcast: yes
        mcastport: 5405
    }
    transport: udpu ❷
}

nodelist { ❸
    node {
        ring0_addr: 192.168.42.1
        ring1_addr: 10.0.42.1
        nodeid: 1
    }
    node {
        ring0_addr: 192.168.42.2
        ring1_addr: 10.0.42.2
        nodeid: 2
    }
}
#...
```

❶ 如果将 broadcast 设置为 yes ，集群心跳将通过广播实现。设置该参数时，不能设置 mcastaddr 。

❷ transport 配置项决定集群通信方式。要完全禁用组播，应该配置单播传输参数 udpu 。这要求将所有的节点服务器信息写入 nodelist ，也就是需要在配署 HA 集群之前确定节点组成。配认配置是 udp 。通信方式类型还支持 udpu 和 iba 。

❸ 在 nodelist 之下可以为某一节点设置只与该节点相关的信息，这些设置项只能包含在 node 之中，即只能对属于集群的节点服务器进行设置，而且只应包括那些与默认设置不同的参数。每台服务器都必须配置 ring0_addr 。

注意

对于 UDPU ，每台节点服务器都需要配置所属的传输组。

可用的节点服务器配置项有：

The ringX_addr specifies the IP address of one of the nodes. X is the ring number.

The nodeid configuration option is optional when using IPv4 and required when using IPv6. This is a 32-bit value specifying the node identifier delivered to the cluster

membership service. When using IPv4, this defaults to the 32-bit IP address to which the system is bound with the ring identifier of 0. The node identifier value of zero is reserved and should not be used.

Set up Corosync with votequorum library for a full-size cluster

This section describes a full-size cluster configuration with three or more members which is appropriate for production deployments. For a two-node configuration that can be used for demostrations and testing, please go to the next section.

Votequorum library is a part of the Corosync project. It provides an interface to the vote-based quorum service and it must be explicitly enabled in the Corosync configuration file. The main role of the votequorum library is to avoid split-brain situations, but it also provides a mechanism to:

- 查询quorum状态
- 获取到quorum 服务的已知节点列表
- 获取quorum状态改变的通知
- 改变分配给节点的投票数
- 改变一个集群合法的期望投票数
- 在节点停止服务期间，连接一个额外的quorum设备来使小集群仍然处于合法状态

Votequorum library has been created to replace and eliminate from advanced cluster configurations qdisk, disk-based quorum daemon for CMAN.

Votequorum服务在Corosync中配置.

```
quorum {
        provider: corosync_votequorum ❶
        expected_votes: 7 ❷
        wait_for_all: 1 ❸
        last_man_standing: 1 ❹
        last_man_standing_window: 10000 ❺
    }
```

❶ 提供者corosync_votequorum 启用votequorum库，这是唯一必须的选项。
❷ 集群拥有expected_votes7个节点(每个节点有1票)才能完整使用，quorum：4。如nodelist中指定的节点列表，expected_votes被忽略。
❸ wait_for_all当启动集群(所有节点关机)时，第一次它将拥有集群quorum直到所有节点在线和加入集群(Corosync 2.0新增)。
❹ last_man_standing enable Last Man Standing (LMS) feature (disabled by default: 0). If a cluster is on the quorum edge (expected_votes: 7; online nodes: 4) for time longer than configured in last_man_standing_window, the cluster can recalculate quorum and continue operating even if the next node will be lost. This logic is repeated until the number of online nodes in the cluster reach 2. In order to allow cluster step down from 2 members to only 1, what is not recommended auto_tie_breaker option needs to be set.
❺ last_man_standing_window is time required to recalculate the quorum after one or most hosts have been lost from the cluster. To do the new quorum recalculation, the

cluster needs to have quorum at least for last_man_standing_window, time in [ms] (default: 10000ms).

Set up Corosync with votequorum library for two-host clusters

The two-node cluster configuration is a special case that Pacemaker supports for demonstration and testing; it is a special feature of the votequorum library and is not recommended for production environments.

Multicast votequorum service configuration for two-host Corosync cluster.

```
quorum {
     provider: corosync_votequorum ❶
     expected_votes: 2
     two_node: 1 ❷
    }
```

❶ corosync_votequorum enables votequorum provider library.
❷ Put the cluser into two-node operational mode (default: 0).

注意

Setting two_node to 1 enables wait_for_all by default. It is still possible to override wait_for_all by explicitly setting it to 0. If more than 2 nodes join the cluster, the two_node option is automatically disabled.

重要

Disabling wait_for_all in a two-node cluster may be dangerous as it can lead to a situation (stonith deathmatch) where each node comes up, assumes the other is down, and fences peer in order to safely start clustered services. The peer eventually comes up and repeats the process until the underlying fault is rectified.

Unicast (UDP) votequorum service configuration for two-host Corosync cluster.

```
quorum {
     provider: corosync_votequorum ❶
     two_node: 1 ❷
    }

    nodelist { ❸
     node {
       ring0_addr: 192.168.1.1
     }
     node {
       ring0_addr: 192.168.1.2
     }
    }
```

❶ corosync_votequorum enables votequorum provider library.
❷ For the cluster to work with two members only two_node artificially sets quorum below mathematical majority.

❸ Unicast Corosync configuratrion requires nodelist option to explicitly provide a list of cluster members.

启动Corosync

Corosync 启动方法和普通的系统服务没有区别，根据 Linux 发行版的不同，可能是 LSB init 脚本、upstart 任务、systemd 服务。不过习惯上，都会统一使用 corosync 这一名称：

- /etc/init.d/corosync start （LSB）
- service corosync start （LSB，另一种方法）
- start corosync （upstart）
- systemctl start corosync （systemd）

使用以下两个工具检查 Corosync 连接状态。

corosync-cfgtool ，执行时加上 -s 参数，可以获取整个集群通信的健康情况：

```
# 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 命令可以列出 Corosync 集群的成员节点列表：

```
# 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
```

status=joined标示着每一个集群节点成员。

注意

如果使用 Corosync v2 版本，请使用 corosync-objctl 命令的替代命令 corosync-cmapctl 。

启动 Pacemaker

Corosync 服务启动之后，一旦各节点正常建立集群通信，就可启动 pacemakerd （ Pacemaker 主进程）：

- /etc/init.d/pacemaker start （LSB）
- service pacemaker start （LSB，另一种方法）

・start pacemaker （upstart）

・systemctl start pacemaker （systemd）

Pacemaker 服务启动之后，会自动建立一份空白的集群配置，不包含任何资源。可以通过 crm_mon 工具查看 Packemaker 集群的状态：

```
============
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 ]
```

设置集群基本属性

Pacemaker 启动之后，建议首先对集群基本属性进行配置。配置时，首先执行 crm 命令，然后输入 configure 进入配置菜单。也可以执行 crm configure 命令直接进入 Pacemaker 配置菜单。

然后，设置下列属性：

```
property no-quorum-policy="ignore"   #❶
 pe-warn-series-max="1000"        #❷
 pe-input-series-max="1000"
 pe-error-series-max="1000"
 cluster-recheck-interval="5min"   #❸
```

❶ 对于 2 个节点的 Pacemaker 集群，集群属性 no-quorum-policy="ignore" 是必须配置的，因为：如果强制集群满足合法节点数要求，当 其中一个节点失效时，剩下的一个节点无法达到集群多数节点在线的要求，从而不能接管原来运行在失效节点上的集群资源。这种情况下，解决方法只能是忽略集群合法节点数要求。但是这一属性只能用于 2 个节点的集群，对于 3 个节点及以上的集群来说，是不应该配置该属性的。需要注意的是，2 个节点的集群配置该属性之后，会出现脑裂（ split-brain）的风险，这是因为当 2 个节点都在线但是互相无法通信时，2 个节点都认为对方出现故障，从而尝试接管对方的集群资源。因此建议布署 3 个节点及以上的集群。

❷ 将 pe-warn-series-max 、 pe-input-series-max 以及 pe-error-series-max 设置为 1000，是为了让 Pacemaker 保存更多 Policy Engine 的处理输入、错误以及警告信息。这些历史记录对排除集群故障会有很大帮忙。

❸ Pacemaker 处理集群状况时使用事件驱动机制。但是某些 Pacemaker 操作只会在固定的时间间隔触发。该时间间隔可以配置，cluster-recheck-interval，默认值是 15 分钟。针对特定的集群，可以适当缩短这一间隔，如 5 分钟或者 3 分钟。

作完这些改变后，可以提交更新配置。

第 3 章 OpenStack 控制服务 HA 集群配置

目录

OpenStack 控制服务运行在管理网络上，可以和其它任何 OpenStack 服务进行交互。

高可用的MySQL

MySQL是许多OpenStack服务所使用的默认数据库服务。确保MySQL服务高可用涉及到：

- 配置被MySQL使用的DRBD设备
- 配置MySQL使用位于DRBD设备上的数据目录
- 选择并绑定一个可以在各集群节点之间迁移的虚拟 IP 地址（即 VIP ）
- 配置 MySQL 监听那个 IP 地址
- 使用 Pacemaker 管理上述所有资源，包括 MySQL 数据库

注意

MySQL/Galera is an alternative method of configuring MySQL for high availability. It is likely to become the preferred method of achieving MySQL high availability once it has sufficiently matured. At the time of writing, however, the Pacemaker/DRBD based approach remains the recommended one for OpenStack environments.

配置 DRBD

基于 Pacemaker 的 MySQL 数据库需要一个 DRBD 设备，并将之挂载到 /var/lib/mysql 目录。在示例中，DRBD 资源被简单命名为 mysql：

mysql DRBD 资源配置文件（ /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;
 }
}
```

该资源使用了一块本地磁盘（DRBD 术语为“后端设备”， a backing device），该磁盘在两台节点服务器（ node1 ， node2 ）上对应相同的设备文件 —— /dev/data/mysql ，一 般情况下，该磁盘是一个专门为此配置的 LVM 逻辑卷。meta-disk 配置项的值是 internal，指的是 DRBD 元数据保存在后端设备的结尾（即元数据和实际数据保存在同一存储设备上）。设备数据同步通过 10.0.42.100 和 10.0.42.254 完成，使用 TCP 7700 端口。当 DRBD 资源激活之后，系统中将对应生成一个 DRBD 设备文件，次设备号为 0 ，设备文件是 /dev/drbd0 。

Enabling a DRBD resource is explained in detail in the DRBD User's Guide. In brief, the proper sequence of commands is this:

```
# drbdadm create-md mysql❶
# drbdadm up mysql❷
# drbdadm -- --force primary mysql❸
```

❶ 初始化 DRBD 元数据，并在 /dev/data/mysql 上初始元数据集。两台节点服务器上都必须完成该操作。

❷ 创建 /dev/drbd0 设备文件，将指定的后端存储设备附加到该 DRBD 资源，同时建立所有节点服务器之间的通信连接。两台节点服务器上都必须完成该操作。

❸ Kicks off the initial device synchronization, and puts the device into the primary (readable and writable) role. See Resource roles (from the DRBD User's Guide) for a more detailed description of the primary and secondary roles in DRBD. Must be completed on one node only, namely the one where you are about to continue with creating your filesystem.

创建文件系统

当 DRBD 资源已经激活并处于 “primay” 角色(可能初始化同步正在进行，还没有完成)，可以开始创建文件系统。 XFS 由于拥有日志系统，分配效率高，性能好等优点，是建议选择的文件系统。

```
# mkfs -t xfs /dev/drbd0
```

也可以使用 DRBD 设备的另外一个名称，该名称有解释含义，更容易记忆：

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

完成后，可以安全地把设备变回 “secondary” 角色。已经启动的设备同步将在后台继续进行：

```
# drbdadm secondary mysql
```

MySQL 针对 Pacemaker HA 架构的前期准备

要通过 Pacemaker 实现 MySQL 高可用，必须首先保证 MySQL 的数据文件使用 DRBD 存储设备。如果使用已有的数据库，最简单的方法是将已有的数据库文件 /var/lib/mysql 迁移到 DRBD 设备之上的文件系统中。

警告

下面的步骤在必须关闭MySQL数据库服务器之后进行。

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

```
# umount /mnt
```

如果使用全新的数据库，可以执行 mysql_install_db 命令：

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

这里列出的这些步骤只需要在其中一个集群节点上操作一遍即可。

在 Pacemaker 中添加 MySQL 资源

现在可以在 Pacemaker 中填加 MySQL 相关资源。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

```
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=192.168.42.101"
  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
```

这个配置创建

- p_ip_mysql，MySQL 服务将会使用的虚拟 IP 地址（192.168.42.101），
- p_fs_mysql，Pacemaker 管理的文件系统，挂载点为 /var/lib/mysql，该文件系统将在运行 MySQL 服务的节点上挂载，
- ms_drbd_mysql，管理 DRBD 设备的主/从资源，

・资源组以及顺序、协同约束条件，会确保资源在正确的节点安照正确次序启动。

crm configure 支持批量输入的配置，因此可以直接复制上述配置示例，粘贴到实际的 Pacemaker 配置环境，然后根据具体情况调整配置项。例如，在 crm configure 菜单中，输入 edit p_ip_mysql，可以对虚拟 IP 地址资源进行编辑。

配置完成后，在 crm configure 菜单下输入 commit 提交所有配置变更。随后 Pacemaker 会其中一台节点服务器上启动 MySQL 服务（包括所有相关资源）。

配置 OpenStack 各服务使用高可用的 MySQL 数据库

现在可以将各 OpenStack 服务配置文件中使用物理 IP 地址的 MySQL 访问方式，更改为访问高可用、使用虚拟 IP 地址的 MySQL 服务。

以 OpenStack 镜像服务为例，如果 MySQL 数据库的虚拟 IP 地址是 192.168.42.101，那么在 OpenStack 镜像服务的配置文件（ glance-registry.conf ）中应该使用如下配置：

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

除此之外，不需要更改其它配置。如果运行数据库服务的节点发生故障，MySQL 服务会自动迁移到其它节点，OpenStack 服务会经历短暂的临时 MySQL 中断，和偶然发生的网络中断类似，之后会继续正常运行。

高可用的 RabbitMQ

RabbitMQ 是多数 OpenStack 服务的默认 AMQP 服务程序。实现 RabbitMQ 的高可用包括以下步骤：

・为 RabbitMQ 配置一个 DRBD 设备

・配置 RabbitMQ 使用建立在 DRBD 设备之上的数据目录，

・选择并绑定一个可以在各集群节点之间迁移的虚拟 IP 地址 （即 VIP ），

・配置 RabbitMQ 监听该 IP 地址，

・使用 Pacemaker 管理上述所有资源，包括 RabbitMQ 守护进程本身。

注意

Active-active mirrored queues is another method for configuring RabbitMQ versions 3.3.0 and later for high availability. You can also manage a RabbitMQ cluster with active-active mirrored queues using the Pacemaker cluster manager.

配置 DRBD

基于 Pacemaker 的 RabbitMQ 服务需要一个 DRBD 设备，并将之挂载到 /var/lib/rabbitmq 目录。在示例中，DRBD 资源被简单命名为 rabbitmq：

rabbitmq DRBD 资源配置文件（ /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;
 }
}
```

该资源使用了一块本地磁盘（DRBD 术语为“后端设备”， a backing device），该磁盘在两台节点服务器（ node1 ， node2 ）上对应相同的设备文件 ——/dev/data/rabbitmq ，一 般情况下，该磁盘是一个专门为此配置的 LVM 逻辑卷。meta-disk 配置项的值是 internal，指的是 DRBD 元数据保存在后端设备的结尾（即元数据和实际数据保存在同一存储设备上）。设备数据同步通过 10.0.42.100 和 10.0.42.254 完成，使用 TCP 7701 端口。当 DRBD 资源激活之后，系统中将对应生成一个 DRBD 设备文件，次设备号为 1 ，设备文件是 /dev/drbd1 。

Enabling a DRBD resource is explained in detail in the DRBD User's Guide. In brief, the proper sequence of commands is this:

```
# drbdadm create-md rabbitmq❶
# drbdadm up rabbitmq❷
# drbdadm -- --force primary rabbitmq❸
```

❶ 初始化 DRBD 元数据，并在 /dev/data/rabbitmq 上初始元数据集。两台节点服务器上都必须完成该操作。

❷ 创建 /dev/drbd1 设备文件，将指定的后端存储设备附加到该 DRBD 资源，同时建立所有节点服务器之间的通信连接。两台节点服务器上都必须完成该操作。

❸ Kicks off the initial device synchronization, and puts the device into the primary (readable and writable) role. See Resource roles (from the DRBD User's Guide) for a more detailed description of the primary and secondary roles in DRBD. Must be completed on one node only, namely the one where you are about to continue with creating your filesystem.

创建文件系统

当 DRBD 资源已经激活并处于 “primay” 角色（可能初始化同步正在进行，还没有完成），可以开始创建文件系统。 XFS 由于拥有日志系统，分配效率高，性能好等优点，是建议选择的文件系统：

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

也可以使用 DRBD 设备的另外一个名称，该名称有解释含义，更容易记忆：

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

完成后，可以安全地把设备变回 “secondary” 角色。已经启动的设备同步将在后台继续进行：

```
# drbdadm secondary rabbitmq
```

RabbitMQ 针对 Pacemaker HA 架构的前期准备

要通过 Pacemaker 实现 RabbitMQ 高可用，必须首先保证 RabbitMQ 的文件 .erlang.cookie 不论在 DRBD 设备有没有挂载的情况下，都完全相同。最简单的方法是将其中一台节点服务器

上已经生成的 .erlang.cookie 文件复制到所有其它节点，同时也复制一份到 DRBD 设备上的文件系统之中。

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

在 Pacemaker 中添加 RabbitMQ 资源

现在可以在 Pacemaker 中填加 RabbitMQ 相关资源。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

```
primitive p_ip_rabbitmq 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
```

这个配置创建

- p_ip_rabbitmq，RabbitMQ 服务将会使用的虚拟 IP 地址（192.168.42.100），
- p_fs_rabbitmq，Pacemaker 管理的文件系统，挂载点为 /var/lib/rabbitmq，该文件系统将在运行 RabbitMQ 服务的节点上挂载，
- ms_drbd_rabbitmq，管理 DRBD 设备的主/从资源，
- 资源组以及顺序、协同约束条件，会确保资源在正确的节点安照正确次序启动。

crm configure 支持批量输入的配置，因此可以直接复制上述配置示例，粘贴到实际的 Pacemaker 配置环境，然后根据具体情况调整配置项。例如，在 crm configure 菜单中，输入 edit p_ip_rabbitmq，可以对虚拟 IP 地址资源进行编辑。

配置完成后，在 crm configure 菜单下输入 commit 提交所有配置变更。随后 Pacemaker 会其中一台节点服务器上启动 RabbitMQ 服务（包括所有相关资源）。

配置 OpenStack 各服务使用高可用的 RabbitMQ 服务

现在可以将各 OpenStack 服务配置文件中使用物理 IP 地址的 RabbitMQ 访问方式，更改为访问高可用、使用虚拟 IP 地址的 RabbitMQ 服务。

以 OpenStack 镜像服务为例，如果 RabbitMQ 服务的虚拟 IP 地址是 192.168.42.100，那么在 OpenStack 镜像服务的配置文件（ glance-api.conf ）中应该使用如下配置：

```
rabbit_host = 192.168.42.100
```

除此之外，不需要更改其它配置。如果运行数据库服务的节点发生故障，RabbitMQ 服务会自动迁移到其它节点，OpenStack 服务会经历短暂的临时 RabbitMQ 中断，和偶然发生的网络中断类似，之后会继续正常运行。

第 4 章 API 服务节点 HA 集群配置

目录

API 服务节点对外（整个互联网）提供 OpenStack API 接口。它们通过管理网络和 OpenStack 控制节点进行交互。

配置VIP

首先选择并绑定一个可以在各集群节点之间迁移的虚拟 IP 地址 （即 VIP ）。

该配置新建了一个 p_ip_mysql 资源，是 API 节点将会使用的虚拟 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 身份认证服务

OpenStack 身份认证服务被很多其他服务使用。实现 OpenStack 身份认证服务主/从模式的高可用包括以下步骤：

- 配置 OpenStack 身份认证服务监听虚拟 IP 地址，
- 使用 Pacemaker 管理 OpenStack 身份认证服务，
- 配置 OpenStack 服务使用该虚拟 IP 地址。

注意

Here is the documentation for installing OpenStack Identity service.

在 Pacemaker 中添加 OpenStack 认证服务资源

首先，下载 Pacemaker 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 *
```

现在可以在 Pacemaker 中填加 OpenStack 身份认证服务相关资源。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

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

该配置增加 p_keystone 资源，对 OpenStack 身份认证服务进行管理。

crm configure 支持批量输入，因此可以拷贝粘贴上面到现有的 Pacemaker 配置中，然后根据需要再作修改。例如，可以从 crm configure 菜单中进入 edit p_ip_keystone，编辑资源以匹配可供使用的虚拟IP地址。

配置完成后，在 crm configure 菜单下输入 commit 提交所有配置变更。随后 Pacemaker 会其中一台节点服务器上启动 OpenStack 身份认证服务（包括所有相关资源）。

配置 OpenStack 身份认证服务

编辑 OpenStack 身份认证服务的配置文件（ keystone.conf ），调整以下配置项：

对于 Havana 版本：

```
bind_host = 192.168.42.103
```

admin_bind_host选项使您可以通过私有网络进行管理任务。

```
public_bind_host = 192.168.42.103
admin_bind_host = 192.168.42.103
```

为了保证所有的数据都是高可用的，应使用 MySQL 数据库服务（同样也要保证 MySQL 服务是高可用的）：

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

配置 OpenStack 各服务使用高可用的 OpenStack 身份认证服务

其它 OpenStack 服务也相应地使用高可用、使用虚拟 IP 地址的 OpenStack 身份认证服务，而不在使用其所在服务器的物理 IP 地址。

以 OpenStack 计算服务为例，如果 OpenStack 身份认证服务的虚拟 IP 地址是 192.168.42.103，那么在 OpenStack 计算服务的配置文件（ api-paste.ini ）中应该使用如下配置：

```
auth_host = 192.168.42.103
```

在 OpenStack 身份认证服务中需要为该 IP 地址创建对应的服务端点。

注意

如果要同时使用私有和公开的 IP 地址，需要创建两个虚拟 IP 地址资源，并建立类似如下的服务端点：

```
$ 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'
```

如果配置了 Horizon 面板，也需要修改 Horizon 的配置文件 local_settings.py ：

```
OPENSTACK_HOST = 192.168.42.103
```

高可用 OpenStack 镜像 API 服务

Openstack镜像服务提供一个发现、注册和获取虚拟机镜像的服务。为了使OpenStack镜像API服务以主/备模式高可用。您必须：

- 配置OpenStack镜像服务监听虚拟IP地址。
- 使用Pacemaker管理OpenStack集群的镜像API守护进程。
- 配置 OpenStack 服务使用该虚拟 IP 地址。

注意

Here is the documentation for installing the OpenStack Image API service.

在 Pacemaker 中添加 OpenStack 镜像服务资源

首先，下载 Pacemaker 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 *
```

现在可以在 Pacemaker 中填加 OpenStack 镜像服务相关资源。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

```
primitive p_glance-api ocf:openstack:glance-api
params config="/etc/glance/glance-api.conf" os_password="secretsecret"
os_username="admin" os_tenant_name="admin" os_auth_url="http://192.168.42.103:5000/v2.0/"
op monitor interval="30s" timeout="30s"
```

这个配置创建

- p_glance-api 资源，对 OpenStack 镜像服务进行管理。

crm configure 支持批量输入，因此可以拷贝粘贴上面到现有的 Pacemaker 配置中，然后根据需要再作修改。例如，可以从 crm configure 菜单中进入 edit p_ip_glance-api，编辑资源以匹配可供使用的虚拟IP地址。

配置完成后，在 crm configure 菜单下输入 commit 提交所有配置变更。随后 Pacemaker 会其中一台节点服务器上启动OpenStack 镜像服务（包括所有相关资源）。

配置OpenStack镜像服务API

编辑 /etc/glance/glance-api.conf：

```
# We have to use MySQL connection to store data:
```

```
sql_connection=mysql://glance:password@192.168.42.101/glance
# Alternatively, you can switch to pymysql,
# a new Python 3 compatible library and use
# sql_connection=mysql+pymysql://glance:password@192.168.42.101/glance
# and be ready when everything moves to Python 3.
# Ref: https://wiki.openstack.org/wiki/PyMySQL_evaluation

# 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 各服务使用高可用的 OpenStack镜像服务

其它 OpenStack 服务也相应地使用高可用、使用虚拟 IP 地址的 OpenStack 镜像服务，而不在使用其所在服务器的物理 IP 地址。

以 OpenStack 计算服务为例，如果 OpenStack 镜像服务的虚拟 IP 地址是 192.168.42.103，那么在 OpenStack 计算服务的配置文件（ nova.conf ）中应该使用如下配置：

```
[glance]
...
api_servers = 192.168.42.103
...
```

注意

对于 Juno 之前的版本，该配置项对应的是 [DEFAULT] 段之下的 glance_api_servers。

在 OpenStack 身份认证服务中需要为该 IP 地址创建对应的服务端点。

注意

如果要同时使用私有和公开的 IP 地址，需要创建两个虚拟 IP 地址资源，并建立类似如下的服务端点：

```
$ 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 OpenStack 块设备存储服务

使得块存储(cinder)API服务在主/被模式中高可用包括:

- 配置块存储监听于VIP地址
- 使用 Pacemaker 管理 OpenStack 块设备存储服务
- 使用该 IP 地址配置 OpenStack 服务

注意

Here is the documentation for installing Block Storage service.

在 Pacemaker 中添加 OpenStack 块设备存储服务资源

首先，下载 Pacemaker 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 *
```

现在可以在 Pacemaker 中填加 OpenStack 块设备存储服务相关资源。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

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

这个配置创建

- p_cinder-api 资源，对 OpenStack 身份认证服务进行管理。

crm configure 支持批量输入，因此可以拷贝粘贴上面到现有的 Pacemaker 配置中，然后根据需要再作修改。例如，可以从 crm configure 菜单中进入 edit p_ip_cinder-api，编辑资源以匹配可供使用的虚拟IP地址。

配置完成后，在 crm configure 菜单下输入 commit 提交所有配置变更。随后 Pacemaker 会其中一台节点服务器上启动OpenStack 块设备存储存服务（包括所有相关资源）。

配置 OpenStack 块设备存储服务

编辑 /etc/cinder/cinder.conf：

```
# We have to use MySQL connection to store data:
sql_connection=mysql://cinder:password@192.168.42.101/cinder
# Alternatively, you can switch to pymysql,
# a new Python 3 compatible library and use
# sql_connection=mysql+pymysql://cinder:password@192.168.42.101/cinder
# and be ready when everything moves to Python 3.
# Ref: https://wiki.openstack.org/wiki/PyMySQL_evaluation

# We bind Block Storage 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 各服务使用高可用的 OpenStack 块设备存储服务

其它 OpenStack 服务也相应地使用高可用、使用虚拟 IP 地址的 OpenStack 块设备存储服务，而不在使用其所在服务器的物理 IP 地址。

在 OpenStack 身份认证服务中需要为该 IP 地址创建对应的服务端点。

注意

如果要同时使用私有和公开的 IP 地址，需要创建两个虚拟 IP 地址资源，并建立类似如下的服务端点：

```
$ 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 网络服务

OpenStack 网络服务为 OpenStack 集群提供网络基础服务。实现 OpenStack 网络服务主/从模式的高可用包括以下步骤：

- 配置 OpenStack 网络服务监听虚拟 IP 地址，
- 使用 Pacemaker 管理 OpenStack 网络服务，
- 配置 OpenStack 服务使用该虚拟 IP 地址。

注意

Here is the documentation for installing OpenStack Networking service.

在 Pacemaker 中添加 OpenStack 网络服务资源

首先，下载 Pacemaker 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 *
```

现在可以在 Pacemaker 中填加 OpenStack 网络服务相关资源。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

```
primitive p_neutron-server ocf:openstack:neutron-server
params os_password="secretsecret" 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"
```

该配置增加 p_neutron-server 资源，对 OpenStack 网络服务进行管理。

crm configure 支持批量输入，因此可以拷贝粘贴上面到现有的 Pacemaker 配置中，然后根据需要再作修改。例如，可以从 crm configure 菜单中进入 edit p_ip_keystone，编辑资源以匹配可供使用的虚拟IP地址

配置完成后，在 crm configure 菜单下输入 commit 提交所有配置变更。随后 Pacemaker 会其中一台节点服务器上启动OpenStack 网络服务（包括所有相关资源）。

配置 OpenStack 网络服务

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 data:
connection = mysql://neutron:password@192.168.42.101/neutron
# Alternatively, you can switch to pymysql,
# a new Python 3 compatible library and use
# connection=mysql+pymysql://neutron:password@192.168.42.101/neutron
# and be ready when everything moves to Python 3.
# Ref: https://wiki.openstack.org/wiki/PyMySQL_evaluation
```

配置 OpenStack 各服务使用高可用的 OpenStack 网络服务

其它 OpenStack 服务也相应地使用高可用、使用虚拟 IP 地址的 OpenStack 网络服务，而不在使用其所在服务器的物理 IP 地址。

以 OpenStack 计算服务为例，在 OpenStack 计算服务的配置文件（ nova.conf ）中应该使用如下配置：

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

在 OpenStack 身份认证服务中需要为该 IP 地址创建对应的服务端点。

注意

如果要同时使用私有和公开的 IP 地址，需要创建两个虚拟 IP 地址资源，并建立类似如下的服务端点：

```
$ 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 Telemetry 监控代理

Telemtry （ ceilometer ）是 OpenStack 系统中的计量和监控服务。监控中心收集包括虚拟机实例和计算节点在内各种资源的使用情况。

注意

由于轮循模型的限制，代理的单个实例只能轮循给定的测量值列表，除非通过配置多个中心代理分担负载。在示例中，在API节点，我们以主/备模式安装这个服务。

Telemetry 监控中心的主/从模式高可用是通过 Pacemaker 管理其后台守护进程实现。

注意

You will find at this page the process to install the Telemetry central agent.

在 Pacemaker 中添加 Telemetry 监控中心资源

首先，下载 Pacemaker 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 *
```

现在可以在 Pacemaker 中填加 Telemetry 监中心相关资源。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

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

这个配置创建

・p_ceilometer-agent-central, 用来管理 Ceilometer 监控代理服务的资源

crm configure 支持批量输入，因此可以拷贝粘贴上面到现有的 Pacemaker 配置中，然后根据需要再作修改。

配置完成后，在 crm configure 菜单下输入 commit 提交所有配置变更。随后 Pacemaker 会其中一台节点服务器上启动 Telemetry 监控中心（包括所有相关资源）。

配置 Telemetry 监中心

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 data:
sql_connection=mysql://ceilometer:password@192.168.42.101/ceilometer
# Alternatively, you can switch to pymysql,
# a new Python 3 compatible library and use
# sql_connection=mysql+pymysql://ceilometer:password@192.168.42.101/ceilometer
# and be ready when everything moves to Python 3.
# Ref: https://wiki.openstack.org/wiki/PyMySQL_evaluation
```

配置 Pacemaker 资源组

最后，创建一个资源组 group，将所有 API 服务资源和该虚拟 IP 地址联系起来。

```
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
```

第 5 章 网络控制节点 HA 集群配置

目录

网络控制节点运行在管理网络和数据网络中，如果虚拟机实例要连接到互联网，网络控制节点也需要具备互联网连接。

注意

Pacemaker 要求所有的节点服务器使用不用的主机名，而 OpenStack 网络服务调度程序只会关注其中一台（例如，一个虚拟路由器只能在其中一台运行 L3 代理的服务上启动），因此需要对 RA （Pacemaker 资源代理程序） 脚本进行相应修改。比如，所有的节点先各自在配置文件中配置不同的主机名，当 Pacemaker 启动 l3-agent 时，自动将该节点的主机名改为 network-controller，这样所有启动 l3-agent 的节点会使用相同的主机名。

高可用 neutron L3 代理程序

Neutron L3 代理程序负责实现 L3/NAT 转发，让运行在租户网络上的虚拟机实例能够访问外部网络。Neutron L3 代理程序实现高可用也基于 Pacemaker 。

注意

Here is the documentation for installing neutron L3 agent.

在 Pacemaker 中添加 neutron L3 代理程序资源

首先，下载 Pacemaker 资源代理：

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

现在可以在 Pacemaker 中填加 neutron L3 代理程序相关资源。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

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

这个配置创建

- p_neutron-l3-agent资源，对 neutron L3 代理程序进行管理。

crm configure 支持批量输入，因此可以拷贝粘贴上面到现有的 Pacemaker 配置中，然后根据需要再作修改。

配置完成后，在 crm configure 菜单下输入 commit 提交所有配置变更。随后 Pacemaker 会其中一台节点服务器上启动 neutron L3 代理程序（包括所有相关资源）。

注意

这种高可用方案不能实现“零停机”需求，原因是neutron L3 代理程序切换时需要重新创建网络命名空间和虚拟路由器。

高可用 neutron DHCP 代理程序

Neutron DHCP 代理程序使用 dnsmasq (默认情况下)为虚拟机实例分配 IP 地址。Neutron DHCP 代理程序高可用也通过 Pacemaker 实现。

注意

Here is the documentation for installing neutron DHCP agent.

在 Pacemaker 中添加 neutron DHCP 代理程序资源

首先，下载 Pacemaker 资源代理：

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

现在可以在 Pacemaker 中填加 neutron DHCP 代理程序相关资源。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

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

该配置会创建：

- p_neutron-agent-dhcp资源，对neutron DHCP代理程序进行管理。

crm configure 支持批量输入，因此可以拷贝粘贴上面到现有的 Pacemaker 配置中，然后根据需要再作修改。

配置完成后，在 crm configure 菜单下输入 commit 提交所有配置变更。随后 Pacemaker 会其中一台节点服务器上启动 neutron DHCP 代理程序（包括所有相关资源）。

高可用 neutron metadata 代理程序

Neutron metadata 代理程序的作用是让运行在租户网络上的虚拟机实例能够访问 OpenStack 计算服务 API 元数据。Neutron metadata 代理程序的高可用也通过 Pacemaker 实现。

注意

Here is the documentation for installing Neutron Metadata Agent.

在 Pacemaker 中添加 neutron metadata 代理程序资源

首先，下载 Pacemaker 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
```

现在可以在 Pacemaker 中填加 neutron metadata 代理程序相关资源。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

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

这个配置创建

- p_neutron-metadata-agent资源，对 neutron metadata 代理程序进行管理。

crm configure 支持批量输入，因此可以拷贝粘贴上面到现有的 Pacemaker 配置中，然后根据需要再作修改。

配置完成后，在 crm configure 菜单下输入 commit 提交所有配置变更。随后 Pacemaker 会其中一台节点服务器上启动 neutron metadata 代理程序（包括所有相关资源）。

组织网络相关资源

创建一个资源组将所有网络服务相关资源联系起来。执行 crm configure 命令进入 Pacemaker 配置菜单，然后加入下列集群资源：

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

部分 II. 主/主模式高可用集群

目录

第 6 章 数据库

目录

第一步部署数据库，数据库是整个集群的心脏。当讨论高可用(HA)时，我们讨论几个数据库(冗余)和一种保持数据库同步的方式。这种情况下，为了多个主控的同步复制，我们选择Galera插件的 MySQL数据库。

Galera集群插件是基于同步复制的多个主的集群。它是一个高可用解决方案，提供高系统运行时间，无数据丢失和为了增长的横向扩展。

注意

MySQL 并不是唯一的选择，以它作为示例是因为目前已有的 OpenStack 布署案例中，使用 MySQL 作为数据库比较常见。

注意

MySQL+Galera绝对不是唯一种实现数据库高可用的方案。MariaDB Galera 集群(https://mariadb.org/) 和Percona XtraDB集群 (http://www.percona.com/) 都可以和Galera一起使用的。除此之外，你可以使用PostgreSQL，PostgreSQL有自己的数据同步方案，或者使用其他的数据库高可用方案

MySQL 和 Galera

Rather than starting with a vanilla version of MySQL, and then adding Galera, you will want to install a version of MySQL patched for wsrep (Write Set REPlication) from https://launchpad.net/codership-mysql. The wsrep API is suitable for configuring MySQL High Availability in OpenStack because it supports synchronous replication.

Note that the installation requirements call for careful attention. Read the guide https://launchpadlibrarian.net/66669857/README-wsrep to ensure you follow all the required steps.

And for any additional information about Galera, you can access this guide: http://galeracluster.com/documentation-webpages/gettingstarted.html

为已经加上 wresp 补丁的 MySQL 数据库安装 Galera ：

1. 为 Ubuntu 14.04 "trusty"(最常用)设置源。安装软件属性，密钥及源：

```
# apt-get install software-properties-common
# apt-key adv --recv-keys --keyserver hkp://keyserver.ubuntu.com:80 0xcbcb082a1bb943db
# add-apt-repository 'deb http://ams2.mirrors.digitalocean.com/mariadb/repo/5.5/ubuntu trusty main'
```

注意

You can change the mirror to one near you on: downloads.mariadb.org

2. 升级您的系统并安装必要的软件包：

```
# apt-get update
# apt-get install mariadb-galera-server galera
```

警告

如果您已经安装了mariaDB，您需要重新申请安装指南中的所有权限。此将清理所有的权限！

3. 调整配置文件：

   在/etc/mysql/my.conf文件中，进行如下修改：

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

4. 创建/etc/mysql/conf.d/wsrep.cnf文件

   将如下内容粘贴到文件中：

```
[mysqld]
wsrep_provider=/usr/lib/galera/libgalera_smm.so
wsrep_cluster_name="Openstack"
wsrep_sst_auth=wsrep_sst:wspass
wsrep_cluster_address="gcomm://PRIMARY_NODE_IP,SECONDARY_NODE_IP"
wsrep_sst_method=rsync
wsrep_node_address="PRIMARY_NODE_IP"
wsrep_node_name="NODE_NAME"
```

   使用您主服务器和次服务器的IP地址替换PRIMARY_NODE_IP 和 SECONDARY_NODE_IP

   使用服务器的主机名替换PRIMARY_NODE_IP。这项配置用于日志。

   将此文件拷贝到所有的其他数据库服务器上并相应的修改wsrep_cluster_address 和wsrep_node_name的值。

5. 使用root启动mysql并执行以下查询：

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

   删除所有用户名为空的 MySQL 帐户（这些帐户会产生安全隐患）：

```
mysql> SET wsrep_on=OFF; DELETE FROM mysql.user WHERE user='';
```

6. 检查各节点之间的网络通信有没有被防火墙拦截。根据布署环境的不同，检查方法也各不相同。如，某些环境中，这一步只需要对 iptables 进行配置：

```
# 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
```

在某些环境中，可能还需要对 NAT 防火墙进行配置，以保证节点服务器之间可以直接通信。另外，可能需要禁用 SELinux，或者允许 mysqld 监听非特权端口。

下一步执行之前创建所有数据库服务器的debian.cnf文件的备份文件，该文件位于/etc/mysql。当有什么错误发生时，仅需拷贝备份文件回来。

```
# cp debian.cnf debian.cnf.old
```

确保您用户其它服务器上的SSH root权限。通过运行一下命令，从主数据库服务器拷贝debian.cnf到其它服务器：

```
# scp /etc/mysql/debian.cnf root@IP-address:/etc/mysql
```

复制之后需确保所有的文件都是相同的，你能通过使用如下命令确认：

```
# md5sum debian.cnf
```

从debian.cnf中获取数据库密码：

```
# cat /etc/mysql/debian.cnf
```

结果将如下所示：

```
[client]
host = localhost
user = debian-sys-maint
password = FiKiOY1Lw8Sq46If
socket = /var/run/mysqld/mysqld.sock
[mysql_upgrade]
host = localhost
user = debian-sys-maint
password = FiKiOY1Lw8Sq46If
socket = /var/run/mysqld/mysqld.sock
basedir = /usr
```

在除了主节点的服务器上运行以下查询。这将确保您能再次重启服务器。不要忘记从debian.cnf天降密码。为了完成这个，运行：

```
mysql> GRANT SHUTDOWN ON *.* TO 'debian-sys-maint'@'localhost' IDENTIFIED BY '<debian.cnf password>';
mysql> GRANT SELECT ON `mysql`.`user` TO 'debian-sys-maint'@'localhost' IDENTIFIED BY '<debian.cnf
 password>';
```

停止所有的mysql服务器然后使用如下命令启动第一个服务器：

```
# service mysql start --wsrep-new-cluster
```

其他所有节点现在能被启动：

```
# service mysql start
```

通过以root身份登录mysql并运行以下命令来确认wsrep复制：

```
mysql> SHOW STATUS LIKE 'wsrep%';
+----------------------------+-----------------------------------+
```

```
| Variable_name          | Value                                |
+------------------------+--------------------------------------+
| wsrep_local_state_uuid     | d6a51a3a-b378-11e4-924b-23b6ec126a13 |
| wsrep_protocol_version     | 5                                    |
| wsrep_last_committed       | 202                                  |
| wsrep_replicated           | 201                                  |
| wsrep_replicated_bytes     | 89579                                |
| wsrep_repl_keys            | 865                                  |
| wsrep_repl_keys_bytes      | 11543                                |
| wsrep_repl_data_bytes      | 65172                                |
| wsrep_repl_other_bytes     | 0                                    |
| wsrep_received             | 8                                    |
| wsrep_received_bytes       | 853                                  |
| wsrep_local_commits        | 201                                  |
| wsrep_local_cert_failures  | 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_local_cached_downto  | 1                                    |
| wsrep_flow_control_paused_ns | 0                                  |
| wsrep_flow_control_paused  | 0.000000                             |
| wsrep_flow_control_sent    | 0                                    |
| wsrep_flow_control_recv    | 0                                    |
| wsrep_cert_deps_distance   | 1.029703                             |
| wsrep_apply_oooe           | 0.024752                             |
| wsrep_apply_oool           | 0.000000                             |
| wsrep_apply_window         | 1.024752                             |
| wsrep_commit_oooe          | 0.000000                             |
| wsrep_commit_oool          | 0.000000                             |
| wsrep_commit_window        | 1.000000                             |
| wsrep_local_state          | 4                                    |
| wsrep_local_state_comment  | Synced                               |
| wsrep_cert_index_size      | 18                                   |
| wsrep_causal_reads         | 0                                    |
| wsrep_cert_interval        | 0.024752                             |
| wsrep_incoming_addresses   | <first IP>:3306,<second IP>:3306     |
| wsrep_cluster_conf_id      | 2                                    |
| wsrep_cluster_size         | 2                                    |
| wsrep_cluster_state_uuid   | d6a51a3a-b378-11e4-924b-23b6ec126a13 |
| wsrep_cluster_status       | Primary                              |
| wsrep_connected            | ON                                   |
| wsrep_local_bf_aborts      | 0                                    |
| wsrep_local_index          | 1                                    |
| wsrep_provider_name        | Galera                               |
| wsrep_provider_vendor      | Codership Oy <info@codership.com>    |
| wsrep_provider_version     | 25.3.5-wheezy(rXXXX)                 |
| wsrep_ready                | ON                                   |
| wsrep_thread_count         | 2                                    |
+----------------------------+--------------------------------------+
```

Galera MariaDB(基于红帽平台)

在主-主，多主环境下，Galera MariaDB提供数据库同步。数据自身的高可用由Galera内部管理，同时访问高可用由HAProxy管理。

本教程假设3个节点用来搭建MariaDB Galera集群。除非明确指定，所有命令需要在所有集群节点上执行。

过程 6.1. 安装带Galera的MariaDB

1. 基于Red Hat的发型版本在它们的仓库中包含Galera包。为了安装最新版本的安装包，运行以下命令：

```
# yum install -y mariadb-galera-server xinetd rsync
```

2. (可选) 配置 clustercheck 工具集。

   如本文档HAProxy章节所描述， 如果HAProxy用来负载均衡MariaDB的访问，您可以用clustercheck工具集改善心跳检测。

   a. 使用以下内容创建文件etc/sysconfig/clustercheck ：

```
MYSQL_USERNAME="clustercheck"
MYSQL_PASSWORD=密码
MYSQL_HOST="localhost"
MYSQL_PORT="3306"
```

警告

确保使用一个便于理解的密码。

   b. 配置监控服务(HAProxy服务使用)：

   使用以下内容创建/etc/xinetd.d/galera-monitor 文件：

```
service galera-monitor
{
 port = 9200
 disable = no
 socket_type = stream
 protocol = tcp
 wait = no
 user = root
 group = root
 groups = yes
 server = /usr/bin/clustercheck
 type = UNLISTED
 per_source = UNLIMITED
 log_on_success =
 log_on_failure = HOST
 flags = REUSE
}
```

   c. 创建clustercheck需要的数据库用户：

```
# systemctl start mysqld
# mysql -e "CREATE USER 'clustercheck'@'localhost' IDENTIFIED BY '密码';"
# systemctl stop mysqld
```

   d. 启动xinetd(clustercheck需要)：

```
# systemctl daemon-reload
# systemctl enable xinetd
# systemctl start xinetd
```

3. 配置带Galera的MariaDB。

a. 使用以下内容创建Galera配置文件/etc/my.cnf.d/galera.cnf ：

```
[mysqld]
skip-name-resolve=1
binlog_format=ROW
default-storage-engine=innodb
innodb_autoinc_lock_mode=2
innodb_locks_unsafe_for_binlog=1
max_connections=2048
query_cache_size=0
query_cache_type=0
bind_address=NODE_IP
wsrep_provider=/usr/lib64/galera/libgalera_smm.so
wsrep_cluster_name="galera_cluster"
wsrep_cluster_address="gcomm://PRIMARY_NODE_IP,SECONDARY_NODE_IP,TERTIARY_NODE_IP"
wsrep_slave_threads=1
wsrep_certify_nonPK=1
wsrep_max_ws_rows=131072
wsrep_max_ws_size=1073741824
wsrep_debug=0
wsrep_convert_LOCK_to_trx=0
wsrep_retry_autocommit=1
wsrep_auto_increment_control=1
wsrep_drupal_282555_workaround=0
wsrep_causal_reads=0
wsrep_notify_cmd=
wsrep_sst_method=rsync
```

b. 打开MariaDB和Galera通信使用的防火墙端口：

```
# firewall-cmd --add-service=mysql
# firewall-cmd --add-port=4444/tcp
# firewall-cmd --add-port=4567/tcp
# firewall-cmd --add-port=4568/tcp
# firewall-cmd --add-port=9200/tcp
# firewall-cmd --add-port=9300/tcp
# firewall-cmd --add-service=mysql --permanent
# firewall-cmd --add-port=4444/tcp --permanent
# firewall-cmd --add-port=4567/tcp --permanent
# firewall-cmd --add-port=4568/tcp --permanent
# firewall-cmd --add-port=9200/tcp --permanent
# firewall-cmd --add-port=9300/tcp --permanent
```

c. 启动MariaDB集群：

i. 在节点1：

```
# sudo -u mysql /usr/libexec/mysqld --wsrep-cluster-address='gcomm://' &
```

ii. 在节点2和3：

```
# systemctl start mariadb
```

iii. 在所有节点上，一旦从clustercheck输出是200，在节点1上重启MariaDB。

```
# kill <mysql 进程号>
# systemctl start mariadb
```

第 7 章 RabbitMQ

目录

RabbitMQ 是多数 OpenStack 服务的默认 AMQP 服务程序。实现 RabbitMQ 的高可用包括以下步骤：

- 安装 RabbitMQ
- 配置 RabbitMQ 实现高可用的消息队列
- 配置其它 OpenStack 服务使用高可用的 RabbitMQ 服务

安装 RabbitMQ

对于 Ubuntu 和 Debian 发行版

RabbitMQ 已经有可用的软件安装包：

```
# apt-get install rabbitmq-server
```

注意

在 Ubuntu 和 Debian 发行版上安装 RabbitMQ 的官方文档

对于 Fedora 和 RHEL 发行版

RabbitMQ 已经有可用的软件安装包：

```
# yum install rabbitmq-server
```

注意

在 Fedora 和 RHEL 发行版上安装 RabbitMQ 的官方文档

对 openSUSE 和 SLES 发行版

过程 7.1. 在 openSUSE 系统中：

- ```
  # zypper install rabbitmq-server
  ```

注意

在 openSUSE 发行版上安装 RabbitMQ 的官方文档
  ```

在 openSUSE 发行版上安装 RabbitMQ 的官方文档

过程 7.2. 在SLES 12中：

1. # zypper addrepo -f obs://Cloud:OpenStack:Kilo/SLE_12 Kilo

注意

这些软件包都使用 GPG 密钥 893A90DAD85F9316 进行了签名，在安装之前可以先验证签名。

```
Key ID: 893A90DAD85F9316
Key Name: Cloud:OpenStack OBS Project <Cloud:OpenStack@build.opensuse.org>
Key Fingerprint: 35B34E18ABC1076D66D5A86B893A90DAD85F9316
Key Created: Tue Oct  8 13:34:21 2013
Key Expires: Thu Dec 17 13:34:21 2015
```

2. # zypper install rabbitmq-server

配置 RabbitMQ

将多个 RabbitMQ 节点组织成一个集群，构建一个 RabbitMQ broker 服务，即一个 Erlang 节点的逻辑集合。

单个节点服务器的故障不会导致消息的交换和绑定完全不可用，但是具体一个消息队列及其中的内容则相反。原因是消息队列和其中的内容只在其中一台节点上，该节点出现故障无法工作时，整个消息队列就丢失了。

RabbitMQ 实现队列镜像更能提高整个集群的高可用性。

示例中会布署 2 台 RabbitMQ 服务器，rabbit1 和 rabbit2。要构建一个 RabbitMQ broker 服务，必须保证所有节点服务器的 Erlang cookie 文件完全相同。

因此，首先在所有节点服务器停止 RabbitMQ 服务，然后将第一台节点服务上的 cookis 文件复制到其它节点：

```
# scp /var/lib/rabbitmq/.erlang.cookie
root@NODE:/var/lib/rabbitmq/.erlang.cookie
```

在目标节点上确保.erlang.cookie文件拥有正确的拥有者、组和权限：

```
# chown rabbitmq:rabbitmq /var/lib/rabbitmq/.erlang.cookie
# chmod 400 /var/lib/rabbitmq/.erlang.cookie
```

在所有节点上启动RabbitMQ，确保所有节点都处于运行状态：

```
# rabbitmqctl cluster_status
Cluster status of node rabbit@NODE...
[{nodes,[{disc,[rabbit@NODE]}]},
 {running_nodes,[rabbit@NODE]},
 {partitions,[]}]
...done.
```

在除第一台节点之外的其它节点服务器上运行下面的命令：

```
# rabbitmqctl stop_app
Stopping node rabbit@NODE...
...done.
```

```
# rabbitmqctl join_cluster rabbit@rabbit1
# rabbitmqctl start_app
Starting node rabbit@NODE ...
...done.
```

检查集群状态：

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

如果集群运行正常，就可以开始为消息队列创建用户和密码。

为了保证除了自动生动的队列之外的其它队列能在所有节点上实现镜像，必须将 RabbitMQ 策略配置项 ha-mode 设置为 all。在集群中任何一台节点服务器上执行下面的命令。

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

注意

More information about highly available queues and clustering can be found in the official RabbitMQ documentation.

配置 OpenStack 各服务使用高可用的 RabbitMQ 服务

现在可以配置 OpenStack 其它组件使用高可用 RabbitMQ 集群（最少使用其中 2 台节点服务器）。

对所有使用 RabbitMQ 的组件进行配置：

- RabbitMQ HA 集群服务地址及端口：

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

- 重新尝试连接 RabbitMQ 服务的时间间隔：

```
rabbit_retry_interval=1
```

- 每次重新尝试连接 RabbitMQ 服务应后延多长时间：

```
rabbit_retry_backoff=2
```

- 连接 RabbitMQ 服务时最大的重试次数（默认没有限制）：

```
rabbit_max_retries=0
```

- 是否使用持久的消息队列：

```
rabbit_durable_queues=true
```

- 否使用 RabbitMMQ 的队列镜像特性（ x-ha-policy: all ）：

```
rabbit_ha_queues=true
```

注意

如果是直接#改没有启用队列镜像特性的 RabbitMQ 服务的配置，那么对服务作一次重置：

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

注意

目前支持高可用 RabbitMQ 服务的 OpenStack 组件有：

- OpenStack 计算服务
- OpenStack 块设备存储服务
- OpenStack 网络服务
- Telemetry

第 8 章 HAProxy 节点服务器

HAProxy 是高效而且可靠的应用程高可用、负载均衡以及代理解决方案，适用于所有适用于基于 TCP 和 HTTP 通信的应用程序，特别是那些在极高负载下仍要保持良好运行同时还必须支持会话保持以及其它 7 层处理操作的 Web 站点。在当前流行的硬件配置条件下，HAProxy 可以轻易地支持同时数十万计的并发连接。

For installing HAProxy on your nodes, you should consider its official documentation. And also, you have to consider that HAProxy should not be a single point of failure so you need to ensure its availability by other means, such as Pacemaker or Keepalived. It is advisable to have multiple HAProxy instances running, where the number of these instances is a small odd number like 3 or 5. Also it is a common practice to collocate HAProxy instances with existing OpenStack controllers.

下面是HAProxy配置文件的示例：

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

defaults
 log global
 maxconn 4000
 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
 server controller3 10.0.0.3:443 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 backup check port 9200 inter 2000 rise 2 fall 5
 server controller3 10.0.0.6:3306 backup 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
 server controller3 10.0.0.3:9292 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
 server controller3 10.0.0.3:9191 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 10.0.0.2:35357 check inter 2000 rise 2 fall 5
 server controller3 10.0.0.3:35357 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
 server controller3 10.0.0.3:5000 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
 server controller3 10.0.0.3:8773 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
 server controller3 10.0.0.3:8774 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
 server controller3 10.0.0.3:8775 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
 server controller3 10.0.0.3:8776 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 10.0.0.1:8777 check inter 2000 rise 2 fall 5
 server controller2 10.0.0.2:8777 check inter 2000 rise 2 fall 5
 server controller3 10.0.0.3:8777 check inter 2000 rise 2 fall 5

listen spice_cluster
 bind <Virtual IP>:6080
 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
 server controller3 10.0.0.3:6080 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
 server controller3 10.0.0.3:9696 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
 server controller3 10.0.0.3:8080 check inter 2000 rise 2 fall 5
```

每次修改配置文件之后，必须重启 HAProxy 服务。

注意

注意，Galera集群配置命令表明3个控制点中的2个节点是备份。因为OpenStack支持多节点写还没有准备好生产使用，应该保证只有一个节点服务器处理写请求。

第 9 章 OpenStack 控制节点服务器

目录

OpenStack 控制节点服务器上运行以下服务：

- 所有 OpenStack API 服务
- 所有 OpenStack 调度相关的服务
- Memcached 服务

运行 OpenStack API 和 调度服务

API 服务

所有的OpenStack项目都包含一个API服务，这些API负责管理整个云平台的各个资源。在主/主模式，通常的方案是让所有这些API服务分布到两台以上的服务节点上，通过负载均衡和虚拟机IP地址(示例中使用HAProxy 和 Keepalived)。

要实现高可用和可扩展的 API 服务，需要保证：

- 为 OpenStack 身份认证中的服务端点配置虚拟 IP 地址。
- 所有 OpenStack 组件的配置中也相应使用虚拟 IP 地址。

注意

监控方法比较简单，只是向 API 服务绑定的端口发起一个 TCP 连接。和主/从模式的高可用方案使用 Corosync 和资源代理脚本不同，这种监控方法不会检查这些服务是否在运行。因此所有的 OpenStack API 服务还需要通过工具作进一步的监控，比如 Nagios 。

调度程序

OpenStack 调度程序决定如何分派对计算资源、网络资源、存储卷源的请求。一般情况下都会使用 RabbitMQ 作为消息队列服务。以下服务使用消息队列作为后端，可以分布在多个节点服务器上运行：

- nova-scheduler
- nova-conductor
- cinder-scheduler
- neutron-server

- ceilometer-collector
- heat-engine

请参阅 RabbitMQ 部分 将 OpenStack 各组件配置为可同时使用多个消息队列服务器。

Telemetry中央代理

The Telemetry Central agent can be configured to partition its polling workload between multiple agents, enabling high availability. Please refer to this section of the OpenStack Cloud Administrator Guide for the requirements and implementation details of this configuration.

Memcached

大部分OpenStack服务使用 Memcached存储瞬时数据，比如令牌。虽然Memcached不支持典型形式的冗余，比如集群，通过配置多个主机名或IP地址OpenStack服务可以使用几乎任意数量的实例。Memcached客户端在实例之间实现对象的哈希均衡。一个实例失败仅影响所有对象的一定比例，并且客户端自动从实例列表中移除它。

To install and configure it, read the official documentation.

基于内存的缓存统一由 oslo-incubator 管理，因此对 OpenStack 服务来说，使用多个 memcached 服务节点作为后端的方法完全相同。

使用 2 个 memcached 节点的配置示例：

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

By default, controller1 handles the caching service but if the host goes down, controller2 does the job. For more information about Memcached installation, see the OpenStack Cloud Administrator Guide.

第 10 章 OpenStack 网络节点服务器

目录

OpenStack 网络节点运行以下服务：

- Neutron DHCP 代理服务
- Neutron L2 代理服务
- Neutron L3 代理服务
- Neutron 元数据代理服务
- Neutron LBaas 代理服务

注意

Neutron L2 代理服务不需要实现高可用。在所有提供数据转发的服务器上都要安装 Neutron L2 代理程序，对诸如 Open vSwitch 、Linux Bridge 等虚拟网络驱动进行管理。每台节点服务器各运行一个 L2 代理程序，负责管理该节点上的虚拟网络接口。这也是 Neutron L2 代理服务无法实现多节点分布以及高可用的原因。

运行 neutron DHCP 代理服务

OpenStack 网络服务拥有一个调度程序，所有可以在多个节点同时运行各种代理程序。同样，DHCP 代理服务自身就是支持高可用的。每个网络使用多少 DHCP 代理程序是可以通过配置文件 neutron.conf 中的 dhcp_agents_per_network 进行配置的。默认值是 1 ，要实现 DHCP 代理服务的高可用，应为每个网络设置多个 DHCP 代理程序。

运行 neutron L3 代理服务

Neutron的三层L3代理是可扩展的，取决于允许跨多个节点虚拟路由器分发的调度器。以下选项用来使一个路由器高可用：

- 通过/etc/neutron/neutron.conf文件中的 allow_automatic_l3agent_failover = True选项配置针对路由器的L3代理失效自动恢复。
- 通过VRRP，使用3层高可用。为了启动它，需要在/etc/neutron/neutron.conf 文件设置以下配置项：

选项	设置的值	描述
l3_ha	True	缺省情况下，所有路由器应该高可用。
max_l3_agents_per_router	2	路由器HA的网络节点最大数。值可以比2大，但是应该至少是2。
min_l3_agents_per_router	2	路由器HA的最小网络节点数。除非有最少的min_l3_agents_per_router网络节点可用，新的路由器创建失败。这个值不能比2小。

- 借助Pacemaker，使用主/备解决方案把Neuttron L3代理运行在失效恢复模式。参考本手册主/备章节

运行 neutron 元数据代理服务

Neutron 元数据代理服务自身是不支持高可用的。目前针对 Neutron L3 代理服务只有主/从模式的高可用方案通过 Pacemaker 可以实现失效切换。参阅本手册的 主/从模式高可用架构部分。

运行 neutron LBaas 代理服务

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。

附录 A. Community support

目录

The following resources are available to help you run and use OpenStack. The OpenStack community constantly improves and adds to the main features of OpenStack, but if you have any questions, do not hesitate to ask. Use the following resources to get OpenStack support, and troubleshoot your installations.

Documentation

For the available OpenStack documentation, see docs.openstack.org.

To provide feedback on documentation, join and use the <openstack-docs@lists.openstack.org> mailing list at OpenStack Documentation Mailing List, or report a bug.

The following books explain how to install an OpenStack cloud and its associated components:

- Installation Guide for openSUSE 13.2 and SUSE Linux Enterprise Server 12
- Installation Guide for Red Hat Enterprise Linux 7, CentOS 7, and Fedora 21
- Installation Guide for Ubuntu 14.04 (LTS)

The following books explain how to configure and run an OpenStack cloud:

- Architecture Design Guide
- Cloud Administrator Guide
- Configuration Reference
- Operations Guide
- Networking Guide
- High Availability Guide
- Security Guide
- Virtual Machine Image Guide

The following books explain how to use the OpenStack dashboard and command-line clients:

- API Quick Start
- End User Guide
- Admin User Guide
- Command-Line Interface Reference

The following documentation provides reference and guidance information for the OpenStack APIs:

- OpenStack API Complete Reference (HTML)
- API Complete Reference (PDF)

ask.openstack.org

During the set up or testing of OpenStack, you might have questions about how a specific task is completed or be in a situation where a feature does not work correctly. Use the ask.openstack.org site to ask questions and get answers. When you visit the https://ask.openstack.org site, scan the recently asked questions to see whether your question has already been answered. If not, ask a new question. Be sure to give a clear, concise summary in the title and provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.

OpenStack mailing lists

A great way to get answers and insights is to post your question or problematic scenario to the OpenStack mailing list. You can learn from and help others who might have similar issues. To subscribe or view the archives, go to http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack. You might be interested in the other mailing lists for specific projects or development, which you can find on the wiki. A description of all mailing lists is available at https://wiki.openstack.org/wiki/MailingLists.

The OpenStack wiki

The OpenStack wiki contains a broad range of topics but some of the information can be difficult to find or is a few pages deep. Fortunately, the wiki search feature enables you to search by title or content. If you search for specific information, such as about networking or OpenStack Compute, you can find a large amount of relevant material. More is being added all the time, so be sure to check back often. You can find the search box in the upper-right corner of any OpenStack wiki page.

The Launchpad Bugs area

The OpenStack community values your set up and testing efforts and wants your feedback. To log a bug, you must sign up for a Launchpad account at https://launchpad.net/+login.

You can view existing bugs and report bugs in the Launchpad Bugs area. Use the search feature to determine whether the bug has already been reported or already been fixed. If it still seems like your bug is unreported, fill out a bug report.

Some tips:

- Give a clear, concise summary.
- Provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.
- Be sure to include the software and package versions that you are using, especially if you are using a development branch, such as, "Juno release" vs git commit bc79c3ecc55929bac585d04a03475b72e06a3208.
- Any deployment-specific information is helpful, such as whether you are using Ubuntu 14.04 or are performing a multi-node installation.

The following Launchpad Bugs areas are available:

- Bugs: OpenStack Block Storage (cinder)
- Bugs: OpenStack Compute (nova)
- Bugs: OpenStack Dashboard (horizon)
- Bugs: OpenStack Identity (keystone)
- Bugs: OpenStack Image service (glance)
- Bugs: OpenStack Networking (neutron)
- Bugs: OpenStack Object Storage (swift)
- Bugs: Application catalog (murano)
- Bugs: Bare metal service (ironic)
- Bugs: Containers service (magnum)
- Bugs: Data processing service (sahara)
- Bugs: Database service (trove)
- Bugs: Orchestration (heat)
- Bugs: Shared file systems (manila)
- Bugs: Telemetry (ceilometer)
- Bugs: Message Service (zaqar)
- Bugs: OpenStack API Documentation (developer.openstack.org)
- Bugs: OpenStack Documentation (docs.openstack.org)

The OpenStack IRC channel

The OpenStack community lives in the #openstack IRC channel on the Freenode network. You can hang out, ask questions, or get immediate feedback for urgent and pressing issues. To install an IRC client or use a browser-based client, go to https://webchat.freenode.net/. You can also use Colloquy (Mac OS X, http://colloquy.info/), mIRC (Windows, http://www.mirc.com/), or XChat (Linux). When you are in the IRC channel and want to share code or command output, the generally accepted method is to use a Paste Bin. The OpenStack project has one at http://paste.openstack.org. Just paste your longer amounts of text or logs in the web form and you get a URL that you can paste into the channel. The OpenStack IRC channel is #openstack on irc.freenode.net. You can find a list of all OpenStack IRC channels at https://wiki.openstack.org/wiki/IRC.

Documentation feedback

To provide feedback on documentation, join and use the <openstack-docs@lists.openstack.org> mailing list at OpenStack Documentation Mailing List, or report a bug.

OpenStack distribution packages

The following Linux distributions provide community-supported packages for OpenStack:

- Debian: https://wiki.debian.org/OpenStack
- CentOS, Fedora, and Red Hat Enterprise Linux: https://www.rdoproject.org/
- openSUSE and SUSE Linux Enterprise Server: https://en.opensuse.org/Portal:OpenStack
- Ubuntu: https://wiki.ubuntu.com/ServerTeam/CloudArchive