

The OpenStack logo, consisting of the word "OpenStack" in white text on a red arrow-shaped background pointing to the right.

OpenStack

云计算平台管理员手册

current (September 26, 2015)



OpenStack 云计算平台管理员手册

current (2015-09-26)

版权 © 2013-2015 OpenStack基金会 Some rights reserved.

摘要

OpenStack 为管理员提供了对 OpenStack 云计算平台进行管理和排障的开源工具。

本手册适用于OpenStack kilo,OpenStack Juno, 以及 OpenStack Icehouse。

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.



Except where otherwise noted, this document is licensed under
Creative Commons Attribution 3.0 License.
<http://creativecommons.org/licenses/by/3.0/legalcode>

目录

前言	ix
约定	ix
文档变更历史	ix
1. 开始使用OpenStack	1
概念架构	2
逻辑架构	2
OpenStack 服务	4
反馈	16
2. 身份认证管理	17
身份概念	17
证书的PKI	22
配置基于SSL的身份服务	26
身份的外部认证	27
集成LDAP身份认证	27
为令牌绑定配置身份服务	35
使用信托	36
缓存层	37
用户 CRUD	38
日志记录	39
启动认证服务	39
使用实例	39
使用用户名和密码认证中间件	40
基于角色访问控制(RBAC)的身份API保护	41
身份服务故障诊断	43
3. Dashboard	46
定制GUI界面	46
为仪表盘设置会话存储	48
4. 计算	53
系统架构	53
镜像和实例	58
使用nova-network的网络	64
系统管理	79
计算故障排查	113
5. 对象存储	117
Object存储介绍	117
特性和好处	117
对象存储特性	118
组件	119
环构建器	124
集群架构	127
复制	129
账户清道夫	131
配置基于对象存储的特定租户的镜像位置	131
对象存储监控	132
对象存储的系统管理	135
对象存储故障排查	135
6. 块存储	138
块存储介绍	138

加大块存储 API 服务的吞吐量	139
管理卷	139
用户安装Troubleshoot	170
7. 网络	180
网络介绍	180
插件配置	184
配置neutron代理	187
网络架构	194
为网络配置身份服务	196
高级配置选项	200
DHCP代理的扩展性和高可用性	201
使用网络	208
通过API扩展的高级功能	212
高级运维特性	222
认证和授权	224
8. Telemetry	228
介绍	228
系统架构	228
数据收集	231
数据恢复	243
警报	249
测量	254
事件	267
Telemetry 故障排查	269
Telemetry 最佳实践	270
9. 数据库	273
介绍	273
创建一个 datastore	273
配置一集群	276
10. 编排	278
介绍	278
编排认证模式	278
栈的域用户	280
A. 社区支持	282
文档	282
问答论坛	283
OpenStack 邮件列表	283
OpenStack 维基百科	283
Launchpad的Bug区	283
The OpenStack 在线聊天室频道	284
文档反馈	285
OpenStack分发版	285

插图清单

1.1. OpenStack概念架构	2
1.2. 逻辑架构	3
4.1. 基础镜像的状态和运行的实例无关	59
4.2. 从镜像创建的实例和运行时状态	59
4.3. 在实例销毁后镜像和卷最后的状态	60
4.4. 多网卡flat管理器	74
4.5. 多网卡flatdhcp管理器	75
4.6. 多网卡 VLAN管理器	76
4.7. noVNC 进程	100
4.8. 可信任的计算池	108
5.1. 对象存储(Swift)	118
5.2. 对象存储构建块	120
5.3. 环	121
5.4. 区	122
5.5. 账户和容器	122
5.6. 分区	123
5.7. 复制	123
5.8. 对象存储可用	124
5.9. 对象存储架构	127
5.10. 对象存储(Swift)	129
7.1. FWaaS架构	182
7.2. VMware NSX 部署实例 — 两台计算节点	195
7.3. VMware NSX部署实例 — 单计算节点	196

表格清单

1.1. OpenStack 服务	1
1.2. 存储类型	8
4.1. Description of IPv6 configuration options	68
4.2. Description of metadata configuration options	71
4.3. 认证服务配置文件	81
4.4. rootwrap.conf 配置选项	92
4.5. .filters 配置选项	92
4.6. Description of live migration configuration options	95
4.7. Description of VNC configuration options	100
4.8. Description of SPICE configuration options	103
4.9. Description of Zookeeper configuration options	105
4.10. Description of trusted computing configuration options	107
5.1. [drive-audit]在drive-audit.conf中的配置属性描述	136
6.1. 镜像设置由glance image-list 报告镜像ID	174
7.1. 网络资源	180
7.2. 负载均衡即服务特性	181
7.3. 网络代理的基本操作	194
7.4. 网络代理	194
7.5. nova.conf API 和凭据设置	198
7.6. nova.conf 安全组设置	199
7.7. nova.conf元数据设置	199
7.8. 设置	200
7.9. 功能	202
7.10. 基本网络操作	208
7.11. 高级网络操作	210
7.12. 基本的计算和网络操作	211
7.13. 高级虚拟机创建操作	211
7.14. 提供者扩展的术语	213
7.15. 供应商网络属性	213
7.16. 基本的L3操作	215
7.17. 基本安全组操作	216
7.18. VMware NSX Qos的基本操作	218
7.19. max_lp_per_bridged_ls 的建议值	218
7.20. NSX 插件中调整可操作的状态同步的配置选项	219
7.21. Big Switch 路由规则属性	220
7.22. 基本的L3操作	221
8.1. OpenStack服务消费的事件类型	232
8.2. 可用转换器的列表	239
8.3. 后端数据库所支持的存活时间	243
8.4. Telemetry计量类型	254
8.5. OpenStack计算服务计量	255
8.6. OpenStack 计算主机计量	258
8.7. OpenStack裸金属模块的计量	259
8.8. 基于IPMI对计量	259
8.9. 基于SNMP的计量	260
8.10. OpenStack镜像服务计量	261
8.11. OpenStack块存储计量	261
8.12. OpenStack对象存储计量	261

8.13. Ceph 对象存储的计量	262
8.14. OpenStack 身份计量	262
8.15. OpenStack 网络计量	263
8.16. SDN 计量	264
8.17. 负载均衡即服务计量	264
8.18. VPN即服务计量	265
8.19. 防火墙即服务计量	266
8.20. 编排模块的计量	266
8.21. OpenStack的数据处理服务计量	266
8.22. 键值存储模块计量	267
8.23. 电表	267

范例清单

2.1. 配置后端的Memcached	38
---------------------------	----

前言

约定

OpenStack文档使用了多种排版约定。

声明

以下形式的标志为注意项



注意

便签或提醒



重要

提醒用户进一步操作之前必须谨慎



警告

关于有丢失数据的风险或安全漏洞的危险信息提示

命令行提示符

\$ 提示符 \$ 提示符表示任意用户，包括 root 用户，都可以运行的命令。

提示符 # 提示符表明命令仅为 root 可以执行。当然，用户如果可以使用 sudo 命令的话，也可以运行此提示符后面的命令。

文档变更历史

此版本的向导完全取代旧版本的，且所有旧版本的在此版本中不再生效。

下面表格描述的是近期的改动：

Revision Date	Summary of Changes
February 20, 2015	· 针对Kilo发行后，本指南更新了Telemetry一章的计量一节。表格包含了所有引进模块的所收集到的发行信息。另外，本文档新增了编排一章，其描述了OpenStack自Havana发布时可用的编排模块的细节。
October 15, 2014	· 在Juno发行后，此向导已经更新了新的Telemetry章节。
July 21, 2014	· 更新变量，使用正确的格式。
April 17, 2014	· 针对 Icehouse 版本，本手册包含了系统管理和系统架构的章节，同时 how-to 相关章节也从 OpenStack Configuration Reference 移动到了这里。
November 12, 2013	· 在 NSX 插件中增加优化操作状态同步过程优化的参数。
October 17, 2013	· Havana 发行版。
September 5, 2013	· 将对象存储服务监控章节移动到本手册。 · 删除多余的对象存储服务相关的内容。
September 3, 2013	· 除去配置过程和安装过程相关内容，将其他部分分离出来形成新的手册：

Revision Date	Summary of Changes
	<ul style="list-style-type: none">• OpenStack 计算服务管理手册• OpenStack 网络服务管理手册• OpenStack 对象存储服务管理手册• OpenStack 块设备服务管理手册

第 1 章 开始使用OpenStack

目录

概念架构 2

逻辑架构 2

OpenStack 服务 4

反馈 16

OpenStack项目是一个适用各种类型云的开源云计算平台，它的目标是：简单的实现，大规模扩展，丰富的功能。来自全世界各地的开发者和云计算技术专家们创建了OpenStack项目。

OpenStack通过一系列相关联的服务提供了基础设施及服务(IaaS) 解决方案。每个服务对外提供应用程序接口(API)便于轻松集。取决于用户的需求，用户可以选择安装部分服务，也可以选择安装全部服务。

以下表格描述了OpenStack架构的OpenStack服务:

表 1.1. OpenStack 服务

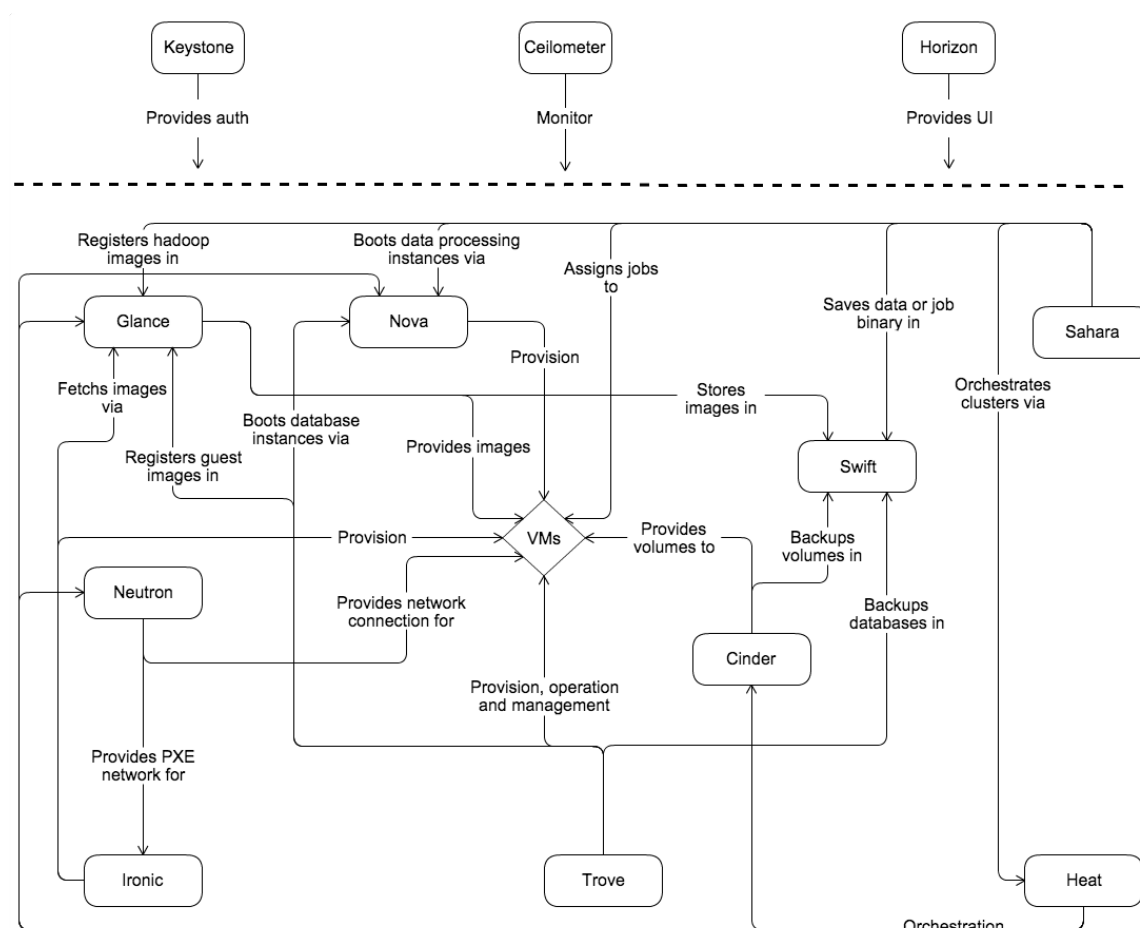
服务	项目名称	描述
仪表盘	控制面板	提供了一个基于web的自服务门户，与OpenStack底层服务交互，诸如启动一个实例，分配IP地址以及配置访问控制。
Compute	Nova	在OpenStack环境中计算实例的生命周期管理。按需响应包括生成、调度、回收虚拟机等操作。
网络	Neutron	确保为其它OpenStack服务提供网络连接即服务，比如OpenStack计算。为用户提供API定义网络和使用。基于插件的架构其支持众多的网络提供商和技术。
存储		
对象存储	Swift	通过一个 RESTful,基于HTTP的应用程序接口存储和任意检索的非结构化数据对象。它拥有高容错机制，基于数据复制和可扩展架构。它的实现并像是一个文件服务器需要挂载目录。在此方式下，它写入对象和文件到多个硬盘中，以确保数据是在集群内跨服务器的多份复制。
块存储	Cinder	为运行实例而提供的持久性块存储。它的可插拔驱动架构的功能有助于创建和管理块存储设备。
共享服务		
认证服务	Keystone	为其他OpenStack服务提供认证和授权服务，为所有的OpenStack服务提供一个端点目录。
镜像服务	Glance	存储和检索虚拟机磁盘镜像，OpenStack计算会在实例部署时使用此服务。
Telemetry	Ceilometer	为OpenStack云的计费、基准、扩展性以及统计等目的提供监测和计量。
高层次服务		
编排	Heat	既可以使用本地HOT模板格式，亦可使用AWS CloudFormation模板格式，来编排多个综合的云应用，通过OpenStack本地REST API或者是CloudFormation相兼容的队列API。
数据库服务	Trove	提供可扩展和稳定的云数据库即服务的功能，可同时支持关系性和非关系性数据库引擎。

服务	项目名称	描述
数据处理服务	Sahara	在OpenStack中通过指定诸如Hadoop版本，集群拓扑结构以及节点硬件细节，来提供部署和扩展Hadoop集群的能力。

概念架构

下面示意图解释了OpenStack服务之间的关系：

图 1.1. OpenStack概念架构



逻辑架构

为了设计、部署和配置OpenStack,管理员必须理解逻辑架构。

正如图 1.1 “OpenStack概念架构” [2]所示，OpenStack是由多个独立的部分组成的，这些独立的部分称之为OpenStack服务。所有服务的认证都通过一个通用的身份服务。分离的服务通过公开的API来彼此之间进行交互，除非哪里需要管理员命令的特权。

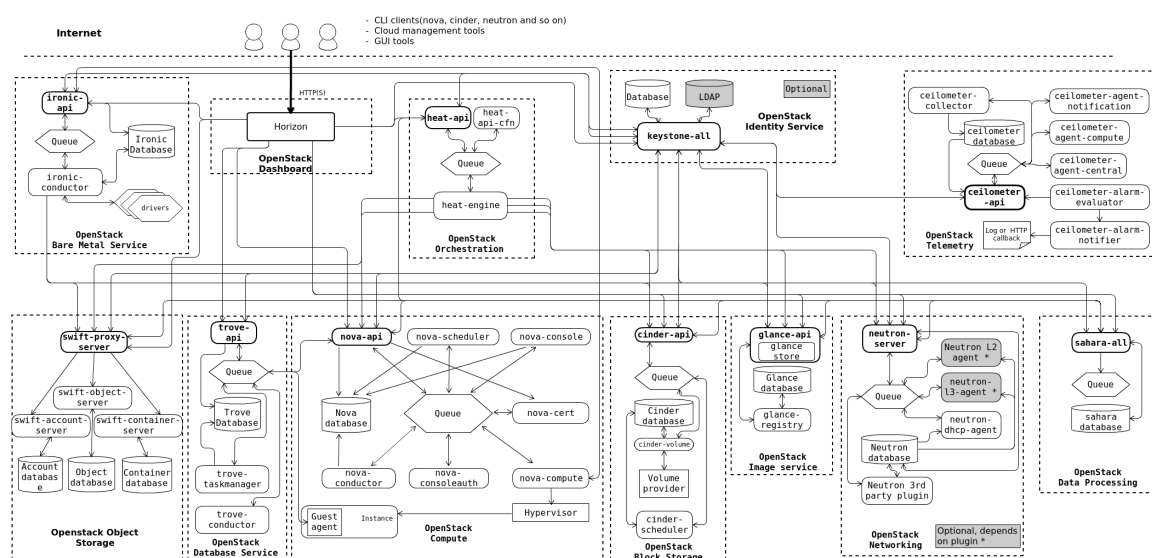
在OpenStack服务的内部往往有多个进程组成。所有的服务至少拥有一个API进程，用来监听API的请求，预先处理它们以及将它们传递给服务的其它部分。除了身份服务之外，其它服务的实际工作都不是一个进程来完成的。

对于一个服务中的进程之间的通信，使用了AMQP消息管家。服务的状态存储在数据库中。当部署和配置你的OpenStack云时，你可以从很多消息管家和数据库解决方案中来作出选择，例如有RabbitMQ, Qpid, MySQL, MariaDB, 以及 SQLite。

用户可以通过基于通过 [dashboard service](#) 所实现web用户界面来访问OpenStack，或者通过 [via 命令行客户端](#)，又或者是通过一些诸如浏览器插件或 curl之类的工具来请求API。对于应用来说，有 [several SDKs](#) 可用。最终，所有的这些访问方法都是调用各个OpenStack服务的REST API。

下面示意图展示了OpenStack云架构的通用的情形，但并非是唯一的可能性：

图 1.2. 逻辑架构



OpenStack 服务

此章描述了OpenStack服务的细节。

OpenStack 计算服务

使用OpenStack计算服务来托管和管理云计算系统。OpenStack计算服务是基础设施即服务(IaaS)系统的主要部分，模块主要由Python实现。

OpenStack计算因为认证和OpenStack身份交互，因为磁盘和服务器的镜像而合OpenStack镜像服务交互，以及为用户和管理员提供图形界面而合OpenStack仪表盘交互。镜像的访问受限于项目和用户；配额是基于项目作限制(实例数量，例如)。OpenStack计算可以在标准的硬件上作横向的扩展，且下载镜像来启动实例。

OpenStack计算服务由下列组件所构成：

应用程序接口

nova-api service	接收和响应来自最终用户的计算API请求。此服务支持OpenStack计算服务API，Amazon EC2 API，以及特殊的管理API用于赋予用户做一些管理的操作。它会强制实施一些规则，发起多数的编排活动，例如运行一个实例。
nova-api-metadata service	Accepts metadata requests from instances. The nova-api-metadata service is generally used when you run in multi-host mode with nova-network installations. For details, see Metadata service in the OpenStack Cloud Administrator Guide. 在Debian系统中，它包含在nova-api包中，可通过debconf来选择。

计算服务核心

nova-compute service	一个持续工作的守护进程，通过Hypervior的API来创建和销毁虚拟机实例。例如： <ul style="list-style-type: none">• XenServer/XCP 的 XenAPI• KVM 或 QEMU 的 libvirt• VMware 的 VMwareAPI 过程是蛮复杂的。最为基本的，守护进程同意了来自队列的动作请求，转换为一系列的系统命令如启动一个KVM实例，然后，到数据库中更新它的状态。
nova-scheduler service	拿到一个来自队列请求虚拟机实例，然后决定那台计算服务器主机来运行它。
nova-conductor 模块	起着在nova-compute 服务和数据库之间的中介交互作用，它消除了由nova-compute服务直接访问数据库的可能

性。nova-conductor模块可横向扩展。然而，不要将其部署在运行 nova-compute服务的节点中。至于原因的更多信息，请参阅[新的 Nova 服务: nova-conductor](#)。

nova-cert 模块

一个服务器的守护进程，为X509证书服务的Nova Cert 服务，用于为euca-bundle-image生成证书，仅用于EC2 API。

虚拟机网络

nova-network 守护进程 和 nova-compute服务类似，从队列[接收网络任务](#)然后[操作网络](#)，执行的任务诸如[设置网桥或更改IPtables规则](#)。

终端接口

nova-consoleauth 守护进程	Authorizes tokens for users that console proxies provide. See nova-novncproxy and nova-xvpvncproxy. This service must be running for console proxies to work. You can run proxies of either type against a single nova-consoleauth service in a cluster configuration. For information, see About nova-consoleauth .
nova-novncproxy 守护进程	提供一个代理，用于访问正在运行的实例，通过VNC协议，支持基于浏览器的novnc客户端。
nova-spicehtml5proxy 守护进程	提供一个代理，用于访问正在运行的实例，通过 SPICE 协议，支持基于浏览器的 HTML5 客户端。
nova-xvpvncproxy 守护进程	提供一个代理，用于访问正在运行的实例，通过VNC协议，支持OpenStack特定的Java客户端。
nova-cert 守护进程	X509 证书。

在Debian中，一个独特的包nova-consoleproxy，提供了包含nova-novncproxy, nova-spicehtml5proxy, and nova-xvpvncproxy三个包，欲选择相应的包，编辑文件/etc/default/nova-consoleproxy或者使用debconf接口。用户也可以手动编辑文件/etc/default/nova-consoleproxy，以及启动和停止终端守护进程。

镜像管理(EC2 场景)

nova-objectstore 守护进程	用于注册镜像到OpenStack镜像服务的S3接口。对于安装它主要是用于支持euca2tools。euca2tool工具使用S3 语言和nova-objectstore对话，然后nova-objectstore翻译S3的请求为OpenStack镜像服务的请求。
euca2ools 客户端	用于管理云资源的一组命令行解释器命令。尽管它不是一个OpenStack的模块，你也可以配置nova-api 来支持EC2的接口。更多信息，请参阅 Eucalyptus 3.4 文档 。

命令行客户端和其他接口

nova 客户端	用于用户作为租户管理员或最终用户来提交命令。
----------	------------------------

其他组件

队列	用于在守护进程之间传送消息到中心枢纽。通常是由 RabbitMQ 来实现的，但是它可由很多AMQP消息队列来实现，例如 Apache Qpid 或 Zero MQ 。
SQL数据库	存储构建时和运行时的状态，为云基础设施，包括有： <ul style="list-style-type: none">· 可用实例类型· 使用中的实例· 可用网络

- 项目

理论上，OpenStack计算可以支持任何和SQLAlchemy所支持的后端数据库，通常使用SQLite3来做测试可开发工作，MySQL和PostgreSQL 作生产环境。

存储概念

OpenStack堆栈使用下列存储类型：

表 1.2. 存储类型

On-instance / ephemeral	块存储 (cinder)	对象存储(Swift)
运行在操作系统中，且提供scratch空间	用于给虚拟机(VM)添加额外的持久存储	用于存放虚拟机镜像和数据
随着VM的销毁而消失	除非人工删除，否则一直存在	除非人工删除，否则一直存在
访问和VM想关联	访问和VM想关联	对任何地方都可用
在OpenStack计算服务下实现的文件系统	通过OpenStack块存储控制协议(例如，iSCSI)来挂载	REST API
加密可用	加密可用	正在开发中 — Mikaka发行时可能可用
系统管理员配置大小，基于flavor	大小根据需要而定	针对于未来的增长，可轻松扩展
Example: 10 GB first disk, 30 GB/core second disk	Example: 1 TB "extra hard drive"	举例：10倍的TB数据单元存储

用户须知：

- 用户不可以想传统的硬盘那样使用OpenStack对象存储。对象存储相对POSIX标准的文件系统放宽了一些约束以获得其他的功能，用户可通过使用HTTP的应用程序接口来访问对象，但是用户不能做一些原子操作(因为最终一致性的原因)，用户可以很容易的扩展存储系统以及避免中心节点的失败。
- OpenStack镜像服务在OpenStack集群中管理虚拟机镜像，但并不存储它们。它提供了对于存储的不同方式的抽象 — 到达存储的一座桥梁，而不是存储本身。
- OpenStack对象存储用于独立的功能。对象存储(swift)可独立于计算(Nova)产品之外对外提供服务。

命令行客户端和其他接口

swift 客户端	用户可以通过此命令行客户端来向REST API提交命令，授权的用户角色可以是管理员用户，经销商用户，或者是swift用户。
swift-init	初始化构建环文件的脚本，以守护进程名称作为其参数，且提供命令行。文档请参阅 http://docs.openstack.org/developer/swift/admin_guide.html#managing-services 。
swift-recon	
swift-ring-builder	存储环的构建和重新平衡工具。文档在 http://docs.openstack.org/developer/swift/admin_guide.html#managing-the-rings 。

OpenStack对象存储

OpenStack对象存储是一个多租户的对象存储系统，它支持大规模扩展，可以以低成本来管理大型的非结构化数据，通过RESTful HTTP 应用程序接口。

它包含下列组件：

代理服务器 (swift-proxy-server)	接收OpenStack对象存储API和纯粹的HTTP请求以上传文件，更改元数据，以及创建容器。它可服务于在web浏览器下显示文件和容器列表。为了改进性能，代理服务可以使用可选的缓存，通常部署的是memcache。
账户服务 (swift-account-server)	管理由对象存储定义的账户。
容器服务 (swift-container-server)	管理容器或文件夹的映射，对象存储内部。
对象服务 (swift-object-server)	Manages actual objects,such as files, on the storage nodes.
各种定期进程	为了驾驭大型数据存储的任务，复制服务需要在集群内确保一致性和可用性，其他定期进程有审计，更新和reaper。
WSGI中间件	掌控认证，使用OpenStack认证服务。

OpenStack块存储

OpenStack块存储服务(cinder)为虚拟机添加持久的存储，块存储提供一个基础设施为了管理卷，以及和OpenStack计算服务交互，为实例提供卷。此服务也会激活管理卷的快照和卷类型的功能。

块存储服务通常包含下列组件：

cinder-api	接收API请求，然后将之路由到cinder-volume这里执行。
cinder-volume	直接和块存储的服务交互，以及诸如 cinder-scheduler这样的流程交互，它和这些流程交互是通过消息队列。cinder-volume 服务响应那些发送到块存储服务的读写请求以维护状态，它可以可多个存储供应商通过driver架构作交互。
cinder-scheduler守护进程	选择最佳的存储节点来创建卷，和它类似的组件是nova-scheduler。
cinder-backup daemon	cinder-backup 服务提供将卷备份到任何的后端存储中。类似于cinder-volume服务，它可通过驱动架构和各式各样的存储提供商交互。
消息队列	在块存储的进程之间路由信息。

OpenStack 网络

OpenStack网络允许用户为了创建和挂接网卡设备而由其他OpenStack服务来管理并连接。插件机制可实现容纳不同的网络设备和软件，为OpenStack架构和部署提供灵活的机制。

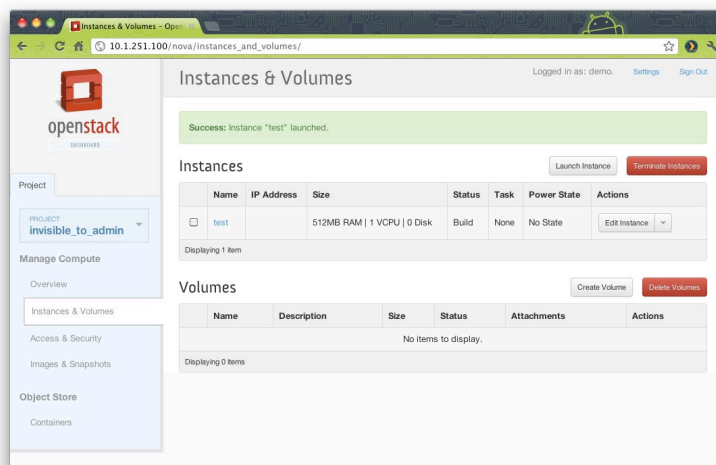
它包含下列组件：

neutron-server	接收和路由API请求到合适的OpenStack网络插件，以达到预想的目的。
OpenStack网络插件和代理	插拔端口，创建网络和子网，以及提供IP地址，这些插件和代理依赖于供应商和技术而不同，OpenStack网络基于插件和代理为Cisco 虚拟和物理交换机、NEC OpenFlow产品，Open vSwitch, Linux bridging以及VMware NSX 产品穿线搭桥。 常见的代理L3(3层)，DHCP(动态主机IP地址)，以及插件代理。
消息队列	用于在neutron-server和各种代理之间路由信息，也会为各种插件将网络状态存储到数据库中。

OpenStack网络主要和OpenStack计算交互，以提供网络连接到它的实例。

OpenStack仪表盘

OpenStack仪表盘是一个模块化的Django web 应用，其提供了访问OpenStack服务的图形化接口。



仪表盘通常通过在Apache中的mod_wsgi来部署的。你可以修改仪表盘的代码来适应不同的站点。

从网络架构的角度来看，此服务必须为最终用户可访问，且需要为所有其他的OpenStack服务公开API。如果要使用其他服务的管理功能，则必须可访问到 Admin API的入口，还得保证不能被最终用户访问到。

OpenStack 身份

OpenStack 身份服务 具有下面的功能：

- 追踪用户及他们的权限。
- 基于API端点提供可用的服务目录。

当安装OpenStack身份服务，用户必须将之注册到其OpenStack安装环境的每个服务。身份服务才可以追踪那些OpenStack服务已经安装，以及在网络中定位它们。

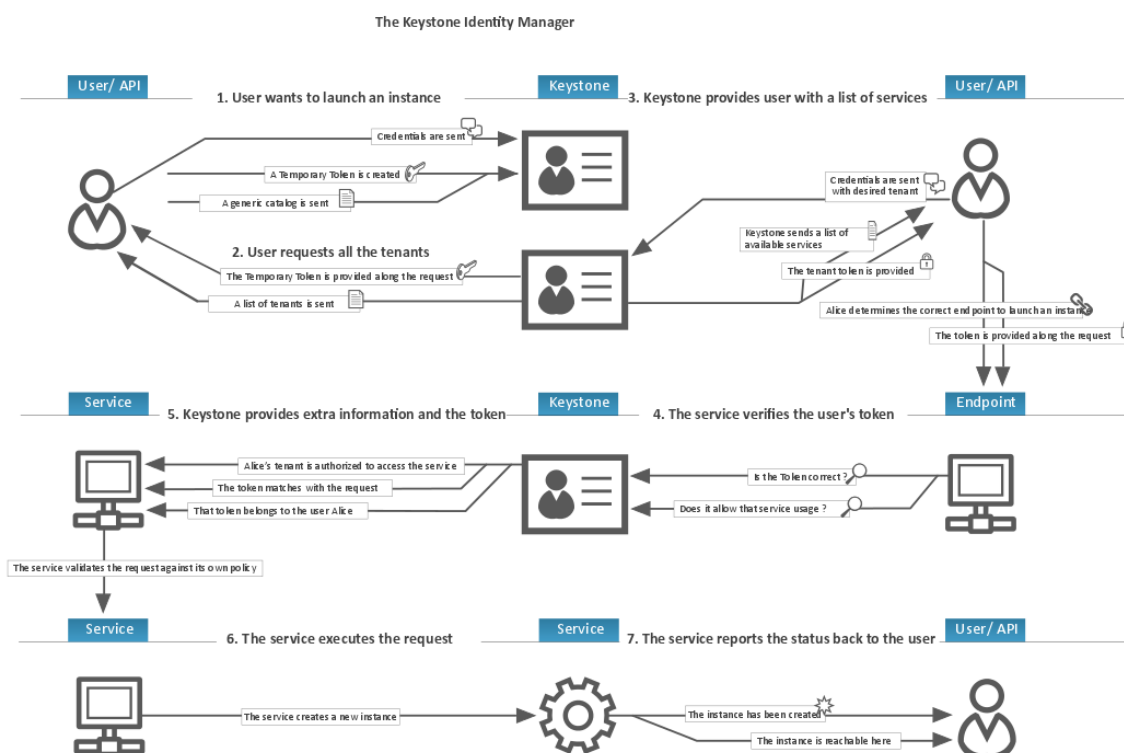
欲理解OpenStack的身份，须先理解以下概念：

用户	那些使用OpenStack云服务的人或系统或服务的数字表示。身份认证服务会验证那些生成调用的用户发过来的请求，用户登录且被赋予令牌以访问资源，用户可以直接被分配到特别的租户和行为，如果他们是被包含在租户中的。
凭证	用户身份的确认数据，例如，用户名和密码，用户名和API密钥，或者是一个有身份服务提供的授权令牌。
认证	确认用户身份的流程，OpenStack身份服务确认过来的请求，即验证由用户提供的凭证。

这些凭证通常是用户名和密码，或者是用户名和API密钥。当用户的凭证被验证通过，OpenStack身份服务给他一个认证令牌，令牌可以为用户提供随后的请求。

令牌	<p>一个字母数字混合的文本字符串，用户访问OpenStack API和资源，令牌可以随时撤销，以及有一定的时间期限。</p> <p>在此版本OpenStack身份服务支持了基于令牌的认证，这意味着会在将来支持更多的协议，它的主要目的是集成服务，而不希望成为一个完整的身份存储和管理解决方案。</p>
租户	用于组成或隔离资源的容器，租户会组成或隔离身份对象，这取决于服务的运维人员，一个租户会映射到一个客户，一个账户，一个组织或一个项目。
服务	一个OpenStack服务，如计算(nova)，对象存储(swift)，或者是镜像服务(glance)。它提供一个或多个端点，让拥护来访问资源和执行操作。
端点	一个当用户访问服务时用到的可访问的网络地址，通常是一个URL地址。如果用户为模板而使用扩展，一个端点可以被创建，它表示模板是为所有可用的跨region的可消费的服务。
角色	<p>定义了一组用户权限的用户，可赋予其执行某些特定的操作。</p> <p>在身份服务中，一个令牌所携带用户信息包含了角色列表。服务在被调用是会看用户是什么样的角色，这个角色赋予的权限能够操作什么资源。</p>
Keystone客户端	为OpenStack身份API提供的一组命令行接口。例如，用户可以运行keystone service-create 和 keystone endpoint-create命令在他们的OpenStack环境中去注册服务。

下面示意图展示了OpenStack身份认证流程:



OpenStack 镜像服务

OpenStack 镜像服务正如在“概念架构”一节 [2] 所展现的那样，是基础设施即服务(IaaS) 活动中心。它为磁盘或服务器镜像接收API请求，以及来自最终用户或OpenStack计算组件的镜像元数据请求。它还支持磁盘或服务器镜像存储到多种仓库类型，包括OpenStack对象存储。

一些定期的进程运行在OpenStack镜像服务至上，为的是支持缓存。复制服务可确保集群内的一致性和可用性。其它定期进程有审计，更新和扫尾。

OpenStack 镜像服务包含以下组件：

glance-api	接收镜像API的调用，诸如镜像发现、恢复、存储。
glance-registry	存储、处理和恢复镜像的元数据，元数据包括诸如大小和类型。



安全须知

注册表是一个私有的内部服务，用于OpenStack镜像内部。不要泄露给用户。

数据库	存放镜像元数据，用户是可以依据个人喜好选择数据库的，多数的部署使用MySQL或SQLite。
镜像文件的存储仓库	支持多种类型的仓库，它们有普通文件系统、对象存储、RADOS块设备、HTTP、以及亚马逊S3。记住，其中一些仓库仅支持只读方式使用。

OpenStack Telemetry 模块

Telemetry 模块表现有如下功能:

- 相关OpenStack服务的有效调查计量数据。
- 通过监测通知收集来自各个服务发送的事件和计量数据。
- 发布收集来的数据到多个目标，包括数据存储和消息队列。
- 当收集的数据超越预订的规则时可创建警告。

Telemetry 模块通常包含下面组件:

一个计算服务代理 (ceilometer-agent-compute)	运行在每个计算节点中，推送资源的使用状态，也许在未来会有其他类型的代理，但是目前来说社区专注于创建计算节点代理。
中心代理 (ceilometer-agent-central)	运行在中心管理服务器以推送资源使用状态，既不捆绑到实例也不在计算节点。代理可启动多个以横向扩展它的服务。
通知代理 (ceilometer-agent-notification)	运行在中心管理服务器(s)中，获取来自消息队列(s)的消息去构建事件和计量数据。
收集者 (ceilometer-collector)	运行在中心管理服务器(s),分发收集的telemetry数据到数据存储或者外部的消费者，但不会做任何的改动。

警告评估 (ceilometer-alarm-evaluator)	运行在一个或多个中心管理服务器，当警告发生是由于相关联的统计趋势超过阈值以上的滑动时间窗口，然后作出决定。
警告通知 (ceilometer-alarm-notifier)	运行在一个或多个中心管理服务器，允许警告为一组收集的实例基于评估阈值来设置。
API服务 (ceilometer-api)	运行在一个或多个中心管理服务器，提供从数据存储的数据访问。

这些服务使用OpenStack消息总线来通信，只有收集者和API服务可以访问数据存储。

OpenStack 编排模块

Orchestration模块提供了一个基于模板的orchestration，用于描述云的应用，通过运行的OpenStack API调用生成运行的云应用。软件和OpenStack其他核心组件集成为一个单一文件的模板系统。模板允许用户创建大多数的OpenStack资源类型，诸如实例，floating IP，卷，安全组，用户等，它也提供高级功能，诸如实例高可用，实例自动扩展，以及嵌套的OpenStack,这给OpenStack的核心项目带来了大量的用户基础。

服务鼓励部署者去直接集成Orchestration模块，或者通过自定义插件实现。

Orchestration模块通常包含下面的组件:

heat 命令行客户端	一个命令行工具，用于和heat-api通信，从而运行OpenStack本地的API。
heat-api 组件	一个OpenStack本地 REST API,发送API请求到heat-engine，通过远程过程调用(RPC)。
heat-api-cfn 组件	AWS 队列API，和AWS CloudFormation兼容，发送API请求到heat-engine，通过远程过程调用。
heat-engine	启动模板和提供给API消费者回馈事件。

OpenStack数据库服务

数据库服务提供可扩展性和可靠的云部署关系型和非关系性数据库引擎的功能。用户可以快速和轻松使用数据库的特性而无须掌控复杂的管理任务，云用户和数据库管理员可以按需部署和管理多个数据库实例。

数据库服务在高性能层次上提供了资源的隔离，以及自动化了复杂的管理任务，诸如部署、配置、打补丁、备份、恢复以及监控。

流程实例. 此例子是一个为使用数据库服务的高级别的流程：

1. OpenStack管理员使用下面的步骤来配置基本的基础设施：
 - a. 安装数据库服务。
 - b. 为每种类型的数据库制作各自的镜像。例如，一个是MySQL,一个是MongoDB。
 - c. 使用命令trove-manage来导入镜像以及为租户提供出去。

2. OpenStack最终用户使用下列步骤部署数据库服务:

- a. 使用命令 `trove create`来创建一个数据库服务的实例。
- b. 使用命令`trove list`获得实例的ID，下面命令`trove show`是获得此实例的IP地址。
- c. 访问数据库服务实例使用普通的数据库访问命令即可。例如，对于MySQL来说：

```
$ mysql -u myuser -p -h TROVE_IP_ADDRESS mydb
```

数据库服务包含下列组件：

python-troveclient 命令行客户端	一个和组件 <code>trove-api</code> 通信的命令行工具。
trove-api 组件	提供OpenStack本地的RESTful API，支持JSON格式的部署和管理Trove实例。
trove-conductor 服务	运行在主机上，接收来自guest实例的消息，然后将之更新在主机上。
trove-taskmanager 服务	能够支持部署实例，管理实例的生命周期，以及对实例的日常操作等复杂系统流的服务。
trove-guestagent 服务	运行在guest实例内部，管理和执行数据库自身的操作。

OpenStack数据处理服务

OpenStack(sahara)的数据处理服务旨在为用户提供一个简单的部署数据处理(Hadoop,Spark)集群，仅通过提供一些简单的参数如Hadoop版本，集群的拓扑，节点硬件系统就可搞定。当用户填写了所有的参数后，sahara花几分钟就可将集群部署完成。Sahara还提供了为已有的集群扩展，按需添加/删除工作节点。

解决方案可交付如下使用场景：

- 为开发和测试环境提供在OpenStack中快速的部署Hadoop集群。
- 在一个通用型OpenStack IaaS云中充分利用其未使用的计算能力。
- 为ad-hoc或突破分析负载提供分析即服务。

关键特性有：

- 基于OpenStack组件设计。
- 通过REST API来管理，其界面已经是OpenStack web界面的一部分，目前已可用。
- 支持不同的Hadoop发行版：
 - Hadoop安装引擎的可插拔系统。
 - 集成了供应商特别的管理工具，诸如Apache Ambari或Cloudera 管理终端。
- 对Hadoop配置模板的预定义，拥有可修改参数的能力。
- 基于Hive或Pig为ad-hoc分析查询提供友好的用户界面。

反馈

要为文档提供反馈，请使用openstack-docs@lists.openstack.org邮件列表加入我们，可在网站[OpenStack 文档邮件列表](#)找到，或者[报告bug](#)。

第 2 章 身份认证管理

目录

身份概念	17
证书的PKI	22
配置基于SSL的身份服务	26
身份的外部认证	27
集成LDAP身份认证	27
为令牌绑定配置身份服务	35
使用信托	36
缓存层	37
用户 CRUD	38
日志记录	39
启动认证服务	39
使用实例	39
使用用户名和密码认证中间件	40
基于角色访问控制(RBAC)的身份API保护	41
身份服务故障诊断	43

OpenStack身份，代号名Keystone,是OpenStack默认的身份管理系统。当用户安装完身份系统后，用户可通过修改配置文件etc/keystone.conf 来做相应的配置，还有可能的话，将日志配置文件分离。用户使用keystone 命令行客户端来为身份系统初始化数据。

身份概念

用户管理

认证用户管理主要的组件有：

- **用户**。就表示现实中的你、我、他。相关信息如用户名、密码以及电子邮箱。此例中是创建了一个名为alice的用户：

```
$ openstack user create --password-prompt --email alice@example.com alice
```

- **项目**。一个租户，组，或者是组织。当你向OpenStack服务发出请求时，你必须指定项目。举例来说，如果你查询计算服务，想获取运行中的实例的话，那么你得到的是在你的查询里所制定项目的所有运行实例。下面的例子是创建了一个名为acme的项目：

```
$ openstack project create acme
```

- **域**。为认证实体的管理定义了行政上的边界。一个域可表示一个个体，一个公司，或者属于运维的空间。它用于直接抛出行政的活动给系统用户。

一个域是项目和用户的集合。一个用户可被赋予域的管理员角色。一个域管理员可以在域内创建项目，用户和组，以及为用户和组分配角色。

此例创建了一个名为emea的域：

```
$ openstack --os-identity-api-version=3 domain create emea
```

- **角色**，在给定的租户中用户可以操作的权限范围。

此例子中创建了一个名称为 `compute-user` 的角色：

```
$ openstack role create compute-user
```



注意

独立的服务，如计算服务和镜像服务，赋予角色的意义。在身份服务中，一个角色就是一个简单的名字。

身份服务为用户分配一个租户和一个角色。用户也许会分配compute-user 角色给用户 alice 在租户 acme中:

```
$ openstack user list
```

```
+-----+-----+
| ID   | Name |
+-----+-----+
| 892585 | alice |
+-----+-----+
```

```
$ openstack role list
```

```
+-----+-----+
| ID   | Name      |
+-----+-----+
| 9a764e | compute-user |
+-----+-----+
```

```
$ openstack project list
```

```
+-----+-----+
| ID   | Name      |
+-----+-----+
| 6b8fd2 | acme      |
+-----+-----+
```

```
$ openstack role add --project 6b8fd2 --user 892585 9a764e
```

一个用户可以在不同的租户中有不同的角色。例如，Alice还是租户 Cyberdyne中的角色是admin。一个用户还可以是在同一个租户下用户多个角色。

/etc/[SERVICE_CODENAME]/policy.json文件控制着用户可执行给定服务的任务。举例来说，/etc/nova/policy.json 指定了计算服务的访问策略，/etc/glance/policy.json指定了镜像服务的访问策略，以及/etc/keystone/policy.json指定了身份服务的访问策略。

在计算，身份和镜像服务来说，默认的政策.json 文件只能识别admin角色：一个租户中任何用户，无论其是什么角色，都无需admin角色，都可以执行所有操作。

若用户希望限制用户执行某些操作，例如，计算服务，用户需要在身份服务中创建一个角色，然后修改文件/etc/nova/policy.json，因为这个角色需要操作计算服务。

举例,在配置文件 `/etc/nova/policy.json`下面这行指出了用户创建卷没有任何的限制:如果用户在租户中拥有角色,他就可以在那个租户中创建卷。

```
"volume:create": "",
```

欲限制拥有 `compute-user` 角色的用户在指定的租户创建卷,用户需要增加 `"role:compute-user"`,类似下面这样:

```
"volume:create": "role:compute-user",
```

要限制所有的计算服务请求,要求此角色,文件最终的结果是类似这样子的:

```
{
  "admin_or_owner": "role:admin or project_id:$(project_id)s",
  "default": "rule:admin_or_owner",
  "compute:create": "role:compute-user",
  "compute:create:attach_network": "role:compute-user",
  "compute:create:attach_volume": "role:compute-user",
  "compute:get_all": "role:compute-user",
  "compute:unlock_override": "rule:admin_api",
  "admin_api": "role:admin",
  "compute_extension:accounts": "rule:admin_api",
  "compute_extension:admin_actions": "rule:admin_api",
  "compute_extension:admin_actions:pause": "rule:admin_or_owner",
  "compute_extension:admin_actions:unpause": "rule:admin_or_owner",
  "compute_extension:admin_actions:suspend": "rule:admin_or_owner",
  "compute_extension:admin_actions:resume": "rule:admin_or_owner",
  "compute_extension:admin_actions:lock": "rule:admin_or_owner",
  "compute_extension:admin_actions:unlock": "rule:admin_or_owner",
  "compute_extension:admin_actions:resetNetwork": "rule:admin_api",
  "compute_extension:admin_actions:injectNetworkInfo": "rule:admin_api",
  "compute_extension:admin_actions:createBackup": "rule:admin_or_owner",
  "compute_extension:admin_actions:migrateLive": "rule:admin_api",
  "compute_extension:admin_actions:migrate": "rule:admin_api",
  "compute_extension:aggregates": "rule:admin_api",
  "compute_extension:certificates": "role:compute-user",
  "compute_extension:cloudpipe": "rule:admin_api",
  "compute_extension:console_output": "role:compute-user",
  "compute_extension:consoles": "role:compute-user",
  "compute_extension:createserverext": "role:compute-user",
  "compute_extension:deferred_delete": "role:compute-user",
  "compute_extension:disk_config": "role:compute-user",
  "compute_extension:evacuate": "rule:admin_api",
  "compute_extension:extended_server_attributes": "rule:admin_api",
  "compute_extension:extended_status": "role:compute-user",
  "compute_extension:flavorextradata": "role:compute-user",
  "compute_extension:flavorextraspecs": "role:compute-user",
  "compute_extension:flavormanage": "rule:admin_api",
  "compute_extension:floating_ip_dns": "role:compute-user",
  "compute_extension:floating_ip_pools": "role:compute-user",
  "compute_extension:floating_ips": "role:compute-user",
  "compute_extension:hosts": "rule:admin_api",
  "compute_extension:keypairs": "role:compute-user",
  "compute_extension:multinic": "role:compute-user",
  "compute_extension:networks": "rule:admin_api",
  "compute_extension:quotas": "role:compute-user",
  "compute_extension:rescue": "role:compute-user",
  "compute_extension:security_groups": "role:compute-user",
  "compute_extension:server_action_list": "rule:admin_api",
  "compute_extension:server_diagnostics": "rule:admin_api",
```

```

"compute_extension:simple_tenant_usage:show": "rule:admin_or_owner",
"compute_extension:simple_tenant_usage:list": "rule:admin_api",
"compute_extension:users": "rule:admin_api",
"compute_extension:virtual_interfaces": "role:compute-user",
"compute_extension:virtual_storage_arrays": "role:compute-user",
"compute_extension:volumes": "role:compute-user",
"compute_extension:volume_attachments:index": "role:compute-user",
"compute_extension:volume_attachments:show": "role:compute-user",
"compute_extension:volume_attachments:create": "role:compute-user",
"compute_extension:volume_attachments:delete": "role:compute-user",
"compute_extension:volumetypes": "role:compute-user",
"volume:create": "role:compute-user",
"volume:get_all": "role:compute-user",
"volume:get_volume_metadata": "role:compute-user",
"volume:get_snapshot": "role:compute-user",
"volume:get_all_snapshots": "role:compute-user",
"network:get_all_networks": "role:compute-user",
"network:get_network": "role:compute-user",
"network:delete_network": "role:compute-user",
"network:disassociate_network": "role:compute-user",
"network:get_vifs_by_instance": "role:compute-user",
"network:allocate_for_instance": "role:compute-user",
"network:deallocate_for_instance": "role:compute-user",
"network:validate_networks": "role:compute-user",
"network:get_instance_uuids_by_ip_filter": "role:compute-user",
"network:get_floating_ip": "role:compute-user",
"network:get_floating_ip_pools": "role:compute-user",
"network:get_floating_ip_by_address": "role:compute-user",
"network:get_floating_ips_by_project": "role:compute-user",
"network:get_floating_ips_by_fixed_address": "role:compute-user",
"network:allocate_floating_ip": "role:compute-user",
"network:deallocate_floating_ip": "role:compute-user",
"network:associate_floating_ip": "role:compute-user",
"network:disassociate_floating_ip": "role:compute-user",
"network:get_fixed_ip": "role:compute-user",
"network:add_fixed_ip_to_instance": "role:compute-user",
"network:remove_fixed_ip_from_instance": "role:compute-user",
"network:add_network_to_project": "role:compute-user",
"network:get_instance_nw_info": "role:compute-user",
"network:get_dns_domains": "role:compute-user",
"network:add_dns_entry": "role:compute-user",
"network:modify_dns_entry": "role:compute-user",
"network:delete_dns_entry": "role:compute-user",
"network:get_dns_entries_by_address": "role:compute-user",
"network:get_dns_entries_by_name": "role:compute-user",
"network:create_private_dns_domain": "role:compute-user",
"network:create_public_dns_domain": "role:compute-user",
"network:delete_dns_domain": "role:compute-user"
}

```

服务管理

身份服务提供了身份，令牌，目录和策略服务。它由下面组件组成：

- keystone Web 服务网关接口 (WSGI) 服务。可以运行在具有WSGI能力的web服务器中，如 Apache httpd就可提供身份服务。服务和API可以是分离的WSGI服务的实例。
- 身份服务功能。每个功能都有一个可插拔后端，从而允许以不同的方法使用特定的服务。大多数支持标准的后端如LDAP或SQL。

- keystone-all。将服务和管理API启在单一的进程中。使用联合的话keystone-all是不支持的。keystone-all是被WSGI服务赞成丢弃的。

身份服务还维护了一个用户对应一个服务，例如，名称为nova的用户是对应计算服务，另外，一个特殊的租户服务就叫做服务。

关于如何创建服务和端点的更多信息，请参阅 [OpenStack 管理员用户指南](#)。

组

一个组就是一个用户的集合。系统管理员可以创建组，然后为其添加用户。再然后，分别为每个用户赋予一个角色，或者赋予组某个角色。每个组都在某个域中。组从身份API v3开始引入。

身份 API V3提供了下列组相关的操作:

- 创建一个组
- 删除一个组
- 更新一个组(修改它的名称或描述)
- 为组添加一个用户
- 从组中删除一个用户
- 列出组成员
- 为用户列出组
- 在租户中为一个组赋予角色
- 在一域中为一个组赋予某角色
- 查询赋予组的角色



注意

身份服务也许不会允许所有的操作。举例来说，如果使用LDAP作为身份服务的后端，且组更新是禁用的，那么创建、删除或更新组的请求都会失败。

这里有两例子:

- A组被赋予在A租户下的A角色。如果用户A是A组的成员，当A用户从A租户获得令牌，此令牌也包含A角色。
- B组被赋予在B域下的B角色。如果用户B是B域的成员，当B用户从B域获得令牌，此令牌也包含B角色。

证书的PKI

PKI代表的是公钥基础设施。令牌均是文档，使用X509标准签署加密。为了能够正常的工作，令牌的生成需要一个公有/私有 密钥对。公钥必须签名到X509证书中，且证书所使用动签名必须能作为证书颁发(CA)来用。这些文件既可以使用 keystone-manage工具来生成，也可以使用外部工具生成。文件的位置须放在上面章节中所指定的身份服务配置文件keystone.conf的顶端。另外，私钥须保证只能让运行在身份服务的系统用户只读。



敬告

证书可以让所有的用户去读取，但是私钥不可以。私钥只能让打算签名令牌的账户读取。当使用`keystone-manage pki_setup`命令生成文件时，你最好是运行在PKI用户名下。如果你以`root`运行的`keystone-manage`，你可以为其添加参数`--keystone-user` 和 `--keystone-group`从而为keystone设置用户名和组。

指定到哪里去读取证书在配置文件中[signing]一节设置，配置的值是：

- `certfile` - 用于验证令牌的证书位置。默认是`/etc/keystone/ssl/certs/signing_cert.pem`。
- `keyfile` - 用于签名令牌的私钥位置。默认是`/etc/keystone/ssl/private/signing_key.pem`。
- `ca_certs` - 颁发上述证书的证书位置。默认是`/etc/keystone/ssl/certs/ca.pem`。
- `ca_key` - CA使用的私钥位置。默认是`/etc/keystone/ssl/private/cakey.pem`。
- `key_size` - 默认是 2048。
- `valid_days` - 默认是 3650。
- `cert_subject` - 证书主题(自动生成的证书)，针对令牌签名。默认是`/C=US/ST=Unset/L=Unset/O=Unset/CN=www.example.com`。

当使用命令`keystone-manage pki_setup`生成证书时，会用到`ca_key`, `key_size`, 以及`valid_days`的配置项。

如果不使用命令`keystone-manage pki_setup`来生成证书，或者你提供自己的证书，这三个值是不需要设置的。

如果在keystone的配置中的[token]一节设置了provider=keystone.token.providers.uuid.Provider，那么典型的令牌看起来是这样53f7f6ef0cc344b5be706bcc8b1479e1。如果是provider=keystone.token.providers.pki.Provider的话，典型的令牌是很长的字符，例如：

[illegible]

```
ZWdpb25PbmUiLCAiaW50ZXJuYWxVUkwiOiAiaHR0cDovLzE5Mi4xNjguMjcMTAwOjUwMDAvdJlUMCIscICJpZCI6ICJiNTY2Y2JlZjA2NjQ0ZmY2OWMyO  
dWJsaWNVUkwiOiAiaHR0cDovLzE5Mi4xNjguMjcMTAwOjUwMDAvdJlUMCJ9XSgImVuZHBvaW50c19saW5rcyI6IFtdLCAidHlwZSI6ICJpZGVudG0  
b25lIn1dLCAidXNlciI6IHsidXNlcm5hbWUiOiAiZGVtbYIsICJyb2xlc19saW5rcyI6IFtdLCAiaWQiOiAiZTVhMTM3NGE4YTRmNDI4NWlZyWQ3MzQ1MWU  
OiBbeyJuYW11IjogImFub3RoZXJyb2x1In0sIHsibmFtZSI6ICJNZW1iZXIifV0sICJuYW11IjogImRlbW8ifSwgIm1ldGFkYXRhIjogeyJpc19hZG1pbil6  
YWRiODM3NDVkyZQzNGJhZmk5ODI1NjBjOTIzYWZhMjgiLCAiMzZTFnJe1N2Y3NGFmZGJhNWUwYTUwYmUwNm5MmYiXX19fTGB-  
zCB-AIBATBcMFcxZzAJBgNVBAYTAIVTMQ4wDAYD  
VQQIEwVVBnNldEOMAwGA1UEBxMFVW5zZXQxDjAMBGNVBAOTBVVuc2VOMRgwFgYDVQQDEw93d3cuZXhhbXBsZS5jb20CAQEWBwYFKw4DAhQwDQY  
nouriuiCgFayIqCsshK3SVdhOMINiuJtqv0sE-wBDFIEj-Prcludqlz-n+6q7VgV4mwMPszz39-rwp  
+P5I4AjrJasUm7Fr0-4l02tPLaaZXU1gBQ1jUG5e5aL5jPDP08HbCWuX6wr-QQQB  
SrWY8lF3HrTcJT23sZlleg==
```

由外部CA颁发的注册证书

你可以使用由外部CA所颁发注册证书来取代keystone-manage生成的证书。然而，由外部CA颁发的证书须满足下列条件：

- 所有的证书和密钥文件必须是增强保密邮件(PEM)格式
- 私钥文件必须是没有被密码保护的

当使用由外部CA颁发的注册证书时，你无需指定key_size, valid_days, 和 ca_password，它们将会被忽略。

使用由外部CA颁发的注册证书的基本工作流涉及：

1. 从外部CA请求注册证书
2. 如果有需要的话，将证书和私钥转换为PEM
3. 安装外部注册证书

从一个外部CA请求一个注册证书

其中从外部CA请求注册证书的一个办法是使用 OpenSSL 命令行来第一次生成一个 PKCS #10 证书请求语法(CRS)。

创建一个证书请求配置文件。例如，创建如下cert_req.conf文件：

```
[ req ]  
default_bits      = 1024  
default_keyfile   = keystonekey.pem  
default_md        = sha1  
  
prompt            = no  
distinguished_name = distinguished_name  
  
[ distinguished_name ]  
countryName       = US  
stateOrProvinceName = CA  
localityName       = Sunnyvale  
organizationName   = OpenStack  
organizationalUnitName = Keystone  
commonName         = Keystone Signing  
emailAddress        = keystone@openstack.org
```

然后用OpenSSL 命令行生成一个CRS。不要为生成的私钥加密，必须使用-nodes属性。

例如：

```
$ openssl req -newkey rsa:1024 -keyout signing_key.pem -keyform PEM  
-out signing_cert_req.pem -outform PEM -config cert_req.conf -nodes
```

如果一切顺利的话，你会得到`signing_cert_req.pem`和`signing_key.pem`。发送`signing_cert_req.pem`给你的CA来请求一个令牌注册证书，且确保所询问的证书是PEM格式。还有，确保你信任的CA证书链也是PEM格式。

安装外部签名证书

假设你已经拥有下列内容项：

- `signing_cert.pem` - (Keystone token) PEM格式的这册证书
- `signing_key.pem` - 对应的(非加密) PEM格式的私钥
- `cacert.pem` - PEM格式的信任CA证书链

复制上述内容到你的证书目录。例如：

```
# mkdir -p /etc/keystone/ssl/certs  
# cp signing_cert.pem /etc/keystone/ssl/certs/  
# cp signing_key.pem /etc/keystone/ssl/certs/  
# cp cacert.pem /etc/keystone/ssl/certs/  
# chmod -R 700 /etc/keystone/ssl/certs
```



注意

确保证书目录之用root可访问：



注意

如果在第一次运行`keystone-manage pki_setup`之后则可改进复制私钥和证书文件的过程，因为此命令会创建其它需要的文件，诸如`index.txt`和`serial`文件。

同样，当拷贝必要的文件到不同的服务器，用来复制功能的话，文件的整个目录也是必要的，不仅仅是私钥和证书文件。

如果你的证书目录不是默认的`/etc/keystone/ssl/certs`，请在配置文件的`[signing]`一节也做相应的修改。

换出过期的签名证书

下面的步骤详细说明了如何换出过期的签名证书，还无须中断。

1. 生成新的签名密钥。
2. 生成新的证书请求。
3. 利用已有的CA签名新的证书来生成新的`signing_cert`。
4. 将新的`signing_cert`附加到旧的`signing_cert`，要确保旧的证书在文件的第一行。
5. 从所有的主机中删除所有的签名证书，强制OpenStack计算下载新的`signing_cert`。
6. 将旧的签名密钥替换为新的签名密钥。在`signing_cert`文件中移动新的签名证书到旧的之上。

7. 在旧的证书过期之后，你就可以从文件中将旧的签名证书安全的删除掉了。

配置基于SSL的身份服务

你可以配置身份服务来支持两种方式的SSL。

你必须遵守x509证书扩展并且配置它们。

身份服务在目录 `examples/pki/certs` 和 `examples/pki/private` 提供了一组证书的实例：

证书类型

<code>cacert.pem</code>	证书颁发机构链来验证。
<code>ssl_cert.pem</code>	身份服务器的公有证书。
<code>middleware.pem</code>	身份服务中间件/客户端端公有证书和私有证书。
<code>cakey.pem</code>	CA的私有证书。
<code>ssl_key.pem</code>	身份服务器的私钥。



注意

你可以为这些证书选择名称，如果你愿意的话，也可以讲公钥和私钥混在同一个文件中。这些证书都提供一个例子。

基于keystone-all的客户端认证

当运行keystone-all时，使用下面说明可将服务器配置为启用SSL的客户端认证。在文件`etc/keystone.conf`中修改 `[eventlet_server_ssl]` 一节。下面的SSL配置例子使用了所包含的实例证书：

```
[eventlet_server_ssl]
enable = True
certfile = <path to keystone.pem>
keyfile = <path to keystonekey.pem>
ca_certs = <path to ca.pem>
cert_required = True
```

属性

- `enable`。为True时表示启用SSL。默认是False。
- `certfile`。到身份服务公共证书文件的路径。
- `keyfile`。到身份服务私钥文件的路径。如果你在`certfile`中包含了私钥，你可以忽略此项配置。
- `ca_certs`。到CA信任链的路径。
- `cert_required`。需要客户端证书。默认为False。

当运行身份服务作为一个WSGI服务在诸如Apache httpd的web服务器时，web服务器中会替代此配置。在这种情况下，`[eventlet_server_ssl]`一节的属性会被忽略。

身份的外部认证

当身份服务运行在 `apache-httpd` 中时，你可以使用外部认证的方法，这不同于由身份存储后端所提供的认证。举例来说，你可以使用一SQL身份后端和X.509认证以及Kerberos，来代替使用用户名和密码的方式。

使用HTTPD认证

web服务器，例如Apache HTTP，支持多种认证方式。身份服务允许web服务器去执行认证。web服务器然后传送所认证过的用户给身份服务，这是通过使用 `REMOTE_USER` 环境变量实现的。此用户必须已经存在于后端的身份中，从而可以从控制器获取到令牌。要使用此方法，认证服务须运行在 `apache-httpd` 中。

使用 X.509

以下是Apache配置的用户摘要认证，基于从一个已知的CA的合法的X.509证书。

```
<VirtualHost _default_:5000>
  SSLEngine on
  SSLCertificateFile /etc/ssl/certs/ssl.cert
  SSLCertificateKeyFile /etc/ssl/private/ssl.key

  SSLCACertificatePath /etc/ssl/allowed_cas
  SSLCARevocationPath /etc/ssl/allowed_cas
  SSLUserName          SSL_CLIENT_S_DN_CN
  SSLVerifyClient       require
  SSLVerifyDepth        10

  (...)
</VirtualHost>
```

集成LDAP身份认证

OpenStack 身份服务支持整合已有的LDAP目录服务来做认证和授权。

当OpenStack身份服务被配置来使用LDAP的后端时，你可以将认证(使用身份 特性)和授权(使用分配 特性)。

身份特性启用了管理员通过每个域活OpenStack身份服务实体来管理用户和组。

分配这个特性启用管理员使用OpenStack认证服务的SQL数据库来管理项目角色认证，通过LDAP目录来提供用户认证。



重要

对于OpenStack认证服务访问LDAP服务器的话，你必须在OpenStack认证服务上启用SELinux的 `authlogin_nsswitch_use_ldap` 布尔值。要启用且确保重启后仍然生效：

```
# setsebool -P authlogin_nsswitch_use_ldap on
```

认证配置可分为两个独立的后端：认证(后端是用户和组)和分配(后端是域，项目，角色，角色分配)。要配置身份，在文件 `/etc/keystone/keystone.conf` 中设置一些属性，关于认证

后端的配置实例，请参阅“[将LDAP当作后端认证的集成](#)”一节 [29]，关于分配后端的配置实例，请参阅“[集成LDAP为分配的后端](#)”一节 [32]。按需修改这些例子。



注意

多后端是被支持的。你可以集成OpenStack认证服务，或基于单个对LDAP服务器（身份和分配都配置为LDAP，或设置身份和分配后端为SQL或LDAP），或者使用指定域配置文件的多后端。

定义目标LDAP服务器. 在文件keystone.conf定义目标LDAP服务：

```
[ldap]
url = ldap://localhost
user = dc=Manager,dc=example,dc=org
password = samplepassword
suffix = dc=example,dc=org
use_dumb_member = False
allow_subtree_delete = False
```



注意

请配置dumb_member，如果你设置了use_dumb_member为true的话。

```
[ldap]
dumb_member = cn=dumb,dc=nonexistent
```

外挂LDAP集成设置. 为单个LDAP服务在文件 /etc/keystone/keystone.conf中设置属性，或者在/etc/keystone/domains/keystone.DOMAIN_NAME.conf 文件中配置多后端。

查询属性 使用query_scope通过LDAP来控制数据表现的范围级别（仅搜索第一级或者是整个子树）。

使用page_size来控制每页到最大结果。值为零时表示弃用分页。

使用alias_dereferencing来控制LDAP查询的非关联化选择。

使用chase_referrals来覆盖系统的默认推荐行为。

```
[ldap]
query_scope = sub
page_size = 0
alias_dereferencing = default
chase_referrals =
```

调试 使用debug_level为LDAP调用设置LDAP调试级别。若值为零的话意味着调试没有启用。

```
[ldap]
debug_level = 0
```



警告

此值为一个位掩码，参考你的LDAP文档来可以设置什么值。

连接池 使用use_pool来启用LDAP连接池。配置连接池的大小，最大尝试次数，重连尝试，超时（-1表示不确定的等待）以及以秒算的生命周期。

```
[ldap]
use_pool = true
pool_size = 10
pool_retry_max = 3
pool_retry_delay = 0.1
pool_connection_timeout = -1
pool_connection_lifetime = 600
```

最终用户认证的连接池 使用use_auth_pool 来为最终用户认证启用LDAP连接池。配置连接池大小和以秒算的生命周期。

```
[ldap]
use_auth_pool = false
auth_pool_size = 100
auth_pool_connection_lifetime = 60
```

当你完成这些配置时，重启OpenStack身份服务：

```
# service keystone restart
```



警告

在服务重启期间，认证和授权不可用。

将LDAP当作后端认证的集成

身份后端包括用户，组，以及组成员列表的信息。与LDAP集成的身份后端允许管理员使用LDAP中的用户和组。



重要

对于OpenStack身份服务访问LDAP服务器，你必须在keystone.conf文件中定义目标LDAP服务器。更多信息，请参阅“集成LDAP身份认证”一节 [27]。

将LDAP当作后端认证的集成

1. 在keystone.conf文件中启用LDAP身份驱动。这允许LDAP当作身份的后端：

```
[identity]
#driver = keystone.identity.backends.sql.Identity
driver = keystone.identity.backends.ldap.Identity
```

2. 在LDAP目录中创建组织单元(OU)，并在keystone.conf文件中定义响应的位置：

```
[ldap]
user_tree_dn = ou=Users,dc=example,dc=org
user_objectclass = inetOrgPerson

group_tree_dn = ou=Groups,dc=example,dc=org
group_objectclass = groupOfNames
```



注意

这些模式属性均是可扩展的，用于和多种模式兼容。举例来说，此实体映射到活动目录中的person属性：


```
user_objectclass = person
```

3. 强烈建议LDAP的集成为只读。这些权限可应用到对象类型均在keystone.conf文件中：

```
[ldap]
user_allow_create = False
user_allow_update = False
user_allow_delete = False

group_allow_create = False
group_allow_update = False
group_allow_delete = False
```

4. 重启OpenStack身份服务：

```
# service keystone restart
```



警告

在服务重启期间，认证和授权不可用。

集成多后端的身份服务

1. 在/etc/keystone/keystone.conf文件中设置以下属性：

- a. 启用LDAP驱动：

```
[identity]
#driver = keystone.identity.backends.sql.Identity
driver = keystone.identity.backends.ldap.Identity
```

- b. 启用特定域驱动：

```
[identity]
domain_specific_drivers_enabled = True
domain_config_dir = /etc/keystone/domains
```

2. 重启服务：

```
# service keystone restart
```

3. 使用仪表盘或OpenStack客户端命令行列出域。关于OpenStack客户端命令行列表请参考[命令列表](#)。
4. 使用OpenStack仪表盘或OpenStack客户端命令行创建域。
5. 针对每个域，在/etc/keystone/domains 目录下创建一个特定的配置文件。使用的命名格式为keystone.DOMAIN_NAME.conf，其中DOMAIN_NAME是上面步骤中所分配动域名。



注意

在文件/etc/keystone/domains/keystone.DOMAIN_NAME.conf中设置的属性会覆盖掉/etc/keystone/keystone.conf 文件中设置的属性。

6. 在/etc/keystone/domains/keystone.DOMAIN_NAME.conf文件中定义目标LDAP服务器。例如：


```
[ldap]
url = ldap://localhost
user = dc=Manager,dc=example,dc=org
password = samplepassword
suffix = dc=example,dc=org
use_dumb_member = False
allow_subtree_delete = False
```

7. 在LDAP目录中创建组件单元(OU)，然后在文件/etc/keystone/domains/keystone.DOMAIN_NAME.conf 中定义它们对应的位置。例如：

```
[ldap]
user_tree_dn = ou=Users,dc=example,dc=org
user_objectclass = inetOrgPerson

group_tree_dn = ou=Groups,dc=example,dc=org
group_objectclass = groupOfNames
```



注意

这些模式属性均是可扩展的，用于和多种模式兼容。举例来说，此实体映射到活动目录中的person属性：

```
user_objectclass = person
```

8. 强烈建议LDAP的集成为只读。这些权限可应用到对象类型均在/etc/keystone/domains/keystone.DOMAIN_NAME.conf文件中：

```
[ldap]
user_allow_create = False
user_allow_update = False
user_allow_delete = False

group_allow_create = False
group_allow_update = False
group_allow_delete = False
```

9. 重启OpenStack身份服务：

```
# service keystone restart
```



警告

在服务重启期间，认证和授权不可用。

外挂LDAP集成设置. 在 /etc/keystone/keystone.conf设置这些属性，用于单个的LDAP服务器，或者是在/etc/keystone/domains/keystone.DOMAIN_NAME.conf中设置，用于多后端。

过滤器 使用过滤器通过LDAP来控制数据表现的范围。

```
[ldap]
user_filter = (memberof=cn=openstack-users,ou=workgroups,dc=example,dc=org)
group_filter =
```

身份属性映射 掩饰账户状态值(包括任何额外的属性映射)，用于兼容多种目录服务。多余的账户由user_filter过滤。

设置属性，忽略属性列表被更新。

举例来说，你可以在keystone.conf中掩饰活动目录账户状态属性：

```
[ldap]
user_id_attribute    = cn
user_name_attribute  = sn
user_mail_attribute  = mail
user_pass_attribute  = userPassword
user_enabled_attribute = userAccountControl
user_enabled_mask     = 2
user_enabled_invert   = false
user_enabled_default  = 51
user_default_project_id_attribute =
user_attribute_ignore = default_project_id,tenants
user_additional_attribute_mapping =

group_id_attribute    = cn
group_name_attribute  = ou
group_member_attribute = member
group_desc_attribute  = description
group_attribute_ignore =
group_additional_attribute_mapping =
```

启用模拟 另一种方法来确定用户是否启用或者是否通过检查,如果用户是模拟组的一员的话。

当使用启用模拟时，使用组实体的DN来掌控已启用的用户。

```
[ldap]
user_enabled_emulation = false
user_enabled_emulation_dn = false
```

集成LDAP为分配的后端

当你配置OpenStack身份服务来使用LDAP服务器时，你可以使用assignment这个特性将认证和授权分离。集成LDAP作为分配的后端允许管理在LDAP中去使用项目(租户)，角色，域和角色分配。



注意

不建议使用LDAP作为分配的后端。



注意

OpenStack身份服务不支持指定域的分配后端。



重要

对于OpenStack身份分配访问LDAP服务器，你必须在 keystone.conf 文件中定义目标LDAP服务器。更多信息，请参考“[集成LDAP身份认证](#)”一节 [27]。

集成LDAP为分配的后端

1. 启用分配驱动，在[assignment]一节，设置driver的配置键为keystone.assignment.backends.sql.Assignment:

```
[assignment]
#driver = keystone.assignment.backends.sql.Assignment
driver = keystone.assignment.backends.ldap.Assignment
```

2. 在LDAP目录中创建组织单元(OU)，然后在keystone.conf文件中定义它们相应的位置：

```
[ldap]
role_tree_dn =
role_objectclass = inetOrgPerson

project_tree_dn = ou=Groups,dc=example,dc=org
project_objectclass = groupOfNames
```



注意

这些模式属性均是可扩展的，用来兼容多种模式。举例来说，此实体映射到活动目录中的 groupOfNames属性：

```
project_objectclass = groupOfNames
```

3. 强烈建议LDAP的集成为只读。这些权限可应用到对象类型均在keystone.conf文件中：

```
[ldap]
role_allow_create = False
role_allow_update = False
role_allow_delete = False

project_allow_create = False
project_allow_update = False
project_allow_delete = False
```

4. 重启OpenStack身份服务：

```
# service keystone restart
```



警告

在服务重启期间，认证和授权不可用。

外挂LDAP集成设置. 在 /etc/keystone/keystone.conf设置这些属性，用于单个的LDAP服务器，或者是在/etc/keystone/domains/keystone.DOMAIN_NAME.conf中设置，用于多后端。

过滤器 使用过滤器通过LDAP来控制数据表现的范围。

```
[ldap]
project_filter = (member=cn=openstack-user,ou=workgroups,dc=example,dc=org)
role_filter =
```



警告

过滤方法

分配属性映射 掩饰账户状态值(包括任何额外的属性映射)，用于兼容多种目录服务。多余的账户由user_filter过滤。

设置属性，忽略属性列表被更新。

```
[ldap]
role_id_attribute = cn
role_name_attribute = ou
role_member_attribute = roleOccupant
role_additional_attribute_mapping =
role_attribute_ignore =

project_id_attribute = cn
project_name_attribute = ou
project_member_attribute = member
project_desc_attribute = description
project_enabled_attribute = enabled
project_domain_id_attribute = businessCategory
project_additional_attribute_mapping =
project_attribute_ignore =
```

启用模拟 另一种方法来确定项目是否启用或者是否通过检查,如果项目是模拟组的一员的话。

当使用启用模拟时,使用组实体的DN来掌控已启用的项目。

```
[ldap]
project_enabled_emulation = false
project_enabled_emulation_dn = false
```

加强OpenStack身份服务到后端LDAP连接的安全性

身份服务支持使用TLS加密LDAP的流量。在配置此之前,你首先必须验证你的证书颁发机构文件在哪里。更多信息,请参阅 [“证书的PKI”一节 \[22\]](#)。

一旦你验证了你的证书颁发机构文件的位置:

配置TLS加密的LDAP流量

1. 打开/etc/keystone/keystone.conf配置文件。
2. 找到 [ldap]一节。
3. 在[ldap]一节中,设置配置键use_tls为True。这样就打开了TLS。
4. 配置身份服务来使用你的证书颁发机构文件。要完成这步,设置ldap节中的tls_cacertfile配置键为证书颁发机构文件的路径。



注意

你也可以设置tls_cacertdir (也在ldap一节)为保存所有的证书颁发机构的目录。如果同时设置了tls_cacertfile 和 tls_cacertdir,后者会被忽略。

5. 指定来自LDAP服务器的TLS会话执行证书检查的行为。要做这些,在 [ldap] 一节设置tls_req_cert配置项。有三个选择,分别是 demand, allow, 或never:
 - demand: 一直要求来自LDAP服务器的证书。如果没有提供证书会话就会终止。如果所提供的证书经验证,不能满足现有的证书颁发机构,会话也会被终止。
 - allow: 一直要求来自LDAP服务器的证书。如果没有提供证书会话进程照样。如果所提供的证书经验证,不能满足现有的证书颁发机构,证书会被忽略且会话照常进行。

- never: 不需要提供证书。

在安装有openstack-config的发行版中，你可以通过运行下面命令来配置TLS加密LDAP的流量：

```
# openstack-config --set /etc/keystone/keystone.conf
ldap use_tls True
# openstack-config --set /etc/keystone/keystone.conf
ldap tls_cacertfile CA_FILE
# openstack-config --set /etc/keystone/keystone.conf
ldap tls_req_cert CERT_BEHAVIOR
```

地点：

- CA_FILE 用于加密LDAP流量的证书颁发机构文件的绝对路径。
- CERT_BEHAVIOR: 指定来自LDAP服务器的TLS会话执行证书检查的行为 (demand, allow, 或 never)。

为令牌绑定配置身份服务

令牌绑定内嵌了外部认证机制的信息，诸如Kerberos服务或X.509证书。通过使用令牌绑定，客户端可强制使用指定外部认证机制的令牌。此额外的安全机制确保，假如令牌被偷，没有额外的认证是无法使用的。

在keystone.conf文件中为令牌绑定配置认证类型：

```
[token]
bind = kerberos
```

或

```
[token]
bind = x509
```

kerberos 和 x509 目前均支持。

要强制检查令牌绑定，为这些模式的其中一种设置enforce_token_bind属性：

- 禁用
禁用令牌绑定检查。
- 许可
启用绑定检查，如果令牌绑定到一个未知的认证机制，服务会忽略它。默认是此模式。
- 严格
启用绑定检查。如果令牌绑定到一个未知的认证机制，服务会拒绝它。
- 必须
启用检查，需要为令牌使用至少一种认证机制。
- kerberos

启用绑定检查，需要为令牌使用kerberos作为认证机制：

```
[token]
enforce_token_bind = kerberos
```

• x509

启用绑定检查。需要为令牌使用X.509作为认证机制：

```
[token]
enforce_token_bind = x509
```

使用信托

OpenStack 身份服务管理着认证和授权。信任是OpenStack身份认证的一个扩展，其启用了委托，而且还可选择通过模拟 keystone。信任的扩展定义如下之间的关系：

委托人 用户委托给其他用户只能是其的一个子集。

受托人 用户信任被委托，是有时间的限制的。

信任可以允许委托者模拟信任者。基于安全的原因，增加了一些保险措施。例如，如果一个信任者丢失了所给定的角色，任何基于此角色的用户都会遇到问题，以及相关的令牌都会自动撤销。

委托的参数是：

用户ID 委托人和受托人的用户ID。

特权 委托的权限是租户ID和多个角色的组合，而角色必须是被分配给信任者的子集。
如果你忽略了所有的授权，没有什么可委托的，你不可能委托所有。

委托深度 定义委托是否为递归。如果它是递归的，定义委托链的长度。
指定下面值的其中一个：

- 0. 该代表不能再委派这些权限。
- 1. 代理人可以权限委托给任何一组的代表，但后者不能再委托。
- inf. 委托是无限递归的。

端点 和委托相关联的端点列表。

此参数仅用于限制委托给特定的端点。如果你忽略了端点，委托就没有了用处。特殊值all_endpoints允许信任用于所有和委托租户相关的端点。

存在时间 (可选)由信任的开始时间和结束时间组成。

缓存层

OpenStack认证支持上述配置的子系统的缓存层(例如, 令牌, 赋值)。OpenStack认证使用 [dogpile.cache](#) 库, 此库支持灵活的后端缓存系统。主要的缓存配置在文件 `keystone.conf` 的 `[cache]` 一节, 然而, 其它的节都有是否支持缓存的布尔值定义, 用来切换是否支持缓存。

所以要仅仅启用令牌的后端缓存的话, 按下面值设置即可:

```
[cache]
enabled=true

[assignment]
caching=false

[token]
caching=true
```



注意

Juno版本发布后, 默认的设置是启用了子系统的缓存的。但是全局的开关是关闭的。那结果就是, 如果不将 `[cache]` 全局开关的值设置为 `true` 的话, 是用不了缓存的。

令牌缓存以及令牌合法性

令牌系统有单独的 `cache_time` 配置项, 既可以设置为上述的项, 也可以设置为下面全局的 `expiration_time`, 允许OpenStack认证中其它系统有不同的缓存行为。此属性在配置文集中的 `[token]` 一节设置。

令牌废弃列表缓存时间由 `[token]` 一节中的配置属性 `revocation_cache_time` 来掌控。该废弃列表会在每个令牌撤销就刷新一次。通常可看到是请求显著的多于令牌检索或令牌的合法验证调用。

这里是一个受缓存时间影响的行为列表: 获取新的令牌, 撤销令牌, 验证令牌, 检查v2令牌, 以及检查v3令牌。

删除令牌API调用无效, 所操作的高速缓存令牌, 以及无效的高速缓存撤销令牌列表和验证/检查令牌调用。

令牌缓存单独配置 `revocation_list` 缓存。从令牌驱动到令牌管理释放过期的检查, 这确保了令牌即使过期了可以发出 `TokenNotFound` 标志。

对于缓存一致性, 所有的令牌ID都在提供者和令牌驱动级别转换为短的令牌哈希值。其中一些方法可以访问全ID(PKI 令牌), 另外一些则不可以。缓存失效是没有令牌ID规范化的不一致。

围绕CRUD分配的缓存

分配系统拥有分离的 `cache_time` 配置属性, 其可以设置为上述的值, 或者是下面的默认全局 `expiration_time`, 允许认证服务内部不同的系统有不同的缓存行为。此属性在配置文件中的 `[assignment]` 一节设置。

目前 分配为project, domain, 和 role指定的需求(主要围绕CRUD动作)提供缓存。缓存目前实现还不是很全面。list方法都是不能实现缓存的。

下面是受影响的分配操作的列表：分配域API，分配项目API，以及分配角色API。

要对域，项目和角色进行创建，更新和删除的操作，得执行上述所列出的缓存方法的合法性验证。



注意

如果使用了只读的assignment 后端，缓存在后端就不会立即体现出变更。在作出反应之前任何的变更都需要cache_time(如果在配置文件设置了[assignment])或者是全局的expiration_time (在配置文件的[cache]一节设置)。如果此类型有延迟的问题的话(当使用了只读 assignment后端)，建议将缓存的assignment禁用。要禁用指定的assignment，在[assignment]一节中的配置，设置caching 为 False。

更多关于不同后端的信息(以及配置属性)，请参考：

- [dogpile.cache.backends.memory](#)
- [dogpile.cache.backends.memcached](#)



注意

内存后端不适用于生产环境。

- [dogpile.cache.backends.redis](#)
- [dogpile.cache.backends.file](#)
- [keystone.common.cache.backends.mongo](#)

例 2.1. 配置后端的Memcached

以下实例展示了如何配置memcached后端:

```
[cache]
enabled = true
backend = dogpile.cache.memcached
backend_argument = url:127.0.0.1:11211
```

你需要给backend_argument指定URL参数以访问memcached实例。

用户 CRUD

认证提供了一个用户 CRUD (Create, Read, Update, and Delete) 过滤，它可以添加到 public_api 管道中。用户 CRUD 过滤允许用户使用一个 HTTP PATCH 来修改他们自己的密码。要启用扩展，您需要定义一个 user_crud_extension 过滤器，将其插入 keystone-paste.ini 文件中 public_api WSGI 管道的 *_body 中间件之后和 public_service 应用之前。例如：


```
[filter:user_crud_extension]
paste.filter_factory = keystone.contrib.user_crud:CrudExtension.factory

[pipeline:public_api]
pipeline = sizelimit url_normalize request_id build_auth_context token_auth admin_token_auth json_body ec2_extension
user_crud_extension public_service
```

每个用户可通过一个HTTP PATCH来修改他们自己的密码:

```
$ curl -X PATCH http://localhost:5000/v2.0/OS-KSCRUd/users/USERID -H "Content-type: application/json"
-H "X_Auth-Token: AUTHTOKENID" -d '{"user": {"password": "ABCD", "original_password": "DCBA"}}'
```

另外，修改他们的密码，所有用户当前的令牌均会无效。



注意

当测试时用于令牌的仅使用KVS后端。

日志记录

您可以在认证的外部配置记录。指定记录配置的文件名称以 `keystone.conf` 文件中 [DEFAULT] 小节的 `log_config` 选项来设置。要通过 `syslog` 路由记录，请设置 [DEFAULT] 小节中的 `use_syslog=true`。

项目中有一个可用的样例记录配置文件在 `etc/logging.conf.sample` 中。类似于 OpenStack 的其他项目，认证使用 Python 记录模块，它会提供广泛的配置选项，让您定义输出级别和格式。

启动认证服务

要启动身份服务，运行下面命令：

```
$ keystone-all
```

这个命令启动了两个由如之前所描述的 `keystone.conf` 文件配置的 `wsgi.Server` 实例。其中一个 `wsgi server` 是 `admin` (管理员 API) 而另一个是 `main` (主/公共 API 接口)。二者均运行在单个进程中。

使用实例

`keystone` 客户端设置为预期命令的 `keystonecommandargument` 形式，后接类似标志的关键字参数，以提供额外的 (通常是可选的) 信息。例如，`user-list` 和 `tenant-create` 命令可以用下列方式调用：

```
# Using token auth env variables
export OS_SERVICE_ENDPOINT=http://127.0.0.1:5000/v2.0/
export OS_SERVICE_TOKEN=secrete_token
keystone user-list
keystone tenant-create --name demo

# Using token auth flags
keystone --os-token secrete --os-endpoint http://127.0.0.1:5000/v2.0/ user-list
keystone --os-token secrete --os-endpoint http://127.0.0.1:5000/v2.0/ tenant-create --name=demo

# Using user + password + project_name env variables
export OS_USERNAME=admin
export OS_PASSWORD=secrete
export OS_PROJECT_NAME=admin
openstack user list
```

```
openstack project create demo

# Using user + password + project-name flags
openstack --os-username admin --os-password secrete --os-project-name admin user list
openstack --os-username admin --os-password secrete --os-project-name admin project create demo
```

使用用户名和密码认证中间件

您还可以使用 `admin_user` 和 `admin_password` 选项来配置身份认证中间件。



注意

属性 `admin_token` 已放弃，且不会再用于配置 `auth_token` 中间件。

对于那些拥有分开粘贴部署的 `.ini` 文件的服务，您可以在主配置文件的 `[keystone_authtoken]` 小节配置认证中间件，如 `nova.conf`。

然后在 `nova.conf` 中设置如下值：

```
[DEFAULT]
...
auth_strategy=keystone

[keystone_authtoken]
auth_uri = http://控制器:5000/v2.0
identity_uri = http://控制器:35357
admin_user = admin
admin_password = SuperSekretPassword
admin_tenant_name = service
```



注意

在粘贴配置参数优先的中间件参数。您必须删除它们使用的值在 `[keystone_authtoken]` 部分。



注意

注释所有 `auth_host`、`auth_port` 和 `auth_protocol` 选项，因为 `identity_uri` 已经包括了它们。

这个示例粘贴了配置过滤利用 `admin_user` 和 `admin_password` 选项：

```
[filter:authtoken]
paste,filter_factory = keystonemiddleware.auth_token:filter_factory
auth_uri = http://控制器:5000/v2.0
identity_uri = http://控制器:35357
auth_token = 012345SECRET99TOKEN012345
admin_user = admin
admin_password = keystone123
```



注意

使用这个选项需要一个管理员的租户/角色。管理员用户被授予了访问管理员租户的管理员角色的权限。



注意

注释所有 `auth_host`、`auth_port` 和 `auth_protocol` 选项，因为 `identity_uri` 已经包括了它们。

基于角色访问控制(RBAC)的身份API保护

正如多数的 OpenStack 项目那样，认证支持通过基于 RBAC 方法定义的策略规则来保护其 API。认证保存一个引用到主要认证配置文件 keystone.conf 的策略 JSON 文件中。通常这个文件命名为 policy.json，并且包含了什么角色拥有访问所定义的服务的特定操作的规则。

每个认证 API v3 调用在策略文件中都有一行规定了访问生效的管理层次。

```
API_NAME: RULE_STATEMENT or MATCH_STATEMENT
```

地点：

RULE_STATEMENT 包含了 RULE_STATEMENT 或 MATCH_STATEMENT。

MATCH_STATEMENT 是认证的一个设置，它必需匹配由 API 调用方提供的令牌和 API 的参数或目标实体调用的问题。例如：

```
"identity:create_user": [{"role:admin", "domain_id:$(user.domain_id)s"}]
```

假设要创建一个用户，您的令牌必需有管理员角色，而且令牌中（这意味着它必需是一个域范围的令牌）的 domain_id 必需匹配您所要创建的用户项目的 domain_id。换言之，您必需在您所要创建的用户域中拥有管理员角色，并且您所使用的令牌必需在该域范围内。

每个匹配的组件使用这样的格式：

```
ATTRIB_FROM_TOKEN:CONSTANT or ATTRIB_RELATED_TO_API_CALL
```

身份服务需要这些属性：

令牌中的属性：user_id、domain_id 或 project_id 依赖于域，以及您在这个范围内所拥有的角色列表。

API 调用相关的属性：任何传递到 API 调用的参数以及任何在查询字符串中指定的过滤器都是可用的。您要以 object.attribute 语法来关联对象的属性（例如，user.domain_id）。API 的目标属性也可以用 target.object.attribute 语法。例如：

```
"identity:delete_user": [{"role:admin", "domain_id:$(target.user.domain_id)s"}]
```

需要确认认证服务仅仅删除了所提供的令牌的同一个域中的用户对象。

每个目标对象都可以有一个 `id` 和一个 `name` 作为 `target.OBJECT.id` 和 `target.OBJECT.name`。认证从数据库获取其他属性，属性因对象类型而不同。认证服务过滤了一些数据库字段，例如用户密码。

对象属性列表：

```
role:
  target.role.id
  target.role.name

user:
  target.user.default_project_id
  target.user.description
  target.user.domain_id
  target.user.enabled
  target.user.id
```

```
target.user.name

group:
  target.group.description
  target.group.domain_id
  target.group.id
  target.group.name

domain:
  target.domain.enabled
  target.domain.id
  target.domain.name

project:
  target.project.description
  target.project.domain_id
  target.project.enabled
  target.project.id
  target.project.name
```

默认的 `policy.json` 文件提供了一个基于 API 保护的示例，并不承担任何域的特殊用途。如果云供应商希望授权域的内容管理员为一个特殊的管理域，可以参考 `policy.v3cloudsample.json` 作为多域配置安装的示例。这个样例策略文件也展示了如何使用 `admin_domain` 来允许云供应商启用云管理员，以通过 API 拥有更广的访问权限。

一个干净的安装可以以标准策略文件开始，允许在其中创建包含第一个用户的 `admin_domain`。然后您可以获取管理域的 `domain_id`，将 ID 粘贴到一个修改了版本的 `policy.v3cloudsample.json` 文件，将其启用为主策略文件。

身份服务故障诊断

要对身份服务进行故障排除，检查/var/log/keystone/keystone.log 文件中的日志。



注意

使用/etc/keystone/logging.conf文件来配置日志文件的位置。

日志显示了向WSGI发出请求的组件，理想情况下会显示，解释了为什么一个授权请求失败的错误。如果在日志中没有看到请求，运行带有参数--debug的keystone命令，请将--debug置于所有命令参数之前。

调试PKI中间件

当你和OpenStack打交道时，如果接收到Invalid OpenStack Identity Credentials的消息，这在Grizzly版中是由从UUID令牌向PKI令牌转换中引起的。学习下如何修复此错误。

基于PKI令牌的验证模式依赖于来自身份服务的证书，而此证书是通过HTTP拿到的并存放在本地目录。此目录的具体位置是由signing_dir配置项所决定的。在你的服务的配置文件中，找到类似如下内容的一节：

```
[keystone_authtoken]
signing_dir = /var/cache/glance/api
auth_uri = http://控制器:5000/v2.0
identity_uri = http://控制器:35357
admin_tenant_name = service
admin_user = glance
```

检查的第一件事就是signing_dir，事实上是看它是否存在，如果存在的话，检查其中的证书文件是否在：

```
$ ls -la /var/cache/glance/api/

total 24
drwx-----, 2 ayoung root 4096 Jul 22 10:58 .
drwxr-xr-x, 4 root root 4096 Nov 7 2012 ..
-rw-r-----, 1 ayoung ayoung 1424 Jul 22 10:58 cacert.pem
-rw-r-----, 1 ayoung ayoung 15 Jul 22 10:58 revoked.pem
-rw-r-----, 1 ayoung ayoung 4518 Jul 22 10:58 signing_cert.pem
```

此目录包含了两个证书和一个令牌废弃列表。如果这些文件都不存在，你的服务就无法从身份服务中获取它们。要修复此问题，如下面所示，尝试和身份服务沟通确保它是正确的文件：

```
$ curl http://localhost:35357/v2.0/certificates/signing
```

此命令获取签名证书：

```
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number: 1 (0x1)
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=US, ST=Unset, L=Unset, O=Unset, CN=www.example.com
Validity
  Not Before: Jul 22 14:57:31 2013 GMT
  Not After : Jul 20 14:57:31 2023 GMT
Subject: C=US, ST=Unset, O=Unset, CN=www.example.com
```

请注意，已过期的证书：

```
Not Before: Jul 22 14:57:31 2013 GMT
Not After : Jul 20 14:57:31 2023 GMT
```

令牌废弃列表会每分钟更新一次，但是证书不会。一个可能的问题是证书是错误的或者是废弃的。你可以删除这些文件，然后运行其它命令来满足你的服务器：它们按需获取。

身份服务日志须显示证书文件的访问记录。你要上调你的日志级别。在你的身份配置文件中设置 `debug = True` 和 `verbose = True`，然后重启身份服务。

```
(keystone.common.wsgi): 2013-07-24 12:18:11,461 DEBUG wsgi __call__
arg_dict: {}
(access): 2013-07-24 12:18:11,462 INFO core __call__ 127.0.0.1 -- [24/Jul/2013:16:18:11 +0000]
"GET http://localhost:35357/v2.0/certificates/signing HTTP/1.0" 200 4518
```

如果完成这些之后，文件还没有出现在你的目录中，它可能是下面原因的其中一种：

- 你的服务配置错误，无法和身份服务通信。检查 `auth_port` 和 `auth_host` 的值，然后如上面所示的那样通过 `cURL` 确保可以访问哪些服务。
- 你的签名目录无法写入。使用命令 `chmod` 来变更权限，以让服务 (POSIX) 用户可以写入。通过 `su` 和 `touch` 来对变更做验证。
- SELinux 规则禁止访问目录。

当你使用基于 Fedora/RHEL 软件包时，若你选择的配置属性和标准的策略不匹配时，经常会遇到 SELinux 的问题。运行命令 `setenforce permissive`。如果你有不同的做法，你须重新给目录打标签。如果你使用了 `/var/cache/` 目录下的子目录，运行下面的命令：

```
# restorecon /var/cache/
```

如果你没有使用 `/var/cache` 子目录，你须为你的服务更改 `signing_dir` 配置项然后重启。

返回 `setenforce enforcing` 的设置，来确认你的变更解决了问题。

如果你的证书是按需获取的，PKI 验证会正常工作。多数情况下，你试图正在执行的操作并不验证来自身份服务的令牌，所以你的用户的操作须切换为不同角色。

调试签名密钥文件错误

如果在打开签名密钥文件时发生了错误，它有可能是运行 `keystone-manage pki_setup` 命令来生成证书和密钥的用户不是正确的用户。当你运行命令 `keystone-manage pki_setup` 时，身份在 `/etc/keystone/ssl*` 中生成一组证书和密钥，它们的属主是 `root:root`。

这可以说明一个问题，那就是当你在 `keystone` 用户账户下 (`nologin`) 运行的身份服务守护进程时，你尝试运行 PKI。除非你运行命令 `chown` 满足了文件属主是 `keystone:keystone`，否则运行带有参数 `--keystone-user` 和 `--keystone-group` 的 `keystone-manage pki_setup` 命令，你会得到如下错误：

```
2012-07-31 11:10:53 ERROR [keystone.common.cms] Error opening signing key file
/etc/keystone/ssl/private/signing_key.pem
140380567730016:error:0200100D:system library:fopen:Permission
denied:bss_file.c:398:fopen('/etc/keystone/ssl/private/signing_key.pem','r')
140380567730016:error:20074002:BI0 routines:FILE_CTRL:system lib:bss_file.c:400:
unable to load signing key file
```

从令牌数据库表刷新过期的令牌

只要你生成令牌，身份服务的令牌的数据库表就在增长。要清理令牌表，管理员用户必须运行命令`keystone-manage token_flush`来刷新令牌。当你刷新令牌时，过期的令牌会被删除掉，而且是永久的。

使用`cron`来规划此命令，让其基于你的负载经常运行。对于大型负载，建议每分钟就运行一次。

第 3 章 Dashboard

目录

定制GUI界面	46
为仪表盘设置会话存储	48

OpenStack dashboard是一个允许用户管理OpenStack资源和服务的基于web的应用，dashboard允许用户和OpenStack计算云控制器交互，使用OpenStack的API，更多关于安装和配置dashboard的内容，请参考 OpenStack 安装指南，当然需要选择下操作系统，因为这本书讲了三种类型的Linux发行版。

Dashboard 资源:

- 要自定义dashboard,请参考 “定制GUI界面” 一节 [46]。
- 为dashboard设置会话存储，请参考 “为仪表盘设置会话存储” 一节 [48]。
- 要部署dashboard，请参考 [Horizon 文档](#)。
- 使用dashboard启动一个实例，请参考[OpenStack 最终用户指南](#)。

定制GUI界面

一旦你已经安装了web图形界面程序，你就可以定制它的外观和感觉，从而适应你自身的需求。



注意

OpenStack web 图形界面，在Ubuntu中默认安装的软件包是openstack-dashboard-ubuntu-theme。

如果你不打算使用此默认的主题，你可以使用下面命令来删除它以及相关的依赖：

```
# apt-get remove --auto-remove openstack-dashboard-ubuntu-theme
```



注意

本书聚焦于文件local_settings.py，它存储在/openstack-dashboard/openstack_dashboard/local/目录。

此书摘选自[如何为 OpenStack "Horizon" 定制自己的品牌](#)。

下列可轻松自定义:

- 站点颜色
- Logo

- HTML 标题
- 站点品牌链接
- 帮助URL

Logo和站点颜色

1. 创建两个png格式的logo文件，使用如下大小的透明背景：
 - 登录界面：365 x 50
 - 登录边框: 216 x 35
2. 上传新的图片到此位置：`/usr/share/openstack-dashboard/openstack_dashboard/static/dashboard/img/`
3. 在此目录下创建一CSS样式表：`/usr/share/openstack-dashboard/openstack_dashboard/static/dashboard/css/`
4. 根据实际情况更改颜色和图片文件名，相对的目录路径须一致。下面的文件是一个例子，说明了如何定制你的CSS文件：

```
/*
 * New theme colors for dashboard that override the defaults:
 * dark blue: #355796 / rgb(53, 87, 150)
 * light blue: #BAD3E1 / rgb(186, 211, 225)
 *
 * By Preston Lee <plee@tgen.org>
 */
h1.brand {
background: #355796 repeat-x top left;
border-bottom: 2px solid #BAD3E1;
}
h1.brand a {
background: url(../img/my_cloud_logo_small.png) top left no-repeat;
}
#splash .login {
background: #355796 url(../img/my_cloud_logo_medium.png) no-repeat center 35px;
}
#splash .login .modal-header {
border-top: 1px solid #BAD3E1;
}
.btn-primary {
background-image: none !important;
background-color: #355796 !important;
border: none !important;
box-shadow: none;
}
.btn-primary:hover,
.btn-primary:active {
border: none;
box-shadow: none;
background-color: #BAD3E1 !important;
text-decoration: none;
}
```

5. 使用你喜爱的编辑器打开下面HTML模版：`filename>/usr/share/openstack-dashboard/openstack_dashboard/templates/_stylesheets.html`
6. 添加一行，将你新创建的样式表包含进来。例如custom.css文件：

```
...
<link href='{ { STATIC_URL } }bootstrap/css/bootstrap.min.css' media='screen' rel='stylesheet' />
<link href='{ { STATIC_URL } }dashboard/css/{% choose_css %}' media='screen' rel='stylesheet' />
<link href='{ { STATIC_URL } }dashboard/css/custom.css' media='screen' rel='stylesheet' />
...
```

7. 重启 Apache:

Ubuntu中:

```
# service apache2 restart
```

Fedora, RHEL, CentOS中:

```
# service httpd restart
```

openSUSE中:

```
# service apache2 restart
```

8. 要查看你的修改的成果，重新载入web页面即可。如果不满意，再回去修改你的CSS文件。

HTML 标题

1. 设置HTML标题，它会出现浏览器的顶部，在配置文件local_settings.py中添加下面几行：

```
SITE_BRANDING = "Example, Inc. Cloud"
```

2. 重启Apache服务以使修改生效。

HTML 标题

1. logo也扮演着一个超链接。默认会重定向到horizon:user_home。要修改此，给local_settings.py添加如下属性

```
SITE_BRANDING_LINK = "http://example.com"
```

2. 重启Apache服务以使修改生效。

帮助URL

1. 默认的帮助URL指向的是<http://docs.openstack.org>。要变更此地址，在local_settings.py编辑下面属性为你要选择的URL

```
'help_url': "http://openstack.mycompany.org",
```

2. 重启Apache服务以使修改生效。

为仪表盘设置会话存储

仪表盘使用Django 会话框架来掌控用户的会话数据。然而，你可以使用任何可用的会话后端。通过在文件local_settings (对于 Fedora/RHEL/CentOS: /etc/openstack-dashboard/local_settings, 对于 Ubuntu and Debian: /etc/openstack-dashboard/local_settings.py, 对于 openSUSE: /srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py)中设置SESSION_ENGINE来定制会话后端。

接下来的几节描述了使用各种会话存储来部署仪表盘的利与弊。

本地内存缓存

设置本地内存存储是最快、最简单的会话后端，它没有任何额外的依赖。但是它有一些显著的缺点：

- 没有跨进程或线程的共享存储。
- 在一个进程终止后没有持久性。

当单独安装Horizon时，默认启用的本地内存后端，因为其没有任何的依赖。强烈建议不要在生产环境中使用，甚至是一些高级的部署工作中。通过下面方法启用：

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'
    }
}
```

键-值存储

你可以使用诸如Memcached或Redis的应用程序来作为外部的缓存。这些应用提供了持久化的和共享的存储，且适用于小型的部署或开发环境。

Memcached

Memcached是一个高性能，分布式的内存对象缓存系统，提供内存内的键-值存储，针对一小块一小块的任意数据。

需求：

- Memcached 服务运行中且可被访问。
- Python 模块 python-memcached 已经安装。

通过下面方法启用：

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': 'my_memcached_host:11211',
    }
}
```

Redis

Redis是一款开源项目，基于BSD许可证，用于高级的键-值存储。It is often referred to as a data structure server.

需求：

- Redis服务运行中且可被访问。
- Python 模块 redis 以及 django-redis 已经安装。

通过下面方法启用：

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    "default": {
        "BACKEND": "redis_cache.cache.RedisCache",
        "LOCATION": "127.0.0.1:6379:1",
        "OPTIONS": {
            "CLIENT_CLASS": "redis_cache.client.DefaultClient",
        }
    }
}
```

初始化和配置数据库

后端的数据库会话保持一致性、扩展性，且要做到高并发和高可用。

然而，后端为关系型数据库的会话是较慢的会话存储，且在大量使用的情况下产生高的负载。正确的配置你的数据库部署也是一项非常耗时的任务，已经超出了本文档的范围。

1. 启动mysql命令行客户端：

```
$ mysql -u root -p
```

2. 按照提示输入MySQL root用户的密码。
3. 要配置为使用MySQL数据库，首先创建数据库：

```
mysql> CREATE DATABASE dash;
```

4. 为新创建的数据库创建一个MySQL用户，其拥有此数据库的全部控制权。以新的用户的密码代替DASH_DBPASS：

```
mysql> GRANT ALL PRIVILEGES ON dash.* TO 'dash'@'%' IDENTIFIED BY 'DASH_DBPASS';
mysql> GRANT ALL PRIVILEGES ON dash.* TO 'dash'@'localhost' IDENTIFIED BY 'DASH_DBPASS';
```

5. 在mysql>提示符下输入quit，退出MySQL。
6. 在文件local_settings(对于 Fedora/RHEL/CentOS: /etc/openstack-dashboard/local_settings, 对于 Ubuntu/Debian: /etc/openstack-dashboard/local_settings.py, 对于 openSUSE: /srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py)中，修改如下这些属性：

```
SESSION_ENGINE = 'django.contrib.sessions.backends.db'
DATABASES = {
    'default': {
        # Database configuration here
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dash',
        'USER': 'dash',
        'PASSWORD': 'DASH_DBPASS',
        'HOST': 'localhost',
        'default-character-set': 'utf8'
    }
}
```

7. 按照上面展示的修改完local_settings之后，你就可以运行命令manage.py syncdb来填写此新创建的数据库了。

```
# /usr/share/openstack-dashboard/manage.py syncdb
```

注意，openSUSE的路径是/srv/www/openstack-dashboard/manage.py。

结果返回下面的输出:

```
Installing custom SQL ...
Installing indexes ...
DEBUG:django.db.backends:(0,008) CREATE INDEX `django_session_c25c2c28` ON `django_session`
(`expire_date`); args=()
No fixtures found.
```

8. 为了避免当你在Ubuntu下重启Apache出现警告，在仪表盘的目录下创建一个blackhole目录，如下所示：

```
# mkdir -p /var/lib/dash/.blackhole
```

9. 重启和刷新Apache

Ubuntu中：

```
# /etc/init.d/apache2 restart
```

在 Fedora/RHEL/CentOS 下:

```
# service httpd restart
```

```
# service apache2 restart
```

openSUSE中:

```
# systemctl restart apache2.service
```

10. 在Ubuntu中，重启nova-api服务以确保此API服务可没有错误的连接到仪表盘：

```
# service nova-api restart
```

缓存数据库

为了减轻数据库查询的性能压力，你可以使用Djangocached_db会话后端，其同时利用数据库和缓存基础设施进行写式高速缓存，高效的检索。

通过按照以前所讨论的同时配置你的数据库和缓存来启用此混合的设置。然后，设置如下值：

```
SESSION_ENGINE = "django.contrib.sessions.backends.cached_db"
```

Cookies

如果你使用Django 1.4 或其后续版本， signed_cookies 后端可避免服务负载和扩展的问题。

此后端存储会话到一个cookie中，cookie会通过用户的浏览器保存在本地。后端使用加密签名技术来确保会话数据在传输过程中不会被篡改。这不同于加密；会话数据仍然可以被攻击者读取。

此引擎的优点在于它不需要额外的依赖或基础设施在上面，且它无限扩展，只要会话数据的存储量融入正常的cookie。

最大的缺点是，它把会话数据到存储在用户的机器上，并通过网络来传输。它也限制会话数据的可存储的量。

请参阅Django [基于cookies的会话](#) 文档。

第 4 章 计算

目录

系统架构	53
镜像和实例	58
使用nova-network的网络	64
系统管理	79
计算故障排查	113

OpenStack计算服务允许用户控制基础设施即服务(IaaS)云计算平台。它给予用户不仅可以控制实例和网络，还可以通过用户和项目管理云的访问。

计算并不包含虚拟化软件，相反，它定义了驱动，这些驱动在用户的主机操作系统运行和其底层的虚拟化机制相交互，然后通过基于web的API抛出相应的功能。

系统架构

OpenStack计算包含多个主要的组件。

- 云控制器表现为全局的状态以及和其他组件的交互，API 服务 是为云控制器扮演前端web service的角色， 计算控制器提供计算服务资源，也通常包含了计算服务。
- 对象存放是一个可选的组件，它提供存储服务；用户可以使用OpenStack对象存储将之替换，通常也是这么做的。
- 认证管理提供了认证和授权的服务，当仅使用计算系统时。用户可以使用单独的认证服务 OpenStack身份认证来将之替换。
- volume controller为计算服务提供了快速、持久化的块级别的存储。
- network controller提供了虚拟网络，用于计算服务和其它组件的交互，以及公有网络。用户可以使用OpenStack网络neutron来将之替代。
- scheduler是用于选择那个计算控制器最适合运行实例。

计算服务使用的是基于消息的， shared nothing架构。所有的组件均可在不同的服务器中，有计算、卷、网络控制器、以及对象存放或镜像服务。整个系统的状态存放在数据库中。云控制器使用HTTP和内部的对象存储通信，但是它和scheduler,网络控制器以及卷控制器的通信使用的是AMQP(高级消息队列协议)。为了避免一个组件因为等待响应而堵塞，计算服务使用异步调用，当一个应答接收到就触发一个回调。

虚拟机管理器

计算节点通过 API 服务器控制 hypervisor。选择最佳的 hypervisor 来使用可能有所不同，且您必需将预算、资源约束、支持的特性和需要的技术说明放入帐户。然而，

大部分 OpenStack 的部署都在使用 KVM 和基于 Xen 的 hypervisor 上完成。要了解详细的特性列表和不同 hypervisor 的访问支持，请阅读 <http://wiki.openstack.org/HypervisorSupportMatrix>。

用户可以在不同的可用域中使用不同的hypervisor来编排自己的云。OpenStack计算支持下列hypervisor：

- [裸金属](#)
- [Docker](#)
- [Hyper-V](#)
- [基于内核的虚拟机\(KVM\)](#)
- [Linux容器\(LXC\)](#)
- [快速模拟器 \(QEMU\)](#)
- [用户模式Linux \(UML\)](#)
- [VMware vSphere](#)
- [Xen](#)

更多关于hypervisor的信息，请参阅OpenStack 配置参考的[Hypervisors](#)一节。

租户、用户和角色

计算服务系统被设计为用于不同的消费者，在一个共享系统的各种租户和基于角色的访问授权。角色控制一个用户是否被允许执行某些操作。

在计算服务内部，租户是重要的组织结构形式来隔离资源的容器。它们由独立的VLAN、卷、实例、镜像、密钥以及用户形成，一个用户可以指定特定的租户，只要在它的访问密钥后加上:project_id。如果在API请求中没有指定租户，计算服务会使用和用户一样的ID来使用租户。

对于租户来说，用户可以使用配额来限制如下内容：

- 多少个卷可以被启动。
- 多少个处理器核和多大内容被分配。
- Floating IP 地址可赋予给任何在启动时的实例。这允许实例可以有同样的公有访问IP地址。
- Fixed IP 地址可赋予给任何在启动时的实例。这允许实例可以有同样的公有或私有访问IP地址。

角色控制扮演的是用户被允许操作什么。默认情况下，绝大多数的动作是不需要指定角色的，但用户可通过编辑文件 policy.json来配置用户角色。举例来说，一个规则被定义为用户若要能够分配公有IP地址则必须拥有admin角色。

一个租户限制用户访问特定的镜像。每个用户会被赋予一个用户名及密码，Keypairs为每个用户赋予访问实例的权限，但是配额也可设置，因此每个租户可以控制资源的消费，且是跨可用的硬件资源。



注意

早期的OpenStack版本使用术语 项目替代 租户。因为这个遗留的术语，一些命令行工具使用参数 `--project_id`，实际上这里提供的应是租户ID。

块存储

OpenStack提供两种类型的块存储：临时存储和持久卷。

临时存储

一个临时存储包括一个root临时卷和一个额外的临时卷。

root虚拟磁盘给了实例使用，而且仅用于此实例的生命周期内。换句话说，它用做实例的根文件系统，持续到客户操作系统重新启动，在实例删除后它也会被删除。root临时卷的容量由实例的flavor来定义。

In addition to the ephemeral root volume, all default types of flavors, except `m1.tiny`, which is the smallest one, provide an additional ephemeral block device sized between 20 and 160 GB (a configurable value to suit an environment). It is represented as a raw block device with no partition table or file system. A cloud-aware operating system can discover, format, and mount such a storage device. OpenStack Compute defines the default file system for different operating systems as Ext4 for Linux distributions, VFAT for non-Linux and non-Windows operating systems, and NTFS for Windows. However, it is possible to specify any other filesystem type by using `virt_mkfs` or `default_ephemeral_format` configuration options.



注意

举例来说，cloud-init软件包含有Ubuntu的云镜像，默认会将之格式化为Ext4文件系统，挂载到目录/mnt。这是cloud-init程序的一个特性，并非OpenStack的机制，OpenStack仅部署裸存储。

持久卷

一个持久卷表现为持久的虚拟化的块设备，独立于任何的特定实例，由OpenStack块存储所提供。

只有单个配置好的实例可以访问一个持续卷。多台实例不能访问一个持续卷。这个配置类型需要一个传统网络文件系统来允许多台实例访问持续卷。还需要一个传统网络文件系统，如NFS、CIFS，或一个集群文件系统，如GlusterFS。这些系统可以在OpenStack集群中构建，或在外部提供，但OpenStack软件还不提供这些特性。

您可以配置一个持续卷为可启动的，并用它来提供一个持续的虚拟实例，就像传统的非基于云的虚拟化系统那样。这对实例保存基于所选择的类型保存短暂的存储仍是可能的。在这个情况下，根文件系统可以在持续卷上，即使实例关闭，卷的状态仍是保持的。要了解更多关于该配置类型的信息，请阅读 [OpenStack Configuration Reference](#)。



注意

一个持久卷并不提供从多个实例并行的访问，若要支持此种情况，需要诸如网络文件系统如NFS，CIFS，或者是集群文件系统如GlusterFS,这些文件系统可以构建到OpenStack集群内部，也可以从外部提供，但是OpenStack软件本身并不提供这些特性。

EC2 兼容应用程序接口

除了原生的 compute API，OpenStack 还提供兼容 EC2 的 API。这个 API 允许 EC2 以传统工作流构建 EC2 来与 OpenStack 一同工作。要了解更多关于该兼容 API 的信息和配置选项，请阅读 [OpenStack Configuration Reference](#)。

一些第三方的工具和特定语言SDK可以和OpenStack云进行交互，同时使用本地和兼容性API。下面是一些常见的第三方工具：

Euca2ools	和EC2 API互动的较流行的开源命令行工具。这在当EC2是通用API 或者从EC2云过渡到OpenStack的多云环境的情况下是非常便利的工具。更多信息，请参阅 euca2ools 网站 。
Hybridfox	一个火狐浏览器的扩展为很多流行的公有或私有云提供了图形化的接口，包括OpenStack。更多信息，请参阅 hybridfox 站点 。
boto	一个和亚马逊Web Service交互的python库。其可用于访问OpenStack，通过EC2兼容的API。更多信息，请参阅 boto 项目在 GitHub上的页面 。
fog	一个Ruby语言的云服务程序库。它提供了和大量云和虚拟化平台交互的方法，包括OpenStack。更多信息，请参阅 fog 站点 。
php-opencloud	一个PHP的SDK，被设计为工作在基于OpenStack云环境下，包括Rackspace公有云。更多信息，请参阅 php-opencloud 站点 。

构建块

在 OpenStack 中，基本的操作系统通常是从一个保存在 OpenStack 镜像服务中的镜像复制的。这是最常见的情况，使得临时的从一个已知模板状态启动的实例并在虚拟机删除是丢失所有的累积状态。将一个操作系统放置在 OpenStack 的块设备存储卷系统上也是可行的。这使一个更传统的持久系统能够积累状态，当删除或重建虚拟机时它能够在 OpenStack 块设备存储卷中获得保护。要获取您系统中的可用镜像，请执行：

```
$ nova image-list
```

ID	Name	Status	Server
ae1d242-730f-431f-88c1-87630c0f07ba	Ubuntu 14.04 cloudimg amd64	ACTIVE	
0b27baa1-0ca6-49a7-b3f4-48388e440245	Ubuntu 14.10 cloudimg amd64	ACTIVE	
df8d56fc-9cea-4dfd-a8d3-28764de3cb08	jenkins	ACTIVE	

显示的镜像属性是：

ID	为镜像自动生成的UUID
名称	任意形式，为镜像设置人类更好读的名字
状态	镜像的状态。镜像标记为ACTIVE，即表示可用。

服务 对于基于运行实例的快照创建的镜像而言，此实例的UUID表示快照的来源，对于上传的镜像来说，此处为空白。

虚拟硬件模板叫做 flavors。默认安装提供五种flavor。默认情况管理员用户可以配置它们，但是此种情况可以由重新定义访问控制来做变更，具体是在compute-api服务器中的/etc/nova/policy.json更改compute_extension:flavormanage。

在用户的系统中可用的类型列表：

```
$ nova flavor-list
```

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
1	m1.tiny	512	1	0	1	1.0	True	
2	m1.small	2048	20	0	1	1.0	True	
3	m1.medium	4096	40	0	2	1.0	True	
4	m1.large	8192	80	0	4	1.0	True	
5	m1.xlarge	16384	160	0	8	1.0	True	

计算服务架构

这些基本的分类描述了服务架构以及关于云控制器的信息。

API服务器

云框架的核心就是API服务，它发出指令和控制hypervisor,存储和网络，为用户提供可编程的方法。

API 端点是基于 HTTP web 服务的，它处理认证、授权和基本命令及使用 Amazon、Rackspace 和相关模块之下的各种 API 接口来控制功能。这允许 API 与多种现有的配置工具相互兼容，这些工具创建来与其他供应商提供的接口交互。这种广泛的兼容性防止了供应商的锁定。

消息队列

消息队列安排计算节点（进程）、网络控制节点（控制节点上的网络基础设施软件）、API 端点、调度器（决定将哪个物理硬件分配给虚拟资源）和类似组件之间的交互。与云控制节点通信和从云控制节点通信是由 HTTP 请求通过多个 API 端点处理的。

典型的消息传递事件以 API 服务器获取一个用户请求开始。API 服务器认证用户并保证他们能够提出命令。与请求相关的可用对象进行评估，如果可用，请求会路由到队列引擎的相关工人。工人继续监听基于它们角色的队列，有时候也监听它们的类型主机名称。当一个合适的工作请求到达队列时，工人接受任务的分配并开始执行。完成后，会调度一个相应到队列中，由 API 服务器接收，并传达给最初的用户。在这过程中，数据库实体根据需要被查询、添加或删除。

计算 worker

主机中的计算worker管理着计算实例。API分发器会给计算worker下达命令，从而让其完成任务：

- 运行实例

- 终结实例
- 重启实例
- 挂接卷
- 分离卷
- 请获取控制台输出

网络控制器

网络控制器管理着主机中的网络资源。API 服务分发器通过消息队列来下达命令，然后网络控制器稍后来处理。指定的操作包括：

- 分配确定的IP地址
- 为项目配置VLAN
- 为计算节点配置网络

镜像和实例

磁盘镜像提供给虚拟机文件系统的一种模板，镜像服务控制着存储和管理镜像。

实例是运行在物理计算节点上的单独的虚拟机，用户可以从同一个镜像启动多个实例。由于每启动一个实例都会复制基础镜像，所以实例的任何改动并不影响原始的基础镜像。用户可以使用快照将当前运行的实例制作成镜像，其基于当前的磁盘状态生成特殊的实例。计算服务管理着实例。

当你要启动一个实例时，你必须选择一个云类型，其代表了一组虚拟资源。云类型定义了实例拥有几颗虚拟CPU，多大的可用内存，以及临时磁盘的大小。用户必须在其的云中选择一个可用的云类型。OpenStack提供了一组预先定义的云类型，当然你也可以编辑它或者添加一些类型。



注意

- 关于创建和故障恢复镜像的更多内容，请参阅[OpenStack 虚拟机镜像指南](#)。
- 关于镜像配置属性的更多内容，请参阅OpenStack 配置参考的章节 [镜像服务](#)。
- 关于类型的更多内容，请参阅OpenStack 操作指南中的 [Flavors](#) 或者是“[云主机类型](#)”一节 [80]。

你也可以给正在运行中的实例添加或删除额外的资源，例如持久的卷存储，或者是公有IP地址。在此章所使用的例子是典型的OpenStack云的系统。它使用cinder-volume服务，此服务提供了持久化的块存储，取代了通过所选择的实例类型所提供的临时存储。

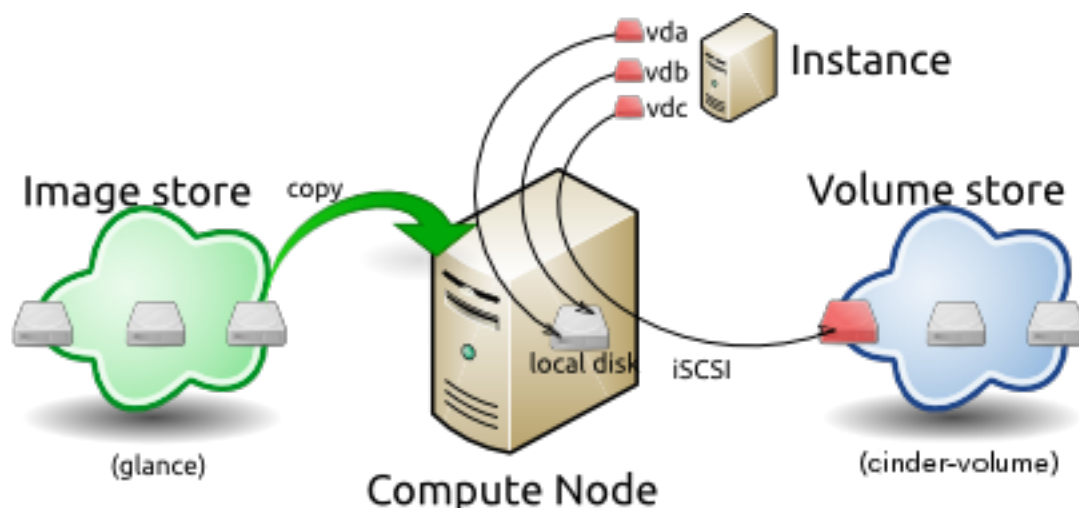
此示意图展示了实例在启动时的系统状态。镜像服务(glance)有一些预先定义的镜像。在云内部，一个计算节点包含了可用的vCPU,内存，和本地磁盘资源。另外，cinder-volume服务也提供了一些预定义的卷。

图 4.1. 基础镜像的状态和运行的实例无关



要启动一个实例，选择一镜像，云类型，以及其它可选的属性，所选择的云类型提供了根卷，在图中标记为vda，以及一个额外的临时存储，标记为vdb。在此例中，cinder-volume存储被映射为此实例的第三个虚拟磁盘，标记为vdc。

图 4.2. 从镜像创建的实例和运行时状态



基本镜像是从镜像存储复制到本地的磁盘。本地磁盘是实例访问的第一块磁盘，实例中被标记为vda。如果使用很小的镜像，实例的启动会非常的快，因为只有很少的数据需要从网络上复制。

也会创建一个新的空的临时磁盘，示意图中的vdb。此磁盘会在当你删除实例时销毁。

计算节点使用 iSCSI 连接到附加的cinder-volume。cinder-volume映射到第三块磁盘，图中标记的vdc。在计算节点部署完vCPU和内存资源后，实例从根卷vda启动。实例开始运行，并更改磁盘上的数据(示意图中的红色高亮)。如果卷的存储是在不同的网络，my_block_storage_ip 属性指定的存储节点，会告诉镜像传输到计算节点。



注意

此例的场景和你实际的环境在细节上可能有些不同。例如，你或许使用了不同类型的后端存储，又或者是网络协议。另外常见的不同是卷vda 和vdb的临时存储的后端使用网络存储取代本地磁盘。

当实例被删除后，持久卷的状态仍然保持。临时存储会被删除；内存和vCPU资源会被释放。此过程对镜像meiyou任何的影响。

图 4.3. 在实例销毁后镜像和卷最后的状态



镜像管理

OpenStack镜像服务发现，寄存和检索虚拟机镜像。服务也包括一个RESTful的API用于用于查询虚拟机镜像的元数据以及通过HTTP获取实际的镜像。关于API的更多信息，请参阅[OpenStack API 完全参考](#) 和 [Python API](#)。

OpenStack镜像服务也支持来自命令行的操作。更多关于使用OpenStack镜像命令行工具的信息，请参阅OpenStack 最终用户指南的章节 [管理镜像](#)。

虚拟机镜像通过镜像服务对外提供服务，但其本身的存储是有多种方式实现的。为了使用镜像服务，必须确保镜像服务安装正常，端点工作正常，且在身份服务上建立了用户。另外，计算服务和镜像服务客户端的环境变量必须满足。

镜像服务支持下面后端存储：

文件系统	OpenStack镜像服务默认会将虚拟机镜像存放在后端的文件系统中，此简单的实现就是将镜像文件写到本地文件系统中。
对象存储	针对对象存储的OpenStack高可用服务。
块存储	针对块存储的OpenStack高可用服务。
VMware	ESX/ESXi 或 vCenter Server 的目标系统。
S3	The Amazon S3 服务。
HTTP	OpenStack镜像服务可以读取来自互联网使用HTTP的虚拟机镜像，此存储是只读的。
RADOS 块设备 (RBD)	使用Ceph的RBD接口存放镜像到Ceph存储集群内部。
Sheepdog	针对QEMU/KVM的分布式存储系统。
GridFS	使用MongoDB存放镜像。

镜像性能和属性保护

镜像的属性是一键值对，此键值对让云管理员或镜像所有者挂接到OpenStack镜像服务的镜像，如下：

- 云管理员会定义核心属性，例如镜像的名称。
- 云管理员和镜像所有者可以定义额外的 属性，诸如许可证和计费信息。

云管理员可以配置任何的protected属性，protected在此属性下限制了规则或用户角色可以执行CRUD的操作。受保护的属性通常是附加属性，只有云管理员可以访问。

对于不受保护的镜像属性来说，云管理员可以管理核心属性，镜像的属主可以管理额外的属性。

要配置属性保护

要配置属性保护，云管理员须完成这些步骤：

1. 在文件policy.json中定义角色或规则：

```
{
  "context_is_admin": "role:admin",
  "default": "",

  "add_image": "",
  "delete_image": "",
  "get_image": "",
  "get_images": "",
  "modify_image": "",
  "publicize_image": "role:admin",
  "copy_from": "",

  "download_image": "",
  "upload_image": "",

  "delete_image_location": "",
  "get_image_location": "",
  "set_image_location": "",

  "add_member": "",
  "delete_member": "",
  "get_member": "",
  "get_members": "",
  "modify_member": "",

  "manage_image_cache": "role:admin",

  "get_task": "",
  "get_tasks": "",
  "add_task": "",
  "modify_task": "",

  "deactivate": "",
  "reactivate": "",

  "get_metadef_namespace": "",
  "get_metadef_namespaces": "",
```

```
"modify_metadef_namespace": "",
"add_metadef_namespace": "",

"get_metadef_object": "",
"get_metadef_objects": "",
"modify_metadef_object": "",
"add_metadef_object": "",

"list_metadef_resource_types": "",
"get_metadef_resource_type": "",
"add_metadef_resource_type_association": "",

"get_metadef_property": "",
"get_metadef_properties": "",
"modify_metadef_property": "",
"add_metadef_property": "",

"get_metadef_tag": "",
"get_metadef_tags": "",
"modify_metadef_tag": "",
"add_metadef_tag": "",
"add_metadef_tags": ""
}
```

对于每个参数，使用“rule:restricted”来限制所有用户的访问，或者使用“role:admin”来限制管理员的访问。例如：

```
"download_image": "rule:restricted"
"upload_image": "role:admin"
```

2. 在属性保护的配置文件中定义哪些用户和规则可以管理那些属性。例如：

```
[x_none_read]
create = context_is_admin
read = !
update = !
delete = !

[x_none_update]
create = context_is_admin
read = context_is_admin
update = !
delete = context_is_admin

[x_none_delete]
create = context_is_admin
read = context_is_admin
update = context_is_admin
delete = !
```

- @ 的值允许允许一个属性相应的操作。
- ! 的值是不允许一个属性做相应的操作。

3. 在文件glance-api.conf中，定义了属性保护配置文件的路径：

```
property_protection_file = {file_name}
```

此文件包括了属性保护的规则，以及角色和策略相关联。

默认情况下，属性保护并非强制。

如果你指定的文件名称且无法找到文件，glance-api服务就无法启动。

要查看样例配置文件，请参阅 [glance-api.conf](#)。

4. 可选项，在glance-api.conf文件中，指定哪些用户或策略是用于道属性保护配置文件：

```
property_protection_rule_format = roles
```

默认是 角色。

要查看样例配置文件，请参阅 [glance-api.conf](#)。

镜像下载：它是如何工作的

在启动一个虚拟机期间，虚拟机镜像的使用必须从镜像服务传输到计算节点中。这是如何工作可以根据所选择的计算节点和图像服务的设置更改。

Typically, the Compute service will use the image identifier passed to it by the scheduler service and request the image from the Image API. Though images are not stored in glance—rather in a back end, which could be Object Storage, a filesystem or any other supported method—the connection is made from the compute node to the Image service and the image is transferred over this connection. The Image service streams the image from the back end to the compute node.

将对象存储节点设置为不同的网络是可以的，而且不会影响到计算节点和对象存储节点的镜像传输流。在存储节点中配置my_block_storage_ip属性，以允许块存储流量可以到达计算节点。

后端支持多种直接的方法，向镜像服务请求，然后返回一个URL，然后可以从此URL来从后端存储直接下载镜像。但是目前能够支持直接现在途径的存储只有基于文件系统的存储。它可以在计算节点的nova.conf文件中的image_file_url一节的filesystems属性来完成配置。

计算节点也实现了镜像的缓存，意思是如果一个镜像在之前用到过，那么就没必要每次都去下载。为计算节点配置缓存的属性的信息可在 [配置参考](#)中找到。

实例构建块

在OpenStack中，基础操作系统一般均是复制来自OpenStack镜像服务中所存储的镜像的。这样的结果就是一个临时的实例从一个已经状态的模版启动，但随着关机就丢失所有的积累的数据。

你可以将操作系统存放在计算节点的持久卷或者是块存储卷系统中。这样的话即使是重启也不会丢失任何的数据，和原有的保持一致。要在系统中获得可用的镜像列表，运行：

```
$ nova image-list
```

ID	Name	Status	Server
ae1d242-730f-431f-88c1-87630c0f07ba	Ubuntu 14.04 cloudimg amd64	ACTIVE	
0b27baa1-0ca6-49a7-b3f4-48388e440245	Ubuntu 14.10 cloudimg amd64	ACTIVE	
df8d56fc-9cea-4dfd-a8d3-28764de3cb08	jenkins	ACTIVE	

显示的镜像属性是：

ID	自动为镜像生成UUID。
名称	任意填写，便于人们阅读的镜像名称。
状态	镜像的状态。镜像标记为ACTIVE，即表示可用。
服务器	对于基于运行实例的快照创建的镜像而言，此实例的UUID表示快照的来源，对于上传的镜像来说，此处为空白。

虚拟机的硬件模版通常称之为 flavors。默认的安装提供了五种预先定义的flavor。

要列出你的系统中可用的类型，运行：

```
$ nova flavor-list
```

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
1	m1.tiny	512	1	0	1	1.0	True	
2	m1.small	2048	20	0	1	1.0	True	
3	m1.medium	4096	40	0	2	1.0	True	
4	m1.large	8192	80	0	4	1.0	True	
5	m1.xlarge	16384	160	0	8	1.0	True	

默认情况下，管理员用户是可以配置类型的。你可以通过重新定义访问控制来改变这一行为，在compute-api服务中修改配置文件/etc/nova/policy.json的compute_extension:flavormanage项。

实例管理工具

OpenStack提供了命令行工具，Web图形界面，以及基于API的实例管理工具。使用本地API或者是提供EC2兼容的API第三方的管理工具也有许多。

OpenStackpython-novaclient软件包提供了一个基本的命令行工具，其使用命令nova。此软件包在大多数的发行版中都有，当然你也可以使用python软件包安装工具pip来安装最新的版本：

```
# pip install python-novaclient
```

更多关于python-novaclient 以及其他命令行工具的内容，请参阅[OpenStack 最终用户指南](#)。

控制实例在哪里运行

[OpenStack 配置参考](#)提供了控制实例运行在哪里的细节信息，包括为了服务负载而将一组实例运行在不同的计算节点，又或者是为实例之间通信的高性能而运行在同一台计算节点。

管理员用户可以指定那个计算节点可运行实例。要实际操作的话，指定参数--availability-zone AVAILABILITY_ZONE:COMPUTE_HOST。

使用nova-network的网络

通过理解有效的网络配置选项你能为你的OpenStack Compute实例设计最好的配置。

你可以选择安装和配置nova-network 也可以使用OpenStack网络服务(neutron)。此章涵盖了nova-network 的简要概述。更多关于OpenStack网络的信息，请参阅 [第 7 章 网络 \[180\]](#)。

网络概念

计算服务分配一个私有的 IP 地址给每一台虚拟机实例。计算服务在固定 IP 和浮动 IP 之间做了区别。固定 IP 是在创建时分配给实例的 IP 地址，并保持不变，直到实例被明确终止。浮动 IP 是可以动态分配给实例的 IP 地址。浮动 IP 可以在任何时候附加和分配给另一台实例。用户可以为他们的项目保留一个浮动 IP。



注意

目前，计算节点上的 nova-network 仅支持 Linux 桥接网络，允许虚拟接口通过物理接口连接到外部网络。

网络控制节点的 nova-network 提供虚拟网络，以允许计算服务器之间以及与公有网络的交互。计算节点的 nova-network 支持以下网络模式，它们是作为网络管理者类型实现的：

扁平化网络管理器 在这个模式下，网络管理者指定了一个子网。虚拟机实例的 IP 地址是从子网中分配的，并在启动时注入镜像中。每个实例从可用地址池中接收一个固定 IP 地址。系统管理员必需在运行 nova-network 服务的系统上创建 Linux 网络桥接（通常命名为 br100，尽管它是可配置的）。所有的系统的实例被附加到由网络管理员手动配置的同一个桥接上。



注意

配置注入目前仅支持linux风格的系统，且将网络配置保持在/etc/network/interfaces。

扁平化DHCP网络管理器 在这个模式下，除了手动配置网络桥接之外，OpenStack 启动了一个 DHCP 服务器 (dnsmasq) 来为虚拟机实例从指定子网中分配 IP 地址。虚拟机实例的 IP 地址由网络管理员从指定的子网中分配。

类似于扁平化模式，所有的实例都被附加到计算节点的单个桥接上。此外，DHCP 服务器与 nova-network 根据单/多主机模式配置实例。在这个模式下，计算节点做了更多的配置。它尝试桥接到一个 Ethernet 设备 (flat_interface，默认是 eth0)。对于每个实例，计算节点分配一个固定 IP 地址并为虚拟机将 dnsmasq 配置为 MAC ID 和 IP 地址。dnsmasq 不会参与 IP 地址的分配进程，它仅根据计算节点完成的映射来分发 IP。实例通过 dhcpdiscover 命令接收它们的固定 IP。这些 IP 没有分配到任何主机的网络接口，仅分配到虚拟机的宿主端接口。

在任意扁平化网络配置中，提供 nova-network 服务的主机负责从私有网络转发流量。它们也运行和配置 dnsmasq 作为一个 DHCP 服务器来监听这个桥接，一般在 IP 地址 10.0.0.1 (详情请看 [DHCP server: dnsmasq](#)) 上。尽管有时并没有使用 NAT，计算节点也可以为每个网络决定 NAT 实体，例如网络被配置为所有的公共 IP，或如果硬件路由被使用（这是一个高可用选项）。在这个情况下，主机需要配置 br100 并物理连接到任何其他托管的节点上。您必需设置 flat_network_bridge 选项或以桥接选项创建网络，以避免出现错误。计算节点为每个项目和项目的实例创建了 iptables 或 ebtables 实体，以防止 MAC ID 或 IP 地址欺骗和 ARP 中毒。



注意

在单节点扁平化 DHCP 模式下，您可以从 nova-network 节点上通过它们的固定 IP 地址 ping 虚拟机，但您不能从其他计算节点上 ping 到它们。这是符合预期的行为。

VLAN网络管理器 这是 OpenStack 计算节点的默认模式。在这个模式下，计算节点为每个租户创建一个 VLAN 和桥接。对于多机器的安装，VLAN 网络模式需要一个支持 VLAN 标签 (IEEE 802.1Q) 的转换器。租户得到私有 IP 的范围，它仅能从内部 VLAN 访问。为了用户可以在他们的租户下访问实例，需要创建一个指定的 VPN 实例 (名为 cloudpipe)。计算节点为用户生成一个证书和密钥，以访问 VPN 并自动启动 VPN。它为每个租户的实例提供了一个私有网络段，可以通过专有 VPN 连接从内部访问。在这个模式下，每个租户得到了他们自己的 VLAN、Linux 网络桥接和子网。

子网是由网络管理员指定的，并在需要时自动分配给租户。DHCP 服务器为每个 VLAN 启动以从分配给租户的子网传递 IP 地址给虚拟机实例。所有属于一个租户的实例被桥接到其租户的同一个 VLAN。OpenStack 计算节点在需要时创建 Linux 网络桥接和 VLAN。

这些网络管理器是可以并存的。然后，因为无法在给定的租户中选择网络的类型，所以不能够在单个的安装的计算中配置多个网络类型。

所有网络管理者都使用网络驱动配置网络。例如，Linux L3 驱动 (l3.py 和 linux_net.py)，可以利用 iptables、route 和其他网络管理设施，以及 libvirt [网络过滤设施](#)。驱动没有绑定到任何特定的网络管理者上，所有的网络管理者使用同一个驱动。驱动通常仅在第一台实例在该主机上启动时初始化。

所有网络管理者在单主机或多主机模式中操作。这个选择非常影响网络的配置。在单主机模式下，单个 nova-network 服务为虚拟机提供一个默认网关并托管一个 DHCP 服务器 (dnsmasq)。在多主机模式下，每个计算节点运行自己的 nova-network 服务。在这两种情况下，所有虚拟机和互联网之间的流量流向 nova-network。每个模式都有优点和缺点。要了解更多与此相关的信息，请阅读 [OpenStack 运维手册](#) 中的 网络拓扑 小节。

所有网络选项都需要在 OpenStack 物理节点之间已经配置好网络连通性。OpenStack 不会配置任何物理网络接口。所有网络管理者会自动创建虚拟机的虚拟接口。一些网络管理者也可以创建网络桥接，如 br100。

内部网络接口用于与虚拟机交互。这个接口不应该在 OpenStack 安装之前附加 IP 地址，它仅作为一个构造服务，其实际端点为虚拟机和 dnsmasq。此外，内部网络接口必需是 promiscuous 模式，这样它就可以接收目标为虚拟机 MAC 地址的包，而不是主机的包。

所有机器必需有一个公共的和内部的网络接口 (由这些选项控制：public_interface 控制公有接口，flat_interface 和 vlan_interface 控制扁平化或 VLAN 内部网络接口)。本指南是指公有网络作为外部网络，而私有网络作为内部和租户网络。

对于 flat 和 flat DHCP 模式，使用命令 nova network-create 来创建网络：

```
$ nova network-create vmnet
--fixed-range-v4 10.0.0.0/16 --fixed-cidr 10.0.20.0/24 --bridge br100
```

此例中使用了下列参数：

- `--fixed-range-v4` 指定网络子网。
- `--fixed-cidr`指定了要分配的固定IP地址范围，且可以跟`--fixed-range-v4`参数。
- `--bridge`指定了每台计算节点的网桥设备连接到此网络。

DHCP服务：dnsmasq

当使用 Flat DHCP Network Manager 或 VLAN Network Manager，计算服务使用 [dnsmasq](#) 作为 DHCP 服务器。对于计算节点在 IPv4/IPv6 dual-stack 模式下操作，请使用至少 [dnsmasq v2.63](#) 的版本。nova-network 服务负责启动 dnsmasq 进程。

dnsmasq 的行为可以通过创建 dnsmasq 配置文件来进行自定义。使用 `dnsmasq_config_file` 配置选项来指定配置文件：

```
dnsmasq_config_file=/etc/dnsmasq-nova.conf
```

关于创建dnsmasq配置文件的更多信息，请参阅[OpenStack 配置参考](#)和 [dnsmasq 文档](#)。

dnsmasq 也扮演一个缓存实例 DNS 服务器的角色。您可以指定 dnsmasq 所使用的 DNS 服务器，通过设置 `/etc/nova/nova.conf` 中的 `dns_server` 配置选项即可。这个示例配置了 dnsmasq 以使用 Google 的公有 DNS 服务器：

```
dns_server=8.8.8.8
```

dnsmasq 日志记录到 syslog 中（通常是 `/var/log/syslog` 或 `/var/log/messages`，取决于 Linux 发行版）。日志对于故障排除是很有用的，特别是当虚拟机启动成功但网络不可达的时候。

管理员可以指定启动点 IP 地址来由 DHCP 服务器保留（格式为 `n.n.n.n`），通过以下命令实现：

```
$nova-manage fixed reserve --address IP_ADDRESS
```

这个保留仅影响虚拟机启动的 IP 地址，不会影响 nova-network 在桥接上设置的固定 IP 地址。

配置计算服务使用IPv6地址

如果您使用的是包含 nova-network 的 OpenStack Compute，您必需将计算节点设置为双栈模式，这样它就会使用 IPv4 和 IPv6 地址来通信。在双栈模式中，实例可以通过使用一个无状态的地址自动配置机制 [RFC 4862/2462] 来获取它们的 IPv6 全局单播地址。IPv4/IPv6 双栈模式同时与 VlanManager 和 FlatDHCPManager 网络模式一同工作。

在 VlanManager 网络模式下，每个项目都使用一个不同的 64 位全局路由前缀。在 FlatDHCPManager 模式下，所有实例都使用同一个全局路由前缀。

此配置在拥有无状态IPv6地址自动配置能力的虚拟机镜像下测试过。任何虚拟机要运行IPv6地址均需要此能力。你必须为无状态地址自动配置使用EUI-64地址。每个运行 nova-*服务的节点必须安装python-netaddr 和 radvd

切换到 IPv4/IPv6 双栈模式

1. 在每个节点运行nova-*服务，且安装python-netaddr:

```
# apt-get install python-netaddr
```

2. 每个节点运行nova-network，安装radvd，且配置IPv6网络：

```
# apt-get install radvd
# echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
# echo 0 > /proc/sys/net/ipv6/conf/all/accept_ra
```

3. 在所有的节点中，编辑文件nova.conf，设置use_ipv6 = True。

4. 重启所有 nova-*服务。



注意

您可以为 IPv6 地址添加一个固定的范围到 nova network-create 命令中。在 network-create 参数之后指定 public 或 private。

```
$ nova network-create public --fixed-range-v4 FIXED_RANGE_V4 --vlan VLAN_ID --vpn VPN_START
--fixed-range-v6 FIXED_RANGE_V6
```

您可以使用 --fixed_range_v6 参数设置 IPv6 全局路由前缀。参数的默认值是 fd00::/48。

如果您使用了 FlatDHCPManager，命令会使用原始 --fixed_range_v6 的值。例如：

```
$ nova network-create public --fixed-range-v4 10.0.2.0/24 --fixed-range-v6 fd00:1::/48
```

如果您使用 VlanManager，命令增加子网 ID 以创建子网前缀。宿主虚拟机使用这个前缀来生成它们的 IPv6 全局单播地址。例如：

```
$ nova network-create public --fixed-range-v4 10.0.1.0/24 --vlan 100 --vpn 1000 --fixed-range-
v6 fd00:1::/48
```

表 4.1. Description of IPv6 configuration options

配置属性=默认值	描述
[DEFAULT]	
fixed_range_v6 = fd00::/48	(StrOpt)固定IPv6地址块
gateway_v6 = 无	(StrOpt)缺省IPv6网关
ipv6_backend = rfc2462	(StrOpt) IPv6生成器的后台
use_ipv6 = False	(BoolOpt) 使用IPv6

元数据服务

计算节点为虚拟机实例使用元数据服务来获取指定实例的数据。实例在 <http://169.254.169.254> 访问元数据服务。元数据服务支持两套 API：OpenStack 元数据 API 和 EC2-兼容的 API。两种 API 都以日期为版本。

要获取元数据API所支持的版本列表，发送GET请求到<http://169.254.169.254/openstack>:

```
$ curl http://169.254.169.254/openstack
2012-08-10
2013-04-04
2013-10-17
```



```
latest
```

要列出所支持的EC2兼容的元数据API版本，发送GET请求到http://169.254.169.254:

```
$ curl http://169.254.169.254
1.0
2007-01-19
2007-03-01
2007-08-29
2007-10-10
2007-12-15
2008-02-01
2008-09-01
2009-04-04
latest
```

如果您编写了一个 API 的 consumer，要优先尝试访问最近的 API 版本来支持您的 consumer，如果最近的一个是不可用的，那么可以追溯到更早期的版本。

来自OpenStack API的元数据分发的是JSON格式。要获取这些元数据，发送GET请求到http://169.254.169.254/openstack/2012-08-10/meta_data.json:

```
$ curl http://169.254.169.254/openstack/2012-08-10/meta_data.json
```

```
{
  "uuid": "d8e02d56-2648-49a3-bf97-6be8f1204f38",
  "availability_zone": "nova",
  "hostname": "test.novalocal",
  "launch_index": 0,
  "meta": {
    "priority": "low",
    "role": "webserver"
  },
  "project_id": "f7ac731cc11f40efbc03a9f9e1d1d21f",
  "public_keys": {
    "mykey": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGDYVEprvtYJXVOBN0XNKVVRNCRX6BlnNbI
+USLGaislsUWPwtSg7z9K9vnbYAPUZcq8c/s5S9dg5vTHbsiyPCIDOKyeHba4MUJq80h5b2i71/3BISpyxTBH/uZDHdsIW2a
+SrPDCEuMMoss9NFhBdKtDkdG9zyi0ibmCP6yMdEX8Q== Generated by Nova  n"
  },
  "name": "test"
}
```

实例也通过元数据服务，使用一个到 http://169.254.169.254/openstack/2012-08-10/user_data 的 GET 请求来获取用户数据（以 API 调用的 user_data 参数或 nova boot 命令的 --user_data 标签传递）：

```
$ curl http://169.254.169.254/openstack/2012-08-10/user_data
#!/bin/bash
echo 'Extra user data here'
```

元数据服务有一个与 [Amazon EC2 元数据服务](#) 兼容的 2009-04-04 版本的 API。这意味着为 EC2 设计的虚拟机镜像在 OpenStack 中也可以正常工作。

EC2 API 为每个元数据元素暴露一个分离的 URL。通过到 http://169.254.169.254/2009-04-04/meta-data/ 的 GET 查寻来获取这些元素的列表：

```
$ curl http://169.254.169.254/2009-04-04/meta-data/
ami-id
ami-launch-index
ami-manifest-path
```

```
block-device-mapping/  
hostname  
instance-action  
instance-id  
instance-type  
kernel-id  
local-hostname  
local-ipv4  
placement/  
public-hostname  
public-ipv4  
public-keys/  
ramdisk-id  
reservation-id  
security-groups
```

```
$ curl http://169.254.169.254/2009-04-04/meta-data/block-device-mapping/  
ami
```

```
$ curl http://169.254.169.254/2009-04-04/meta-data/placement/  
availability-zone
```

```
$ curl http://169.254.169.254/2009-04-04/meta-data/public-keys/  
0=mykey
```

实例可以通过向 `http://169.254.169.254/2009-04-04/meta-data/public-keys/0/openssh-key` 发送 GET 请求获取公有 SSH key (在用户请求一个新的实例时由密钥对名称确定)：

```
$ curl http://169.254.169.254/2009-04-04/meta-data/public-keys/0/openssh-key  
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGDYVEprvtYJXVOBN0XNKVVRNCRX6BlnNbI  
+USLGais1sUWPwtSg7z9K9vnbYAPUZcq8c/s5S9dg5vTHbsiyPCIDOKyeHba4MUJq80h5b2i71/3BISpyxBH/uZDHdsIW2a  
+SrPDCeuMMoss9NFhBdKtDkdG9zyi0ibmCP6yMdEX8Q== Generated by Nova
```

通过以 GET 请求 `http://169.254.169.254/2009-04-04/user-data`，实例可以获取到用户的数据：

```
$ curl http://169.254.169.254/2009-04-04/user-data  
#!/bin/bash  
echo 'Extra user data here'
```

元数据服务是由 `nova-api` 服务或 `nova-api-metadata` 服务实现的。请注意 `nova-api-metadata` 服务一般仅在运行多主机模式下使用，因为它会检索指定实例的元数据。如果您正在运行 `nova-api` 服务，您必需将 `metadata` 作为在 `/etc/nova/nova.conf` 文件中的 `enabled_apis` 配置选项中列出的元素之一。默认的 `enabled_apis` 配置选项包括了元数据服务，因此您不需要修改它。

主机在 `169.254.169.254:80` 上访问服务，而且它会通过 `nova-network` 服务建立的 `iptables` 规则被转化到 `metadata_host:metadata_port` 中。在多主机模式下，您可以将 `metadata_host` 设置为 `127.0.0.1`。

为了实例可以获得元数据服务，`nova-network` 服务必需配置 `iptables` 为 NAT port `169.254.169.254` 地址的 `80` 端口到 `metadata_host` 中指定的 IP 地址 (这默认是到 `$my_ip` 的，即 `nova-network` 服务的 IP 地址) 和 `/etc/nova/nova.conf` 的 `metadata_port` 中指定的端口 (默认是 `8775`)。



注意

`metadata_host` 的配置属性必须是 IP 地址，而并非主机名。

默认的计算服务设置是假设nova-network 和 nova-api是运行在同一台主机中的。若事实的情况不是这样的话，在运行nova-network的主机的/etc/nova/nova.conf文件中，设置metadata_host配置属性，即指定运行nova-api的主机的IP地址。

表 4.2. Description of metadata configuration options

配置属性=默认值	描述
[DEFAULT]	
metadata_cache_expiration = 15	(IntOpt) Time in seconds to cache metadata; 0 to disable metadata caching entirely (not recommended). Increasing this should improve response times of the metadata API when under heavy load. Higher values may increase memory usage and result in longer times for host metadata changes to take effect.
metadata_host = \$my_ip	(StrOpt) 元数据API服务器的IP地址
metadata_listen = 0.0.0.0	(StrOpt) 元数据API监听的IP地址
metadata_listen_port = 8775	(IntOpt) 元数据API监听端口
metadata_manager = nova.api.manager.MetadataManager	(StrOpt) OpenStack元数据服务管理者
metadata_port = 8775	(IntOpt) 元数据API端口
metadata_workers = 无	(IntOpt) 元数据服务的工作者数量。缺省是可用CPU的数量。
vendordata_driver = nova.api.metadata.vendordata_json.JsonFileVendorData	(StrOpt) 使用赞助商数据的驱动
vendordata_jsonfile_path = 无	(StrOpt) 从文件加载JSON格式赞助商数据

在虚拟机激活ping和SSH

需要为虚拟机访问网络启用ping 和 ssh 。这可以通过nova 和 euca2ools命令来完成。



注意

只有在用于和 nova-api交互的凭证都在/root/.bashrc时以root运行这些命令。如果EC2凭证在.bashrc文件中都是未经授权的用户，就必须替代这些用户来运行这些命令。

通过nova命令来打开ping和SSH:

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

通过euca2ools命令来打开ping和SSH:

```
$ euca-authorize -P icmp -t -1:-1 -s 0.0.0.0/0 default
$ euca-authorize -P tcp -p 22 -s 0.0.0.0/0 default
```

如果你已经运行了这些命令，仍然ping不到或SSH不到实例，请检查运行中的dnsmasq进程数量，必须是两个。如果不是两个的话，杀死这些进程然后基于下面命令重启服务：命令：

```
# killall dnsmasq
# service nova-network restart
```

配置公共(扁平)IP地址

此节阐释了如何基于nova-network来配置浮动IP地址。关于OpenStack网络完成此内容的信息，请参阅“[3层路由和网络地址转换](#)”一节 [214]。

私有和公有IP地址

在此节中，术语浮动 IP 地址是用于公有的IP地址，其可以动态的添加到正在运行中的虚拟机实例。

每个虚拟机实例都会自动的被分配一个私有IP地址。你可以选择分配一个公有(或浮动)IP地址来替代。OpenStack计算服务使用网络地址转换(NAT)来分配浮动IP给虚拟机实例。

为了能够分配一个浮动IP地址，编辑/etc/nova/nova.conf 文件，指定nova-network服务的哪块网卡须绑定公有IP地址到：

```
public_interface=VLAN100
```

如果更改/etc/nova/nova.conf 文件时，nova-network还正在运行中，那么要使这些变更生效，需要重启此服务。



使用浮动IP的虚拟机之间的流量

浮动 IP 是通过使用源 NAT (iptables 中的 SNAT 规则) 来实现的，因此如果虚拟机使用了他们的浮动 IP 来与其他虚拟机通信，特别是在同一台主机上的两台虚拟机，那么安全组的显示有时会不一致。虚拟机到虚拟机的流量通过固定网络不会产生这个问题，因此这是推荐的配置。要保证流量不会获得 SNATed 的浮动范围，请直接设置：

```
dmz_cidr=x.x.x.x/y
```

x.x.x.x/y 值指定了您定义的浮动 IP 的浮动 IP 范围。如果虚拟机在源组上有浮动 IP，这个配置也是需要的。

打开IP转发

在大多数的Linux发行版中IP转发是被禁用的，你需要打开它以使用扁平IP。



注意

IP 转发仅需要在运行 nova-network 的节点上启用。但是，您如果使用多主机模式，您需要在所有计算节点上启用它。

要检查IP转发是否打开，运行:

```
$ cat /proc/sys/net/ipv4/ip_forward
0
```

交互式的，运行:

```
$ sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0
```

在这些例子中，IP转发是禁用的。

要动态的打开IP转发，运行:

```
# sysctl -w net.ipv4.ip_forward=1
```

交互式的，运行:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

要永久的变更，变见文件/etc/sysctl.conf，然后更新IP转发设置：

```
net.ipv4.ip_forward = 1
```

保存文件然后运行此命令来应用变更：

```
# sysctl -p
```

你也可以通过重启网络服务来应用变更：

- 对于 Ubuntu 和 Debian 发行版

```
# /etc/init.d/networking restart
```

- 在 RHEL, Fedora, CentOS, openSUSE 以及 SLES中：

```
# service network restart
```

创建一个浮动IP地址的可用列表

计算服务维护了浮动IP地址的一个列表，用于分配给实例，使用命令nova-manage floating create为列表添加项：

```
# nova-manage floating create --pool nova --ip_range 68.99.26.170/31
```

使用这些nova-manage命令来执行浮动IP的操作：

- # nova-manage floating list

在池中列出浮动IP地址。

- # nova-manage floating create --pool POOL_NAME --ip_range CIDR

创建指定的浮动IP，无论是单个的地址还是子网。

- # nova-manage floating delete CIDR

删除浮动IP地址，使用和创建时命令一样的参数。

更多关于管理员如何给实例分配浮动IP的信息，请参阅OpenStack 管理用户手册中的 [管理IP地址](#) 一章。

自动添加浮动IP

你可以配置nova-network去在实例启动时自动的分配和制定浮动IP地址。添加下面几行内容到文件/etc/nova/nova.conf：

```
auto_assign_floating_ip=True
```

保存文件，然后重启 nova-network



注意

一旦启用了此属性，后果就是所有的浮动IP地址其实已经分配完毕了，命令nova boot会失败。

从一个项目中删除一个网络

已经分配给项目的网络是无法删除的。此节讲述的就是撤销分配然后删除的正确步骤。

为了撤销已经分配的网络，需要知道所分配的项目的ID，要获得项目的ID，需要是管理员的身份。

使用命令scrub来将已经分配给项目的网络撤销，将项目ID作为参数的最后部分：

```
# nova-manage project scrub --project ID
```

实例的多网卡 (multinic)

多网卡的特性允许实例使用多于一块网卡，这在多个场景中非常的有用：

- SSL 配置(VIPs)
- 服务失效切换/高可用
- 带宽分配
- 管理/公共访问实例

每个VIP代表具有其自己的IP块单独的网络。每个网络模式都有其自身针对多网络使用的设置：

图 4.4. 多网卡flat管理器

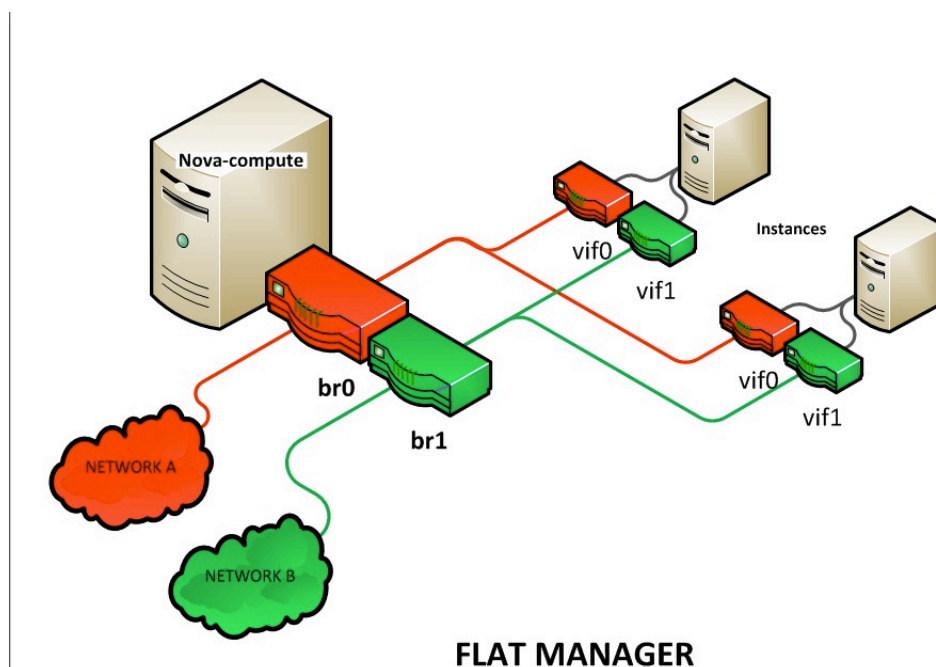


图 4.5. 多网卡flatdhcp管理器

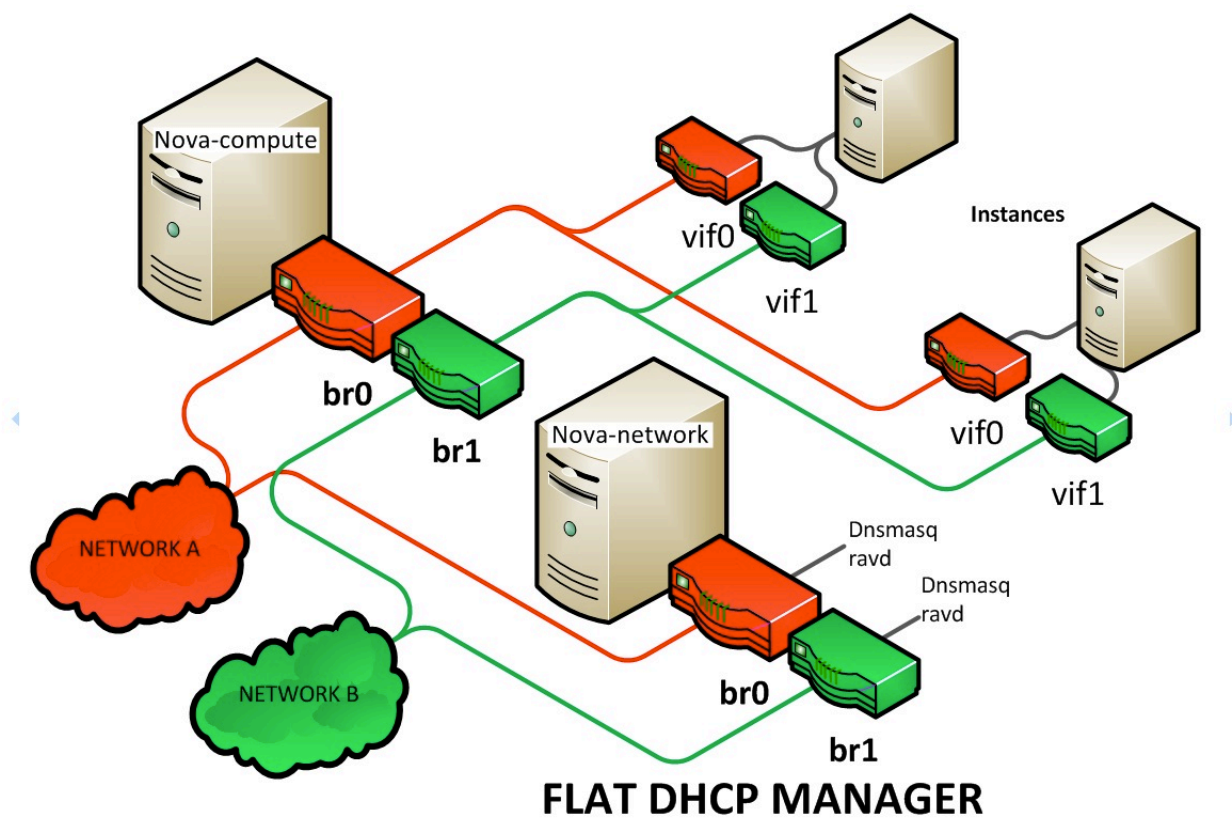
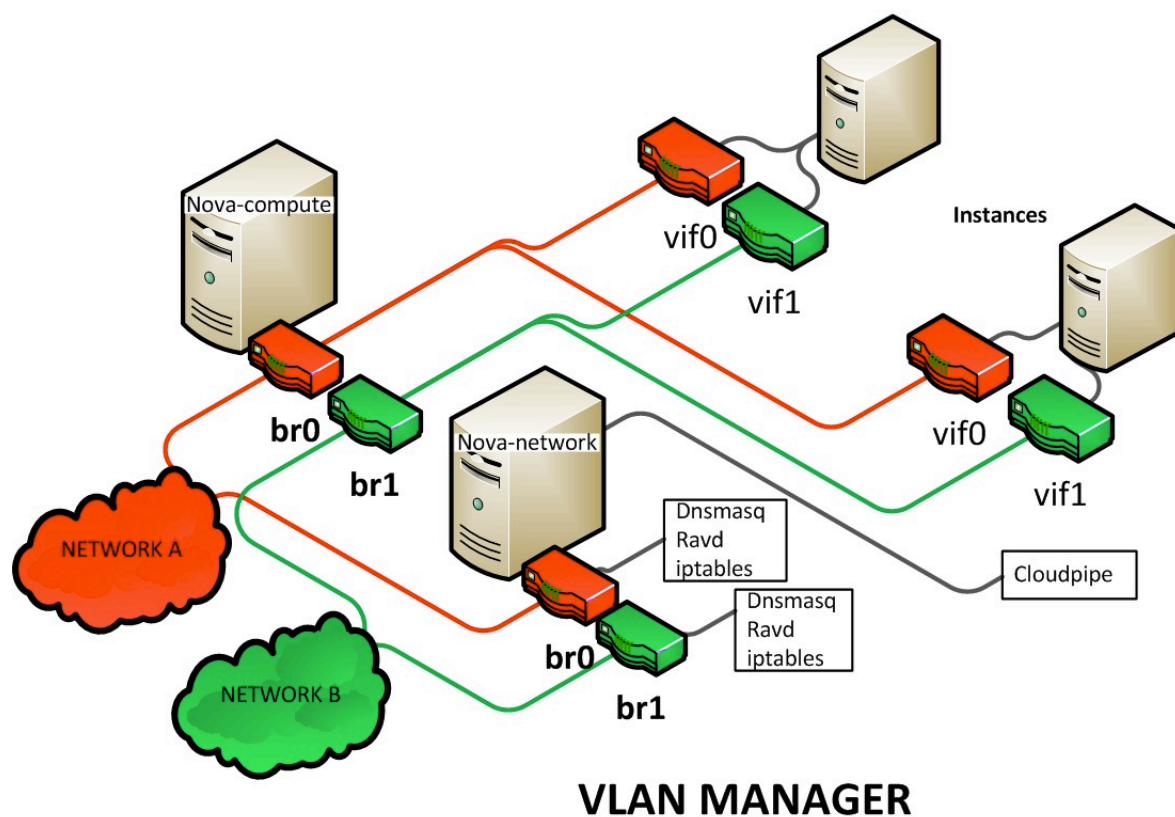


图 4.6. 多网卡 VLAN管理器



使用多网卡

为了使用多网卡，创建两个网络，然后挂接它们到租户(命令行下的project名称)：

```
$ nova network-create first-net --fixed-range-v4 20.20.0.0/24 --project-id $your-project
$ nova network-create second-net --fixed-range-v4 20.20.10.0/24 --project-id $your-project
```

每个新的实例都会从它们DHCP服务获得两个IP地址：

```
$ nova list
+-----+
| ID | Name | Status | Networks |
+-----+
| 124 | Server 124 | ACTIVE | network2=20.20.0.3; private=20.20.10.14 |
+-----+
```



注意

确保在实例中启动了第二块网卡，否则第二个IP就会不可达。

此例示范了如何在实例内部设置网卡。此配置需要应用到镜像内部。

编辑文件 `/etc/network/interfaces`：

```
# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet dhcp
```

如果虚拟网络服务Neutron已经安装，可以使用命令nova 带上参数 --nic 来指定网络挂接到网卡：

```
$ nova boot --image ed8b2a37-5535-4a5f-a615-443513036d71 --flavor 1 --nic net-id=NETWORK1_ID --nic net-id=NETWORK2_ID test-vm1
```

网络故障排查

浮动IP不可达

假如你通过浮动IP地址无法访问你的实例：

- 检查默认的安全组是允许ICMP (ping) 和SSH (端口 22)的，然后就可以访问实例了：

```
$ nova secgroup-list-rules default
+-----+-----+-----+-----+
| IP Protocol | From Port | To Port | IP Range | Source Group |
+-----+-----+-----+-----+
| icmp       | -1        | -1      | 0.0.0.0/0 |              |
| tcp        | 22        | 22      | 0.0.0.0/0 |              |
+-----+-----+-----+-----+
```

- 检查运行nova-network的节点上的iptables已经添加了NAT的规则：

```
# iptables -L -nv -t nat
-A nova-network-PREROUTING -d 68.99.26.170/32 -j DNAT --to-destination 10.0.0.3
-A nova-network-floating-snat -s 10.0.0.3/32 -j SNAT --to-source 68.99.26.170
```

- 检查公网地址(此例中是68.99.26.170)已经添加到公共的网卡。当使用命令ip addr 时可以看到其出现在列表中：

```
$ ip addr
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
link/ether xx:xx:xx:17:4b:c2 brd ff:ff:ff:ff:ff:ff
inet 13.22.194.80/24 brd 13.22.194.255 scope global eth0
inet 68.99.26.170/32 scope global eth0
inet6 fe80::82b:2bf:fe1:4b2/64 scope link
valid_lft forever preferred_lft forever
```



注意

路由的配置是不允许在相同的服务器上将有IP ssh给实例的。

- 如果包被路由到计算主机上的到达的接口，那么使用 tcpdump 来认证。如果包到达计算主机但连接失败了，问题可能是包被逆向路径过滤丢弃了。尝试在到达接口上禁用逆向路径过滤。例如，如果到达接口是 eth2，请执行：

```
# sysctl -w net.ipv4.conf.ETH2.rp_filter=0
```


如果解决了此问题，将下面这几行内容添加到文件/etc/sysctl.conf 以使变更为永久生效：

```
net.ipv4.conf.rp_filter=0
```

临时禁用防火墙

为了能够调试网络的问题对访问虚拟机有所帮助，在 /etc/nova/nova.conf 中设置下面的属性来禁用防火墙：

```
firewall_driver=nova.virt.firewall.NoopFirewallDriver
```

我们强烈建议在网络问题解决后将此行删除，从而重新启用防火墙。

从实例到nova-network服务丢包(VLANManager 模式)

如果你可以SSH到你的实例，但是网络特别的慢，又或者是你发现了其他比平时慢很多的操作(例如sudo)，那么就是连接到实例的网络发生了丢包现象。

丢包可能是和网桥相关的Linux网络配置有关。比较确定的是在VLAN网卡(例如, vlan100)和对应的运行nova-network主机上的网桥(例如, br100) 之间的包被丢弃的情况。

检查是否出现此问题的一个办法是打开三个终端然后运行下面的命令：

1. 在第一个终端，在运行nova-network的主机上执行，在VLAN的网卡使用tcpdump 来监控DNS相关的流量(UDP,端口53)。以root用户运行：

```
# tcpdump -K -p -i vlan100 -v -vv udp port 53
```

2. 在第二个终端，也是在运行nova-network主机上执行，使用tcpdump来监控网桥接口的DNS相关流量。以root运行：

```
# tcpdump -K -p -i br100 -v -vv udp port 53
```

3. 在第三个终端，SSH登录到实例，然后使用nslookup命令来生成DNS的请求：

```
$ nslookup www.google.com
```

表现可能时好时坏，所以尝试多次运行 nslookup。如果网络配置是正确的，命令每次都会立即返回，如果是错误的配置，命令会在返回之前有几秒钟的挂起。

4. 如果命令nslookup有时候会挂起，而且在第一个终端上还能看到网络包的出现，但并不是在秒内，所以可能的问题在于网络设置了过滤。尝试关闭过滤，以root运行下面的命令：

```
# sysctl -w net.bridge.bridge-nf-call-arptables=0
# sysctl -w net.bridge.bridge-nf-call-iptables=0
# sysctl -w net.bridge.bridge-nf-call-ip6tables=0
```

如果解决了此问题，将下面这几行内容添加到文件/etc/sysctl.conf 以使变更为永久生效：

```
net.bridge.bridge-nf-call-arptables=0
net.bridge.bridge-nf-call-iptables=0
net.bridge.bridge-nf-call-ip6tables=0
```


KVM：网络开始时远行良好，不久失效

基于KVM hypervisor运行Ubuntu 12.04的实例有时会在运行正常一段时间后突然网络失去连接。尝试加载内核模块vhost_net来修复此问题(请参阅 [bug #997978](#))。此内核模块也会[改进KVM的网络性能](#)。要加载此内核模块：

```
# modprobe vhost_net
```



注意

加载模块对正在运行的实例没有影响。

系统管理

为了更好的管理计算服务，你必须理解安装到不同的节点后它们之间时如何交互的。计算服务可以被安装到不同的服务器中，但是通常多数计算节点控制虚拟化的主机，且有一个云控制器节点囊括了计算服务。

计算服务使用了名为nova-*一系列守护进程，它们都持久存在于主机服务器中。这些二进制的程序可以全部运行在同一台机器中，也可以分开各自部署到大规模的环境中。这些服务和驱动有：

服务

- nova-api:接收XML的请求，然后发送它们到其它系统。需要WSGI应用路由喝认证。支持EC2和OpenStack API。当计算服务安装好后会创建nova.conf配置文件。
- nova-cert: 证书管理。
- nova-compute: 管理虚拟机。负载服务对象，通过远程过程调用(RPC) 抛出公用的方法给计算管理器。
- nova-conductor: 为计算节点提供了数据库访问的支持(因此降低了安全风险)。
- nova-consoleauth: 管理终端认证。
- nova-objectstore:简单的基于文件的存储镜像的系统，复制了大多数的S3 API。它可以被OpenStack镜像服务所替代，无论是简单镜像管理还是OpenStack对象存储，都可作为虚拟机镜像存储。它必须和nova-compute服务处于同一主机。
- nova-network：管理浮动和固定 IP、DHCP、桥接和 VLAN。加载一个在NetworkManager 上暴露公有方法的服务对象。设置不同的网络策略可以通过修改network_manager 配置选项，可用选项为 network_manager configuration option to FlatManager、FlatDHCPManager 或 VLANManager（如果没有指定，默认是 VLANManager）。
- nova-scheduler: 分发新建虚拟机的请求到合适的节点。
- nova-novncproxy: 为浏览器提供一个VNC的代理，允许VNC终端访问虚拟机。



注意

一些服务有改变了服务如何实现其核心功能的驱动。例如，nova-compute 服务支持您选择它能够使用的 hypervisor 类型。nova-network 和 nova-scheduler 也有驱动。

管理计算用户

Euca2ools (ec2) API 的访问是由一个访问 key 和 secret key 控制的。需要在请求中包含用户的访问 key，且请求必需以 secret key 标记。当收到了 API 请求，计算服务验证其签名并代表用户执行命令。

要开始使用计算机，你必须在认证服务那里创建一个用户。

管理卷

取决于你的云的提供商的步骤，他们可能会给出端点来用于管理卷，或者是给定一个扩展。无论是哪种情况，你都可以使用 nova 命令行来管理卷：

volume-attach	Attach a volume to a server.
volume-create	Add a new volume.
volume-delete	Remove a volume.
volume-detach	Detach a volume from a server.
volume-list	List all the volumes.
volume-show	Show details about a volume.
volume-snapshot-create	Add a new snapshot.
volume-snapshot-delete	Remove a snapshot.
volume-snapshot-list	List all the snapshots.
volume-snapshot-show	Show details about a snapshot.
volume-type-create	Create a new volume type.
volume-type-delete	Delete a specific flavor
volume-type-list	Print a list of available 'volume types'.
volume-update	Update an attached volume.

例如，要列出计算卷的ID和名称，运行：

```
$ nova volume-list
+-----+-----+-----+-----+-----+-----+
| ID              | Status | Display Name | Size | Volume Type | Attached to |
+-----+-----+-----+-----+-----+-----+
| 1af4cb93-d4c6-4ee3-89a0-4b7885a3337e | available | PerfBlock  | 1    | Performance |              |
+-----+-----+-----+-----+-----+-----+
```

云主机类型

管理员可以使用 nova flavor- 命令来定制和管理云类型。要查看可用的云类型命令，运行：

```
$ nova help | grep flavor-
flavor-access-add Add flavor access for the given tenant.
flavor-access-list Print access information about the given flavor.
flavor-access-remove
                    Remove flavor access for the given tenant.
flavor-create      Create a new flavor
flavor-delete      Delete a specific flavor
flavor-key         Set or unset extra_spec for a flavor.
flavor-list        Print a list of available 'flavors' (sizes of
flavor-show        Show details about the given flavor.
```



注意

- 配置权限可以委托给其他用户，通过重新定义访问控制来实现，这步在 nova-api 服务器上的 /etc/nova/policy.json 文件中定义 compute_extension:flavormanage。

- 要在仪表盘中修改现有的云主机类型，你必须删除某个云主机类型，然后再创建一个同名的即可。

云主机类型可定义的元素：

表 4.3. 认证服务配置文件

元素	描述
名称	一个描述性的名称。XX.SIZE_NAME 通常不是必需的，尽管有些第三方工具可能依赖它。
Memory_MB	虚拟机的内存，以兆字节表示。
磁盘	虚拟机根磁盘，已GB表示。这是一个临时的磁盘，其复制了基础镜像。当从一个持久卷启动时用不到这个。大小为"0"是特殊情况，使用了本地基础镜像大小作为临时根卷的大小。
临时卷	指定第二个临时数据磁盘的大小。这是空的，没有格式化的，仅在实例的生存期有效。
Swap	可选项，为实例分配swap空间。
虚拟内核	为实例提供的虚拟CPU的数量。
RXTX_Factor	可选项，属性允许创建服务器中定义的带宽上限不同于它们所挂接的网络。此因素由网络的属性 rxtx_base 成倍的增加所决定。默认值是1.0。表示的和所挂接的网络一样。此参数仅适用于基于Xen或NSX的系统。
Is_Public	布尔值，决定云主机类型是所有用户都可访问，还是所创建的租户自用。默认是True。
extra_specs	键值对，定义了那台计算节点可以让云主机类型在其上运行。这些键值对匹配对应的计算节点上的键值对。用户实现特殊的资源，比如一个云主机类型只能运行在拥有GPU硬件的计算节点上。

云主机类型的定制受限于所使用的Hypervisor。举例来说，libvirt驱动启用了针对虚拟机的CPU配额，磁盘调整，I/O带框，看门狗行为，随机数生成器设备控制，以及实例虚拟网卡的流量控制。

CPU限制 你可以使用nova客户端的控制参数来限制CPU。例如，要配置I/O限制的话，使用：

```
$ nova flavor-key m1.small set quota:read_bytes_sec=10240000
$ nova flavor-key m1.small set quota:write_bytes_sec=10240000
```

使用这些可选的参数来控制加权共享,执行间隔运行时配额,配额最大允许带宽:

- **cpu_shares**。为域指定一个比例的加权共享。如果此元素被忽略的话，服务的默认值就是操作系统所提供的默认值。此值没有单位；它和其它虚拟机的设置相关联。举例来说，一个虚拟机配置为2048，那么它比比配置为1024的虚拟机拥有多2倍的CPU时间。
- **cpu_period**。为QEMU和LXC hypervisor指定执行间隔(单位：微秒)。在一个执行的间隔期，域内的每个VCPU是不允许消费多于配额所限定的。此值的配置范围是 [1000, 1000000]。值为0的则意味着没有限制。
- **cpu_limit**。指定VMware机器分配CPU的上限，已MHz为单位。此参数确保一台虚拟机永远不能使用超过指定CPU时间。它可用于强制限定机器的CPU性能。

- `cpu_reservation`。指定了保证最低CPU预订VMware，以MHz计算。这意味着，如果需要，机器肯定会分配预留数量的CPU周期。

- `cpu_quota`。指定最大允许带宽(单位：微秒)。一个为负值的域的配额则表明此域拥有无限的带宽，也就意味着没有带宽控制。值的范围须是在[1000, 18446744073709551]之间，或者干脆为0。一个为0的值意思是没有值。你可以使用此特性来确保所有的vCPU运行在同一个速度下。例如：

```
$ nova flavor-key m1.low_cpu set quota:cpu_quota=10000
$ nova flavor-key m1.low_cpu set quota:cpu_period=20000
```

在此例中，`m1.low_cpu`的实例最大仅能消费物理CPU计算能力的50%。

磁盘调整

使用磁盘I/O配额，你可以设置为一个虚拟机用户设置最大的磁盘写入速度为每秒10MB。例如：

```
$ nova flavor-key m1.medium set quota:disk_write_bytes_sec=10485760
```

磁盘I/O属性有：

- `disk_read_bytes_sec`
- `disk_read_iops_sec`
- `disk_write_bytes_sec`
- `disk_write_iops_sec`
- `disk_total_bytes_sec`
- `disk_total_iops_sec`

I/O带框

虚拟网卡的I/O属性有：

- `vif_inbound_average`
- `vif_inbound_burst`
- `vif_inbound_peak`
- `vif_outbound_average`
- `vif_outbound_burst`
- `vif_outbound_peak`

入口和出口流量可以分别设置。带宽元素至多是一个入口和至多一个出口。如果你不使用这些子元素，就没有服务质量(QoS)应用于流量。所以，如果你打算仅配置网络的入口流量，只使用入口(反之也一样)。每个元素都有一个强制性的属性，它指定了已经配置的接口多平均比特率。

还有两个可选项(整型)：`peak`，其指定了一个网桥可以发送数据的最大速率(kilobytes/秒)，以及`burst`，在速度的峰值可以达到的最大bytes数(kilobytes)。在连接到网络的域内速率可以被等同的共享。

以下例子是为已经存在的云主机类型设置网络流量带宽：

- 出口流量：
 - 平均: 256 Mbps (32768 kilobytes/秒)
 - 峰值 : 512 Mbps (65536 kilobytes/秒)
 - burst: 100毫秒
- 入口流量：
 - 平均: 256 Mbps (32768 kilobytes/秒)
 - 峰值 : 512 Mbps (65536 kilobytes/秒)
 - burst: 100毫秒

```
$ nova flavor-key nlimit set quota:vif_outbound_average=32768
$ nova flavor-key nlimit set quota:vif_outbound_peak=65536
$ nova flavor-key nlimit set quota:vif_outbound_burst=6553
$ nova flavor-key nlimit set quota:vif_inbound_average=16384
$ nova flavor-key nlimit set quota:vif_inbound_peak=32768
$ nova flavor-key nlimit set quota:vif_inbound_burst=3276
```



注意

以上例子中所有的速度限制值均以kilobytes/秒计算，只有burst的值是以kilobytes计算的。

看门狗行为 对于libvirt驱动来说，你可以为每个云主机类型启用和设置一个虚拟硬件看门狗。看门狗设备监视着客户服务器，且如果服务器宕机，会做出相应的举动。看门狗使用 i6300esb设备(模拟Intel的6300ESB)。如果没有指定hw:watchdog_action，则说明禁用了看门狗。

要设置行为，使用：

```
$ nova flavor-key FLAVOR-NAME set hw:watchdog_action=ACTION
```

合法的ACTION值有：

- disabled—(default) The device is not attached.
- reset—Forcefully reset the guest.
- poweroff—Forcefully power off the guest.
- pause—Pause the guest.
- none—Only enable the watchdog; do nothing if the server hangs.



注意

看门狗行为设置使用了一个特定的镜像属性，会覆盖掉使用云主机类型的行为。

随机数生成器

如果通过实例镜像的属性为实例添加了一块随机数生成器设备，可使用下面来启用和配置此设备：

```
$ nova flavor-key FLAVOR-NAME set hw_rng:allowed=True
$ nova flavor-key FLAVOR-NAME set hw_rng:rate_bytes=RATE-BYTES
$ nova flavor-key FLAVOR-NAME set hw_rng:rate_period=RATE-PERIOD
```

地点：

- RATE-BYTES—(Integer) Allowed amount of bytes that the guest can read from the host's entropy per period.
- RATE-PERIOD—(Integer) Duration of the read period in seconds.

项目私有云主机类型

云主机类型可以分配给特定的项目。默认情况下，一个云主机类型是公有的且所有项目均可用。私有云主机类型仅能让访问列表中项目访问，且对其它项目不可见。要为某项目创建和分配一个私有云主机类型，运行下面命令：

```
$ nova flavor-create --is-public false p1.medium auto 512 40 4
$ nova flavor-access-add 259d06a0-ba6d-4e60-b42d-ab3144411d58
86f94150ed744e08be565c2ff608eef9
```

计算节点的防火墙需求

终端连接到虚拟机，或者是直接连接或者是通过代理，都要通过端口5900到5999。每个计算服务节点的防火墙都需要允许这些端口上的网络流量通过。

此过程时修改了iptables防火墙，以允许连接到计算服务。

配置计算节点的防火墙

1. 使用 root 用户登录到计算服务所在主机。
2. 编辑文件/etc/sysconfig/iptables，添加一个INPUT规则，允许从端口5900到5999到TCP流量通过。要确保新的规则是在REJECT流量之前：

```
-A INPUT -p tcp -m multiport --dports 5900:5999 -j ACCEPT
```

3. 保存变更到/etc/sysconfig/iptables，然后重启iptables服务以让变更生效：

```
$ service iptables restart
```

4. 在每个计算节点重复此过程。

注入管理员密码

计算服务可以生成一个随机管理员 (root) 密码并将密码注入到实例中。如果启用了该特性，用户可以 ssh 到实例中而不使用 ssh 密钥对。随机密码出现在 nova boot 命令的输出中。您也可以从仪表盘中浏览和设置管理员密码。

使用面板来实现密码注入

默认情况下，面板会显示admin 密码且允许用户修改它。

如果你不打算支持密码注入的，通过编辑面板的local_settings文件，将密码那栏禁用。在Fedora/RHEL/CentOS,文件路径是 /etc/openstack-dashboard/local_settings。在Ubuntu 和 Debian, 它是 /etc/openstack-dashboard/local_settings.py。在openSUSE 和 SUSE Linux Enterprise Server, 它是 /srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py

```
OPENSTACK_HYPERVISOR_FEATURES = {  
    ...  
    'can_set_password': False,  
}
```

基于libvirt管理的Hypervisor的密码注入

使用libvirt的Hypervisor(诸如KVM，QEMU，以及LXC)，管理员密码注入默认是禁用的。要启用它，在/etc/nova/nova.conf中设置此属性：

```
[libvirt]  
inject_password=true
```

当启用后，计算服务将会修改实例操作系统的管理员账户密码，通过编辑虚拟机实例内部的文件/etc/shadow来实现。



注意

如果虚拟机镜像时一个Linux发行版用户只能ssh到实例，且已经配置了允许用户ssh为根用户。但这对于Ubuntu 云镜像来说不可能，因为其不允许用户ssh 到根账户。

密码注入和XenAPI (XenServer/XCP)

当hypervisor后端使用了XenAPI时，计算服务是使用XenAPI代理来为客户操作系统注入密码的。虚拟机镜像必须被配置代理以让密码注入可正常工作。

密码注入和Windows镜像(所有的hypervisor)

对于Windows虚拟机，配置Windows镜像在启动时获取管理员密码，可安装代理如 [cloudbase-init](#)。

管理云

系统管理员可以使用nova 客户端和 Euca2ools命令来管理他们的云。

nova客户端和euca2ools可被用于所有的用户，尽管身份服务的基于角色的访问控制对于一些指令仍然有所限制。

使用nova 客户端来管理云

1. 软件包python-novaclient提供了nova shell，用来从命令行和计算服务API的交互。安装客户端，然后提供用户名和密码(其也可以为便利设置为环境变量)，在命令行下也具有了管理员点能力。

要安装python-novaclient，从 <http://pypi.python.org/pypi/python-novaclient/#downloads> 下载tar包，然后安装到你喜欢的Python环境中。

```
$ curl -O http://pypi.python.org/packages/source/p/python-novaclient/python-novaclient-2.6.3.tar.gz
$ tar -zxvf python-novaclient-2.6.3.tar.gz
$ cd python-novaclient-2.6.3
```

以root运行:

```
# python setup.py install
```

2. 确认安装时成功的:

```
$ nova help
usage: nova [--version] [--debug] [--os-cache] [--timings]
        [--timeout SECONDS] [--os-username AUTH_USER_NAME]
        [--os-password AUTH_PASSWORD]
        [--os-tenant-name AUTH_TENANT_NAME]
        [--os-tenant-id AUTH_TENANT_ID] [--os-auth-url AUTH_URL]
        [--os-region-name REGION_NAME] [--os-auth-system AUTH_SYSTEM]
        [--service-type SERVICE_TYPE] [--service-name SERVICE_NAME]
        [--volume-service-name VOLUME_SERVICE_NAME]
        [--endpoint-type ENDPOINT_TYPE]
        [--os-compute-api-version COMPUTE_API_VERSION]
        [--os-cacert CA_CERTIFICATE] [--insecure]
        [--bypass-url BYPASS_URL]
        SUBCOMMAND ...
```

此命令会返回nova 命令和参数的列表。要获得其子命令的帮助，运行：

```
$ nova help SUBCOMMAND
```

对于命令nova及其参数的完整列表，请参阅[OpenStack 命令行参考](#)。

3. 将所需要的参数设置为环境变化从而让运行命令更加的简单。例如，可以增加parameter>--os-username

```
$ export OS_USERNAME=joecool
$ export OS_PASSWORD=coolword
$ export OS_TENANT_NAME=coolu
```

4. 身份服务会给出一个认证端点，它就是计算服务所可以识别的 OS_AUTH_URL。

```
$ export OS_AUTH_URL=http://hostname:5000/v2.0
$ export NOVA_VERSION=1.1
```

使用euca2tools管理云

euca2ools命令行工具提供了访问EC2 API调用的命令行接口。关于euca2ools的更多信息，请参阅 http://open.eucalyptus.com/wiki/Euca2oolsGuide_v1.3。

显示主机和实例的使用状态

你可以现实主机和实例资源使用的基本状态。



注意

更多复杂的监控，请参阅[ceilometer](#)项目。你也可以使用诸如[Ganglia](#)或者[Graphite](#)工具来收集更多的细节数据。

显示主机使用状态

以下例子是主机调用了devstack而显示的主机使用状态。

- 列出主机以及运行在其之上的nova相关的服务：

```
$ nova host-list
+-----+-----+-----+
| host_name | service | zone |
+-----+-----+-----+
| devstack | conductor | internal |
| devstack | compute | nova |
| devstack | cert | internal |
| devstack | network | internal |
| devstack | scheduler | internal |
| devstack | consoleauth | internal |
+-----+-----+-----+
```

- 获取运行在主机上所有实例的资源使用总量：

```
$ nova host-describe devstack
+-----+-----+-----+-----+-----+
| HOST | PROJECT | cpu | memory_mb | disk_gb |
+-----+-----+-----+-----+-----+
| devstack | (total) | 2 | 4003 | 157 |
| devstack | (used_now) | 3 | 5120 | 40 |
| devstack | (used_max) | 3 | 4608 | 40 |
| devstack | b70d90d65e464582b6b2161cf3603ced | 1 | 512 | 0 |
| devstack | 66265572db174a7aa66eba661f58eb9e | 2 | 4096 | 40 |
+-----+-----+-----+-----+-----+
```

cpu一栏显示了运行在主机上的实例的虚拟CPU总量。

memory_mb一栏显示了运行在主机上实例所分配内存总量(以MB计算)。

disk_gb一栏显示了运行在主机上实例的根磁盘和临时磁盘的大小总量(以GB计算)。

在PROJECT 一栏中used_now一排的值显示的是运行在主机上所分配给实例的资源总量，加上主机本身的所分配的虚拟机的资源。

在PROJECT 一栏中used_max一排的值显示的是运行在主机上所分配给实例的资源总量。



注意

这些值是使用运行在主机上实例的类型的信息计算所得。此命令不会查询物理主机的CPU的使用量，内存的使用量或者磁盘使用量。

显示实例使用状态

- 获取某实例的CPU，内存，I/O，以及网络的状态。

- 实例列表：

```
$ nova list
```

ID	Name	Status	Task State	Power State	Networks
84c6e57d-a6b1-44b6-81eb-fcb36afd31b5	myCirrosServer	ACTIVE	None	Running	private=10.0.0.3
8a99547e-7385-4ad1-ae50-4ecfaaad5f42	myInstanceFromVolume	ACTIVE	None	Running	private=10.0.0.4

2. 获取诊断状态：

```
$ nova diagnostics myCirrosServer
```

Property	Value
vnet1_rx	1210744
cpu0_time	19624610000000
vda_read	0
vda_write	0
vda_write_req	0
vnet1_tx	863734
vnet1_tx_errors	0
vnet1_rx_drop	0
vnet1_tx_packets	3855
vnet1_tx_drop	0
vnet1_rx_errors	0
memory	2097152
vnet1_rx_packets	5485
vda_read_req	0
vda_errors	-1

· 获取每个租户的状态汇总：

```
$ nova usage-list
```

Usage from 2013-06-25 to 2013-07-24:

Tenant ID	Instances	RAM MB-Hours	CPU Hours	Disk GB-Hours
b70d90d65e464582b6b2161cf3603ced	1	344064.44	672.00	0.00
66265572db174a7aa66eba661f58eb9e	3	671626.76	327.94	6558.86

日志记录

日志模块

日志的行为可通过配置文件来改变。要指定配置文件的话，在文件/etc/nova/nova.conf中添加下面几行内容：

```
log-config=/etc/nova/logging.conf
```

要更改日志级别，以参数对形式添加DEBUG, INFO, WARNING, 或 ERROR。

日志的配置文件时INF风格的配置，其必须包含称之为 logger_nova 的一节，这个控制了nova-*服务的日志行为，举例：

```
[logger_nova]
level = INFO
handlers = stderr
qualname = nova
```

此例是将调试级别的日志置为INFO(这比默认的DEBUG设置要少一些内容)。

更多关于日志配置的语法，包括handlers 和qualname等变量，请参阅 [Python 文档](#) 的日志配置。

对于定义多种handler的 logging.conf实例文件，请参阅[OpenStack 配置参考](#)。

系统日志

OpenStack计算服务可以发送日志信息到syslog。如果你打算使用rsyslog将日志存放在远程的机器中，这个就非常的有用。将计算服务(nova),身份服务(keystone)，镜像服务(glance)分别来配置，块存储服务(cinder)也发送日志消息到syslog。打开这些配置文件：

- /etc/nova/nova.conf
- /etc/keystone/keystone.conf
- /etc/glance/glance-api.conf
- /etc/glance/glance-registry.conf
- /etc/cinder/cinder.conf

在每个配置文件中，增加下面这些行：

```
verbose = False
debug = False
use_syslog = True
syslog_log_facility = LOG_LOCAL0
```

另外启用syslog需要说明的一点，这些设置通常是将日志详细和调试输出关闭的。



注意

尽管这个示例为每个服务使用了同一个本地设施 (LOG_LOCAL0，对应于 syslog 设施 LOCAL0)，我们建议您为每个服务配置一个分开的本地设施，因为这可以提供更好的隔离性和灵活性。例如，您可以获取不同服务级别的服务的日志信息。syslog 允许您定义最多 8 个本地设施，LOCAL0, LOCAL1, ..., LOCAL7。要了解更多信息，请阅读 syslog 文档。

远程系统日志

rsyslog 对于在多台机器上设置一个集中日志服务器是非常有用的。本小节简单地描述了如何配置 rsyslog 服务器。完整的 rsyslog 培训超出了本书的范围。本小节假设 rsyslog 已经在您的主机上安装好 (对于大多数的 Linux 发行版，它是默认安装的)。

此例提供了一个在日志服务主机上的/etc/rsyslog.conf最小化配置，此主机用来接收日志文件：

```
# provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 1024
```

添加一个寻找主机名的过滤规则到 `/etc/rsyslog.conf` 中。这个示例使用了 `COMPUTE_01` 作为计算节点的主机名称：

```
:hostname, isequal, "COMPUTE_01" /mnt/rsyslog/logs/compute-01.log
```

在每个计算节点中创建名称为 `/etc/rsyslog.d/60-nova.conf` 的文件，然后添加下面内容：

```
# prevent debug from dnsmasq with the daemon,none parameter
*.*:auth,authpriv,none,daemon,none,local0,none -/var/log/syslog
# Specify a log level of ERROR
local0,error @@172.20.1.43:1024
```

一旦你创建了文件，重启 `rsyslog` 服务。计算节点的 `Error` 级别的日志消息就会发送到日志服务器中。

串口终端

串口终端提供了诊断内核输出的途径，以及在实例缺乏网连接的情况下修复其它系统故障时的解决途径。

OpenStack Icehouse 及其更早的版本支持只读访问使用串口终端来使用 `os-GetSerialOutput` 服务的动作。多数的镜像默认都会启动此特性。更多信息，请参阅 [故障修复](#)。

OpenStack Juno 及其后续版本支持读写访问使用串口终端来使用 `os-GetSerialConsole` 的服务行为。此特性也需要 `websocket` 客户端的支持来访问串口终端。

配置串口终端的读-写访问

在计算节点，编辑文件 `/etc/nova/nova.conf`：

1. 在 `[serial_console]` 一节，启用串口终端：

```
[serial_console]
...
enabled = true
```

2. 在 `[serial_console]` 章节中，配置串口终端代理类似于图形终端代理：

```
[serial_console]
...
base_url = ws://控制器:6083/
listen = 0.0.0.0
proxycient_address = MANAGEMENT_INTERFACE_IP_ADDRESS
```

`base_url` 属性指定基本的 URL，此 URL 是客户端获取来自 API 上所请求的串口终端地址。典型情况，它为控制器节点的主机名称。

`listen` 属性指定了 `nova-compute` 须监听的网卡，用于虚拟终端的连接。典型的，使用 `0.0.0.0` 来监听所有的网卡。

`proxycient_address` 属性指定了代理应连接到哪块网卡，典型的情况是管理网卡的 IP 地址。

当你启用了读写访问串口终端时，计算服务会为实例将串口终端的信息添加到Libvirt的XML文件中。例如：

```
<console type='tcp'>
  <source mode='bind' host='127.0.0.1' service='10000'>
  <protocol type='raw'>
  <target type='serial' port='0'>
  <alias name='serial0'>
</console>
```

通过串口终端访问实例

1. 使用命令 `nova get-serial-proxy` 来获取实例串口终端的websocket URL：

```
$ nova get-serial-proxy INSTANCE_NAME
+-----+
| Type | Url |
+-----+
| serial | ws://127.0.0.1:6083/?token=18510769-71ad-4e5a-8348-4218b5613b3d |
+-----+
```

另外，可直接使用API：

```
$ curl -i 'http://<controller>:8774/v2/<tenant_uuid>/servers/<instance_uuid>/action'
-X POST
-H "Accept: application/json"
-H "Content-Type: application/json"
-H "X-Auth-Project-Id: <project_id>"
-H "X-Auth-Token: <auth_token>"
-d '{"os-getSerialConsole": {"type": "serial"}}'
```

2. 使用Python websocket加上URL，会为访问串口终端生成 `.send`、`.recv`，以及 `.fileno` 方法。例如：

```
import websocket
ws = websocket.create_connection(
    'ws://127.0.0.1:6083/?token=18510769-71ad-4e5a-8348-4218b5613b3d',
    subprotocols=['binary', 'base64'])
```

另外，使用Python websocket客户端，例如<https://github.com/larsks/novaconsole/>。



注意

当你启用了串口终端，实例使用命令 `nova console-log` 的记录就会禁用。这样的话，除非你是在串口终端下，否则内核的输出以及其它系统消息都不会显示。

基于rootwrap的安全

Rootwrap是允许未授权的用户以root用户来安全的运行计算服务的一些动作的机制。计算服务原来使用sudo来实现此目的，但是这样非常的难以维护，况且还不允许有过滤。所以rootwrap 命令替代了sudo。

要使用rootwrap,须为计算命令加上前缀nova-rootwrap。例如：

```
$ sudo nova-rootwrap /etc/nova/rootwrap.conf 命令
```

通用的 `sudoers` 入口使计算用户能够以 `root` 执行 `nova-rootwrap`。`nova-rootwrap` 代码寻找其配置文件中的过滤器定义目录，并从中加载命令过滤器。然后，它会检查计算节点的请求

是否匹配过滤器之一，如果匹配，(以 root) 执行命令。如果没有过滤器匹配，它会拒绝请求。



注意

请主意使用 NFS 和属于 root 的文件的问题。NFS 共享必需配置启用 no_root_squash 选项，这是为了使 rootwrap 正常工作。

Rootwrap 是完全由 root 用户控制的。root 用户拥有 sudoers 入口，这个入口允许计算节点以 root 且仅以特定配置文件 (这些文件被 root 所有) 运行一些指定的 rootwrap。nova-rootwrap 命令导入 Python 模块，它需要从一个干净的、系统默认的 PYTHONPATH 导入。被 root 所有的配置文件指向被 root 所有的过滤器定义目录，这些目录包含了被 root 所有的过滤定义文件。这样的链保证了计算节点用户自己不被配置或由 nova-rootwrap 可执行文件使用的模块的控制。

Rootwrap 是通过 rootwrap.conf 文件进行配置的。由于它在安全路径的信任之下，它必需被 root 用户所有并可以被 root 用户编辑。文件的位置在 sudoers 入口和 nova.conf 配置文件中的 rootwrap_config=entry 参数指定。

rootwrap.conf 文件使用一个包含这些小节和参数的 INI 文件格式：

表 4.4. rootwrap.conf 配置选项

配置属性=默认值	(类型) 描述
[DEFAULT] filters_path=/etc/nova/rootwrap.d,/usr/share/nova/rootwrap	(ListOpt) 逗号分开的目录列表包含了过滤器定义文件。定义了 rootwrap 过滤器保存的地方。在这一行中定义的目录都应该存在，并被 root 用户拥有和对 root 用户可写。

如果 root wrapper 没有正常运作，您可以添加一个解决选项到 nova.conf 配置文件中。这个解决办法重新配置了 root wrapper 的配置来回到以 sudo 执行命令，且这是一个 Kilo 版本的发布特性。

将这个解决办法包括到您的配置文件防止您环境出现可能损害 root wrapper 性能的问题。工具修改影响了 [Python Build Reasonableness \(PBR\)](#)，例如，一个影响 root wrapper 性能的已知问题。

要设置此临时修复的话，在配置文件 nova.conf 中的 [workaround] 一节设置 disable_rootwrap。

过滤器定义文件包含了 rootwrap 会使用的过滤器来允许或拒绝一个特定的命令。他们一般以 .filters 作为后缀。由于它们是受信任的安全路径，它们需要被 root 用户所有并对 root 用户可写。它们的位置在 rootwrap.conf 文件中指定。

过滤器定义文件使用一个 INI 文件格式，包含 [Filters] 小节和几行内容，每个都使用了一个唯一的参数名称，这些参数名称在您定义的过滤器中应该是不同的：

表 4.5. .filters 配置选项

配置属性=默认值	(类型) 描述
[Filters] filter_name=kpartx: CommandFilter, /sbin/kpartx, root	(ListOpt) 逗号分开的列表包含了使用的过滤器类，紧跟着过滤器参数 (取决于选定过滤器类的不同)。

配置迁移



注意

只有云管理员用户可以执行在线迁移。如果云是被配置为使用单元，非管理员用户可以执行在线迁移，但不是在单元之间进行。

迁移可让管理员将虚拟机实例从一台计算节点迁移到其它的节点。此特性对于当一个计算节点需要维护时特别的有用处。当然迁移也可用于当某台计算节点运行了太多虚拟机实例而过载需要重现分发的情况。

迁移的类型有：

- 非在线迁移（有时也称之为‘迁移’）。也就是在迁移到另外的计算节点时的这段时间虚拟机实例是处于宕机状态的。在此情况下，实例需要重启才能工作。
- 在线迁移（或‘真正的在线迁移’）。实例几乎没有宕机时间。用于当实例需要在迁移时保持运行。在线迁移有下面几种类型：
 - 基于共享存储的在线迁移。所有的Hypervisor都可以访问共享存储。
 - 块在线迁移。无须共享存储。但诸如CD-ROM 和配置驱动 ([config_drive](#))之类的只读设备是无法实现的。
 - 基于卷的在线迁移。实例都是基于卷的而不是临时的磁盘，无须共享存储，也支持迁移（目前仅支持基于libvirt的hypervisor）。

下面章节描述了如何配置主机和计算节点来作迁移，Hypervisor有KVM和XenServer。

KVM-Libvirt

共享存储

先决条件

- Hypervisor: 基于libvirt的 KVM
- 共享存储:NOVA-INST-DIR/instances/（例如，/var/lib/nova/instances）挂载了共享存储。此向导使用了NFS，没有特殊的属性，也包括了 [OpenStack Gluster Connector](#)。
- 实例: 基于iSCSI卷的实例是可以迁移的。



注意

- 因为计算服务默认不使用libvirt的在线迁移功能，客户操作系统在迁移之前会休眠，或许可能有几分钟的宕机。更多细节，请参考 [“激活真实的在线迁移”一节 \[96\]](#)。
- 此向导假设了文件nova.conf中 instances_path的默认值 (NOVA-INST-DIR/instances)。如果你更改了state_path 或 instances_path 变量，请在命令行中也做对应的修改。
- 你必须指定 vncserver_listen=0.0.0.0，否则在线迁移将无法正常工作。

- 必须在每个运行nova-compute的节点中指定instances_path。每个节点上instances_path的挂载点也必须相同。否则在线迁移就无法正常工作。

计算环境安装实例

- 准备至少三台服务器。在此例中，我们的服务器分别是HostA, HostB, 和 HostC:
- HostA 云控制器, 且须运行这些服务: nova-api, nova-scheduler, nova-network, cinder-volume, 以及 nova-objectstore。
- HostB 和 HostC均是运行nova-compute的计算节点。

确保所有节点上的NOVA-INST-DIR(在nova.conf文件中设置state_path)是相同的。

- 在此例中, HA是NFSv4服务, 其抛出目录NOVA-INST-DIR/instances。HostB 和 HostC 是NFSv4的客户端, 挂载HostA抛出的NFSv4目录。

配置系统

1. 配置DNS或者/etc/hosts, 确保所有主机保持一致。确保三台主机可以相互彼此解析。作为测试, 使用命令 ping从其中一台ping其它的主机。

```
$ ping HostA
$ ping HostB
$ ping HostC
```

2. 确保计算服务和libvirt的用户的UID和GID是在所有主机都是一样的。这可确保NFS挂载权限正常的工作。
3. 从 HostA抛出NOVA-INST-DIR/instances, 同时确保HostB 和 HostC中的计算服务用户可读写此目录。

更多信息, 请参阅[SettingUpNFSHowTo](#) 或 [CentOS/Red Hat: 配置 NFS v4.0 文件服务器](#)。

4. 在 HostA中配置NFS服务, 添加下面几行内容到/etc/exports 文件:

```
NOVA-INST-DIR/instances HostA/255.255.0.0(rw,sync,fsid=0,no_root_squash)
```

更改子网掩码(255.255.0.0)为HostB 和HostC IP地址所对应的值。然后重启NFS服务:

```
# /etc/init.d/nfs-kernel-server restart
# /etc/init.d/idmapd restart
```

5. 在所有的计算节点中, 启用共享目录的'execute/search'属性, 以允许qemu在此目录内使用镜像。在所有的节点中, 运行下面的命令:

```
$ chmod o+x NOVA-INST-DIR/instances
```

6. 在 HostB 和 HostC中配置NFS, 添加下面几行内容到文件/etc/fstab:

```
HostA:/NOVA-INST-DIR/instances nfs4 defaults 0 0
```

确保你可以挂载导出的目录:

```
$ mount -a -v
```

检查HostA已经抛出NOVA-INST-DIR/instances/" 目录:


```
$ ls -ld NOVA-INST-DIR/instances/
drwxr-xr-x 2 nova nova 4096 2012-05-19 14:34 nova-install-dir/instances/
```

在HostB and HostC执行相同的检查，尤其要注意的是权限(计算服务用户可写)：

```
$ ls -ld NOVA-INST-DIR/instances/
drwxr-xr-x 2 nova nova 4096 2012-05-07 14:34 nova-install-dir/instances/
```

```
$ df -k
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/sda1        921514972  4180880 870523828  1% /
none            16498340    1228 16497112  1% /dev
none            16502856      0 16502856  0% /dev/shm
none            16502856    368 16502488  1% /var/run
none            16502856      0 16502856  0% /var/lock
none            16502856      0 16502856  0% /lib/init/rw
HostA:          921515008 101921792 772783104 12% /var/lib/nova/instances (<--- this line is
important.)
```

- 更新libvirt的配置以实现安全的调用。这些方法通过TCP来远程访问，但是这里篇幅所限，就不提及了。

- SSH隧道到libvirtd的 UNIX 套接字
- libvirtd TCP 套接字，基于 GSSAPI/Kerberos的认证+数据加密
- libvirtd TCP 套接字，基于TLS的加密和x509客户端证书的认证
- libvirtd TCP 套接字，基于TLS的加密和Kerberos的认证

重启 libvirt。在你运行命令之后，确保libvirt已经重启成功：

```
# stop libvirt-bin && start libvirt-bin
$ ps -ef | grep libvirt
root 1145 1 0 Nov27 ? 00:00:03 /usr/sbin/libvirtd -d -l
```

- 配置你的防火墙以允许libvirt在节点之间通信。

默认情况下，libvirt监听的是TCP端口16509，且临时的TCP范围是从49152到49261用于KVM的通信。基于所选择的TCP安全远程访问，请小心打开的端口，且要理解谁可以访问。关于libvirt使用的端口更多信息，请参阅 [libvirt 文档](#)。

- 你现在可以为在线迁移配置属性了。在大多数情况下，你不需要配置任何的属性。下面的表仅针对高级用户。

表 4.6. Description of live migration configuration options

配置属性=默认值	描述
[DEFAULT]	
live_migration_retry_count = 30	(IntOpt) Number of 1 second retries needed in live_migration
[libvirt]	
live_migration_bandwidth = 0	(IntOpt) Maximum bandwidth to be used during migration, in Mbps
live_migration_flag = VIR_MIGRATE_UNDEFINE_SOURCE, VIR_MIGRATE_PEER2PEER, VIR_MIGRATE_LIVE, VIR_MIGRATE_TUNNELLED	(StrOpt) Migration flags to be set for live migration

配置属性=默认值	描述
live_migration_uri = qemu+tcp://%/system	(StrOpt) Migration target URI (any included "%s" is replaced with the migration target hostname)

激活真实的在线迁移

截至Kilo的发布，计算服务默认并不使用libvirt的在线迁移。要启用它的话，添加下面几行内容到nova.conf 中[libvirt]一节：

```
live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE,
VIR_MIGRATE_TUNNELLED
```

比Kilo更早的版本中，计算服务默认不使用libvirt的在线迁移是因为有迁移进程不会终止的bug。这尤其会发生在客户操作系统使用的块存储块于其迁移的过程。

块迁移

配置KVM的块迁移和上面的配置“共享存储”一节[93]非常的类似，不同的是，NOVA-INST-DIR/instances是在每台主机的本地，无须贡献，也无须一致。也不需要配置NFS的服务和客户端。



注意

- 要使用块迁移，必须在在线迁移的命令中加上参数--block-migrate。
- 块在线迁移。但诸如CD-ROM 和配置驱动 (config_drive)之类的只读设备是无法实现的。
- 在块迁移中临时的驱动器是要在整个网络中做复制的，所以迁移实例会给I/O负载带来很重的负担，在极端的情况下，如驱动器写入快于网络的数据复制的话，迁移将是无限期来完成的。

XenServer

共享存储

先决条件

- 兼容 XenServer hypervisors. 更多信息, 请参阅 XenServer 管理员手册的章节[创建资源池的需求](#)。
- 共享存储。一个NFS抛出目录，对于所有的XenServer主机都可见。



注意

至于所支持动NFS版本，请参考XenServer 系统管理员手册章节 [NFS VHD](#)。

要使用基于XenServer hypervisor的共享存储在线迁移的话，主机必须加入XenServer池。要创建此池的话，必须基于特定的元数据创建主机的聚合，此元数据是通过XAPI插件来用于建立池的。

基于XenServer hypervisors来使用共享存储，实现在线迁移

1. 给主的XenServer添加一NFS VHD 存储，然后设置它作为默认的存储仓库。更多信息，请参阅XenServer 管理员指南的 NFS VHD。

2. 配置所有的计算节点使用默认的存储仓库(sr) 用于池的操作。在所有的计算节点中的配置文件nova.conf添加下面几行内容：

```
sr_matching_filter=default-sr:true
```

3. 创建一个主机聚合。此命令会创建聚合并返回一个包含新的聚合的ID的表：

```
$ nova aggregate-create POOL_NAME AVAILABILITY_ZONE
```

添加元数据到并集，然后标记其为一个hypervisor池：

```
$ nova aggregate-set-metadata AGGREGATE_ID hypervisor_pool=true
```

```
$ nova aggregate-set-metadata AGGREGATE_ID operational_state=created
```

将第一台计算节点添加到聚合：

```
$ nova aggregate-add-host AGGREGATE_ID MASTER_COMPUTE_NAME
```

主机现在时XenServer池的一部分了。

4. 添加主机到池中：

```
$ nova aggregate-add-host AGGREGATE_ID COMPUTE_HOST_NAME
```



注意

所添加的计算节点和主机将会关闭以加入到XenServer池。如果其中有的计算节点处于运行状态或休眠状态，操作就会失败。

块迁移

先决条件

- 兼容的 XenServer hypervisors。hypervisor必须支持存储XenMotion特性。请参阅你的XenServer文档，从而确保你的版本支持此特性。



注意

- 要使用块迁移，必须在在线迁移的命令中加上参数--block-migrate。
- 块迁移仅能工作在EXT本地存储仓库下，且服务器必须没有任何的挂接的卷。

迁移实例

此章讨论了如何将实例从一台OpenStack计算节点迁移到另外的OpenStack计算节点。

在开始迁移之前，请重新阅读[配置迁移](#)一节的内容。



注意

尽管命令nova调用了live-migration，默认的计算服务配置属性，仍然会在迁移之前将实例休眠。更多信息，请参阅OpenStack 配置参考的 [配置迁移](#) 一节。

迁移实例

1. 检查要迁移的实例ID：

```
$ nova list
```

ID	Name	Status	Networks
d1df1b5a-70c4-4fed-98b7-423362f2c47c	vm1	ACTIVE	private=a,b,c,d
d693db9e-a7cf-45ef-a7c9-b3ecb5f22645	vm2	ACTIVE	private=e,f,g,h

2. 检查实例所关联的信息，在此例中，vm1 运行在HostB上:

```
$ nova show d1df1b5a-70c4-4fed-98b7-423362f2c47c
```

Property	Value
OS-EXT-SRV-ATTR:host	HostB
flavor	m1.tiny
id	d1df1b5a-70c4-4fed-98b7-423362f2c47c
name	vm1
private network	a,b,c,d
status	ACTIVE

3. 为实例选择所要迁移到的计算节点。在此例中，我们将会将实例迁移到HostC，因为HostC运行了nova-compute：

```
# nova service-list
```

Binary	Host	Zone	Status	State	Updated_at	Disabled Reason
nova-consoleauth	HostA	internal	enabled	up	2014-03-25T10:33:25.000000	-
nova-scheduler	HostA	internal	enabled	up	2014-03-25T10:33:25.000000	-
nova-conductor	HostA	internal	enabled	up	2014-03-25T10:33:27.000000	-
nova-compute	HostB	nova	enabled	up	2014-03-25T10:33:31.000000	-
nova-compute	HostC	nova	enabled	up	2014-03-25T10:33:31.000000	-
nova-cert	HostA	internal	enabled	up	2014-03-25T10:33:31.000000	-

4. 检查 HostC 拥有足够的资源来进行迁移：

```
# nova host-describe HostC
```

HOST	PROJECT	cpu	memory_mb	disk_gb
HostC	(total)	16	32232	878
HostC	(used_now)	13	21284	442
HostC	(used_max)	13	21284	442
HostC	p1	13	21284	442
HostC	p2	13	21284	442

- cpu: CPU 数量
- memory_mb: 内存总量，按 MB 算
- disk_gb: NOVA-INST-DIR/instances 空间的总量，以GB计算

在此表中，第一行展现的是物理服务器资源的总量。第二行展示的是当前已经使用的资源。第三行显示了最大可以使用的资源。第四行以及下面所显示的是每个项目可用的资源。

5. 迁移实例使用命令 `nova live-migration` :

```
$ nova live-migration SERVER HOST_NAME
```

在此例中，SERVER 可以是实例的ID，也可以是实例的名称。另外一个例子：

```
$ nova live-migration d1df1b5a-70c4-4fed-98b7-423362f2c47c HostC
Migration of d1df1b5a-70c4-4fed-98b7-423362f2c47c initiated.
```



警告

当在Icehouse和Juno的计算节点之间进行在线迁移来平衡负载的话，可能会引发数据丢失，因为libvirt在版本3.32之前对于基于共享块存储的在线迁移是有问题的(丢过数据)。此问题将RPC API更新到4.0后解决了。

6. 使用`nova list`来检查实例已经成功的迁移了。如果实例仍然运行在HostB上，检查源主机和目标主机的`nova-compute` 和 `nova-scheduler`) 日志文件来查找原因。

配置远程终端访问

为了提供一个远程的控制台或远程的桌面来访问虚拟机，请使用 VNC 或 SPICE HTML5，可以通过 OpenStack 仪表盘或命令行访问。最佳的实践是选择一个或另一个来运行。

VNC 终端代理

VNC 代理是一个 OpenStack 组件，允许计算服务用户通过 VNC 客户端访问它们的实例。



注意

web 代理控制台 URL 不支持低于 2.7.4 版本的 python 的 websocket 协议方案 (`ws://`)。

VNC终端连接工作流程如下：

1. 用户连接上API，然后获取到`access_url`，如`http://ip:port/?token=xyz`。
2. 用户将此URL粘贴到浏览器或者是当作客户端的参数。
3. 浏览器或客户端连接到代理。
4. 代理和`nova-consoleauth`通信，为用户授权令牌，然后映射令牌到实例所在的VNC服务的`private`主机和端口。

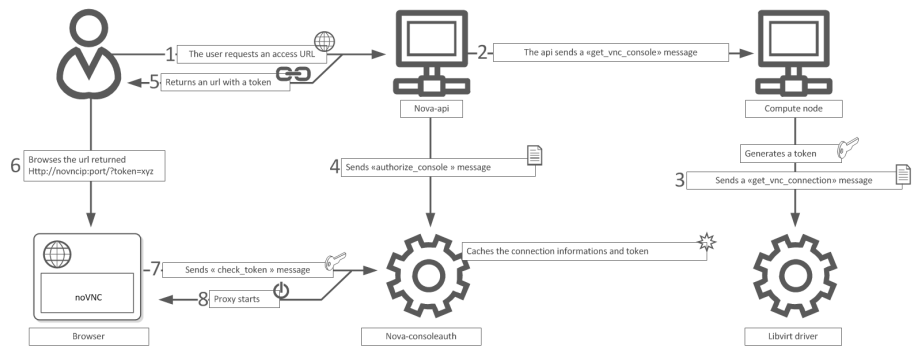
计算主机指定代理的地址，这要通过配置文件`nova.conf`的`vncserver_proxycient_address`属性指定。这样的话，VNC代理的工作就相当于连接公有网络和私有主机网络的桥梁。

5. 代理发起连接到VNC服务，并保持连接知道会话结束。

代理也在WebSocket之上为VNC协议提供隧道，以让noVNC客户端能够和VNC服务通信。通常，VNC代理是：

- 客户端所在的公有网络和VNC服务所在的私有网络之间的桥梁。
- 协调令牌认证
- 透明的处理特定Hypervisor的连接是为了提供一致的客户体验。

图 4.7. noVNC 进程



关于 nova-consoleauth

客户端和代理均会利用一共享服务来管理令牌授权，它叫做nova-consoleauth。此服务运行在代理所在服务器。在集群配置中单个nova-consoleauth 服务可运行多个代理。

请不要讲共享服务nova-consoleauth和nova-console相混淆，nova-console是特定的Xen API服务，多数的VNC代理架构不再使用它了。

典型部署

一个典型的配置有下列组件：

- nova-consoleauth进程。通常运行在控制器节点。
- 一个或多个nova-novncproxy 服务。支持基于浏览器对noVNC客户端。对于简单的部署，此服务通常和nova-api运行在同台主机中，因为nova-api也是作为公有网络和私有计算主机网络之间的桥梁。
- 一个或多个nova-xvpvncproxy服务。支持这里所讨论的特定的Java客户端。对于简单的部署来说，此服务通常和nova-api运行在同台主机中，因为nova-api也是作为公有网络和私有计算主机网络之间的桥梁。
- 一个或多个计算节点。这些计算节点必须被正确的配置，如下所示：

VNC 配置属性

要定制VNC终端，在文件nova.conf中使用下列配置属性：

表 4.7. Description of VNC configuration options

配置属性=默认值	描述
[DEFAULT]	

配置属性=默认值	描述
daemon = False	(BoolOpt)成为守护进程(后台进程)
key = 无	(StrOpt)SSL键值文件(从证书中隔离)
novncproxy_host = 0.0.0.0	(StrOpt)监控入口请求的主机
novncproxy_port = 6080	(IntOpt)监听入口请求的端口
record = False	(BoolOpt) Record sessions to FILE.[session_number]
source_is_ipv6 = False	(BoolOpt) 源是ipv6
ssl_only = False	(BoolOpt)不允许非加密连接
web = /usr/share/spice-html5	(StrOpt)在在同样地端口运行websever。服务器文件在目录DIR。
[vmware]	
vnc_port = 5900	(IntOpt)VNC启动端口
vnc_port_total = 10000	(IntOpt) Total number of VNC ports
[vnc]	
激活 = True	(BoolOpt) 启用VNC相关特性
keymap = en-us	(StrOpt) Keymap for VNC
novncproxy_base_url = http://127.0.0.1:6080/vnc_auto.html	(StrOpt) Location of VNC console proxy, in the form "http://127.0.0.1:6080/vnc_auto.html"
vncserver_listen = 127.0.0.1	(StrOpt) vncsever监听的IP地址
vncserver_proxycient_address = 127.0.0.1	(StrOpt)代理客户端(比如nova-xvpvncproxy)应该连接的地址。
xvpvncproxy_base_url = http://127.0.0.1:6081/console	(StrOpt)nova xvp VNC控制台代理的位置，格式为"http://127.0.0.1:6081/console"



注意

要支持[在线迁移](#)，你不可以为vncserver_listen指定特定的IP地址，因为在目标主机中不存在这样一个IP地址。



注意

- vncserver_proxycient_address默认是127.0.0.1，它是计算主机的地址，当连接到实例的vncservers时计算所指示代理所使用的。
- 对于一体化的XenServer domU的部署，设置此值为169.254.0.1。
- 对于多主机的XenServer domU部署来说，在相同的网络设置一个dom0管理IP作为代理。
- 对于多主机libvirt部署来说，在相同的网络设置一个主机管理IP作为代理。

nova-novncproxy (noVNC)

你必须安装 noVNC软件包，其包含了nova-novncproxy服务。以root身份运行下面命令：

```
# apt-get install nova-novncproxy
```

在初始化安装时服务自动启动。

要重启服务，运行：

```
# service nova-novncproxy restart
```

应在nova.conf文件中指出配置属性参数，其包括消息队列服务地址和凭证。

默认情况下，nova-novncproxy 绑定到 0.0.0.0:6080。

要连接服务到你的计算部署环境中，在你的nova.conf文件中添加以下配置属性：

- vncserver_listen=0.0.0.0

指定VNC服务须绑定的地址。确保它已经分配给了计算节点的网卡。这个地址是一个域所使用的文件。

```
<graphics type="vnc" autoport="yes" keymap="en-us" listen="0.0.0.0"/>
```



注意

要使用在线迁移，须使用地址0.0.0.0。

- vncserver_proxyclient_address=127.0.0.1

这是计算主机的地址，当连接到实例的vncservers时计算所指示代理所使用的。

关于通过VNC访问虚拟机的常见问题

- 问:nova-xvpvncproxy 和 nova-novncproxy之间有何区别？

答：nova-xvpvncproxy，由OpenStack计算提供，是一个支持简单Java客户端的代理。nova-novncproxy使用noVNC来通过web浏览器提供VNC的支持。

- 问：我打算在OpenStack仪表盘中支持VNC。有哪些服务时我需要的？

答：你需要nova-novncproxy, nova-consoleauth，以及正确配置的计算节点。

- 问：当我使用nova get-vnc-console或者是在OpenStack仪表盘中点击VNC项，它挂起了，为什么？

答：确保你运行了nova-consoleauth(以及 nova-novncproxy)。代理依赖nova-consoleauth来验证令牌，然后等待它们的应答，知道超时。

- 问：我的VNC代理在我的一体化测试机中工作正常，但是到了多主机的环境中就不工作了。为什么？

答：在一体机安装中默认的属性工作正常，但是一旦你切换到集群环境也必须将你的计算节点做出变更。这里以实例说明，希望你的环境也有两台服务器：

```
PROXYSERVER (public_ip=172.24.1.1, management_ip=192.168.1.1)
COMPUTESERVER (management_ip=192.168.1.2)
```

nova-compute配置文件必须设置下面的值：

```
# These flags help construct a connection data structure
vncserver_proxyclient_address=192.168.1.2
novncproxy_base_url=http://172.24.1.1:6080/vnc_auto.html
xvpvncproxy_base_url=http://172.24.1.1:6081/console

# This is the address where the underlying vncserver (not the proxy)
# will listen for connections.
vncserver_listen=192.168.1.2
```




注意

novncproxy_base_url 和 xvpvncproxy_base_url 使用一个公有IP；此URL最终会返回给客户端，客户端是无法访问你的私有网络的。你的PROXYSERVER必须能够访问vncserver_proxyclient_address，因为这里VNC连接代理的地址。

- 问：我的noVNC无法在最近的web浏览器版本下工作。为什么？

答：确保你已经安装了python-numpy，此软件包时WebSocket协议新版本所需要的(HyBi-07+)。

- 问：我应该如何在OpenStack仪表盘中调整VNC窗口大小？

答：这些值固定的写入了Django HTML模版中了。要修改它们，编辑_detail_vnc.html模版文件。此文件的位置在不同的发型版中还不一样。在Ubuntu 14.04中，此文件的路径是/usr/share/pyshared/horizon/dashboards/nova/instances/templates/instances/_detail_vnc.html。

修改 width 和 height属性，如下：

```
<iframe src="{ { vnc_url } }" width="720" height="430"></iframe>
```

- 问：为什么noVNC会出现验证错误：原有的协议头文件不匹配到连接失败。为什么？

答：确保 base_url的配置和TLS的设置一致。如果你使用https终端连接，确保明确的在运行nova-novncproxy中设置了novncproxy_base_url的值。

SPICE 控制台

OpenStack Compute supports VNC consoles to guests. The VNC protocol is fairly limited, lacking support for multiple monitors, bi-directional audio, reliable cut-and-paste, video streaming and more. SPICE is a new protocol that aims to address the limitations in VNC and provide good remote desktop support.

SPICE support in OpenStack Compute shares a similar architecture to the VNC implementation. The OpenStack dashboard uses a SPICE-HTML5 widget in its console tab that communicates to the nova-spicehtml5proxy service by using SPICE-over-websockets. The nova-spicehtml5proxy service communicates directly with the hypervisor process by using SPICE.

VNC must be explicitly disabled to get access to the SPICE console. Set the vnc_enabled option to False in the [DEFAULT] section to disable the VNC console.

Use the following options to configure SPICE as the console for OpenStack Compute:

表 4.8. Description of SPICE configuration options

配置属性=默认值	描述
[spice]	
agent_enabled = True	(BoolOpt) 启用 spice 客户代理支持
激活 = False	(BoolOpt) 启用spice相关特性

配置属性=默认值	描述
html5proxy_base_url = http://127.0.0.1:6082/spice_auto.html	(StrOpt) Location of spice HTML5 console proxy, in the form "http://127.0.0.1:6082/spice_auto.html"
html5proxy_host = 0.0.0.0	(StrOpt) 监控入口请求的主机
html5proxy_port = 6082	(IntOpt) 监听入口请求的端口
keymap = en-us	(StrOpt) spice键映射
server_listen = 127.0.0.1	(StrOpt) 实例spice 服务器监听的 IP地址
server_proxyclient_address = 127.0.0.1	(StrOpt) 代理客户端(比如nova-spice html5代理)应该连接的地址。

配置计算服务组

计算服务必须知道每台计算节点的状态才能很好的去管理和使用它们。这包括的事件诸如用户启动一个新的虚拟机，调度器发送给运行的节点一个请求，或者是查询服务组 API来决定节点是否在线。

当计算的线程随着nova-compute守护进程启动而运行时，它会调用join应用程序接口来加入到计算组。任何的服务(例如调度器)都可以查询组的成员以及其节点的状态。在内部，ServiceGroup的客户端驱动自动更新计算线程的状态。

数据库服务组驱动

默认情况下，计算服务使用数据库的驱动来跟踪节点是在线的。在计算的线程中，此驱动会隔一段时间发送命令db update到数据库，称“I'm OK”，且打上时间戳。计算服务使用预先定义的超时(service_down_time)来决定节点已经失效。

此驱动有局限性，其是否出问题取决于你的环境。如果太多的计算线程节点需要被检查，数据库就承受巨大的负载，那就可能引起超时，那么对于一个正常的节点就会被认为是失效的。默认情况下，超时的值为60秒，减少超时值在某些情况下有用，但是这必然导致数据库被频繁访问，这又导致给数据库带来了压力。

数据库中的数据既有暂时性的(诸如节点是否是在线的)也有持久性的(诸如虚拟机的属主)。基于服务组的抽象，计算服务可以区别对待每种类型。

ZooKeeper 服务组驱动

ZooKeeper服务组驱动通过使用ZooKeeper临时节点来工作。ZooKeeper不像数据库，它是一分布式系统，可将负载分发到不同的服务器。在计算线程节点中，驱动可以建立ZooKeeper会话，然后在组目录创建临时的znode。临时的znode和会话有着同样的生命周期。如果一个计算节点或nova-compute守护进程崩溃了，又或者是节点之间的网络出现了问题，ZooKeeper服务重新选举了，那么临时节点都会自动的移除。驱动可以在组目录中运行ls命令来给到组的成员。

ZooKeeper的驱动需要ZooKeeper服务和客户端的库。设置ZooKeeper服务不在本书的范围之内(更多信息，请参阅 [Apache Zookeeper](#))。这些客户端的Python库必须安装在每台计算节点中：

python-zookeeper	Zookeeper官方的Python绑定
evzookeeper	此库让绑定工作在基于eventlet线程模式下。

此例设定ZooKeeper服务器的地址和端口是192.168.2.1:2181, 192.168.2.2:2181, 以及192.168.2.3:2181。

每台运行ZooKeeper驱动力的节点都需要在文件/etc/nova/nova.conf中配置这些值：

```
# Driver for the ServiceGroup service
servicegroup_driver="zk"

[zookeeper]
address="192.168.2.1:2181,192.168.2.2:2181,192.168.2.3:2181"
```

要定制计算服务组，使用这些配置项来设置：

表 4.9. Description of Zookeeper configuration options

配置属性=默认值	描述
[zookeeper]	
address = 无	(StrOpt) Zookeeper 服务组服务的地址格式为 host1:port,host2:port,host3:port
recv_timeout = 4000	(IntOpt) zk 会话的 recv_timeout 参数
sg_prefix = /servicegroups	(StrOpt) Zookeeper 用于保存临时节点的前缀
sg_retry_interval = 5	(IntOpt) 等待重新尝试加入会话的时间秒数

Memcache服务组驱动

memcache 服务组驱动使用了memcache，一个分布式的内存对象缓存系统，用于提升站点到性能。更多细节，请参阅memcached.org。

要使用memcache驱动，必须安装memcache。或许已经像OpenStack对象存储和OpenStack面板那样早已经安装过了。如果需要安装memcache，请参阅[OpenStack 安装指南](#)都详细说明。

这些文件 /etc/nova/nova.conf所需要的值须在每台memcache驱动的节点上配置：

```
# Driver for the ServiceGroup service
servicegroup_driver="mc"

# Memcached servers. Use either a list of memcached servers to use for caching (list value),
# or "<None>" for in-process caching (default).
memcached_servers=<None>

# Timeout; maximum time since last check-in for up service (integer value).
# Helps to define whether a node is dead
service_down_time=60
```

安全强化

OpenStack计算可以整合各种第三方的技术来增强安全性。更多信息，请参阅[OpenStack 安全指南](#)。

可信计算池

管理员可以划定一组计算主机作为可信的计算池。可信主机使用了基于硬件的安全特性，比如英特尔可信执行技术(TXT)，来提供额外的安全级别。结合外部独立的，基于Web的远程认

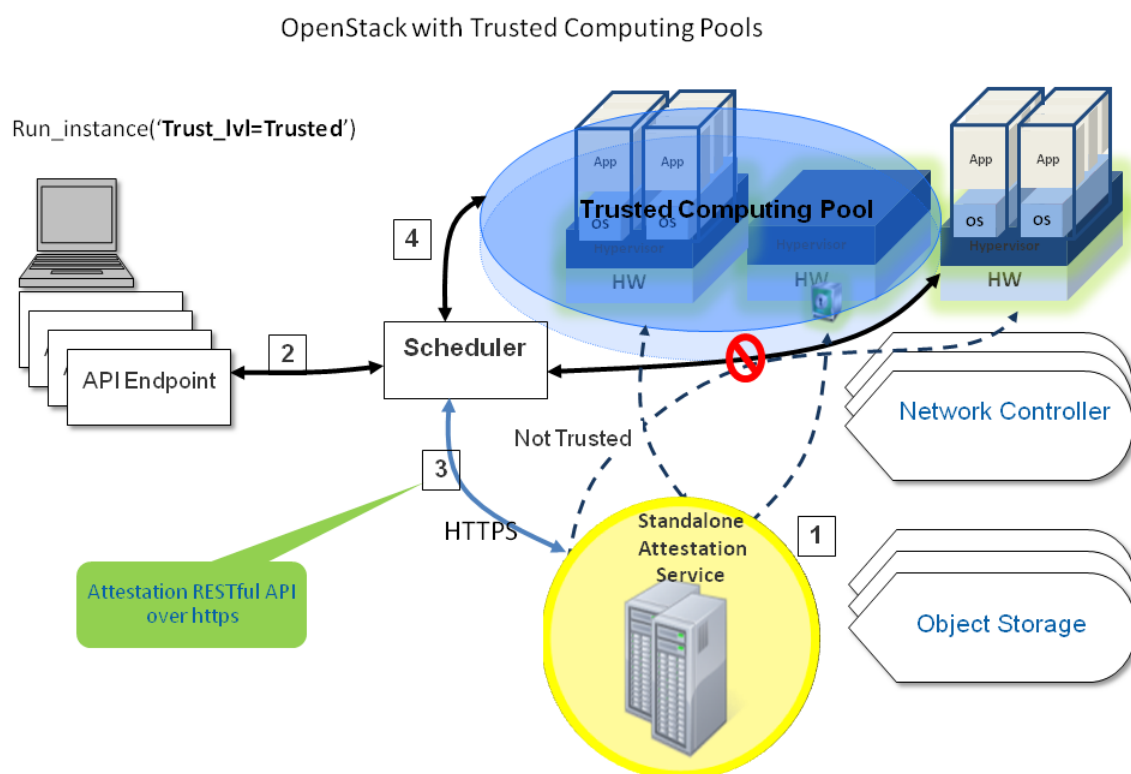
证服务器，云提供商可以保证计算节点上运行通过验证测量的只有软件，并能保证安全的云堆栈。

可信计算池提供了云用户只能请求通过验证的计算节点上所运行服务的能力。

远程认证服务所执行的节点验证类似如下：

1. 激活Intel TXT技术启动计算节点。
2. 计算节点的BIOS，Hypervisor，以及操作系统均是可测量的。
3. 当认证服务器质询计算节点，所测量的数据会被发送到认证服务器。
4. 认证服务器验证测量的内容和一个已知的好的数据库中的内容做比对，然后来决定节点是否值得信任。

关于如何设置一个认证服务器的描述已经超出本书的范围，这里推荐一个开源实现的认证服务，请参考 [Open Attestation](#) 项目。



配置计算节点以使用可信的计算池

1. 为可信计算池启用调度的支持，在文件`/etc/nova/nova.conf`中的`DEFAULT`一节中添加下面几行内容：

```
[DEFAULT]
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler
scheduler_available_filters=nova.scheduler.filters.all_filters
scheduler_default_filters=AvailabilityZoneFilter,RamFilter,ComputeFilter,TrustedFilter
```

- 为认证服务制定连接信息，在文件/etc/nova/nova.conf 中的trusted_computing一节添加下面几行内容：

```
[trusted_computing]
attestation_server = 10.1.71.206
attestation_port = 8443
# If using OAT v2.0 after, use this port:
# attestation_port = 8181
attestation_server_ca_file = /etc/nova/ssl.10.1.71.206.crt
# If using OAT v1.5, use this api_url:
attestation_api_url = /AttestationService/resources
# If using OAT pre-v1.5, use this api_url:
# attestation_api_url = /OpenAttestationWebServices/V1.0
attestation_auth_blob = i-am-openstack
```

在此例中：

服务器	运行证书服务的主机的主机名或IP地址
端口	证书服务的HTTPS端口
server_ca_file	证书文件用来验证认证服务器的身份
api_url	证书服务的 URL 路径
auth_blob	认证服务需要认证点。

- 保存文件，然后重启nova-compute 和 nova-scheduler服务以使设置生效。

要定制可信任的计算池的话，使用如下这些配置属性设置：

表 4.10. Description of trusted computing configuration options

配置属性=默认值	描述
[trusted_computing]	
attestation_api_url = /OpenAttestationWebServices/V1.0	(StrOpt) Attestation web API URL
attestation_auth_blob = 无	(StrOpt) Attestation authorization blob - must change
attestation_auth_timeout = 60	(IntOpt) 证书状态缓存有效期长度
attestation_insecure_ssl = False	(BoolOpt)为证书服务禁用SSL证书校验
attestation_port = 8443	(StrOpt) Attestation server port
attestation_server = 无	(StrOpt) Attestation server HTTP
attestation_server_ca_file = 无	(StrOpt) Attestation server Cert file for Identity verification

指定可信类型

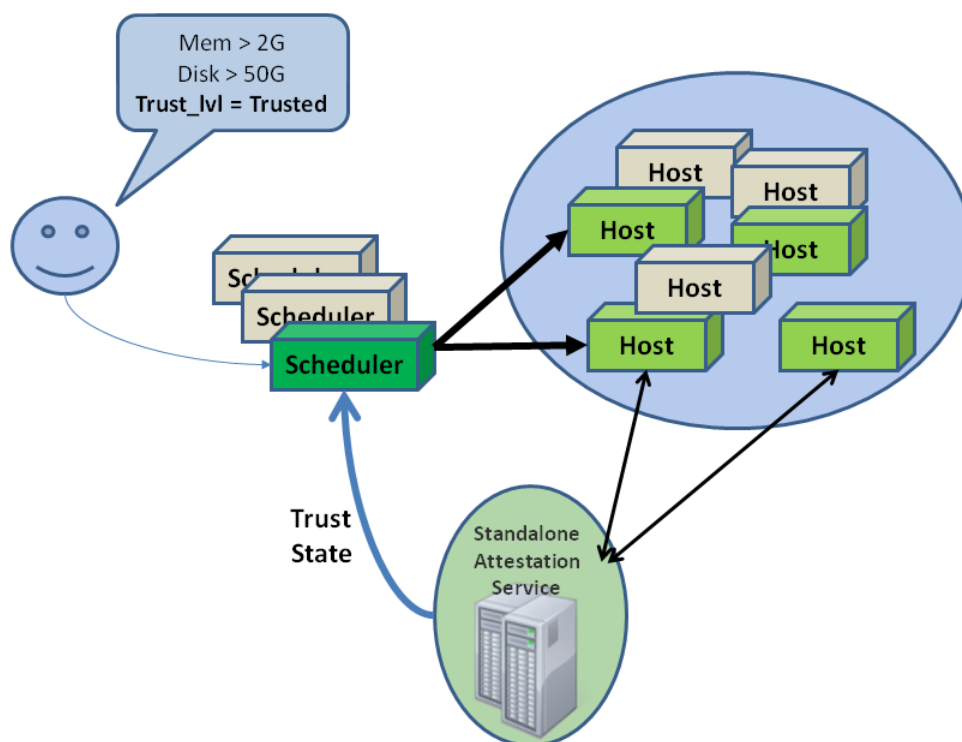
- 使用命令nova flavor-key set命令可以让某个类型组成可信的实例。在此例中，类型m1.tiny就被设置为了可信的：

```
$ nova flavor-key m1.tiny set trust:trusted_host=trusted
```

- 你可以请求你的实例运行在可信的主机中，通过在启动实例的时候指定可信的类型即可：

```
$ nova boot --flavor m1.tiny --key_name myKeypairName --image myImageID newInstanceName
```

图 4.8. 可信任的计算池



加密计算元数据传输

OpenStack支持通过HTTPS来传输加密的计算元数据。在配置文件 `metadata_agent.ini` 启用SSL加密。

激活SSL加密

1. 激活HTTPS协议：

```
nova_metadata_protocol = https
```

2. 决定是否接受计算元数据服务的非安全的SSL连接请求。默认的值是False:

```
nova_metadata_insecure = False
```

3. 指定客户端证书的路径：

```
nova_client_cert = PATH_TO_CERT
```

4. 指定私钥的路径:

```
nova_client_priv_key = PATH_TO_KEY
```

将一个失效的计算节点恢复

如果计算服务是基于共享文件系统来部署的，如果其中一个节点失效了，有多种办法可以快速地从失败中恢复。此章就是讨论手动的恢复。

拯救实例

如果一台云计算节点由于硬件故障或其他原因而故障，您可以拯救实例以使它们重新能够使用。您可以在 `evacuate` 命令中包含目标主机。如果您省略了主机，调度器会决定目标主机。

为了保留用户在云主机磁盘中的数据，您必须在目标主机上配置共享的存储。而且，您需要验证当前虚拟机主机是 `down` 的状态；否则，拯救会以错误失败。

1. 要列出主机并为拯救实例找到一台不同的主机，请执行：

```
$ nova host-list
```

2. 拯救实例。您可以使用 `--password <pwd>` 选项将实例的密码传递给命令。如果您没有指定密码，将会生成一个密码并在命令成功结束时打印出来。下列的命令从一台没有共享存储的主机上拯救一台 `down` 的云主机，指定主机为 `host_b`：

```
$ nova evacuate evacuated_server_name host_b
```

实例从一个新的磁盘启动，但保留了它的配置，包括它的 ID、名称、uid、IP 地址等等。命令返回了密码：

```
+-----+-----+
| Property | Value |
+-----+-----+
| adminPass | kRAJpErnT4xZ |
+-----+-----+
```

3. 要保留用户在所拯救的云主机上的磁盘数据，请以一个共享的文件系统部署 OpenStack Compute。要配置您的系统，请阅读 OpenStack 云管理员指南的 [Configure migrations](#) 部分。下列示例中，密码不变：

```
$ nova evacuate evacuated_server_name host_b --on-shared-storage
```

手动恢复

要恢复一个KVM或libvirt的计算节点，请参阅 [“手动恢复”一节 \[109\]](#)。对于所有其它的hypervisor，使用下面步骤：

1. 在受影响的主机中确认虚拟机，可以使用 `nova list` 和 `nova show` 或 `euca-describe-instances`。例如，此命令展示了关于实例i-000015b9运行在节点 np-rcc54上的信息：

```
$ euca-describe-instances
i-000015b9 at3-ui02 running nectarkey (376, np-rcc54) 0 m1.xxlarge 2012-06-19T00:48:11.000Z 115.146.93.60
```

2. 查询计算服务的数据库来检查主机的状态。此例将一个EC2 API的实例ID转换为一个OpenStack ID。如果使用命令 `nova`，你可以直接替换ID(在此例中截取了部分的输出)：

```
mysql> SELECT * FROM instances WHERE id = CONV('15b9', 16, 10) G;
***** 1, row *****
created_at: 2012-06-19 00:48:11
```



```
updated_at: 2012-07-03 00:35:11
deleted_at: NULL
...
id: 5561
...
power_state: 5
vm_state: shutoff
...
hostname: at3-ui02
host: np-rcc54
...
uuid: 3f57699a-e773-4650-a443-b4b37eed5a06
...
task_state: NULL
...
```



注意

数据库中的证书可以在/etc/nova.conf中找到。

3. 决定了受影响的虚拟机应该迁移到哪台计算主机上，然后运行此数据库命令将虚拟机迁移到新的主机节点：

```
mysql> UPDATE instances SET host = 'np-rcc46' WHERE uuid = '3f57699a-e773-4650-a443-b4b37eed5a06';
```

4. 如果你使用的hypervisor依赖于libvirt(诸如KVM)，更新这些更改到libvirt.xml文件(可在 /var/lib/nova/instances/[instance ID]目录中找到)：

- 更改DHCPSEVER的值为新的计算主机的IP地址。
- 更新 VNC 的IP为 0.0.0.0。

5. 重启虚拟机：

```
$ nova reboot 3f57699a-e773-4650-a443-b4b37eed5a06
```

从一个失效的主机中恢复虚拟机同时需要更新数据库和使用命令nova reboot。然而，如果使用virsh尝试重新创建网络过滤配置还是有问题的话，重启计算服务，或者在数据库中更新 vm_state 和 power_state。

从UID/GID 不匹配中恢复

在一些情况下，计算节点中的文件使用了错误的UID或GID。这通常是运行的OpenStack计算服务使用了共享文件系统或者是自动配置工具所导致，这会引起很多的问题，诸如在线迁移无法执行，无法启动虚拟机等。

此步骤运行在nova-compute主机中，基于KVMhypervisor：

从UID/GID 不匹配中恢复

1. 在所有的主机中的/etc/passwd设置nova UID的为相同的数字(例如，112)。



注意

确保你所选择的UID或GID是其他用户或组没有使用的。

2. 在所有的主机中的/etc/passwd设置libvirt-qemu的为相同的数字(例如，119)。

3. 在所有的主机中的/etc/group设置nova的为相同的数字(例如, 120)。
4. 在所有的主机中的/etc/group设置libvirt组为相同的数字(例如, 119)。
5. 在计算节点上停止服务。
6. 变更所有文件的属主为nova用户或组, 例如:

```
# find / -uid 108 -exec chown nova {} ; # note the 108 here is the old nova UID before the change
# find / -gid 120 -exec chgrp nova {} ;
```

7. 如果需要的话, 为文件 libvirt-qemu重复所有的步骤。
8. 重启服务。
9. 运行命令find来验证所有的文件都使用了正确的标识。

云的灾难恢复

此章涵盖了云在发生灾难后的管理步骤, 以及备份持久的存储卷。备份是强制的, 那怕是超出了灾难的一些场景。

灾难恢复规划(DRP)的定义, 请参阅 http://en.wikipedia.org/wiki/Disaster_Recovery_Plan。

灾难会发生在架构中的多个组件(例如, 磁盘损坏, 断网, 或者是掉电)。在此例中, 下面的组件是被配置的:

- 一个云控制器(nova-api, nova-objectstore, nova-network)
- 一个计算节点 (nova-compute)
- 一个OpenStack块存储所用到的存储区域网络 (SAN) (cinder-volumes)

对于云来说, 最糟糕的灾难莫过于掉电了, 三个组件都怕这个。在掉电之前:

- 从SAN到云控制器创建一个活动的iSCSI会话(用于cinder-volumes的LVM的VG)。
- 从云控制器到计算节点创建一个活动的iSCSI会话(由cinder-volume管理)。
- 为每个卷创建一个iSCSI会话(所以14 EBS的卷需要14个iSCSI会话)。
- 创建从云控制器到计算节点的iptables或ebtables规则。允许来自云控制器访问到运行的实例。
- 保存数据库当前的状态, 当前运行实例的状态, 以及挂接卷(挂载点, 卷ID, 卷状态等), 至少从云控制器到计算节点都需要。

在恢复电力后, 所有的硬件组件都重启过了:

- 从SAN到云的iSCSI不存在了。
- 从云控制器到计算节点的iSCSI会话不存在了。
- 从云控制器到计算节点的iptables和ebtables都重新创建, 这是因为nova-network在启动时配置会重置。
- 没有运行的实例。

注意，实例不会丢失，因为无论是destroy 还是 terminate 都会被调用，实例的文件仍然存在于计算节点中。

- 数据库已经更新过了。

开始恢复



警告

不要再在此过程增加任何额外的步骤，也不要不按照顺序执行。

1. 检查卷和实例之间的目前关系，然后再重新附加。

此信息可从使用nova volume-list来查看。注意nova客户端同样拥有从OpenStack块存储获取卷信息的能力。

2. 更新数据库已清理停滞的状态。使用下面语句为每个卷都执行：

```
mysql> use cinder;
mysql> update volumes set mountpoint=NULL;
mysql> update volumes set status="available" where status <>"error_deleting";
mysql> update volumes set attach_status="detached";
mysql> update volumes set instance_id=0;
```

使用命令 nova volume-list列出所有卷。

3. 使用命令 nova reboot INSTANCE重启实例。



重要

有些实例会完成重启并且可访问，有些或许就停留在了 plymouth阶段。这是意料中的结果，请不要重启第二次。

在该阶段中的实例状态取决于您是否添加了 /etc/fstab 入口给卷。以 cloud-init 包构建的镜像保持在 pending 状态，而其他的跳过了一些错误卷并启动。执行这个步骤是为了让计算服务重启每一台实例，这样就可以保留保存的状态。是否所有的实例都成功启动是不重要的。要了解更多关于 cloud-init 的信息，请阅读 help.ubuntu.com/community/CloudInit。

4. 重新附加卷到其各自的实例上，如果需要，使用 nova volume-attach 命令。这个示例使用了一个列出卷的文件来重新附加它们：

```
#!/bin/bash

while read line; do
    volume=`echo $line | $CUT -f 1 -d " "`
    instance=`echo $line | $CUT -f 2 -d " "`
    mount_point=`echo $line | $CUT -f 3 -d " "`
    echo "ATTACHING VOLUME FOR INSTANCE - $instance"
    nova volume-attach $instance $volume $mount_point
    sleep 2
done < $volumes_tmp_file
```

停止在 plymouth 阶段的实例现在会自动地继续启动和正常开始。之前成功启动的实例现在是可以看到卷的。

5. SSH到实例并重启它们。

如果一些基于服务依赖于卷，或者如果卷在 `fstab` 有一个入口，您现在应该可以重启实例。直接从实例本身重启，而不是通过 `nova`：

```
# shutdown -r now
```

当你计划和执行灾难恢复时，请记住这一点：

- 在文件 `fstab` 中使用参数 `errors=remount` 来防止数据损坏。

这个参数会使系统禁用当 I/O 错误时写入磁盘的功能。这个配置选项应该添加到 `cinder-volume` 服务器（即进行 iSCSI 到 SAN 连接的服务器）和虚拟机的 `fstab` 文件中。

- 不要将 SAN 的磁盘项添加到 `cinder-volume` 的 `fstab` 文件中。

一些系统会在此步挂起，这意味着你丢失了访问到你云控制器的连接。要手动的重新运行会话，请在执行 `mount` 之前运行此命令：

```
# iscsiadm -m discovery -t st -p $SAN_IP $ iscsiadm -m node --target-name $IQN -p $SAN_IP -l
```

- 在您的实例中，如果您在磁盘中有整个 `/home/` 目录，将以用户的 `bash` 文件和 `authorized_keys` 文件保留一个用户目录（而不是空的 `/home` 目录然后在其上映射磁盘）。

此允许你在没有挂接卷的情况下连接到实例，如果你仅允许通过公钥来连接的话。

如果你需要灾难恢复计划(DRP)的脚本，<https://github.com/Razique> 恰好提供了这一版本，其会执行下面的步骤：

- 为实例及其挂接的卷创建了阵列。
- MySQL 数据库已更新。
- 使用 `euca2ools` 重启所有的实例。
- 重新挂接卷。
- SSH连接使用计算服务凭证在每个实例中执行。

脚本包括了 `test mode`，其允许你执行整个序列，但仅针对一个实例。

要重现断掉电源的情况，连接到运行实例的节点，然后关闭 iSCSI 会话。不要使用命令 `nova volume-detach` 来分离卷，手动关闭 iSCSI 会话。下面的例子就是将 15 号的 iSCSI 会话关闭了：

```
# iscsiadm -m session -u -r 15
```

不要忘记 `-r` 标记，否则，你会关闭所有会话。

计算故障排查

计算常见的错误一般是网络配置错误，或者没有拿到正确的凭证。同样，多数 `flat` 网络配置不启用从一个计算节点到另外运行实例的节点的 `ping` 或 `ssh`。另外常见的问题是在 64 位计算节点上尝试运行 32 位的镜像。此章和你展示了如何对计算进行故障排查。

计算服务日志

计算组件为每个服务将日志文件保存在 `/var/log/nova` 中。例如，`nova-compute.log` 是 `nova-compute` 服务的日志。您可以在 `nova.conf` 文件中设置下列选项以将 `nova.log` 模块格式化日志字符串。

- `logging_context_format_string`
- `logging_default_format_string`

如果日志级别设置为 `debug`，您也可以指定 `logging_debug_format_suffix` 来添加额外的格式。要了解更多变量可用的格式设置，请阅读 <http://docs.python.org/library/logging.html#formatter>。

根据配置的设置，您有两个记录 OpenStack Compute 的选项。在 `nova.conf` 中，包含了启用日志记录的 `logfile` 选项。您还可以选择性地设置 `use_syslog = 1`，那么 `nova` 守护进程将记录到 `syslog` 中。

Guru Meditation 报告

Guru Meditation 报告是由计算服务在接收到 `SIGUSR1` 信号时发送的。这份报告是一个通用的错误报告，包括一个完整的服务当前状态报告，并发送到 `stderr`。

例如，如果您使用 `nova-api 2>>/var/log/nova/nova-api-err.log` 将错误输出重定向到 `nova-api-err.log` 中，其进程 ID 为 8675，您可以运行：

```
# kill -USR1 8675
```

这个命令使得了 Guru Meditation 报告被粘贴到 `/var/log/nova/nova-api-err.log` 中。

该报告包含如下小节：

- `Package`：显示关于进程所属的包的信息，包括版本信息。
- `Threads`：显示栈的追踪和进程中的每个线程的线程 ID。
- `Green Threads`：显示每个进程中的绿色线程的栈追踪（绿色线程没有线程 ID）。
- `Configuration`：列出当前进程目前可以通过 `CONF` 对象访问的所有的配置选项。

要了解更多信息，请阅读 [Guru Meditation 报告](#)。

计算组件的常见错误和修复

ask.openstack.org 站点提供了一个询问和回答问题的地方，您也可以将问题标记为频繁提问的问题。该小节描述了一些人们之前发布的错误。缺陷在不断的被修复，因此在线资源是一个获得最新错误和修复的非常好的方式。

证书错误、401 和 403 拒绝错误

丢失证书而引发了 `403forbidden` 错误。请使用以下方法之一解决这个问题：

1. 手动方式。从 ZIP 文件中获取 `novarc` 文件，为防止覆盖，先保存已有的证书，然后手动 `source novarc` 文件。

2. 脚本方式。从项目 ZIP 文件中生成 novarc，然后 source 这个文件。

如果您是第一次运行 nova-api，它会生成证书认证信息，包括 openssl.cnf。如果在这之前启动了 CA 服务，您可能无法创建您的 ZIP 文件。请重启服务。在您的 CA 信息可用后，创建您的 ZIP 文件。

此外，请检查您的 HTTP 代理设置，查看是否是它们引发了 novarc 创建的问题。

实例错误

有时候某个实例显示 pending 或您不能 SSH 到这台实例上。有时候镜像本身是有问题的。例如，如果您使用了扁平管理者网络，您没有 DHCP 服务器，且某些镜像不支持接口注入；您就无法连接它们。这个问题的解决办法是使用一个支持该方法的镜像，如 Ubuntu，它包含了一个正确设置了 FlatManager 网络的 IP 地址。

要对一台实例的可能问题进行故障排除，如处于 spawning 状态的实例，请在 nova-compute 主机上的 /var/lib/nova/instances 目录上检查这些实例，并保证这些文件是存在的：

- libvirt.xml
- 磁盘
- disk-raw
- 内核
- ramdisk
- 实例启动后，console.log

如果有文件丢失、为空或非常小，说明 nova-compute 没有成功地从镜像服务中下载镜像。

此外，检查 nova-compute.log 中的 exceptions。有时它们不会在控制台输出中显示。

接着，在 /var/log/libvirt/qemu 目录下检查实例的日志文件，以查看其是否存在并有一些有用的错误信息。

最后，在实例的 /var/lib/nova/instances 目录下，查看这个命令是否返回了错误：

```
# virsh create libvirt.xml
```

Linux实例的日志输出为空

您可以在仪表盘的 Log 标签或通过 nova console-log 的输出浏览正在运行的实例的日志输出。在某些情况下，正在运行的 Linux 实例的日志输出可能是空的或仅显示一个字符（例如，? 字符）。

如果实例本身没有配置发送输出信息到控制台，那么当计算服务尝试通过一些列控制台获取实例的日志输出时，这个情况会发生。要进行补救，请在实例的引导装载程序中添加下列参数到指定的内核参数中：

```
console=tty0 console=ttyS0,115200n8
```

重启之后，该实例将被配置为发送输出到计算服务中。

重置实例的状态

如果实例保持在中间状态，如 `deleting`，您可以使用 `nova reset-state` 命令来手动重置实例的状态为 `error` 状态。然后您就可以将实例删除。例如：

```
$ nova reset-state c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
$ nova delete c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
```

您也可以使用 `--active` 参数来强制实例回到 `active` 状态而不是 `error` 状态。例如：

```
$ nova reset-state --active c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
```

注入问题

如果实例没有启动或启动非常缓慢，请审查是否为注入文件的原因。

要禁用 `libvirt` 中的注入文件，请在 `nova.conf` 中设置以下选项：

```
[libvirt]
inject_partition = -2
```



注意

如果您没有启用配置驱动，而希望使用户指定的文件在元数据服务器上可用于增加性能并避免在注入失败时的启动失败，您必须禁用注入。

禁用在线快照

如果您使用的 `libvirt` 版本是 1.2.2，您在创建在线快照时可能会面临一些问题。特定版本的 `libvirt` 在多个快照同时创建的负载下进行在线快照时偶尔会失败。

在问题解决之前为了有效地禁用 `libvirt` 在线快照，请设置 `disable_libvirt_livesnapshot` 选项。您可以通过在 `nova.conf` 配置文件中的 `[workarounds]` 小节将其值设置为 `True`，以关闭在线快照机制：

```
[workarounds]
disable_libvirt_livesnapshot = True
```

第 5 章 对象存储

目录

Object存储介绍	117
特性和好处	117
对象存储特性	118
组件	119
环构建器	124
集群架构	127
复制	129
账户清道夫	131
配置基于对象存储的特定租户的镜像位置	131
对象存储监控	132
对象存储的系统管理	135
对象存储故障排查	135

Object存储介绍

OpenStack对象存储(项目代号 swfit)是为创建冗余，可扩展的数据存储的开放源代码软件，使用标准的服务组件的集群来存放petabyte级别的可访问的数据。它是一个长期的存储系统，用于大量静态数据的检索，杠杆，和更新。对象存储使用了分布式的架构，没有中心控制点，提供良好的扩展型，冗余性和高性能。对象可以写入到多个硬件设备，OpenStack软件来负责确保数据的复制和跨集群的整合。存储集群通过增加新到节点来作横向扩展。一旦有节点失效，OpenStack就会以其他活动的节点原来所复制的内容来提供服务。因为OpenStack使用了软件逻辑来确保数据的复制和分布在不同的设备，便宜的商品的硬盘驱动器和服务器可以代替更昂贵的设备来使用。

对象存储是拥有理想的性价比和高扩展的存储。它提供了一个完全分布式的，基于API访问的存储平台，且可以直接集成到应用中，或者用于备份，归档，以及数据留存。

特性和好处

特性	好处
利用旧硬件	没有厂商锁定，低价/GB.
HDD/节点故障无关	自我修复，可靠，数据冗余以在失效时提供保障。
无限的存储	大而扁平的命名空间，高度可扩展的读/写访问，能够直接从存储系统中提供内容。
多维度可扩展性	扩展架构：纵向和横向扩展的分布式存储。备份和归档大量增长的数据也是线性性能。
账户／容器／对象结构	没有嵌套，没有旧有的文件系统：为扩展而优化，它可以扩展到petabytes级别以及亿级的对象。
内建复制 3# + 的数据冗余(相比于RAID的2X)	为高可用而可配置的账户数量，容器数量和对象复制数量。
轻松扩容(不像RAID重置大小)	弹性数据轻松扩展
无中心数据库	高性能，无瓶颈
无须RAID	处理许多小的，随机读取和写入效率

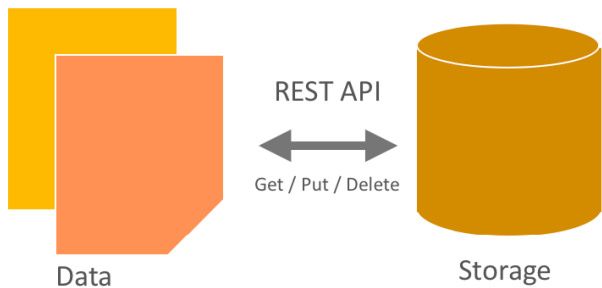
特性	好处
内建管理工具	账户管理：创建，添加，验证，以及删除用户；容器管理：上传，下载，以及验证；监控：容量，主机，网络，日志跟踪，以及集群健康。
驱动审计	检测驱动器故障，在数据损坏之前作出反应
过期对象	用户可以设置一过期时间或一TTL给对象以控制访问
直接对象访问	启动浏览器直接访问内容，类似于控制面板
实时查看客户端的请求	了解用户在请求什么
支持S3 API	所设计的工具支持流行的S3 API。
限制每个帐户的容器	通过用户来限制访问从而控制使用量。
支持 NetApp, Nexenta, SolidFire	使用多种存储系统为块卷提供统一支持。
为块卷提供快照和备份API	用户虚拟机数据的数据保护和恢复。
可用的标准卷API	分离端点和API，从而可以集成其它的计算系统。
和计算集成	与计算完全集成，用于挂接块卷和报告使用情况。

对象存储特性

对象存储的关键特性有：

- 所有的保存在对象存储中的对象都有一个URL。
- 所有对象的保存都复制3#份，在尽可能唯一的区域中，区域的定义可以使一组磁盘，也可以是一组节点，也可以是一组机架，等等。
- 所有的对象都有其自己的元数据。
- 开发者通过RESTful HTTP API来和对象存储系统交互。
- 对象数据可以存放在集群的任何地方。
- 通过增加额外的节点来扩展集群而无序牺牲任何的性能，它允许一个更具成本效益的线性存储扩展比fork-lift升级。
- 数据不必是迁移到一个完全新的存储系统。
- 无宕机任意添加新的节点到集群。
- 失效的节点或磁盘可在无宕机的情况下无缝换出。
- 可运行在工业标准的硬件上，如戴尔，惠普，以及超微。

图 5.1. 对象存储(Swift)



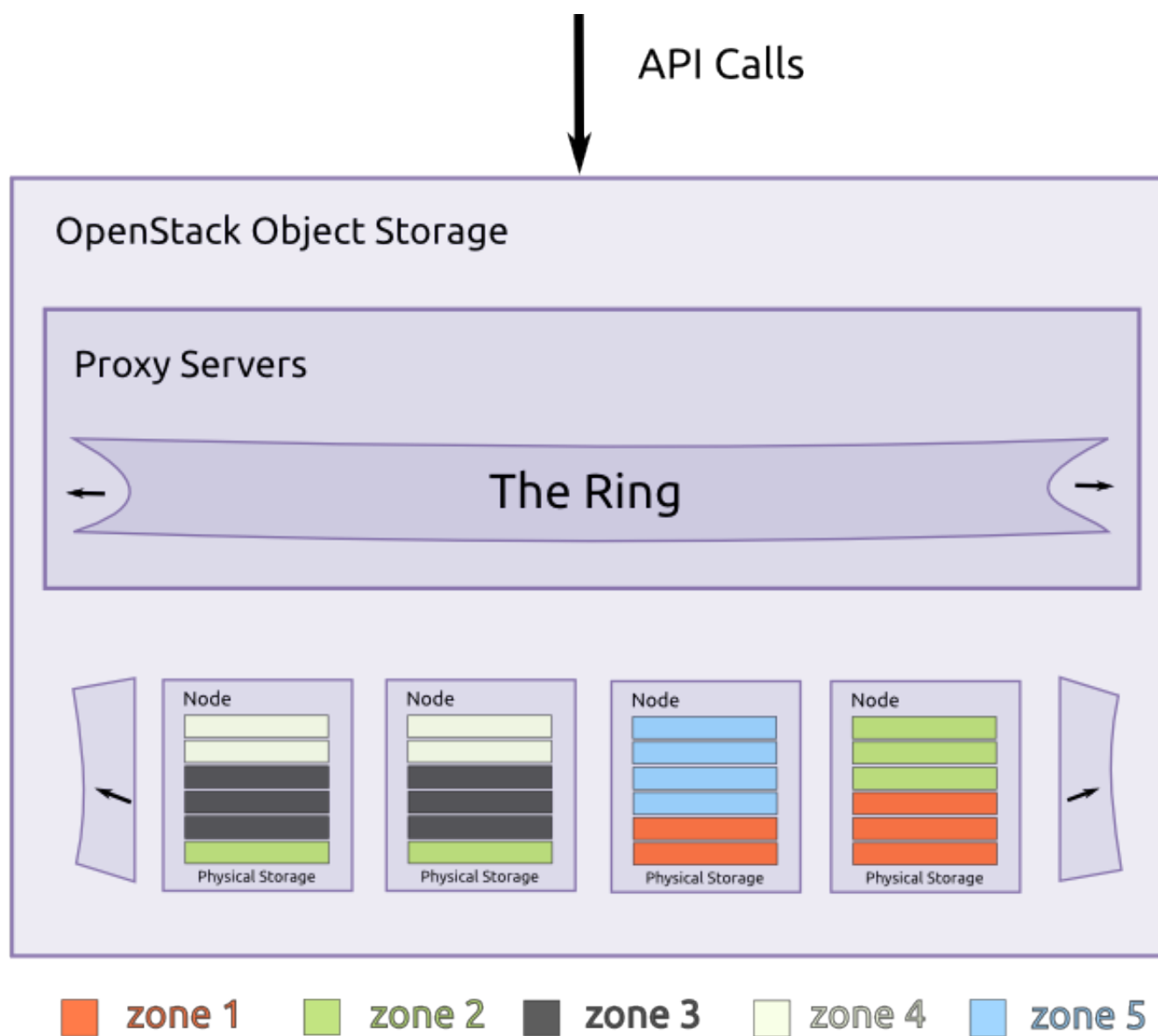
开发者既可以直接写入到Swift API可以使用众多的客户端库，客户端库支持所有存在的流行的编程语言，如Java，Python，Ruby，C#。亚马逊 S3和RackSpace 云文件用户对于OpenStack存储不会有一点陌生。当然，对于一个对象存储系统的新手来说，相比于传统的文件系统而言，就不得不适应一个全新的方法和思考方式。

组件

让对象存储能够交付高可用，高耐久以及高并发的服务的组件有：

- 代理服务器. 掌控所有到来的API请求。
- 环. 映射数据的逻辑名称到指定磁盘的位置。
- 区. 和另外一些区将数据隔离。其中一个区的失效并不会影响到集群中其它区，因为数据是跨区复制的。
- 账户和容器. 每个账户和容器都是独立的数据库，跨集群分布。一个账户数据库包含该账户的容器列表，一个容器数据库包含该容器的对象列表。
- 对象. 数据本身。
- 分区. 一个分区存储对象，账户数据库，以及容器数据库，且协助管理集群中数据的位置。

图 5.2. 对象存储构建块



代理服务

代理服务器是对象存储的公开的接口，且掌控着所有到来的API请求。一旦代理服务接收到一个请求，它会基于对象的URL来决定存储的节点，举例来说，`https://swift.example.com/v1/account/container/object`。代理服务器也协调响应，处理故障，以及协调时间戳。

代理服务器使用了无共享的架构，可以根据项目的负载按需扩展。一个最小化的理应是部署两台代理服务器，是为了冗余。如果其中一个代理服务器挂了，另一个就会接替。

关于代理服务器配置的更多信息，请参阅[配置参考](#)。

环

环代表了存储在磁盘的实体名称和它们物理位置的一个映射。账户，容器和对象都有各自的环。当其它组件需要对一个对象，容器或账户执行任何操作时，它们都需要和对应的环交互，从而获取集群中它们的物理位置。

环使用区，设备，分区和副本来维护此映射。环中的每个分区都会被复制，默认情况下，跨集群复制三份，且分区的位置均存储在映射中由环来维护。环也负责在失效的场景下那个设备不可用。

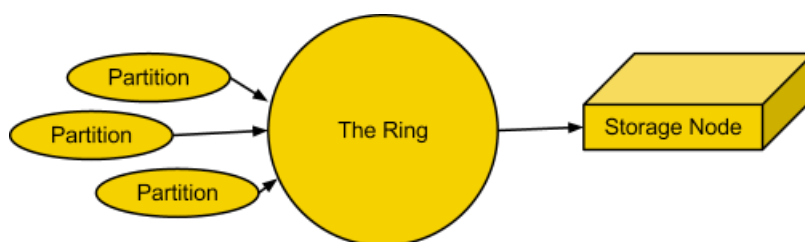
在环中数据可以用区来隔离。每个分区副本保证在不同的区中。一个区可以表示是一个磁盘，一台服务器，一个机架，一台交换机，或者甚至是一个数据中心。

环的分区会将所安装的OpenStack存储中的所有设备平分。当分区需要移动时(举例来说，如果要给集群添加一个设备)，环要确保一次移动最小的分区数量，且一次只能移动一个分区的一个副本。

你可以使用权重来平衡跨集群的磁盘上的分区的分布。这很有用，例如，当集群中使用了不同大小的磁盘。

环被代理服务器和一些后台进程(例如复制)所使用。

图 5.3. 环



这些环都是由外部来管理的，服务进程本身并不会修改环，而是通过其它工具给出一个新的环替代，从而达到修改的目的。

环使用一个来自路径的MD5哈希的可配置的比特数来作为分区的索引，此索引指定设备。来自哈希的比特数是作为分区的平方数的，还有2的分区平方数表明分区的数量。完整的MD5哈希分区环允许集群的其他部分在批项目工作,最终更有效或者至少比单独处理每一项复杂或整个集群。

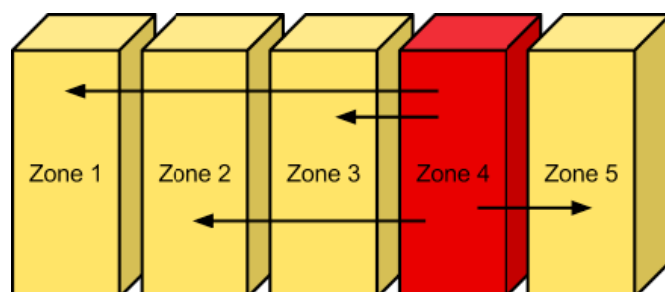
另外一个可配置的值是复制数量，它表明了多少个分区—设备作业组成一个单一的环。对于给定的分区数来说，每个复制的设备都将不会和其它的复制设备在同一个区中。区可以用于设备组，基于物理位置，电源分离，网络分离，或者任何能够在同时提高多份复制的属性。

区

对象存储允许配置区是为了能够有一道隔离失效的界限。如果可能的话，每个数据副本都会驻留在一个区中。在最小的级别下，一个区只有一单个的驱动或一组几个的驱动。如果拥有五台对象存储服务器的话，则每台服务器都表示它本身的区。大型的部署会是一整个机架(或多个机架)的对象服务器，每一个都表示一个区。区域的目标是允许存储服务器的集群即使发生了重大问题也不会丢失数据的所有副本。

正如以前所强调的，对象存储中的一切都会被保存，默认情况下，是三次。Swift会将安置每个副本“尽可能的使之独一无二”来同时确保高可用和可持续性。这就意味着当选择了一个副本位置时，对象存储在一个未使用过的区中选择了一台服务器之前，区中的未使用过的服务器也已经拥有了数据的副本。

图 5.4. 区

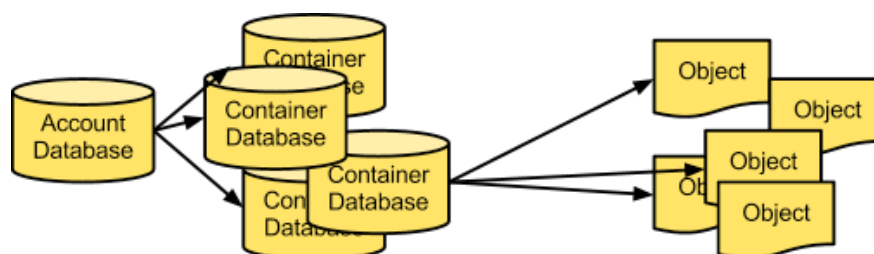


当一个磁盘失效时，副本数据会自动地分发到其它的区，从而确保集群中的数据拥有三份拷贝。

账户和容器

每个账户和容器都是独立的SLIte数据库，它们跨集群分布。一个账户数据库包含该账户的容器列表，一个容器数据库包含该容器的对象列表。

图 5.5. 账户和容器



要保持对对象数据位置的跟踪，系统中的每个账户都有一个数据库，其中参考了它的所有的容器，以及每个容器数据库都参考其每个对象。

分区

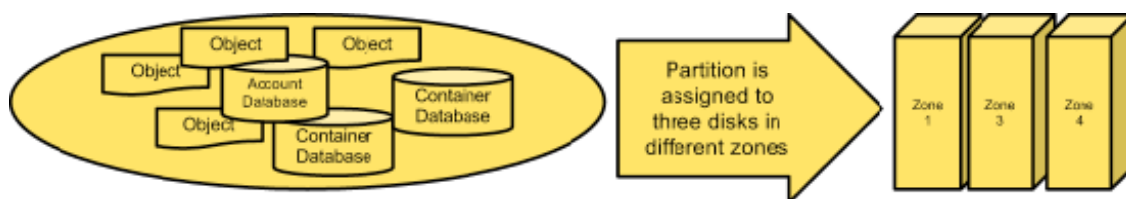
一个分区就是所存储的数据的集合，包括账户数据库，容器数据库，以及对象。分区时复制系统的核心。

想象下，分区就好比在一个巨大的中心仓库中一个个箱子的移动。将单独的小件放到箱子中，系统将箱子视为非常有凝聚力的实体，从而可以在系统中随意的移动。一个箱子可轻松处理许多小的物件。它可以使整个系统少一些可移动的部分。

系统复制器和对象的上传/下载等操作均在分区之上进行。作为系统的可伸缩性，它的行为仍然是可以预测的，因为分区的数量是固定的。

实现分区的概念很简单，就是一个分区就是一个磁盘上的目录，再加上相应的它包含了什么的哈希表。

图 5.6. 分区



复制器

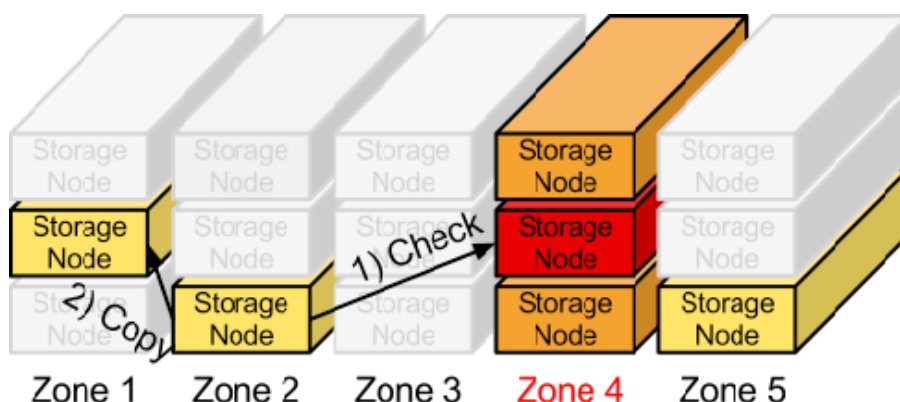
为了保证各个地方的数据都有三份拷贝，复制器会持续的检查每个分区。在每个本地分区，复制器会和另外一个区的拷贝作对比，以查看它们是否有不同。

复制器知道复制需要通过检查哈希。为每个分区创建一个哈希文件，分区包含了每个目录的哈希。会比较每三个的哈希文件。对于给定的分区，每个分区拷贝的哈希文件都会被比较。如果哈希是不一样的，那就是时间复制了，而且目录也需要复制过去。

这是分区派上用场的地方。用系统中更少的东西,做着大量的数据传输(而不是大量的低效的TCP连接,)而且有一个一致的哈希来作比较。

当最新的数据拥有优先级时，集群会产生一致性的行为。

图 5.7. 复制



如果一个区失效了，其中一个节点包含一个副本通知，而且会主动复制数据到切换的位置。

用例

下面一节展示了对对象上传和下载的用例，并介绍了相关组件。

上传

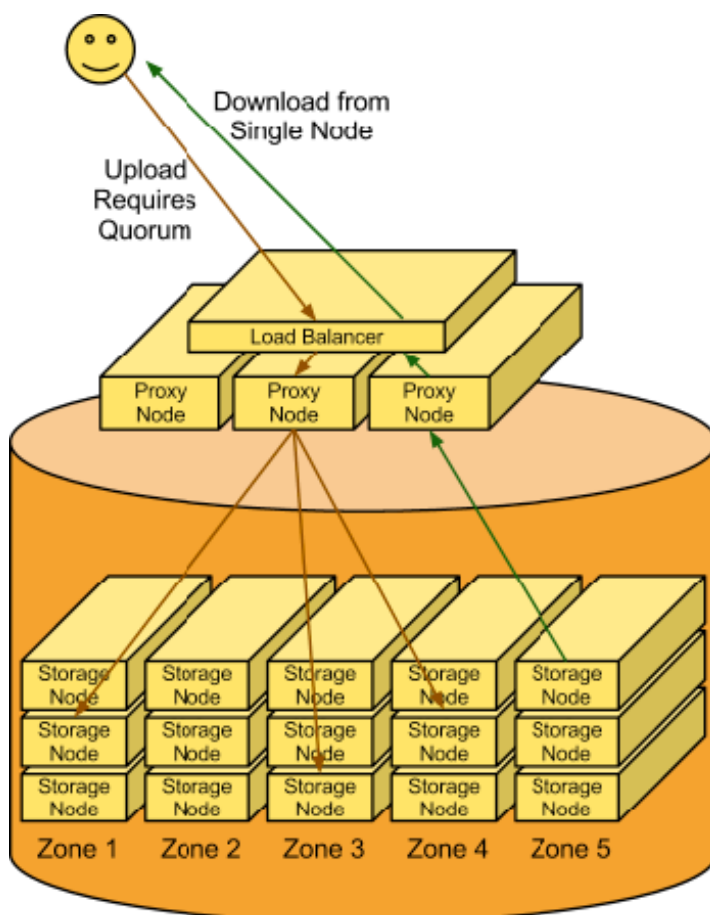
客户端使用REST API 向一个已经存在的容器发起HTTP请求来PUT一个对象。集群接收到这个请求后。首先，系统必须指出数据将去哪里。要完成这步，账户名称，容器名称以及对象名称所有的这些都使用由分区所决定的对象放在哪里。

然后寻找问题中的由环指出的那个存储节点包含的分区。

数据然后会发送到每个存储节点，这些存储节点都是分区所对应的。在客户端收到上传成功的通知之前，必须至少有三分之二的成功写入。

接下来，容器数据库会异步的更新，反映出一个新的对象存入。

图 5.8. 对象存储可用



下载

一个请求以account/container/object形式发送过来了，使用相同的一致性哈希，生成的分区的名称。在环中找到谁是包含分区的存储节点。请求会从存储节点的其中一个去获取对象，如果失败，则请求会切换到另外的节点。

环构建器

使用swift-ring-build套件来构建和管理环。此套件可将分区分配给设备，并且可写入优化过的Python结构用来压缩，串行化磁盘上的文件，从而传输到服务器。服务器进程偶尔检查文件的修改时间，且按需加载环结构的拷贝到内存中。如果你使用了稍旧版本的环，分区子集的三个副本中的一个会出现错误，因为环构建器管理着环的变更。此问题是可以解决的。

环构建器也会保持一份带有环信息的其自身的构建器文件，以及将来构建环所需要的额外的数据。保存这些构建器文件的多个备份是非常重要的。其中一个建议就是在拷贝环文件它们时同时要拷贝构建器文件到其他服务器中。另外一个建议是上传构建器文件到集群内部。如果你丢失了构建器文件，你就得从头开始创建新的环。几乎所有的分区都会分配给不同的设备，因此，几乎所有保存的数据都会复制到新的位置。所以，恢复一个丢失的构建器文件是可能的，但是数据在这段时间是不可以访问的。

环数据结构

环数据结构由三个顶级的字段组成：一个集群中的设备列表，一个设备ID列表的列表，表明了分配给设备的分区，以及一个整型，表明了转换为MD5哈希的bit数，用来计算分区的哈希。

分区分配列表

此是一个设备ID的 `array('H')` 列表。对于每个副本的外层都包含一个 `array('H')` 列表。每个 `array('H')` 长度等于环的分区数。每个 `array('H')` 中的整型数都是上述设备列表的索引。此分区列表是已知内部的环类即 `_replica2part2dev_id`。

所以，要创建一个设备列表字典分配到一个分区，Python的代码类似这样：

```
devices = [self.devs[part2dev_id[partition]] for  
part2dev_id in self._replica2part2dev_id]
```

此代码有些简单，因为它没有将已删除的重复的设备计算在内。如果环拥有的副本多于设备，分区亦会多于每个设备一个副本。

`array('H')` 是用于内存的转换，可能会有数百万的分区。

副本计数

为了支持副本逐渐递增的变化，环可拥有真实的副本数，而且并不局限于一个整数的副本。

一个部分的副本计数是针对环的整体而不是某个单独的分区。它表明每个分区的平均数量的副本。例如，一个副本计数为3.2意思是20%的分区拥有4个副本，而80%的拥有三个副本。

副本计数是可调整达。

例子：

```
$ swift-ring-builder account.builder set_replicas 4  
$ swift-ring-builder account.builder rebalance
```

你必须在全局的分布式集群中重新平衡副本环。这些集群的运维人员通常打算将副本和区域划上等号。因此，当运维人员添加或删除一个区域时，就会同样添加或删除一个副本。删除不需要的副本可节省磁盘的损耗。

你可以逐渐增加副本计数率，但不能不利于集群的性能。

例如：

```
$ swift-ring-builder object.builder set_replicas 3.01  
$ swift-ring-builder object.builder rebalance  
<distributed rings and wait>...  
  
$ swift-ring-builder object.builder set_replicas 3.02  
$ swift-ring-builder object.builder rebalance  
<create distributed rings and wait>...
```

变更在环重新平衡后才生效。因此，如果你打算从3副本更改为3.01，但是不小心敲入了2.01，不会有数据丢失的。

另外，`swift-ring-builder X.builder create`现在可以跟一个副本数量的小数参数了。

分区调整值

分区调整值在内部即是环类的 `_part_shift`。此值用于调整MD5哈希，从而来计算分区上的哪里的数据应驻留。在此过程中只有顶级的4byte哈希用得到。例如，要计算 `/account/container/object` 路径的分区，Python代码类似下面的代码：

```
partition = unpack_from('>I',
md5('/account/container/object').digest())[0]>>
self._part_shift
```

对于 `part_power P` 生成的环，分区调整值是 $32 - P$ 。

构建环

环构建器的过程包括三个主要步骤：

1. 此工具计算分配给每个设备的分区数量是基于设备的权重。举例来说，对于一个设备是2的20次方，那么环就有1,048,576个分区。成千个有此权重的设备每个都需要1,048,576个分区。设备是按照分区的数量来排序的，而且通过初始化的进程来保持顺序。



注意

每个设备会被分配一个随机的决胜值，此值是用于在两个设备都想要同样数量的分区时使用。此决胜值并不在磁盘中保存，所以基于相同的参数创建了两个不同的环会有不同的分区分配。对于可重复的分区分配来说，`RingBuilder.rebalance()` 需要一个可选的种子值，此值使用Python `pseudo-random` 数字生成器生成。

2. 环构建器分配每个分区副本给设备，需要在此点的多数分区尽可能的远离其它副本。环构建器优先分配副本到那些区域中还没有副本的设备。如果没有此种区域，环构建器会搜索不同区的设备或是不同的服务器。如果还是没有找到结果，它就会找没有副本的设备。最后，如果所有的选项都不满足，环构建器就会分配副本到已经分配到最新副本的设备。



注意

环构建器分配多个副本给一个设备的情况，仅在环拥有的设备少于其副本。

3. 当从一个旧的环境构建一个新的环时，环构建器会重新计算每个设备打算要的分区的数量。
4. 构建器不会分配用于重新分配的分区和聚集这些分区，如下所示：
 - 环构建器不会分配已经分配过的分区，即使是这些分区来自任何已经删除的设备，或者是添加这些分区到聚集列表。
 - 环构建器不会分配任何的可以展开更好的耐用性分区副本，以及添加这些分区到聚集列表。
 - 环构建器不会分配哪些来自拥有的分区多于实际需要的设备的随机分区，以及添加这些分区到聚集列表。
5. 环构建器重新分配聚集的分区给设备，通过使用前面所描述的非常类似大方法。

6. 当环构建器重新分配一个副本给一个分区时，环构建器会记录重新分配的时间。在收集分区用于重新分配时环构建器使用此值，所以在一个可配置的时间内不会出现移动两次分区的情形。RingBuilder类通过min_part_hours获知可配置的时间。环构建器忽略此在已经删除的设备上的分区副本的限制，因为移除设备仅在设备失效的情况下才会发生，而且重新分配时仅有的选择。

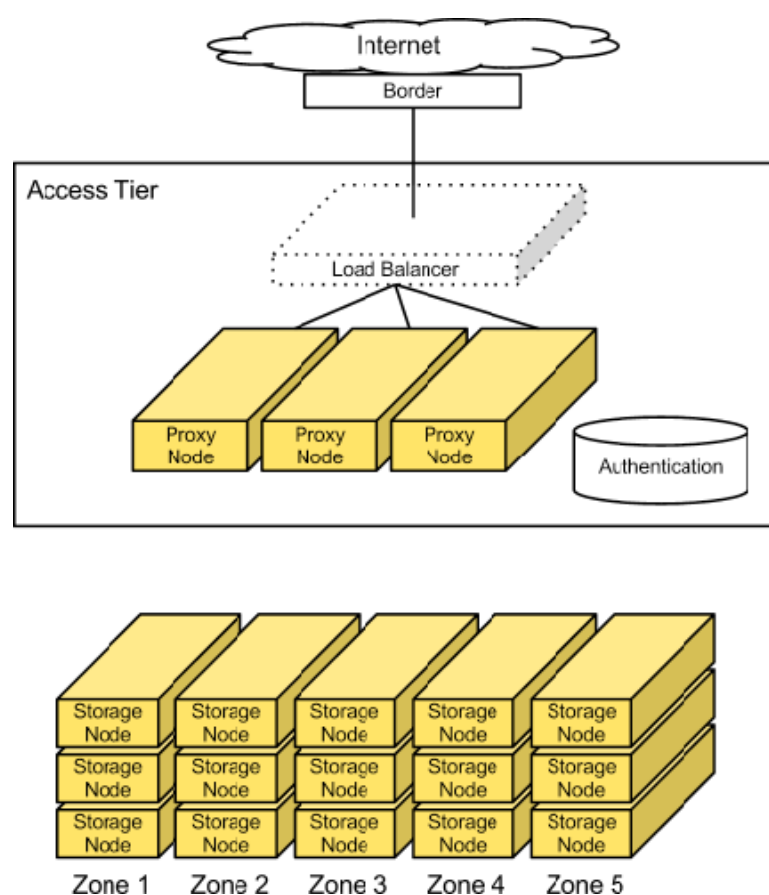
这些步骤并不能保持一直完美，重新平衡一个环由于收集的随机性质分区来用于重新分配。要帮助达到更加平衡的环，重新平衡进程会一再重复直到接近完美(少于1%)或者当平衡在至少1%下无法再进一步(表明了我们可能无法获得最完美的平衡，在大范围没有平衡的区域或者最近移动了太多的分区。)

集群架构

访问层

大型可扩展的部署会将访问层区分出来，而访问层被认为是对象存储系统的中心枢纽。访问层接受来自客户端的API请求，且是移动数据的系统。此层由前端负载均衡器，ssl-终端，以及认证服务组成。它运行着对象存储系统的(分布式)大脑：代理服务进程。

图 5.9. 对象存储架构



因为访问服务都集中在它自己的层，所以你可以横向扩展读写的访问而无须顾忌存储的容量。举例来说，如果一个集群在互联网中，还需要SSL终端，并且对数据访问有很高的要

求，那么你就须部署多个访问服务。然而，如果集群是在私网中，而且目的仅仅是作为归档来用，那么你只需少量访问服务即可。

因为这是HTTP地址的存储服务，你可以在访问层中放入一个负载均衡器。

典型地，应用层使用1U的服务器来组成。这些机器仅使用适量的内存，以及网络I/O密集型的特点。因为这些系统会接受每个外来的API请求，所以你须要给它们部署两块高吞吐量(10GbE)的网卡—其中一块用于“前端”来的API请求，另外一块用于“后端”访问对象存储节点的存放和获取数据。

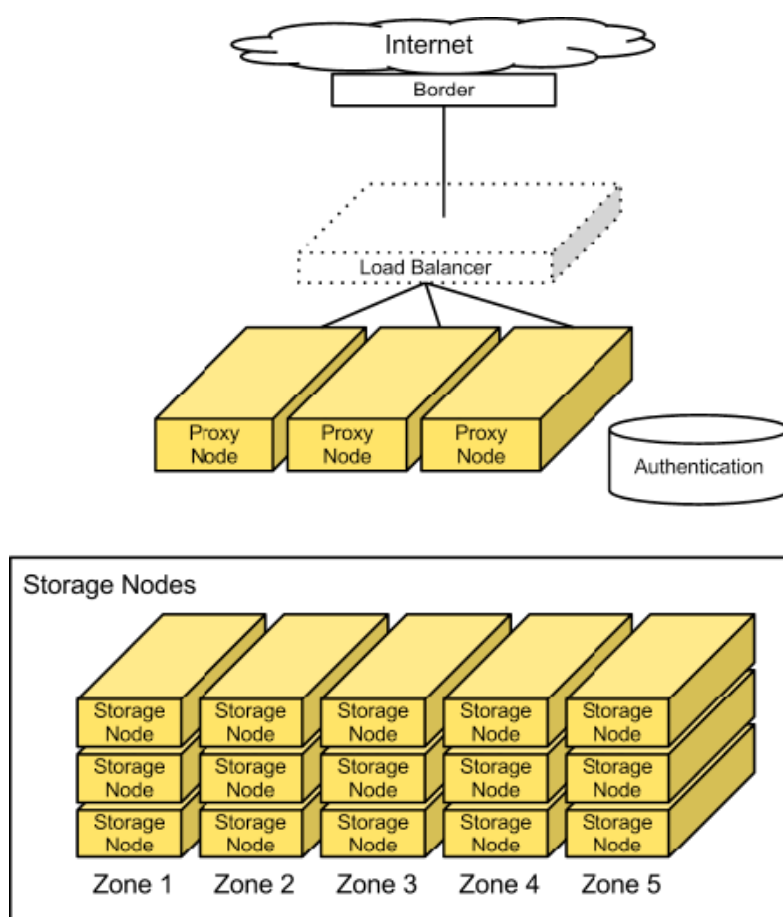
需要考虑的因素

对于大多数面向公网的部署以及那些跨超远距离的企业网络的私有部署来说，都使用了SSL来加密和客户端之间的网络。SSL增加了签名的过程来创建和客户端之间的会话，而这是要增加负载的，这就是为什么在访问层部署更多的机器的原因。SSL在私有信任网络的部署中不是必须的。

存储节点

在多数的配置中，每五个区就须拥有相同数量的存储。存储节点使用一个合理的内存和CPU数量即可。元数据需要被读取能够快速返回对象。对象存储运行的服务不仅仅是响应来自访问层的请求，还运行着复制器，审计和回收器。你可以为对象存储部署单个的千兆或万兆网卡，这取决于期望能够承受多大的负载和想要多好的性能。

图 5.10. 对象存储(Swift)



Currently, a 2 TB or 3 TB SATA disk delivers good performance for the price. You can use desktop-grade drives if you have responsive remote hands in the datacenter and enterprise-grade drives if you don't.

需要考虑的因素

你应该记住为单线程请求所需的I/O性能。此系统不使用RAID,所以单个磁盘处理每个请求对象。磁盘性能影响单线程响应率。

为了实现更高的吞吐量,对象存储系统被设计为并行的掌控上传和下载。那么网络 I/O 的能力(1GbE, 绑定一对 1GbE, 或者 10GbE) 就须满足你想要并行吞吐量。

复制

因为每个副本在对象存储中都是功能上独立的,而且客户端通常只需要有一个节点来响应,这样就被认为是一次成功的操作,诸如网络分区这样的瞬间失效就会立刻引发副本的发散。所造成的差异最终是通过异步来协调,点对点的复制器来处理的。复制器会遍历这些本地的文件系统,而且同时执行操作跨物理磁盘的负载均衡动作。

复制使用推送的模式,通常仅从本地复制纪录和文件到远程的复制器。这非常的重要,因为节点上的数据可能不属于哪里(如发生偏移或环变更的情况),而且复制器无法得知哪些数据

需要从集群中的拉入的。任何包含数据的节点必须确保数据归属于哪里。环掌控着副本的所在。

复制除了创建之外就是删除了，系统中每个删除的纪录或文件都会被标记为一个墓碑。复制清理墓碑是在一致性窗口过时之后进行。此窗口定义了复制的持续时间以及失效的节点多长时间可从集群中删除。墓碑的清理必须联系到复制达到副本已收敛。

如果复制器发现了远程的磁盘驱动已经失效，复制器就会使用`get_more_nodes`接口，为环选择相关与之同步的节点提供依据。复制器可以在磁盘失效期间维护期望水平的复制，尽管一些副本可能不能在本地立即使用。



注意

复制器不会维护哪些诸如整个节点都宕机的失效时的期望水平的复制；多数的失效都是瞬时的。

主要的复制类型有：

- 数据库复制，复制容器和对象。
- 对象复制。复制对象数据。

数据库复制

数据库复制是一次低代价的哈希对比来决定两个副本是否匹配。通常，系统中多数已经同步的数据库作这样的检查会非常快的得到验证。但是如果哈希不一致了，复制器通过分享最后一次同步点所增加的纪录来同步数据库。

此同步点是一高水印，此水印记下了已知的同步过的两数据库的最后纪录，且作为一远程数据库ID和纪录ID的元组保存在各自的数据库。数据库ID在跨所有副本的数据库中是唯一的，纪录ID按整数单调递增。在所有的新的纪录都推送到了远程数据库之后，会讲整个本地数据库的同步表推送，所以远程数据库可以保证同步所有的本地数据库是过去同步过的。

如果一个副本丢失了，就会通过`rsync(1)`将整个本地数据库文件传输到远端，然后再为其分配一个新的唯一的ID。

在实践中，数据库复制每秒可以处理数以百计的每个并发数据库设置(可用cpu的数量或磁盘)和绑定必须执行数据库事务的数量。

对象复制

对象的初始实现复制执行`rsync`将数据从本地分区所有远程服务器将驻留。虽然这在小规模，复制倍飙升一旦目录结构可以不再驻留在内存中。这个方案被修改保存一个散列的每个后缀目录的内容到每个分区散列文件。后缀目录的散列是不再有效时修改后缀目录的内容。

对象复制过程读入和计算任何无效哈希散列文件。然后,它将散列应持有的每个远程服务器分区,只有后缀目录与远程服务器上不同的散列`rsync`。把文件到远程服务器后,复制过程通知重新计算散列的`rsync`后缀目录。

对象复制必须遍历的那些没有缓存的目录的数量,通常是由于无效后缀目录的散列,这会阻碍性能。为了提供可接受的复制速度,对象复制被设计为每天在正常的节点上合理的使用约2%的散列空间。

账户清道夫

在后台，账户清道夫从已经删除的账户中移除数据库。

通过向账户存储URL发出一个DELETE请求来标记账户要删除。此动作会设置账户数据库中的表account_stat的status一栏，以及副本为DELETED,标记账户的数据要删除。

通常，系统是不提供指定保留时间和标记不删除功能的。然而，你可以在account-server.conf 中的[account-reaper]一节设置 delay_reaping的值来延迟实际删除数据的行。此时，要撤销删除的话，你要直接更新账户数据库副本，设置状态一栏为空字符，然后更新put_timestamp要大于delete_timestamp。



注意

执行此任务的工具还处于开发者的开发列表中，所以最好是通过REAST来调用。

在每个账户服务中均运行了账户清道夫，偶尔会扫描服务中的账户数据库，标记某些账户为删除。它仅仅是激发了那个服务是主节点上的账户，所以并不会同时影响到多个账户服务。使用多服务来删除一个账户也许会改进删除的速度，但是需要协调来避免重复。数据删除的速度并不是大的需要关心的问题，更何况大型的账户并不需要经常性的删除。

删除一个账户很简单。对于每个账户容器来说，所有的对象都删除了，那就意味着容器也被删除了。失败的删除请求将不会停止整个过程，但会导致整个过程最终失败(举例来说，如果删除一个对象超时，你就不能再去删除其容器或账户)。账户清道夫会一直尝试删除一个账户直到它为空，之后，数据库回收过程内部的db_replicator会移除数据库文件。

持久性错误状态可能会阻止删除的对象或容器。如果发生了此类事件，你可以在日志文件中看到诸如"Account <name> has not been reaped since <date>"的信息。你可以通过设置 account-server.conf 文件的 [account-reaper] 一节中的reap_warn_after的值来控制此种记录。此值的默认值是30天。

配置基于对象存储的特定租户的镜像位置

对于一些部署者来说，他们没有打算将所有的镜像都存放在一个地方，这样的话，所有的租户和用户都可以访问这些镜像。你可以配置镜像服务按照指定租户镜像位置的方式来存储镜像。这样，只有下列租户可以使用镜像服务来访问已经创建好的镜像：

- 镜像所有者的租户
- 租户在 swift_store_admin_tenants中被定义，且是拥有管理员级别的账户

要配置特定租户镜像位置

1. 在 glance-api.conf 文件中配置swift作为你的 default_store 。
2. 在glance-api.conf 文件中设置这些配置选项：
 - swift_store_multi_tenant。设置为True，从而启用特定租户存储位置。默认是False。
 - swift_store_admin_tenants。指定一租户ID列表，这些租户可以被赋予所有对象存储容器的读写访问，而这些对象存储容器均是由镜像服务所创建。

有了这些配置，镜像均是存储在一个对象存储服务(swift)端点，已授权的用户可以从服务目录中来访问。

对象存储监控

Darrell Bishop的博客可以说明这些。

OpenStack 对象存储集群是一组跨多节点的多个守护进程协同工作，拥有多个不同的组件，用户必须清楚在集群内部发生了什么，追踪一些计量如CPU使用、负载、内存消耗、磁盘使用量和容量等等都是必要的，但还不够。

在每个服务器中不同的守护进程在做什么？是什么卷在节点8上做对象复制？会花费多久？有错误吗？如果有，什么时候发生的？

在如此一个复杂的系统中，用户可以使用很多合适的工具来回答这些问题，此节描述了一些。

Swift Recon

中间件Swift Recon(请参阅http://swift.openstack.org/admin_guide.html#cluster-telemetry-and-monitoring) 提供了常见的机器状态，诸如平均负载，套接字状态，/proc/meminfo内容，等等，以及特定的Swift计量：

- 每个ring文件的MD5值。
- 最近的对象复制时间。
- 统计隔离文件的每种类型:账户、容器、或对象。
- 统计 磁盘上的"async_pendings" (容器更新延迟)

Swift Recon 是一个中间件，它安装在对象服务器的管道中，需要一个选项：一个本地的缓存目录。要追踪 async_pendings，您必需设置为每个对象服务器一个额外的 cron 任务。您通过发送 HTTP 请求直接访问对象服务器或使用 swift-recon 命令行客户端来访问数据均可。

有一些好的对象存储集群统计数据，但一般的服务器计量与现有的服务器监控系统重叠了。要获取指定 Swift 的计量数据到监控系统中，它们必需被修剪。Swift Recon 本质上作为一个计量收集器的中间件。向计量提供数据到您统计系统的进程，如 collectd 和 gmond，可能已经运行在存储节点上。所以，您可以选择与 Swift Recon 会话或直接收集计量数据。

Swift-Informant

Florian Hines开发了Swift-Informant中间件(请参阅 <https://github.com/pandemicsyn/swift-informant>)用来获取实时的对象存储客户端请求。其驻扎在代理服务的管道中，且在每个请求代理服务都会发送三个计量到statsD服务(请参阅 <http://codeascraft.etsy.com/2011/02/15/measure-anything-measure-everything/>):

- 计量的增长计数器类似于obj.GET.200 或 cont.PUT.404。
- 用于计量的时间数据类似acct.GET.200 或 obj.GET.200。[README中说计量会类似duration.acct.GET.200这样，但是我在代码中没有看到 duration。我不确定Etsy服务

做了什么，但是StatsD服务会更改时间计量为五个衍生的计量，使用了新的附加段，所以它是可以正常工作的。改变后的计量是这样 `acct.GET.200.lower`, `acct.GET.200.upper`, `acct.GET.200.mean`, `acct.GET.200.upper_90`, 以及 `acct.GET.200.count`]。

- 计数器的增加会由字节传输到计量，如 `tfer.obj.PUT.201`。

这对获取经历事件计量的服务客户端的质量和请求的服务器类型、命令和响应代码的各种排列的卷的感受是有好处的。`Swift-informant` 也不需要修改核心对象存储代码，因为它是以中间件实现的。但是，它会使您不了解集群传递给代理服务器的过程是如何工作的。如果一个存储节点的响应性降低了，您仅仅能看到一些请求是坏的，高延时或错误状态代码。您不会确切地知道为什么或哪里的请求出现问题。也许出现问题的容器服务器在一个好的节点上，但对对象服务器在另一个性能不好的节点上。

Statsdlog

Florian's [Statsdlog](#) 项目增加基于记录事件的 StatsD 计数器。类似于 `Swift-informant`，它也是非入侵性的，但 `statsdlog` 可以从所有对象存储的守护进程中追踪事件，而不仅仅是代理服务器。守护进程监听一个 `syslog` 消息的 UDP 流，当记录行匹配一个正则表达式时，StatsD 计数器会增加。计量名称会映射到正则表达式匹配 JSON 文件格式，允许灵活配置什么计量要从记录流中提取出来。

目前，只有第一个匹配正则表达式会触发 StatsD 计数器的增长，计数器总是由一增长的。无法让计数器由大于一的数目增长或发送实时数据到基于记录内容的 StatsD 中。工具可以扩展来为每一行和数据提取处理更多的计量，包括实时数据。但耦合仍然存在于记录文本格式和记录解析表达式之间，他们自己会变得更加复杂以为每一行和数据提取支持多匹配。并且，记录进程介绍了一个触发事件和发送数据给 StatsD 之间的延时。这将是最好的增量错误计数器，他们会发生并在知道要避免记录字符串和正则比阿大时解析之间的耦合是发送实时数据，防止事件和发送数据到 StatsD 之间的时间延时。

下一节描述了对对象存储运维计量的两外一种方法。

Swift StatsD 日志

StatsD (请阅读 <http://codeascraft.etsy.com/2011/02/15/measure-anything-measure-everything/>) 被设计为应用代码，以深入仪表；计量是由仅仅通知或做某些事情的代码实时发送的。发送一个计量的开销是非常低的：一个 UDP 包的 `sendto`。如果这个开销仍然太高，StatsD 客户端库可以仅仅发送一个样例的随机部分，且冲刷计量到上游时，StatsD 非常仅近于实际数值。

为了避免使用基于中间件的监控和事后日志处理的内部问题，StatsD 计量的发送被整合到对象存储本身中。提交修改设置 (详见 <https://review.openstack.org/#change,6058>) 当前通过 15 个对象存储守护进程和临时路径中间件报告 124 个计量。详细的计量方针在 [Administrator's Guide](#) 文档中。

计量的发送已与记录框架整合。要将它启用，请配置相关配置文件中的 `log_statsd_host` 字段。您也可以指定端口和一个默认的采样率。除非有一个特定的调用，指定的默认采样率才会被用于 `statsd` 记录方法 (见下表) 覆盖。目前，没有记录调用覆盖采样率，但可以想象的是，一些计量可能需要精度 (`sample_rate == 1`) 而其他的不需要。

```
[DEFAULT]
...
log_statsd_host = 127.0.0.1
log_statsd_port = 8125
```

```
log_statsd_default_sample_rate = 1
```

然后，由 `get_logger()` 返回的 `LogAdapter` 对象，通常保存在 `self.logger`，有了这些新的方法：

- `set_statsd_prefix(self, prefix)` 设置客户端库 `stat` 的前缀值，它会获取前缀给每个计量。默认的前缀是记录者的名称，如 `"object-server"`、`"container-auditor"` 等。现在，当确定了控制器对象和示例化请求时，用于将 `"proxy-server"` 转换为 `"proxy-server.Account"`、`"proxy-server.Container"` 或 `"proxy-server.Object"`。
- `update_stats(self, metric, amount, sample_rate=1)` 按照给定的数目增加所提供的计量。当您需要从计数器中添加或减去大于一的数目时，这将会使用到，例如以对象重复符中的计算哈希来增加 `"suffix.hashes"`。
- `increment(self, metric, sample_rate=1)` 以一来增加给定的计数器。
- `decrement(self, metric, sample_rate=1)` 以一减少给定的计数器。
- `timing(self, metric, timing_ms, sample_rate=1)` 记录给定的计量采取了提供的毫秒数。
- `timing_since(self, metric, orig_time, sample_rate=1)` 记录一个其值为 `"now"` 减去一个已有时间戳的实时计量的便捷方法。

请注意，这些记录方法可能会被任何您有记录者的对象安全调用。如果 `StatsD` 记录没有配置，这些方法是无操作的。这避免了每个地方记录一个计量的杂乱逻辑。这些示例的使用展示了新的记录方法：

```
# swift/obj/replicator.py
def update(self, job):
    # ...
    begin = time.time()
    try:
        hashed, local_hash = tpool.execute(tpooled_get_hashes, job['path'],
            do_listdir=(self.replication_count % 10) == 0,
            reclaim_age=self.reclaim_age)
        # See tpooled_get_hashes "Hack".
        if isinstance(hashed, BaseException):
            raise hashed
        self.suffix_hash += hashed
        self.logger.update_stats('suffix.hashes', hashed)
    # ...
    finally:
        self.partition_times.append(time.time() - begin)
        self.logger.timing_since('partition.update.timing', begin)
```

```
# swift/container/updater.py
def process_container(self, dbfile):
    # ...
    start_time = time.time()
    # ...
    for event in events:
        if 200 <= event.wait() < 300:
            successes += 1
        else:
            failures += 1
    if successes > failures:
        self.logger.increment('successes')
    # ...
```



```
else:
    self.logger.increment('failures')
    # ...
    # Only track timing data for attempted updates:
    self.logger.timing_since('timing', start_time)
else:
    self.logger.increment('no_changes')
    self.no_changes += 1
```

StatsD 的开发团队打算使用 `pystatsd` 客户端库 (不要与 Github 上的 [similar-looking 项目](#) 混淆)，但在 PyPI 上发布的版本缺少两个 Github 上的最新版本所需要的特性：在客户端对象配置计量前缀的功能和一个发送实时数据到 “now” 和 “start” 的您已有时间戳之间的便捷方法。所以他们只使用同样的接口从中实现了一个简单的 StatsD 客户端库。这有良好的附加效益，不会引入另一个额外的依赖于对象存储的库。

对象存储的系统管理

欲理解对象存储概念，用户最好是监测和管理其存储方案。主要的系统管理信息有开发者文档所维护，在 docs.openstack.org/developer/swift/。

关于对象存储的配置属性列表详情，请参考 [OpenStack 配置参考](#)。

对象存储故障排查

对于对象存储，所有的日志记录都在 `/var/log/syslog` (或在某些发行版是 `/var/log/messages`)。在对象服务器中的配置文件中，有多个设置可进一步的定制日志，诸如 `log_name`, `log_facility`, 以及 `log_level`，。

驱动失效

在一个磁盘驱动失效的事件中，首要的步骤是要确认磁盘是否已卸载。已卸载的磁盘驱动会让对象存储从失效中恢复相比挂载的来说容易一些。如果磁盘要立即替代的话，只需要替代磁盘，格式化它，重新挂载它，然后让复制填满它即可。

如果磁盘驱动不能被理解的替换，那么最好是将之先卸载，然后将之从环中移除。这会让此磁盘上所有的副本复制到其它地方，直到磁盘驱动替换为止。一旦替换完成，它就可以重新添加到环中。

你可到 `/var/log/kern.log` 中查找错误信息，以捕捉到失效的磁盘。

服务器失效

如果服务器有了硬件的故障，这非常的容易确定那个对象存储服务停止了运行。当你排除了故障后，可以将此对象存储从失效中恢复回来。

如果仅仅是服务器需要重启，或者是停止工具一两个小时的话，最好是让对象存储自身来恢复此类暂时的失效，自身获得已恢复的机器并恢复在线。当机器回到在线时，复制会确保更新它在宕机时错过的任何东西。

如果服务器发生了严重的问题，那么最好是从环中将所有的服务器的设备都移除。一旦服务器修复且恢复在线，服务器的设备就可以添加回环中。对于设备在添加回到环之前将设备重新格式化是非常重要的，正如原来负责不同的分区一样。

发现失效的磁盘驱动器

根据我们的经验,当驱动器即将失败时,会在/var/log/kern.log中出现错误消息。这里有一个脚本,叫做swift-drive-audit,可以通过计划任务来运行从而时刻监视着损坏的磁盘。一旦发现错误,它会卸载损坏的磁盘,然后对象存储会自我修复。此脚本的配置文件需要下面的配置:

表 5.1. [drive-audit]在drive-audit.conf中的配置属性描述

配置属性=默认值	描述
device_dir = /srv/node	Directory devices are mounted under
error_limit = 1	Number of errors to find before a device is unmounted
log_address = /dev/log	Location where syslog sends the logs to
log_facility = LOG_LOCAL0	Syslog log facility
log_file_pattern = /var/log/kern.*[!][g][!z]	Location of the log file with globbing pattern to check against device errors locate device blocks with errors in the log file
log_level = INFO	Logging level
log_max_line_length = 0	Caps the length of log lines to the value given; no limit if set to 0, the default.
log_to_console = False	No help text available for this option.
minutes = 60	Number of minutes to look back in `/var/log/kern.log`
recon_cache_path = /var/cache/swift	Directory where stats for a few items will be stored
regex_pattern_1 = berror b,* b(dm-[0-9]{1,2} d?) b	No help text available for this option.
unmount_failed_device = True	No help text available for this option.

此脚本仅在Ubuntu 10.04下测试过,所以如果你使用的是不同的发行版或不同的操作系统,请在使用到生产环境中时务必小心。

紧急恢复环构建者文件

你须一直保持对swift环构建者文件的备份。然而,一旦紧急事件发生了,此步骤可帮助你集群恢复到可操作状态。

使用现有的swift工具,是没有办法从一个ring.gz文件来恢复构建者文件的。然后,如果拥有些Python的知识的话,就有可能构造一个构建者文件,它可以非常的接近你所丢失的。下面是你需要做的内容。



警告

此步骤是在紧急情况下最后的绝招。它需要swift python代码的知识,而且还有可能不会成功。

首先,在一个Python REPL中加载环和一个新的环构建者的对象:

```
>>> from swift.common.ring import RingData, RingBuilder
>>> ring = RingData.load('/path/to/account.ring.gz')
```

现在,开始拷贝环中的数据到构建者。

```
>>> import math
```

```
>>> partitions = len(ring._replica2part2dev_id[0])
>>> replicas = len(ring._replica2part2dev_id)

>>> builder = RingBuilder(int(math.log(partitions, 2)), replicas, 1)
>>> builder.devs = ring.devs
>>> builder._replica2part2dev = ring._replica2part2dev_id
>>> builder._last_part_moves_epoch = 0
>>> from array import array
>>> builder._last_part_moves = array('B', (0 for _ in xrange(partitions)))
>>> builder._set_parts_wanted()
>>> for d in builder._iter_devs():
>>>     d['parts'] = 0
>>> for p2d in builder._replica2part2dev:
>>>     for dev_id in p2d:
>>>         builder.devs[dev_id]['parts'] += 1
```

这是一个在多大程度上可恢复的事情。对于`min_part_hours`，你不仅要记住你所使用的值，还要组成一个新的。

```
>>> builder.change_min_part_hours(24) # or whatever you want it to be
```

接下来，验证构建者。如果此会抛出异常，检查你原来的代码。当它通过验证，你就准备好保存构建者，然后创建一个新的`account.builder`。

```
>>> builder.validate()
```

保存构建者。

```
>>> import pickle
>>> pickle.dump(builder.to_dict(), open('account.builder', 'wb'), protocol=2)
>>> exit()
```

你现在须在当前的工作目录中有一个名字叫做`'account.builder'`的文件。接下来，运行`swift-ring-builder account.builder write_ring`，然后比较下你开始时的`account.ring.gz`和新的`account.ring.gz`。他们不可能每个字节都是一样的，但是如果你加载它们到REPL中，它们的`_replica2part2dev_id` and `devs`属性是一致(或者非常接近)的话，那么你成功的几率就非常的大了。

接下来，为`container.ring.gz` 和 `object.ring.gz`重复上述步骤，然后，你就可能得到可用的构建者文件

第 6 章 块存储

目录

块存储介绍	138
加大块存储 API 服务的吞吐量	139
管理卷	139
用户安装Troubleshoot	170

OpenStack块存储的服务是由在一台或多台主机上的一系列叫做 cinder-*的守护进程相关作用而工作的，这些守护进程可以运行在单独的一台节点，也可以跨多节点，它们也可以和OpenStack其他服务一起运行在同一主机。

块存储介绍

要管理OpenStack块存储服务，理解一些概念是非常有帮助的。当你在OpenStack中配置存储服务时必须做出一定的选择。这些选项的大部分归结为两种选择，单节点或多节点安装。你可以阅读关于存储决定的长篇大论，在OpenStack 运维手册中的[存储决策](#)一文。

OpenStack块存储可为用户的OpenStack计算实例提供额外的块设备级别的存储，此服务类似于亚马逊所提供的 EC2 伸缩块存储(EBS)。

加大块存储 API 服务的吞吐量

默认情况下，块存储API服务运行一个进程。这就限制了一些API请求，块存储服务可以在任何时候处理这些请求的能力也就遭到了限制。在生产环境中，用户需要增加块存储API服务吞吐量，允许块存储API服务在主机能力范围可以的情况下尽可能的多运行几个进程。



注意

块存储API服务在下列发行版中的名称为 `openstack-cinder-api`：
CentOS, Fedora, openSUSE, Red Hat Enterprise Linux 以及 SUSE Linux Enterprise。
若是 Ubuntu 或 Debian 发行版的话，块存储API服务叫做 `cinder-api`。

要实现上述的话，使用块存储API服务属性 `osapi_volume_workers`。此属性允许用户指定API服务的工作数量(或者是操作系统的进程)，然后去启动相应数量的块存储API服务。

欲配置此属性，打开配置文件 `/etc/cinder/cinder.conf`，然后设置 `osapi_volume_workers` 的配置键的值为主机CPU的核心或线程数。

在装有 `openstack-config` 程序的发行版中，你可以运行下面命令来配置此步：

```
# openstack-config --set /etc/cinder/cinder.conf  
DEFAULT osapi_volume_workers CORES
```

将 `CORES` 用主机的CPU 核心数或线程数替代即可。

管理卷

默认的OpenStack块存储服务的实现是在Linux下，使用逻辑卷管理(LVM)的iSCSI解决方案。



注意

OpenStack块存储服务不是像存储区域网络(SAN)的NFS卷那样的共享存储解决方案，NFS的话你可以在多个服务器上挂载同一个卷，而对于OpenStack块存储服务来说，你同时只能将卷挂载到一个实例。

OpenStack块存储服务也提供了让用户使用多个供应商的后端存储设备的驱动机制，增强或替代最基本的LVM实现。

此高度抽象的步骤想用户展示了如何创建和挂接一个卷到服务器的实例。

创建卷并将之挂接到实例

1. 通过修改文件 `cinder.conf` 来配置OpenStack计算和OpenStack块存储服务。
2. 使用命令 `cinder create` 来创建一个卷。此命令是在 `cinder-volumes` 卷组(VG)中创建了一个LV。
3. 使用命令 `nova volume-attach` 将某个卷挂接到实例上。此命令创建了一个抛给计算节点的唯一iSCSI IQN。
 - a. 运行着实例的计算节点，此时会激活iSCSI会话，且有了新的本地存储(常见的是一个 `/dev/sdX` 磁盘)。

- b. `libvirt`使用本地存储来作为实例的存储。实例会得到一个新的磁盘(常见的是一个 `/dev/vdX` 磁盘)。

For this particular walk through, one cloud controller runs `nova-api`, `nova-scheduler`, `nova-objectstore`, `nova-network` and `cinder-*` services. Two additional compute nodes run `nova-compute`. The walk through uses a custom partitioning scheme that carves out 60 GB of space and labels it as LVM. The network uses the `FlatManager` and `NetworkManager` settings for OpenStack Compute.

网络的模式不会影响到OpenStack块存储的操作，但是你必须设置可让块存储工作的网络。更多细节，请参考 [第 7 章 网络 \[180\]](#)。

要设置计算节点使用卷，确保块存储已经安装好，且安装了软件包 `lvm2`。此向导描述了如何为你的安装环境做故障排查以及备份你的计算卷。

从云硬盘启动

在某些情况下，你可以在卷内部存储和运行实例。关于此方面的信息，请参阅 [OpenStack 最终用户手册](#) 章节中的 [从卷启动一个实例](#) 一节。

在后端配置一NFS存储

此节解释了如何配置OpenStack块存储使用NFS作为后端。你须保证cinder卷服务所在主机可以访问到NFS的共享。



注意

cinder 卷服务在下面的发行版中的名称叫做 openstack-cinder-volume：

- CentOS
- Fedora
- openSUSE
- 红帽企业 Linux
- SUSE Linux企业版

在Ubuntu和Debian的发行版中，cinder卷服务名称叫做cinder-volume。

配置块存储以使用NFS存储后端

1. 以root 登录到托管cinder卷服务的系统。
2. 在目录 /etc/cinder/下创建一个名称为 nfsshare的文本文件。
3. OpenStack块存储用于后端存储的每个cinder 卷服务都须为/etc/cinder/nfsshare 添加实体，每个实体须是单独的一行，且须使用如下格式：

```
HOST:SHARE
```

地点：

- HOST 填写IP地址或是NFS服务器的主机名。
- SHARE使已经存在的且可访问的NFS共享的绝对路径。

4. 设置/etc/cinder/nfsshare的属主为root用户，组为cinder。

```
# chown root:cinder /etc/cinder/nfsshare
```

5. 设置/etc/cinder/nfsshare为可由组cinder成员可读：

```
# chmod 0640 /etc/cinder/nfsshare
```

6. 要配置OpenStack块存储使用早些时候所创建的文件/etc/cinder/nfsshare，打开配置文件/etc/cinder/cinder.conf然后设置配置项nfs_shares_config 的值为/etc/cinder/nfsshare。

在装有openstack-config程序的发行版中，你可以运行下面命令来配置此步：

```
# openstack-config --set /etc/cinder/cinder.conf  
DEFAULT nfs_shares_config /etc/cinder/nfsshare
```

以下发行版包括openstack-config:

- CentOS
- Fedora
- openSUSE
- 红帽企业 Linux
- SUSE Linux企业版

7. 可选，添加额外的NFS挂载点属性需要在你的环境中设置/etc/cinder/cinder.conf的nfs_mount_options 键值。如果你的NFS共享无须任何额外的挂载属性(或者是你不能确定)的话，请忽略此步。

在装有openstack-config程序的发行版中，你可以运行下面命令来配置此步：

```
# openstack-config --set /etc/cinder/cinder.conf  
DEFAULT nfs_mount_options OPTIONS
```

替换OPTIONS为用于访问NFS共享时的挂载点属性。请参阅帮助文档，来获得可用的挂载属性更多的信息 (man nfs)。

8. 配置cinder 卷服务使用正确的卷驱动，即名称为cinder.volume.drivers.nfs.NfsDriver驱动。要完成此，打开/etc/cinder/cinder.conf配置文件，然后设置volume_driver 的键值为cinder.volume.drivers.nfs.NfsDriver。

在装有openstack-config程序的发行版中，你可以运行下面命令来配置此步：

```
# openstack-config --set /etc/cinder/cinder.conf  
DEFAULT volume_driver cinder.volume.drivers.nfs.NfsDriver
```

9. 你现在可以重启服务以应用这些配置。

在运行 CentOS, Fedora, openSUSE, Red Hat Enterprise Linux, 或 SUSE Linux Enterprise, 的发行版中，重启cinder 卷服务：

```
# service openstack-cinder-volume restart
```

在Ubuntu或Debian中要重启cinder卷服务的话，运行：

```
# service cinder-volume restart
```



注意

nfs_sparsed_volumes 配置关键字定义了卷是否作为稀疏文件创建并按需要分配或完全预先分配。默认和建议的值为 true，它会保证卷初始化创建为稀疏文件。

设置 nfs_sparsed_volumes为false的结果就是在卷创建的时候就完全分配了。这会导致卷创建时间的延长。

然而，你若选择设置 nfs_sparsed_volumes 为false的话，你可以直接编辑/etc/cinder/cinder.conf。

在装有openstack-config程序的发行版中，你可以运行下面命令来配置此步：


```
# openstack-config --set /etc/cinder/cinder.conf  
DEFAULT nfs_sparsed_volumes false
```



重要

如果客户端主机启用了SELinux，若此主机需要访问NFS共享上的实例的话就需要设置virt_use_nfs布尔值。以root用户运行下面的命令：

```
# setsebool -P virt_use_nfs on
```

此命令可是布尔值永久生效，即使是重启。在所有需要访问NFS共享的主机上运行此命令。这包括所有的计算节点。

配置后端为GlusterFS

此节解释了如何配置OpenStack块存储使用GlusterFS作为后端。你须保证cinder卷服务所在主机可以访问到GlusterFS的共享。



注意

cinder 卷服务在下面的发行版中的名称叫做 openstack-cinder-volume：

- CentOS
- Fedora
- openSUSE
- 红帽企业 Linux
- SUSE Linux企业版

在Ubuntu和Debian的发行版中，cinder卷服务名称叫做cinder-volume。

挂载GlusterFS卷需要软件包glusterfs-fuse 所提供的工具和库。此软件包必须安装到所有需要访问GlusterFS后端的节点中。



注意

挂载GlusterFS卷所需要的工具和库，在Ubuntu和Debian发行版中替代的软件包是glusterfs-client。

如何安装和配置GlusterFS的信息，请参阅页面[Gluster文档](#)。

配置GlusterFS作为OpenStack块存储

GlusterFS服务须配置为允许OpenStack块存储来访问GlusterFS共享：

1. 以root登录到GlusterFS服务器。
2. 为每个Gluster卷设置使用和cinder用户一样的UID和GID:

```
# gluster volume set VOL_NAME storage.owner-uid CINDER_UID  
# gluster volume set VOL_NAME storage.owner-gid CINDER_GID
```

地点：

- VOL_NAME 是Gluster卷名称。
- CINDER_UID 是用户cinder 的UID。
- CINDER_GID 是用户cinder 的GID。



注意

在大多数的发行版中用户cinder默认的UID和GID是165。

3. 配置每个Gluster卷接受libgfapi的连接，要完成此配置，设置每个Gluster卷打开不安全到端口：

```
# gluster volume set VOL_NAME server,allow-insecure on
```

4. 每个客户端的连接均来自未授权的端口。要完成此，在文件/etc/glusterfs/glusterd.vol加入下面这几行：

```
option rpc-auth-allow-insecure on
```

5. 重启 glusterd服务:

```
# service glusterd restart
```

配置块存储，以让其可使用GlusterFS后端

当你完成GlusterFS服务的配置，完成这些步骤:

1. 以root 登录到托管块存储服务的系统。
2. 在目录 /etc/cinder/下创建一个名称为 glusterfs的文本文件。
3. OpenStack块存储用于后端存储的每个GlusterFS共享都须为/etc/cinder/glusterfs 添加实体，每个实体须是单独的一行，且须使用如下格式：

```
HOST:/VOL_NAME
```

地点：

- HOST 是红帽存储服务器的IP地址或主机名。
- VOL_NAME 是GlusterFS服务器已存在且可访问达卷名。

可选，如果你的环境需要额外的对外共享的挂载点，你可以添加它们到共享条目里：

```
HOST:/VOL_NAME -o OPTIONS
```

将OPTIONS以逗号分隔的挂载属性替换。

4. 设置/etc/cinder/glusterfs的属主为root用户，组为cinder。

```
# chown root:cinder /etc/cinder/glusterfs
```

5. 设置/etc/cinder/glusterfs为可由组cinder成员可读：

```
# chmod 0640 文件
```

6. 要配置OpenStack块存储使用早些时候所创建的文件/etc/cinder/glusterfs，打开配置文件/etc/cinder/cinder.conf然后设置配置项glusterfs_shares_config的值为/etc/cinder/glusterfs。

在装有openstack-config程序的发行版中，你可以运行下面命令来配置此步：

```
# openstack-config --set /etc/cinder/cinder.conf  
DEFAULT glusterfs_shares_config /etc/cinder/glusterfs
```

以下发行版包括openstack-config:

- CentOS
- Fedora
- openSUSE
- 红帽企业 Linux
- SUSE Linux企业版

7. 要配置OpenStack块存储使用名称为cinder.volume.drivers.glusterfs的正确的卷驱动器，打开配置文件/etc/cinder/cinder.conf然后设置配置项volume_driver的值为cinder.volume.drivers.glusterfs。

在装有openstack-config程序的发行版中，你可以运行下面命令来配置此步：

```
# openstack-config --set /etc/cinder/cinder.conf  
DEFAULT volume_driver cinder.volume.drivers.glusterfs.GlusterfsDriver
```

8. 你现在可以重启服务以应用这些配置。

在运行 CentOS, Fedora, openSUSE, Red Hat Enterprise Linux, 或 SUSE Linux Enterprise, 的发行版中，重启cinder 卷服务：

```
# service openstack-cinder-volume restart
```

在Ubuntu或Debian中，重启cinder 卷服务运行：

```
# service cinder-volume restart
```

OpenStack块存储现在可以使用GlusterFS后端了。



注意

在/etc/cinder/cinder.conf中，glusterfs_sparsed_volumes配置项决定了卷的创建使用稀疏文件以及按需还是全部分配空间。默认和建议此键的值为 true，这样可确保初始化创建的卷是基于稀疏文件的。

设置 glusterfs_sparsed_volumes为false的结果就是在卷创建的时候就完全分配了。这会导致卷创建时间的延长。

然而，你若选择设置 glusterfs_sparsed_volumes 为false的话，你可以直接编辑/etc/cinder/cinder.conf。

在装有openstack-config程序的发行版中，你可以运行下面命令来配置此步：

```
# openstack-config --set /etc/cinder/cinder.conf  
DEFAULT glusterfs_sparsed_volumes false
```



重要

如果客户端主机启用了SELinux，若此主机需要访问GlusterFS卷上的实例的话就需要设置virt_use_fusefs布尔值。以root用户运行下面的命令：

```
# setsebool -P virt_use_fusefs on
```

此命令可是布尔值永久生效，即使是重启。在所有需要访问GlusterFS卷的主机上运行此命令。这包括所有的计算节点。

配置多后端存储

当您配置了多存储后端，您可以创建一些后端存储解决方案，服务于同一个 OpenStack 计算配置，并为每个后端存储或后端存储池启动一个 cinder-volume。

在一个多存储后端的配置中，每个后端都有一个名称 (volume_backend_name)。一些后端可以有相同的名称。在这个情况下，调度器会适当地决定使用哪个后端来创建卷。

后端的名称声明为一个额外指定的卷类型 (例如，volume_backend_name=LVM_iSCSI)。当卷创建后，调度器根据用户指定的卷类型选择一个合适的后端来处理请求。

激活后端多存储

要启用多存储后端，您必需设置 cinder.conf 文件中的 enabled_backends 标签。这个标签定义了不同后端的配置组的名称 (以逗号隔开)：一个名称关联到一个配置组后端 (例如，[lvmdriver-1])。



注意

配置的组名和volume_backend_name没有关联。



注意

在一个已有 cinder 服务设置了 enabled_backends 标签后并重启块设备存储服务后，原来的 host 服务被新的 host 服务所代替。新服务会以一个类似于 host@backend 的名称出现。使用：

```
$ cinder-manage volume update_host --currentname CURRENTNAME --  
newname CURRENTNAME@BACKEND
```

来将当前的块设备转换为新的主机名称。

配置组的选项必须在组中 (或默认使用选项中) 定义。所有的标准块设备存储配置选项 (volume_group、volume_driver 等) 都可以在配置组中使用。[DEFAULT] 配置组中的配置值没有被使用。

这些例子展示了三个后端：

```

enabled_backends=lvmdriver-1,lvmdriver-2,lvmdriver-3
[lvmdriver-1]
volume_group=cinder-volumes-1
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI
[lvmdriver-2]
volume_group=cinder-volumes-2
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI
[lvmdriver-3]
volume_group=cinder-volumes-3
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI_b

```

在这个配置中，lvmdriver-1 和 lvmdriver-2 有相同的 volume_backend_name。如果一个卷的创建请求 LVM_iSCSI 后端名称，调度器会使用容量过滤调度器来选择最合适的驱动，即 lvmdriver-1 或 lvmdriver-2。容量过滤调度器是默认启用的。下一个小节提供了更多相关信息。另外，这个示例使用了 lvmdriver-3 后端。



注意

对于支持多路径的 Fiber Channel 驱动，配置组需要 use_multipath_for_image_xfer=true 选项。在下面的示例中，您可以看到 HP 3PAR 和 EMC Fiber Channel 驱动的信息。

```

[3par]
use_multipath_for_image_xfer = true
volume_driver = cinder.volume.drivers.san.hp.hp_3par_fc,HP3PARFCDriver
volume_backend_name = 3parfc

[emc]
use_multipath_for_image_xfer = true
volume_driver = cinder.volume.drivers.emc.emc_smis_fc,EMCSMISFCDriver
volume_backend_name = emcfc

```

配置块存储多个后端调度

你要使用多存储后端的话必须启用 filter_scheduler。过滤调度器：

1. 后端的过滤器是可用的。默认情况下，启用的有 AvailabilityZoneFilter, CapacityFilter 以及 CapabilitiesFilter。
2. 权衡预先过滤的后端。默认情况下，CapacityWeigher 是启用的。如果这个选项启用，那么过滤调度器会分配最高权重给拥有最多可用容量的后端。

调度器使用过滤器和权重来挑选最佳后端来应答请求。调度器使用卷类型来明确指定后端来创建卷。

卷类型

在使用之前，一个卷的类型在块存储中声明。这可由下面的命令来完成：

```
$ cinder --os-username admin --os-tenant-name admin type-create lvm
```

然后，创建一个额外的说明，关联到卷的类型到后端的名称，运行此命令：

```
$ cinder --os-username admin --os-tenant-name admin type-key lvm set volume_backend_name=LVM_iSCSI
```

此例创建了一个lvm卷类型，volume_backend_name=LVM_iSCSI是其额外的说明。

创建另外的卷类型：

```
$ cinder --os-username admin --os-tenant-name admin type-create lvm_gold
```

```
$ cinder --os-username admin --os-tenant-name admin type-key lvm_gold set volume_backend_name=LVM_iSCSI_b
```

此处第二个卷类型的名称是 lvm_gold，后端的名称是LVM_iSCSI_b。



注意

要列出额外的特定的内容，使用此命令：

```
$ cinder --os-username admin --os-tenant-name admin extra-specs-list
```



注意

如果在块存储的配置中卷类型指向的volume_backend_name不存在，filter_scheduler就会返回无法找到合法的合适的后端主机的错误。

用量

当你创建一个卷的时候，你必须制定卷的类型。卷类型的额外说明用于决定使用那个后端。

```
$ cinder create --volume_type lvm --display_name test_multi_backend 1
```

考虑到之前cinder.conf的描述，调度器在lvmdriver-1 或 lvmdriver-2之上创建卷。

```
$ cinder create --volume_type lvm_gold --display_name test_multi_backend 1
```

此第二个卷创建到了lvmdriver-3。

备份块存储服务磁盘

其实你使用LVM快照来创建快照的话，就意味着你也可以使用LVM快照来做备份。通过使用LVM快照，你可以有效减少备份的占用空间；它会仅备份已经存在的数据，而不是整个卷。

要备份卷的话，你必须给它创建一个快照。一个LVM快照就是确切的逻辑卷复制，其包含了冰冻状态的数据。这保证数据的可靠性，因为在卷创建的过程中数据是不可操作的。记住，通过命令nova volume-create在已有的LVM逻辑卷上创建卷。

你也必须保证操作系统没有使用此卷，且客户文件系统已经将所有数据刷回。这通常意味着在快照创建期间这些文件系统已经卸载。当然在逻辑卷创建完成后可立刻再次挂载。

在你创建快照之前，你必须确保拥有足够的空间来存放它。作为预防措施，你须拥有潜在快照大小至少两倍的空间。如果可用空间不足，快照可能会损坏。

For this example, assume that a 100 GB volume named volume-00000001 was created for an instance while only 4 GB are used. This example uses these commands to back up only those 4 GB:

- lvm2命令。直接操作卷。
- kpartx命令。发现实例内部的所创建的分区表。
- tar命令。创建最小的备份。

- `sha1sum`命令。计算备份校验来检查其一致性。

你可以应用此过程给任何大小的卷。

欲备份块存储服务磁盘

1. 为一个正在使用中的卷创建一个快照

- a. 使用此命令来列出所有的卷:

```
# lvdisplay
```

- b. 创建快照；在卷已经挂接到实例上时你仍可使用:

```
# lvcreate --size 10G --snapshot --name volume-00000001-snapshot /dev/cinder-volumes/  
volume-00000001
```

使用 `--snapshot` 配置选项来告诉 LVM 您想要一个已有卷的快照。该命令包含了快照卷保留空间的大小、快照的名称和已有卷的路径。一般这个路径为 `/dev/cinder-volumes/VOLUME_NAME`。

大小不必与快照的卷相同。`--size` 参数定义了 LVM 为快照卷保留的空间。作为一个防范措施，大小应该与原本的卷相同，即使整个空间目前没有完全被快照所使用。

- c. 再次运行 `lvdisplay`命令来验证快照:

```
--- Logical volume ---  
LV Name      /dev/cinder-volumes/volume-00000001  
VG Name      cinder-volumes  
LV UUID      gI8hta-p21U-IW2q-hRN1-nTzN-UC2G-dKbdKr  
LV Write Access    read/write  
LV snapshot status  source of  
                  /dev/cinder-volumes/volume-00000026-snap [active]  
LV Status      available  
# open         1  
LV Size        15,00 GiB  
Current LE     3840  
Segments       1  
Allocation     inherit  
Read ahead sectors    auto  
- currently set to 256  
Block device   251:13  
  
--- Logical volume ---  
LV Name      /dev/cinder-volumes/volume-00000001-snap  
VG Name      cinder-volumes  
LV UUID      HlW3Ep-g5I8-KGQb-IRvi-IRYU-IiKe-wE9zYr  
LV Write Access    read/write  
LV snapshot status  active destination for /dev/cinder-volumes/volume-00000026  
LV Status      available  
# open         0  
LV Size        15,00 GiB  
Current LE     3840  
COW-table size  10,00 GiB  
COW-table LE    2560  
Allocated to snapshot 0,00%  
Snapshot chunk size 4,00 KiB  
Segments       1  
Allocation     inherit
```

```
Read ahead sectors   auto
- currently set to    256
Block device          251:14
```

2. 发现分区表

- a. 要使用命令 `tar` 来利用快照的话，将你的分区挂载到块存储服务的服务器上。

`kpartx` 工具发现和映射分区表，你可以使用它来查看实例中所创建的分区。不使用实例内部创建的分区，你就无法查看其内容，且无法创建效率较高的备份。

```
# kpartx -av /dev/cinder-volumes/volume-00000001-snapshot
```



注意

在基于Debian的发行版中，你可以使用 `apt-get install kpartx` 命令来安装 `kpartx`。

如果工具成功的发现和映射了分区表，没有错误返回。

- b. 要检查分区表映射，运行此命令：

```
$ ls /dev/mapper/nova*
```

你可以看到分区 `cinder--volumes-volume--00000001--snapshot1`。

如果你在此卷上创建了更多的分区的话，你可以看到多个分区；例如：`cinder--volumes-volume--00000001--snapshot2`, `cinder--volumes-volume--00000001--snapshot3`, 等等。

- c. 挂载用户的分区：

```
# mount /dev/mapper/cinder--volumes-volume--volume--00000001--snapshot1 /mnt
```

如果分区成功挂载，没有错误返回。

你可以直接访问实例内部的数据。如果有信息提示你无法挂载它，那么去看是否为快照分配了足够的空间，或者是 `kpartx` 命令无法找到分区表。

给快照分配更多空间，然后再次尝试。

3. 使用 `tar` 命令来创建归档

创建一个卷的备份：

```
$ tar --exclude="lost+found" --exclude="some/data/to/exclude" -czf volume-00000001.tar.gz -C /mnt/ /backup/destination
```

此命令创建了一个 `tar.gz` 文件，其包含了数据，且仅有数据。这确保了你不会因为空的扇区而浪费空间。

4. 校验计算

你须经常性的去校验你的备份文件。尤其是你通过网络传输了某个文件，你就需要运行校验计算来确保此文件没有在传输过程中有所损害。校验的文件ID是唯一的。如果校验值变得不同了，那就意味着文件已损坏。

运行此命令来为你的文件和保存的文件结果运行校验:

```
$ sha1sum volume-00000001.tar.gz > volume-00000001.checksum
```



注意

使用命令sha1sum要小心，因为完成计算所需要的时间直接与文件的大小成正比。

For files larger than around 4 to 6 GB, and depending on your CPU, the process might take a long time.

5. 时候清理工作

现在你已经有了有效性和一致性的备份了，使用此命令来清理下文件系统:

a. 卸载卷:

```
umount /mnt
```

b. 删除分区表:

```
kpartx -dv /dev/cinder-volumes/volume-00000001-snapshot
```

c. 删除快照:

```
lvremove -f /dev/cinder-volumes/volume-00000001-snapshot
```

重复上述步骤，针对所有的卷。

6. 自动化备份

随着越来越多的卷分配到块存储服务，你就需要将这些都自动化了，尤其是备份，脚本[SCR_5005_V01_NUAC-OPENSTACK-EBS-volumes-backup.sh](#)可以帮助你完成此任务。脚本所执行的操作来自上述的实例，但是其还提供了邮件的报告，以及基于backups_retention_days设置的备份。

在运行块存储服务的机器上启动此脚本。

此例展示了一个邮件报告：

```
Backup Start Time - 07/10 at 01:00:01
Current retention - 7 days

The backup volume is mounted. Proceed...
Removing old backups... : /BACKUPS/EBS-VOL/volume-00000019/volume-00000019_28_09_2011.tar.gz
/BACKUPS/EBS-VOL/volume-00000019 - 0 h 1 m and 21 seconds, Size - 3,5G

The backup volume is mounted. Proceed...
Removing old backups... : /BACKUPS/EBS-VOL/volume-0000001a/volume-0000001a_28_09_2011.tar.gz
/BACKUPS/EBS-VOL/volume-0000001a - 0 h 4 m and 15 seconds, Size - 6,9G
-----
Total backups size - 267G - Used space : 35%
Total execution time - 1 h 75 m and 35 seconds
```

脚本也会让你使用SSH登录到实例中，然后在实例中运行命令`mysqldump`。要使这正常工作，启用计算项目的密钥连接。如果你不打算运行`mysqldump`命令，你可以将`enable_mysql_dump=0` 添加到脚本中来禁用此功能。

迁移卷

OpenStack 可以在两个支持其卷类型的后端之间迁移卷。迁移一个卷会透明地将它的数据库从当前的后端移动到新的后端上。这是一个管理员功能，使用在一些功能上，包括存储撤离（用于维护或停用），或手动优化（例如，性能、可靠性或消耗）等。

对于一个迁移来说这些工作流时可能的：

1. 如果存储可以自行进行迁移，那么它是有机会这么做到的。这允许块设备存储驱动启用存储优化。如果后端不能够进行迁移，块设备存储会使用以下两种通用流程之一。
2. 如果卷没有挂接的话，块存储服务会创建一个卷且从旧的卷复制数据到新的卷。



注意

多数的后端都是支持此功能的，但不是全部都支持。请参阅[OpenStack 配置参考](#) 中驱动文档获取更多细节。

3. 如果是一个已经挂接到虚拟机实例的卷，块存储创建一个卷，然后调用计算从旧的复制数据到一个新的卷。目前此特性仅支持驱动为libvirt的计算。

举个例子，此场景展示了后端为两个LVM且可以讲挂接的卷从其中一个迁移到另外一个。此场景使用了第三个迁移流程。

首先，列出可用的后端:

```
# cinder get-pools
+-----+-----+
| Property | Value |
+-----+-----+
| name | server1@lvmstorage-1#lvmstorage-1 |
+-----+-----+
| Property | Value |
+-----+-----+
| name | server2@lvmstorage-2#lvmstorage-2 |
+-----+-----+
```



注意

仅有块存储V2 API支持 `get-pools`。

你可以类似如下获取可用的后端:

```
# cinder-manage host list
server1@lvmstorage-1 zone1
server2@lvmstorage-2 zone1
```

但是它需要在后面增加池的名称。例如，`server1@lvmstorage-1#zone1`。

接下来，作为管理员用户，你可以看到当前卷的状态(将例子中ID替换为你自己的):

```
$ cinder show 6088f80a-f116-4331-ad48-9afb0dfb196c
```

Property	Value
attachments	[...]
availability_zone	zone1
bootable	False
created_at	2013-09-01T14:53:22.000000
display_description	test
display_name	test
id	6088f80a-f116-4331-ad48-9afb0dfb196c
metadata	{}
os-vol-host-attr:host	server1@lvmstorage-1#lvmstorage-1
os-vol-mig-status-attr:migstat	None
os-vol-mig-status-attr:name_id	None
os-vol-tenant-attr:tenant_id	6bdd8f41203e4149b5d559769307365e
size	2
snapshot_id	None
source_volid	None
status	in-use
volume_type	None

注意这些属性:

- os-vol-host-attr:host - 卷的当前后端。
- os-vol-mig-status-attr:migstat - 卷迁移的状态(None意味着当前的过程没有做迁移)。
- os-vol-mig-status-attr:name_id - 卷在后端中所基于的卷的 ID。在卷迁移之前，卷在后端存储中的名称可能是基于卷 ID 的 (详见 volume_name_template 配置参数)。例如，如果 volume_name_template 作为默认只保留 (volume-%s)，您的第一个 LVM 后端有一个名为 volume-6088f80a-f116-4331-ad48-9afb0dfb196c 的逻辑卷。在迁移的过程中，如果您创建了一个卷并复制了数据，卷会得到新的名称，但保留了原本的 ID。这是由 name_id 显示的。



注意

如果你打算将某个存储节点撤下的话，你必须在执行完迁移之后停止cinder卷服务。

在运行 CentOS, Fedora, openSUSE, Red Hat Enterprise Linux, 或 SUSE Linux Enterprise,的节点上运行：

```
# service openstack-cinder-volume stop
# chkconfig openstack-cinder-volume off
```

在运行Ubuntu或Debian的节点上，运行:

```
# service cinder-volume stop
# chkconfig cinder-volume off
```

停止cinder卷服务能够防止已经分配到节点上的卷。

迁移此卷到第二个LVM后端:

```
$ cinder migrate 6088f80a-f116-4331-ad48-9afb0dfb196c server2@lvmstorage-2
```

你可以使用命令 `cinder show` 来查看迁移的状态。当正在迁移时，`migstat` 的属性会展示诸如 `migrating` 或 `completing`。若有错误发生，`migstat` 会显示为 `None`，且 `host` 属性会显示原来的主机。迁移成功到话，找本例而言输出类似如下结果：

Property	Value
attachments	[...]
availability_zone	zone1
bootable	False
created_at	2013-09-01T14:53:22.000000
display_description	test
display_name	test
id	6088f80a-f116-4331-ad48-9afb0dfb196c
metadata	{}
os-vol-host-attr:host	server2@lvmstorage-2#lvmstorage-2
os-vol-mig-status-attr:migstat	None
os-vol-mig-status-attr:name_id	133d1f56-9ffc-4f57-8798-d5217d851862
os-vol-tenant-attr:tenant_id	6bdd8f41203e4149b5d559769307365e
size	2
snapshot_id	None
source_volid	None
status	in-use
volume_type	None

注意 `migstat` 为 `None`，`host` 为新的主机，以及 `name_id` 由迁移所创建的卷的ID滞留值。如果你查看LVM后端的第二个值的话，你会看到逻辑卷 `volume-133d1f56-9ffc-4f57-8798-d5217d851862`。



注意

迁移对于非管理员用户是不可见的(举例来说，如通过卷的 `status`)。还有，在迁移的过程中一些操作是不被允许的，诸如挂接/分离卷或者删除卷。如果一个用户在迁移过程中执行了诸如此来的一些操作，只会返回错误。



注意

目前还不允许带有快照的卷做迁移。

从正在使用中的GlusterFS卷毫无压力的删除

配置 `cinder` 卷服务以使用 `GlusterFS` 来参与创建一个共享文件(例如，`/etc/cinder/glusterfs`)。这个共享文件列出 `cinder` 卷服务可以用于后端存储的每个 `GlusterFS` 卷(及其对应的存储服务器)。

要从已使用中将一个 `GlusterFS` 卷从后端移除，请从共享文件中删除卷对应的实体。完成后，重启块设备存储服务。

在 `CentOS`, `Fedora`, `openSUSE`, `Red Hat Enterprise Linux`, 或 `SUSE Linux Enterprise` 的节点上重启块存储服务的话，运行：

```
# for i in api scheduler volume; do service openstack-cinder-$i restart; done
```

在Ubuntu或Debian下重启块存储服务，运行：

```
# for i in api scheduler volume; do service cinder-${i} restart; done
```

重启块设备存储服务会阻止 cinder 卷服务导出已删除的 GlusterFS 卷。这会阻止任何实例在之前的挂载点中挂载该卷。

但是，移除的 GlusterFS 卷可能仍然挂载在实例的该挂载点上。通常这是卷已经挂载而它的入口已经从共享文件中删除的情况。如果这个情况发生了，您将必需在块设备存储服务重启后以常规的方式卸载卷。

备份和恢复卷

cinder 命令行接口提供了创建卷备份的工具。只要块设备存储数据库中的备份相关的数据库信息（或备份元数据）是完整的，您就可以从一个备份中恢复卷。

运行此命令以创建一个卷的备份：

```
$ cinder backup-create [--incremental] 值
```

VOLUME是卷的名称或ID，incremental是一个说明是否要执行增量备份的标志。

若没有incremental标志，默认会创建全备份，若有incremental标志，就会创建增量备份。



注意

incremental 标签只对块设备存储的 API v2 可用。您必须在 cinder 命令行接口中指定 [--os-volume-api-version 2] 以使用该参数。

增量备份是基于一个已经存在的打了时间戳的备份来进行的。其上游的备份可以是全备份也可以是增量的备份，主要取决于时间戳。



注意

第一个卷的备份是一个全备份。尝试进行没有基于已存在的备份来做增量备份会失败。

对于默认的Swift备份驱动来说有一个新的属性backup_swift_block_size被引进到cinder.conf。那就是多大byte的变化以跟踪增量备份。至于backup_swift_object_size，是Swift备份对象的大小，也以byte算，拥有多个backup_swift_block_size，backup_swift_block_size默认是32768，backup_swift_object_size默认是52428800。

此命令会返回一个备份ID。当恢复卷时使用此备份ID：

```
$ cinder backup-restore BACKUP_ID
```

当从一个完全备份恢复时，也会是一个完全的恢复。

当从一个增量备份恢复时，会列出其所基于动上游的备份的ID。一个全部的恢复的执行首先得基于全备份，然后是恢复基于增量的备份，就是这么个顺序。

由于卷备份是依赖于块设备存储数据库的，您还必须定期备份您的块设备存储数据库来保证数据可恢复。



注意

您可以选择导出和保存所选择的卷备份的元数据。这么做就除去了备份整个块设备存储数据库的需要。如果您仅仅想要备份卷的一小部分来挽回一个灾难性的数据库故障，那么这将是很有用的。

如果您在创建卷规格时指定了一个 UUID 密钥，那么当您备份和恢复卷时，备份元数据会保证密钥保持有效。

关于如何导出和导入卷备份元数据的更多内容，请参阅[“导入和到处备份元数据”一节 \[156\]](#)。

默认情况下，备份的仓库会使用swift对象存储。

如果您想转而使用 NFS 导出作为备份仓库，请添加下列配置选项到 `cinder.conf` 文件的 [DEFAULT] 小节中，并重启块设备存储服务：

```
backup_driver = cinder.backup.drivers.nfs
backup_share = HOST:EXPORT_PATH
```

对于 `backup_share` 选项，请将 `HOST` 替换为可以被 DNS 解析的主机名或 NFS 共享存储服务器的 IP 地址，将 `EXPORT_PATH` 替换为共享的路径。如果您的环境需要指定共享的非默认选项，请设置如下：

```
backup_mount_options = MOUNT_OPTIONS
```

`MOUNT_OPTIONS`是逗号分隔的字符，即NFS挂载的属性，更多细节请参考NFS 文档页面。

有多个默认的有关属性会覆盖掉你环境对应的值：

```
backup_compression_algorithm = zlib
backup_sha_block_size_bytes = 32768
backup_file_size = 1999994880
```

属性`backup_compression_algorithm`可以设置为`bz2` 或 `None`。后者用于当服务器提供了共享作为存储的仓库时会用到后者，此仓库上的备份数据执行重复数据删除或压缩。

选项 `backup_file_size` 必需是多重的 `backup_sha_block_size_bytes`。它有效地在您的环境中使用了最大文件大小，以保持备份数据。大于它的卷会被保存到备份仓库的多文件中。`backup_sha_block_size_bytes` 选项决定了 `cinder` 卷所备份的块设备的大小，它计算数字签名，以启用增长的备份容量。

导入和到处备份元数据

卷的备份职能存放在相同的块存储服务下。这是因为恢复的时候需要数据库中的元数据，而这是有块存储服务所控制的。



注意

关于如何备份和恢复一个卷的信息，请参考[“备份和恢复卷”一节 \[155\]](#)。

当然，你可以将卷备份的元数据导出来。要这么的话，以OpenStack admin 用户运行下面命令(当然，需要在创建卷备份之后)：

```
$ cinder backup-export BACKUP_ID
```

BACKUP_ID 备份卷的ID。此命令须返回备份相应的作为编码字符元数据的数据库信息。

导出并保存这个编码字符串元数据让您能够完全恢复备份，即使是在发生了灾难性的数据库故障的情况下。这除去了备份整个块设备存储数据库的需要，特别是您只需要保持完整的卷的一小部分备份时。

如果您替换了卷上的加密，当创建卷是如果制定了 UUID 加密关键字，您恢复卷时加密仍然会被替换。使用备份元数据支持，为卷设置 UUID 关键字会保留加密，并在您的证书下是可以访问的。

另外，拥有一个卷的备份机器备份元数据也提供了卷的可移植性。特别是备份一个卷并导出起元数据时您可以在一个完全不同的块设备存储数据库中恢复卷，或这甚至在一个不同的云服务上也是可以的。为了达到这个目的，首先要导入备份元数据到块设备存储数据库中，然后将其恢复。

要导入备份元数据，以OpenStackadmin运行下面命令：

```
$ cinder backup-import METADATA
```

METADATA 早些时候抛出的用于备份元数据的地。

一旦你完成了备份元数据到一个块设备的数据库的导入，就可以做恢复了（“[备份和恢复卷](#)”一节 [155]）。

使用LIO 所支持的iSCSI

iscsi_helper默认的工具是 tgtadm。要使用 LIO iSCSI,安装python-rtplib 软件包，然后在文件cinder.conf中设置iscsi_helper=lioadm。

一旦配置完成，你就可以使用命令 cinder-rtstool来管理卷了。此命令能够让你创建，删除，和验证卷，以及决定target，且可添加iSCSI initiator到系统。

配置和使用卷的权重

OpenStack块存储让你在free_capacity 和 allocated_capacity卷的后端做出选择。卷的权重号可让调度器基于卷后端的卷号来选择选择一个卷的后端。这样点话就意味着可以改进卷后端的I/O 平衡和卷的I/O 性能。

启用卷的权重

要启用卷的权重号，在cinder.conf文件设置scheduler_default_weighters 的VolumeNumberWeigher 标记，定义VolumeNumberWeigher 作为所选择的权重。

配置多后端存储

要配置VolumeNumberWeigher，使用LVMISCSIDriver来作为卷的驱动。

This configuration defines two LVM volume groups: stack-volumes with 10 GB capacity and stack-volumes-1 with 60 GB capacity. This example configuration defines two back ends:


```
scheduler_default_weighers=VolumeNumberWeigher
enabled_backends=lvmdriver-1,lvmdriver-2
[lvmdriver-1]
volume_group=stack-volumes
volume_driver=cinder,volume.drivers.lvm,LVMISCSIDriver
volume_backend_name=LVM_iSCSI

[lvmdriver-2]
volume_group=stack-volumes-1
volume_driver=cinder,volume.drivers.lvm,LVMISCSIDriver
volume_backend_name=LVM_iSCSI
```

卷类型

在块存储中定义一卷类型:

```
$ cinder type-create lvm
```

创建一个额外的说明，关联到卷的类型到后端的名称：

```
$ cinder type-key lvm set volume_backend_name=LVM_iSCSI
```

此例创建了一个lvm卷类型，volume_backend_name=LVM_iSCSI是其额外的说明。

用量

要创建6个1GB卷，运行6遍命令 cinder create --volume-type lvm 1：

```
$ cinder create --volume-type lvm 1
```

此命令在stack-volumes中创建了一个卷，在stack-volumes-1中也创建了一个卷。

列出可用卷:

```
# lvs
LV          VG          Attr   LSize Pool Origin Data%  Move Log Copy%  Convert
volume-3814f055-5294-4796-b5e6-1b7816806e5d stack-volumes -wi-a---- 1.00g
volume-72cf5e79-99d2-4d23-b84e-1c35d3a293be stack-volumes -wi-a---- 1.00g
volume-96832554-0273-4e9d-902b-ad421dfb39d1 stack-volumes -wi-a---- 1.00g
volume-169386ef-3d3e-4a90-8439-58ceb46889d9 stack-volumes-1 -wi-a---- 1.00g
volume-460b0bbb-d8a0-4bc3-9882-a129a5fe8652 stack-volumes-1 -wi-a---- 1.00g
volume-9a08413b-0dbc-47c9-afb8-41032ab05a41 stack-volumes-1 -wi-a---- 1.00g
```

一致性组

OpenStack块存储服务支持一致性组，为一致性组创建快照来实现的。此特性利用了存储级别的一致性技术。它允许在同一个一致性组在同一个点快照多个卷从而实现数据的一致性。使用块存储的命令即可操作一致性组。



注意

只有块存储 V2 API支持一致性组。当使用块存储命令行来操作一致性组时，你可以指定参数--os-volume-api-version 2。

在使用一致性组之前，要确保所使用的块存储驱动是支持一致性组的，这可以通过阅读块存储帮助文档或者咨询驱动的维护者。只有一小部分的驱动实现了此特性。默认的LVM驱动目前还不支持一致性组，因为在此级别还没有一致性技术的实现。

在使用一致性组之前，你必须为一致性组API更改规则，这在配置文件/etc/cinder/policy.json 中修改。默认情况下，一致性组API是禁用的。在运行一致性组操作之前启用它。

这里是一致性组已经存在的规则：

```
"consistencygroup:create": "group:nobody",
"consistencygroup:delete": "group:nobody",
"consistencygroup:get": "group:nobody",
"consistencygroup:get_all": "group:nobody",
"consistencygroup:create_cgsnapshot": "group:nobody",
"consistencygroup:delete_cgsnapshot": "group:nobody",
"consistencygroup:get_cgsnapshot": "group:nobody",
"consistencygroup:get_all_cgsnapshots": "group:nobody",
```

通过删除 group:nobody 来启用这些API，按照下面来变更：

```
"consistencygroup:create": "",
"consistencygroup:delete": "",
"consistencygroup:update": "",
"consistencygroup:get": "",
"consistencygroup:get_all": "",
"consistencygroup:create_cgsnapshot": "",
"consistencygroup:delete_cgsnapshot": "",
"consistencygroup:get_cgsnapshot": "",
"consistencygroup:get_all_cgsnapshots": "",
```

在变更规则后要重启块存储API服务。

下面的一致性组操作时被支持的：

- 创建一个一致性组，指定卷的类型。



注意

一个一致性组可支持多于一个卷类型。调度器会响应所发现的后端所支持的所有给定的卷的类型。



注意

由同一个后端，一个一致性组只包含卷的托管。



注意

一个一致性组在创建后是空的，后续创建卷然后才能加入进来。

- 展示一个一致性组。
- 列出一致性组。
- 创建一个卷然后将其添加到一致性组，需要指定卷的类型和一致性组ID。
- 为一个一致性组创建一快照：
- 展现一致性组的快照：
- 列出一致性组快照：

- 删除一致性组的快照:
- 删除一致性组。
- 修改一个一致性组。
- 从另外的一致性组的快照创建一个一致性组。

如果卷在一致性组中，下面的操作是不被允许的：

- 卷迁移。
- 卷类型
- 卷删除。



注意

删除一个一致性组，其包含的所有的卷都会被删除。

若卷的快照在一致性组的快照中，以下操作时不被允许的：

- 卷快照删除。



注意

删除一个一致性组快照，其包含的所有的卷的快照都会被删除。

一致性组操作的细节如下所示。

创建一个一致性组：

```
cinder consisgroup-create
[--name name]
[--description description]
[--availability-zone availability-zone]
volume-types
```



注意

需要参数volume-types，可以是卷类型的名称或UUID的列表，中间使用逗号隔开，注意不是空格。举例来说， volumetype1,volumetype2,volumetype3。

```
$ cinder consisgroup-create --name bronzeCG2 volume_type_1
```

Property	Value
availability_zone	nova
created_at	2014-12-29T12:59:08.000000
description	None
id	1de80c27-3b2f-47a6-91a7-e867cbe36462
name	bronzeCG2
status	creating

显示一致性组:

```
$ cinder consisgroup-show 1de80c27-3b2f-47a6-91a7-e867cbe36462
```

Property	Value
availability_zone	nova
created_at	2014-12-29T12:59:08.000000
description	None
id	2a6b2bda-1f43-42ce-9de8-249fa5cbae9a
name	bronzeCG2
status	available

列出一致性组:

```
$ cinder consisgroup-list
```

ID	Status	Name
1de80c27-3b2f-47a6-91a7-e867cbe36462	available	bronzeCG2
3a2b3c42-b612-479a-91eb-1ed45b7f2ad5	error	bronzeCG

创建一个卷且将其添加到一个一致性组:



注意

当创建一个卷然后将其添加到一致性组时，卷的类型和一致性组的ID是必须提供的。这是因为一致性组可以支持多个卷类型。

```
$ cinder create --volume-type volume_type_1 --name cgBronzeVol --consisgroup-id 1de80c27-3b2f-47a6-91a7-e867cbe36462 1
```

Property	Value
attachments	[]
availability_zone	nova
bootable	false
consistencygroup_id	1de80c27-3b2f-47a6-91a7-e867cbe36462
created_at	2014-12-29T13:16:47.000000
description	None
encrypted	False
id	5e6d1386-4592-489f-a56b-9394a81145fe
metadata	{}
name	cgBronzeVol
os-vol-host-attr:host	server-1@backend-1#pool-1
os-vol-mig-status-attr:migstat	None
os-vol-mig-status-attr:name_id	None
os-vol-tenant-attr:tenant_id	1349b21da2a046d8aa5379f0ed447bed
os-volume-replication:driver_data	None
os-volume-replication:extended_status	None
replication_status	disabled
size	1
snapshot_id	None
source_volid	None
status	creating
user_id	93bdea12d3e04c4b86f9a9f172359859
volume_type	volume_type_1

为一个一致性组创建快照:

```
$ cinder cgsnapshot-create 1de80c27-3b2f-47a6-91a7-e867cbe36462
```

Property	Value
consistencygroup_id	1de80c27-3b2f-47a6-91a7-e867cbe36462
created_at	2014-12-29T13:19:44.000000
description	None
id	d4aff465-f50c-40b3-b088-83feb9b349e9
name	None
status	creating

展现一致性组的快照:

```
$ cinder cgsnapshot-show d4aff465-f50c-40b3-b088-83feb9b349e9
```

列出一致性组快照:

```
$ cinder cgsnapshot-list
```

ID	Status	Name
6d9dfb7d-079a-471e-b75a-6e9185ba0c38	available	None
aa129f4d-d37c-4b97-9e2d-7effda29de0	available	None
bb5b5d82-f380-4a32-b469-3ba2e299712c	available	None
d4aff465-f50c-40b3-b088-83feb9b349e9	available	None

删除一致性组的快照:

```
$ cinder cgsnapshot-delete d4aff465-f50c-40b3-b088-83feb9b349e9
```

删除一致性组:



注意

当卷在一致性组时需要强制标记。

```
$ cinder consisgroup-delete --force 1de80c27-3b2f-47a6-91a7-e867cbe36462
```

修改一个一致性组：

```
cinder consisgroup-update
[--name NAME]
[--description DESCRIPTION]
[--add-volumes UUID1,UUID2,.....]
[--remove-volumes UUID3,UUID4,.....]
CG
```

需要参数CG，它可以使一致性组的名称或UUID。UUID1,UUID2,.....是由逗号隔开的已经添加到一致性组的卷，默认为None。UUID3,UUID4,.....是已经删除的一致性组的卷，默认也为None。

```
$ cinder consisgroup-update --name 'new name' --description 'new description' --add-volumes
0b3923f5-95a4-4596-a536-914c2c84e2db,1c02528b-3781-4e32-929c-618d81f52cf3 --remove-volumes
8c0f6ae4-efb1-458f-a8fc-9da2afcc5fb1,a245423f-bb99-4f94-8c8c-02806f9246d8 1de80c27-3b2f-47a6-91a7-
e867cbe36462
```

从另外的一致性组的快照创建一个一致性组。

```
cinder consisgroup-create-from-src
```

```
[--cgsnapshot CGSNAPSHOT]
[--name NAME]
[--description DESCRIPTION]
```

参数CGSNAPSHOT是一致性组的快照的名称或UUID。

```
$ cinder consisgroup-create-from-src --cgsnapshot 6d9dfb7d-079a-471e-b75a-6e9185ba0c38 --name 'new cg'
--description 'new cg from cgsnapshot'
```

为调度配置和使用驱动过滤器和权重

OpenStack 块设备存储允许您选择一个基于卷后端指定属性的卷后端，这些属性是通过 DriverFilter 和 GoodnessWeigher 来调度的。驱动过滤器和权重调度可以帮助保证调度器选择基于请求的卷属性和各种卷后端指定属性的最佳卷后端。

什么是驱动过滤和权重，以及何时使用它

The driver filter and weigher gives you the ability to more finely control how the OpenStack Block Storage scheduler chooses the best back end to use when handling a volume request. One example scenario where using the driver filter and weigher can be if a back end that utilizes thin-provisioning is used. The default filters use the "free capacity" property to determine the best back end, but that is not always perfect. If a back end has the ability to provide a more accurate back-end specific value you can use that as part of the weighing. Another example of when the driver filter and weigher can prove useful is if a back end exists where there is a hard limit of 1000 volumes. The maximum volume size is 500 GB. Once 75% of the total space is occupied the performance of the back end degrades. The driver filter and weigher can provide a way for these limits to be checked for.

启用驱动过滤和权重

要启用驱动过滤器，请将 cinder.conf 文件中的 scheduler_default_filters 选项设置为 DriverFilter 或将其添加到其他已有的过滤器列表中。

要启用 goodness 过滤器作为权重，请将 cinder.conf 文件中的 scheduler_default_weighers 设置为 GoodnessWeigher 或将其添加到其他已有权重的列表中。

您可以选择不包含 GoodnessWeigher 的 DriverFilter，反之亦然。过滤器或权重协同工作，在帮助调度器选择最佳后端过程中创造了最佳利益。



重要

DriverFilter 和 GoodnessWeigher 的支持对后端是可选的。如果您使用了一个不支持过滤器和权重功能的后端，那么您可能无法从中得到完全的利益。

配置文件cinder.conf实例:

```
scheduler_default_filters = DriverFilter
scheduler_default_weighers = GoodnessWeigher
```



注意


使用 OpenStack 中的其他过滤器和权重与这些自定义的过滤器和权重结合使用是非常有用的。例如，CapacityFilter 和 CapacityWeigher 可以结合使用。

定义你自己的过滤器和合适的函数

您可以通过使用各种 OpenStack 块设备存储可用的属性来定义自己的过滤器和 goodness 功能。属性的暴露包括关于卷请求的信息、volume_type 的设置和后端关于驱动的指定信息。所有这些允许了很多对卷请求的最佳后端的决定的控制。

filter_function 是一个字符串，定义了一个等式，它会决定一个后端在调度器中是否应该看成一个潜在的候选。

goodness_function 选项时一个字符串，定义了一个等式，它会评估潜在主机的质量 (0 到 100，0 最低，100 最高)。




重要

如果您没有定义的话，过滤器和 goodness 功能的默认值会用于所有后端。如果需要完全控制，那么过滤器和 goodness 功能应该在每个后端的 cinder.conf 文件中定义。

在过滤器中所支持的操作和良好的功能

下面是一个可以使用在您所创建的自定义的过滤器和 goodness 功能中目前所有可用的操作的表格：

操作	类型
+, -, *, /, ^	标准数字运算符
not, and, or, &, , !	逻辑
>, >=, <, <=, ==, <>, !=	等于
!, ~	标志
x ? a : b	三元
abs(x), max(x, y), min(x, y)	数学函数



小心

您所定义的过滤器和 goodness 字符串时出现的语法错误会在卷请求时引起错误抛出。

当创建自定义的功能时可用的属性

有各种个样的属性可以用在 filter_function 或 goodness_function 字符串中。这些属性允许访问卷的信息、qos 配置、额外规格等等。

这里是属性的一个列表，以及当前可用的子属性：

- stats—These are the host stats for a back end.

主机	主机名称。
volume_backend_name	后端卷名称
vendor_name	供应商名称
driver_version	驱动版本。
storage_protocol	存储协议。

QoS_support	布尔值，决定是否支持QoS。
total_capacity_gb	以GB计的总容量。
allocated_capacity_gb	以GB计的已分配容量。
reserved_percentage	保留存储的百分比。

- capabilities—These are the capabilities specific to a back end.

在此可用的属性是由指定的后端决定的，您为后端创建了过滤器和 goodness 功能。一些后端可能包含了这里的一些可用属性。

- volume—The requested volume properties.

状态	所请求的卷的状态。
volume_type_id	卷类型的ID。
display_name	卷的显示名称。
volume_metadata	任何卷拥有的元数据。
预订	任何卷的预订。
user_id	卷的用户ID。
attach_status	卷的挂接状态。
display_description	卷的描述显示。
id	卷的ID。
replication_status	卷的复制状态。
snapshot_id	卷的快照ID。
encryption_key_id	卷的加密键ID。
source_volid	源卷ID。
volume_admin_metadata	此卷的任何管理的元数据。
source_replicaid	复制源ID。
consistencygroup_id	一致性组ID。
大小	以GB为单位的卷大小。
元数据	常规元数据。

在此使用最多的属性应该是 size 子属性了。

- extra—The extra specs for the requested volume type.

查看卷类型的可用属性，运行：

```
$ cinder extra-specs-list
```

- qos—The current QoS specs for the requested volume type.

查看卷类型的可用属性，运行：

```
$ cinder qos-list
```

为了以自定义的字符访问这些属性，使用下面的格式：

<property>.<sub_property>

驱动过滤和权重使用实例

下面是一些分开或共同使用过滤器和权重以及使用指定驱动属性的示例。

为 cinder.conf 配置定制的过滤器功能实例：

```
[default]
scheduler_default_filters = DriverFilter
enabled_backends = lvm-1, lvm-2

[lvm-1]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
filter_function = "volume.size < 10"

[lvm-2]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
filter_function = "volume.size >= 10"
```

The above example will filter volumes to different back ends depending on the size of the requested volume. Default OpenStack Block Storage scheduler weighing is done. Volumes with a size less than 10 GB are sent to lvm-1 and volumes with a size greater than or equal to 10 GB are sent to lvm-2.

针对goodness功能所定制的cinder.conf 配置文件实例：

```
[default]
scheduler_default_weighers = GoodnessWeigher
enabled_backends = lvm-1, lvm-2

[lvm-1]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
goodness_function = "(volume.size < 5) ? 100 : 50"

[lvm-2]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
goodness_function = "(volume.size >= 5) ? 100 : 25"
```

The above example will determine the goodness rating of a back end based off of the requested volume's size. Default OpenStack Block Storage scheduler filtering is done. The

example shows how the ternary if statement can be used in a filter or goodness function. If a requested volume is of size 10 GB then lvm-1 is rated as 50 and lvm-2 is rated as 100. In this case lvm-2 wins. If a requested volume is of size 3 GB then lvm-1 is rated 100 and lvm-2 is rated 25. In this case lvm-1 would win.

针对同时支持goodness和过滤器功能所定制的cinder.conf 配置文件实例：

```
[default]
scheduler_default_filters = DriverFilter
scheduler_default_weighers = GoodnessWeigher
enabled_backends = lvm-1, lvm-2

[lvm-1]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
filter_function = "stats.total_capacity_gb < 500"
goodness_function = "(volume.size < 25) ? 100 : 50"

[lvm-2]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
filter_function = "stats.total_capacity_gb >= 500"
goodness_function = "(volume.size >= 25) ? 100 : 75"
```

上面的例子结合了前两个例子的方法。现在最佳后端将基于总的后端容量和所请求的卷大小来决定。

为 cinder.conf配置访问驱动指定属性功能实例：

```
[default]
scheduler_default_filters = DriverFilter
scheduler_default_weighers = GoodnessWeigher
enabled_backends = lvm-1, lvm-2, lvm-3

[lvm-1]
volume_group = stack-volumes-lvmdriver-1
volume_driver = cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name = lvmdriver-1
filter_function = "volume.size < 5"
goodness_function = "(capabilities.total_volumes < 3) ? 100 : 50"

[lvm-2]
volume_group = stack-volumes-lvmdriver-2
volume_driver = cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name = lvmdriver-2
filter_function = "volumes.size < 5"
goodness_function = "(capabilities.total_volumes < 8) ? 100 : 50"

[lvm-3]
volume_group = stack-volumes-lvmdriver-3
volume_driver = cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name = lvmdriver-3
goodness_function = "55"
```

The above is an example of how back-end specific properties can be used in the filter and goodness functions. In this example the LVM driver's 'total_volumes' capability is being used to determine which host gets used during a volume request. In the above example, lvm-1 and lvm-2 will handle volume requests for all volumes with a size less than 5 GB. The lvm-1

host will have priority until it contains three or more volumes. After than lvm-2 will have priority until it contains eight or more volumes. The lvm-3 will collect all volumes greater or equal to 5 GB as well as all volumes once lvm-1 and lvm-2 lose priority.

卷复制的带宽限制率

当您从镜像或现有卷中创建了一个新卷，或上传了一个卷镜像到镜像服务中，大数据的复制会给磁盘和网络带宽带来压力。为了减轻从实例访问数据的迟钝，OpenStack 块设备存储支持对卷数据复制的带宽进行速率限制。

配置卷复制带宽的限制

要配置卷复制带宽限制，请为每个后端设置 cinder.conf 文件的配置组中的 volume_copy_bps_limit 选项。该选项使用最大带宽的整数，允许卷数据以字节每秒复制。如果该选项设置为 0，那么速率限制被禁用。

当多个卷数据的复制操作是在同一个后端进行时，需指定分离每个复制的带宽。

将lvmdriver-1 的卷复制带宽限制到最大100 MiB/s的cinder.conf 配置文件实例：

```
[lvmdriver-1]
volume_group=cinder-volumes-1
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI
volume_copy_bps_limit=104857600
```



注意

此特性需要libcgroup来设置blkio的cgroup，来限制磁盘I/O的带宽。在Debian和Ubuntu下libcgroup是由软件包cgroup-bin提供的。在Fedora，Red Hat Enterprise Linux，CentOS，openSUSE，以及SUSE Linux Enterprise下由软件包libcgroup-tools提供。



注意

一些后端所使用的远程文件系统如NFS在此特性中时不被支持的。

在精简置配中超额认购

OpenStack 块设备存储允许您使用超分配来为轻量供应选择一个基于虚拟容量的后端。

一个相关的实现提供了默认的 LVM 驱动。下面的插图使用了 LVM 驱动作为示例。

配置超额设置

要在轻量供应中支持超分配，需要设置 cinder.conf 中的 max_over_subscription_ratio 标签。这是在使用轻量供应时的一个浮动的超分配比率的表示。默认比率为 20.0，意味着提供的容量可以是总的物理容量的 20 倍。10.5 比率意味着提供的容量可以是总物理容量的 10.5 倍。1.0 比率意味着提供的容量不可以超过总的物理容量。小于 1.0 的比率会被无视并使用默认值。



注意

启用了存储后端后，`max_over_subscription_ratio` 可以为每个后端配置。它会作为一个相关实现提供，并由 LVM 驱动使用。但是，这不是驱动使用该选项的 `cinder.conf` 中的依赖。`max_over_subscription_ratio` 用于配置一个后端。对于支持每个后端中的多池的驱动，它可以为每个池报告该比率。LVM 驱动不支持多池。

现有的 `reserved_percentage` 用于防止过度供应。该标签表示保留的后端容量百分比。



注意

该变化显示了 `reserved_percentage` 是如何使用的。它在以前测量了空闲的容量。现在它测量了总的容量。

能力

驱动程序可以报告后端或池的以下功能：

```
thin_provisioning_support=True(or False)
thick_provisioning_support=True(or False)
provisioned_capacity_gb=PROVISIONED_CAPACITY
max_over_subscription_ratio=MAX_RATIO
```

`PROVISIONED_CAPACITY` 是可见的已分配的空间，表示已经分配了多少空间，而 `PROVISIONED_CAPACITY` 是最大的超分配比率。对于 LVM 驱动，其为 `cinder.conf` 中的 `max_over_subscription_ratio`。

两个容量被添加到此处以允许后端或池要求支持轻量级供应或重量级供应，或者二者都要求。

如果在 `cinder.conf` 中 `lvm_type` 的值是 `thin` 的话，LVM 驱动会报告 `thin_provisioning_support=True` 和 `thick_provisioning_support=False`，其它情况下，会报告 `thin_provisioning_support=False` 和 `thick_provisioning_support=True`。

云硬盘类型扩展规格

如果卷类型作为卷创建请求的一部份提供，它可以有以下额外规格定义：

```
'capabilities:thin_provisioning_support': '<is> True' or '<is> False'
'capabilities:thick_provisioning_support': '<is> True' or '<is> False'
```



注意

`capabilities` 范围关键字在 `thin_provisioning_support` 和 `thick_provisioning_support` 之前是不需要的。所以下面的方式也可用：

```
'thin_provisioning_support': '<is> True' or '<is> False'
'thick_provisioning_support': '<is> True' or '<is> False'
```

上面的额外规格用于调度器，来寻找支持轻量级供应、重量级供应或二者都支持的后端，以匹配指定卷类型的需要。

容量过滤

在容量过滤中，`max_over_subscription_ratio` 是用于当后端选择了 `thin_provisioning_support` 为 `True`，且 `max_over_subscription_ratio` 大于 `1.0`。

容量权重

在容量权重中，虚拟空闲容量用于排行，如果 `thin_provisioning_support` 是 `True` 的情况，其它情况下，像以前一样使用真实的空闲容量。

用户安装Troubleshoot

此节是一些有用的小帖子，望能帮到用户在安装块存储时自己解决掉。

块存储配置的故障排查

多数引起块存储的错误是由于错误的卷配置，进而导致卷的创建失败。要解决这些问题，检阅这些日志：

- cinder-api 日志 (`/var/log/cinder/api.log`)
- cinder-volume 日志 (`/var/log/cinder/volume.log`)

cinder-api 日志对于是否有连接到端点的错误非常的有帮助。如果你发送了一个创建卷的请求，但是其失败了，检阅cinder-api 日志来判断请求是否到达了块存储服务。如果请求被记录而且你没有看到任何的错误或回溯信息，请检查cinder-volume的错误或回溯信息。



注意

cinder-api日志中会列出创建命令。

这些 `cinder.openstack.common.log` 文件中的项可用于块存储配置的故障排查。

```
# Print debugging output (set logging level to DEBUG instead
# of default WARNING level). (boolean value)
#debug=false

# Print more verbose output (set logging level to INFO instead
# of default WARNING level). (boolean value)
#verbose=false

# Log output to standard error (boolean value)
#use_stderr=true

# Default file mode used when creating log files (string
# value)
#logfile_mode=0644

# format string to use for log messages with context (string
# value)
#logging_context_format_string=%(asctime)s,%(msecs)03d %(levelname)s %(name)s [%(request_id)s
%(user)s %(tenant)s] %(instance)s%(message)s

# format string to use for log mes #logging_default_format_string=%(asctime)s,%(msecs)03d %(process)d
%(levelname)s %(name)s [-] %(instance)s%(message)s
```

```

# data to append to log format when level is DEBUG (string
# value)
#logging_debug_format_suffix=%(funcName)s %(pathname)s:%(lineno)d

# prefix each line of exception output with this format
# (string value)
#logging_exception_prefix=%(asctime)s,%(msecs)03d %(process)d TRACE %(name)s %(instance)s

# list of logger=LEVEL pairs (list value)
#default_log_levels=amqpplib=WARN,sqlalchemy=WARN,boto=WARN,suds=INFO,keystone=INFO,eventlet.wsgi.
server=WARNsages without context
# (string value)

# If an instance is passed with the log message, format it
# like this (string value)
#instance_format="[instance: %(uuid)s]"

# If an instance UUID is passed with the log message, format
# it like this (string value)
# A logging.Formatter log message format string which may use
# any of the available logging.LogRecord attributes. Default:
# %(default)s (string value)
#log_format=%(asctime)s %(levelname)s %(name)s %(message)s

# Format string for %(asctime)s in log records. Default:
# %(default)s (string value)
#log_date_format=%Y-%m-%d %H:%M:%S

# (Optional) Name of log file to output to. If not set,
# logging will go to stdout. (string value)
#log_file=<None>

# (Optional) The directory to keep log files in (will be
# prepended to --log-file) (string value)
#log_dir=<None>
#instance_uuid_format="[instance: %(uuid)s]"

# If this option is specified, the logging configuration file
# specified is used and overrides any other logging options
# specified. Please see the Python logging module
# documentation for details on logging configuration files.
# (string value) # Use syslog for logging. (boolean value)
#use_syslog=false

# syslog facility to receive log lines (string value)
#syslog_log_facility=LOG_USER
#log_config=<None>

```

这些常见的问题一般是发生在配置的时候，要更改它，使用这些建议的解决方案。

- state_path 和 volumes_dir 设置带来的错误。

OpenStack块存储使用tgt作为默认的iSCSI助手以及实现持久的目标。这意味着tgt重启的情况甚至是节点重启，原来节点上已存在的卷回自动的基于它们原来的IQN存储。

为了使其成为可能，iSCSI target 信息需要在创建时保存在一个文件中，当重启 tgt 守护进程时它可以被查询到。默认情况下，块设备存储会使用 state_path 变量，它在用 YUM 或 APT 安装时会设置在 /var/lib/cinder/ 中。下一个部分是 volumes_dir 变量，默认情

况下这仅仅追加一个 "volumes" 目录到 state_path 中。其结果是一个文件树 /var/lib/cinder/volumes/。

安装程序掌控了这一切的话，将会出错。如果你创建卷遇到了错误且此目录不存在，你须查看 cinder-volume 日志中的错误信息，其中有说 volumes_dir 不存在，然后就须提供关于寻找路径的信息。

- 持久的 tgt 包括的文件。

除了 volumes_dir 选项之外，iSCSI target 驱动也需要配置为在正确的地方寻找持久文件。这是 /etc/tgt/conf.d 文件中的一个简单入口，这是您在安装 OpenStack 时就设置好的。如果出现了问题，请检查您是否有 /etc/tgt/conf.d/cinder.conf 文件。

加入文件不存在，使用此命令来创建：

```
# echo 'include /var/lib/cinder/volumes/*' >> /etc/tgt/conf.d/cinder.conf
```

- cinder-api 日志中没有挂接调用的标志。

这可能需要微调 nova.conf 文件。确保 nova.conf 有此项：

```
volume_api_class=nova.volume.cinder.API
```

- 在文件 cinder-volume.log 中有创建 iscsi target 失败的错误。

```
2013-03-12 01:35:43 1248 TRACE cinder.openstack.common.rpc.amqp ISCSITargetCreateFailed: Failed to create iscsi target for volume volume-137641b2-af72-4a2f-b243-65fdccd38780.
```

You might see this error in cinder-volume.log after trying to create a volume that is 1 GB. To fix this issue:

更改 /etc/tgt/targets.conf 的内容，将其内容为 include /etc/tgt/conf.d/*.conf 改变为 include /etc/tgt/conf.d/cinder_tgt.conf，如下所示：

```
include /etc/tgt/conf.d/cinder_tgt.conf
include /etc/tgt/conf.d/cinder.conf
default-driver iscsi
```

重启 tgt 和 cinder-* 服务，以应用新的配置。

多路径调用退出失败

问题

多路径调用退出失败。这是计算节点日志里生成的警告，是没有在计算节点安装软件包 multipath-tools 所导致。这虽然是个可选的软件包，但是若没有安装卷的挂接就不能正常工作。如果在计算节点安装了 multipath-tools 软件包，它也就是用来作卷的挂接。其在系统的信息中的 ID 是唯一的。

```
WARNING nova.storage.linuxscsi [req-cac861e3-8b29-4143-8f1b-705d0084e571 admin
admin|req-cac861e3-8b29-4143-8f1b-705d0084e571 admin admin] Multipath call failed exit
(96)
```

解决方案

在计算节点中运行下面的命令来安装软件包 multipath-tools。

```
# apt-get install multipath-tools
```

EqualLogic存储的报告中的卷大小差异寻址

问题

在EqualLogic(EQL)存储的实际卷的大小和镜像服务中的镜像大小之间有点差异，其中镜像服务是OpenStack数据库所报告。如果用户从镜像创建了卷然后上传到EQL卷(通过镜像服务)会导致某些困惑。镜像大小要略大于目标卷的大小；这是因为EQL所报出的大小是EQL使用了其内部的卷元数据。

复制这个问题遵循下列程序中的步骤。

此步骤假设EQL阵列已经预先分配，且连接到EQL阵列所对应的配置设置已经包含进了/etc/cinder/cinder.conf。

1. 创建一个新的卷。记下卷的ID和大小。下面的例子ID和带下分别是74cf9c04-4543-47ae-a937-a9b7c6c921e7 和 1：

```
$ cinder create --display-name volume1 1
```

Property	Value
attachments	[]
availability zone	nova
bootable	false
created_at	2014-03-21T18:31:54.248775
display_description	None
display_name	volume1
id	74cf9c04-4543-47ae-a937-a9b7c6c921e7
metadata	{}
size	1
snapshot_id	None
source_volid	None
status	creating
volume type	None

2. 通过启用EQL阵列自身的命令行接口来验证卷的大小。

The actual size (VolReserve) is 1.01 GB. The EQL Group Manager should also report a volume size of 1.01 GB.

```
eq1> volume select volume-74cf9c04-4543-47ae-a937-a9b7c6c921e7
eq1 (volume_volume-74cf9c04-4543-47ae-a937-a9b7c6c921e7)> show
```

Volume Information
Name: volume-74cf9c04-4543-47ae-a937-a9b7c6c921e7
Size: 1GB
VolReserve: 1.01GB
VolReserveInUse: 0MB
ReplReserveInUse: 0MB
iSCSI Alias: volume-74cf9c04-4543-47ae-a937-a9b7c6c921e7
iSCSI Name: iqn.2001-05.com.equallogic:0-8a0906-19f91850c-067000000b4532cl-volume-74cf9c04-4543-47ae-a937-a9b7c6c921e7
ActualMembers: 1
SnapWarn: 10%

```
Snap-Deletion: delete-oldest
Description:
Snap-Reserve: 100%
Snap-Reserve-Avail: 100% (1.01GB)
Permission: read-write
DesiredStatus: online
Status: online
Connections: 0
Snapshots: 0
Bind:
Type: not-replicated
ReplicationReserveSpace: 0MB
```

3. 从此卷创建一个新的镜像:

```
$cinder upload-to-image --disk-format raw
--container-format bare volume1 image_from_volume1
+-----+-----+
| Property | Value |
+-----+-----+
| container_format | bare |
| disk_format | raw |
| display_description | None |
| id | 74cf9c04-4543-47ae-a937-a9b7c6c921e7 |
| image_id | 3020a21d-ba37-4495-8899-07fc201161b9 |
| image_name | image_from_volume1 |
| size | 1 |
| status | uploading |
| updated_at | 2014-03-21T18:31:55.000000 |
| volume_type | None |
+-----+-----+
```

4. When you uploaded the volume in the previous step, the Image service reported the volume’s size as 1 (GB). However, when using glance image-list to list the image, the displayed size is 1085276160 bytes, or roughly 1.01 GB:

表 6.1. 镜像设置由glance image-list 报告镜像ID

名称	磁盘格式	容器格式	配置	状态
image_from_volume1	raw	bare	1085276160	活跃

5. Create a new volume using the previous image (image_id 3020a21d-ba37-4495-8899-07fc201161b9 in this example) as the source. Set the target volume size to 1 GB; this is the size reported by the cinder tool when you uploaded the volume to the Image service:

```
$cinder create --display-name volume2
--image-id 3020a21d-ba37-4495-8899-07fc201161b9 1
ERROR: Invalid input received: Size of specified image 2 is larger
than volume size 1. (HTTP 400) (Request-ID: req-4b9369c0-dec5-4e16-a114-c0cd16bSd210)
```

若基于cinder 工具所报出的卷的大小来创建卷的话，会失败。

解决方案

To work around this problem, increase the target size of the new image to the next whole number. In the problem example, you created a 1 GB volume to be used as volume-backed image, so a new volume using this volume-backed image should use a size of 2 GB:

```
$ cinder create --display-name volume2
--image-id 3020a21d-ba37-4495-8899-07fc201161b9 1
```

Property	Value
attachments	[]
availability_zone	nova
bootable	false
created_at	2014-03-21T19:25:31.564482
display_description	None
display_name	volume2
id	64e8eb18-d23f-437b-bcac-b352afa6843a
image_id	3020a21d-ba37-4495-8899-07fc201161b9
metadata	[]
size	2
snapshot_id	None
source_volid	None
status	creating
volume_type	None



注意

当你基于一个卷后端的镜像创建一个新的卷时，前端会给你提供一个合适的大小值的建议。

你可以到EQL盘阵中查看此新的卷：

```
eql> volume select volume-64e8eb18-d23f-437b-bcac-b352afa6843a
eql (volume_volume-61e8eb18-d23f-437b-bcac-b352afa6843a)> show
```

Volume Information
Name: volume-64e8eb18-d23f-437b-bcac-b352afa6843a
Size: 2GB
VolReserve: 2.01GB
VolReserveInUse: 1.01GB
ReplReserveInUse: 0MB
iSCSI Alias: volume-64e8eb18-d23f-437b-bcac-b352afa6843a
iSCSI Name: iqn.2001-05.com.equallogic:0-8a0906-e3091850e-eae000000b7S32cl-volume-64e8eb18-d23f-437b-bcac-b352afa6843a
ActualMembers: 1
Snap-Warn: 10%
Snap-Depletion: delete-oldest
Description:
Snap-Reserve: 100%
Snap-Reserve-Avail: 100% (2GB)
Permission: read-write
DesiredStatus: online
Status: online
Connections: 1
Snapshots: 0
Bind:
Type: not-replicated
ReplicationReserveSpace: 0MB

挂接卷失败，缺失 sg_scan

问题

给实例挂接卷失败，文件sg_scan不存在，此警告和错误的发生是计算节点没有安装 sg3-utils 软件包。在系统中此错误信息有唯一的ID：

```
ERROR nova.compute.manager [req-cf2679fd-dd9e-4909-807f-48fe9bda3642 admin admin|req-cf2679fd-dd9e-4909-807f-48fe9bda3642 admin admin]
[instance: 7d7c92e0-49fa-4a8e-87c7-73f22a9585d5|instance: 7d7c92e0-49fa-4a8e-87c7-73f22a9585d5]
Failed to attach volume 4cc104c4-ac92-4bd6-9b95-c6686746414a at /dev/vdcTRACE nova.compute.manager
[instance: 7d7c92e0-49fa-4a8e-87c7-73f22a9585d5|instance: 7d7c92e0-49fa-4a8e-87c7-73f22a9585d5]
Stdout: '/usr/local/bin/nova-rootwrap: Executable not found: /usr/bin/sg_scan'
```

解决方案

在计算节点中运行下面的命令来安装sg3-utils软件包：

```
# apt-get install sg3-utils
```

在取消挂接卷之后再次挂接卷失败

问题

这些错误会出现在文件cinder-volume.log 中。

```
2013-05-03 15:16:33 INFO [cinder.volume.manager] Updating volume status
2013-05-03 15:16:33 DEBUG [hp3parclient.http]
REQ: curl -i https://10.10.22.241:8080/api/v1/cpgs -X GET -H "X-Hp3Par-Wsapi-Sessionkey: 48dc-b69ed2e5f259c58e26df9a4c85df110c-8d1e8451" -H "Accept: application/json" -H "User-Agent: python-3parclient"

2013-05-03 15:16:33 DEBUG [hp3parclient.http] RESP: {'content-length': 311, 'content-type': 'text/plain', 'status': '400'}

2013-05-03 15:16:33 DEBUG [hp3parclient.http] RESP BODY: Second simultaneous read on file no 13 detected. Unless you really know what you're doing, make sure that only one greenthread can read any particular socket. Consider using a pools.Pool. If you do know what you're doing and want to disable this error, call eventlet.debug.hub_multiple_reader_prevention(False)

2013-05-03 15:16:33 ERROR [cinder.manager] Error during VolumeManager._report_driver_status: Bad request (HTTP 400)
Traceback (most recent call last):
File "/usr/lib/python2.7/dist-packages/cinder/manager.py", line 167, in periodic_tasks task(self, context)
File "/usr/lib/python2.7/dist-packages/cinder/volume/manager.py", line 690, in _report_driver_status volume_stats = self.driver.get_volume_stats(refresh=True)
File "/usr/lib/python2.7/dist-packages/cinder/volume/drivers/san/hp/hp_3par_fc.py", line 77, in get_volume_stats stats = self.common.get_volume_stats(refresh, self.client)
File "/usr/lib/python2.7/dist-packages/cinder/volume/drivers/san/hp/hp_3par_common.py", line 421, in get_volume_stats cpg = client.getCPG(self.config.hp3par_cpg)
File "/usr/lib/python2.7/dist-packages/hp3parclient/client.py", line 231, in getCPG cpgs = self.getCPGs()
File "/usr/lib/python2.7/dist-packages/hp3parclient/client.py", line 217, in getCPGs response, body = self.http.get('/cpgs')
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 255, in get return self._cs_request(url, 'GET', **kwargs)
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 224, in _cs_request **kwargs)
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 198, in _time_request resp, body = self.request(url, method, **kwargs)
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 192, in request raise exceptions.
from_response(resp, body)
HTTPBadRequest: Bad request (HTTP 400)
```

解决方案

你需要更新你复制的 `hp_3par_fc.py` 驱动，其包含了同步的代码。

复制 3PAR 主机

问题

此错误是由于OpenStack所使用的主机名和系统的名称不一致所导致。此错误会显示IQN，如果使用了iSCSI所期望的主机。

```
Duplicate3PARHost: 3PAR Host already exists: Host wwn 50014380242B9750 already used by host  
cld4b5ubuntuW(id = 68, The hostname must be called 'cld4b5ubuntu'.
```

解决方案

修改 3PAR 的主机名称以匹配 OpenStack 所期望的名称。由驱动构建的 3PAR 主机仅使用了本地主机名，而不是计算主机的完全合格域名 (FQDN)。例如，如果 FQDN 为 `myhost.example.com`，那么仅仅会使用 `myhost` 作为 3PAR 的主机名。在 3PAR 存储服务器上，IP 地址不允许设置为主机名。

在取消挂接卷之后再次挂接卷失败

问题

在挂接相同的卷之后挂接卷会失败。

解决方案

你必须在命令 `nova-attach` 更改设备的名称。在运行命令 `nova-detach` 之后或许虚拟机不能清空。这个例子说明了当你使用 `vdb`, `vdc`, 或 `vdd` 的设备名称时命令 `nova-attach` 是如何失败的：

```
# ls -al /dev/disk/by-path/  
total 0  
drwxr-xr-x 2 root root 200 2012-08-29 17:33 .  
drwxr-xr-x 5 root root 100 2012-08-29 17:33 ..  
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:04,0-virtio-pci-virtio0 -> ../../vda  
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:04,0-virtio-pci-virtio0-part1 -> ../../vda1  
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:04,0-virtio-pci-virtio0-part2 -> ../../vda2  
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:04,0-virtio-pci-virtio0-part5 -> ../../vda5  
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:06,0-virtio-pci-virtio2 -> ../../vdb  
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:08,0-virtio-pci-virtio3 -> ../../vdc  
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:09,0-virtio-pci-virtio4 -> ../../vdd  
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:09,0-virtio-pci-virtio4-part1 -> ../../vdd1
```

你可能会在同一个虚拟机挂载相同的挂载点，挂接和分离同一个卷多次之后遇到此问题，若发生此种情况，重启KVM主机。

挂接卷失败，没有安装sysstool软件包

问题

如果计算节点没有安装所需要的软件包 `sysfsutils`，此警告和错误就会发生。

```
WARNING nova.virt.libvirt.utils [req-1200f887-c82b-4e7c-a891-fac2e3735dbb admin admin|req-1200f887-c82b-4e7c-a891-fac2e3735dbb admin admin] systool is not installed
ERROR nova.compute.manager [req-1200f887-c82b-4e7c-a891-fac2e3735dbb admin admin|req-1200f887-c82b-4e7c-a891-fac2e3735dbb admin admin]
[instance: df834b5a-8c3f-477a-be9b-47c97626555c|instance: df834b5a-8c3f-477a-be9b-47c97626555c]
Failed to attach volume 13d5c633-903a-4764-a5a0-3336945b1db1 at /dev/vdk.
```

解决方案

在计算节点中运行下面命令来安装sysfsutils软件包。

```
# apt-get install sysfsutils
```

在光纤SAN中连接卷失败

问题

计算节点无法连接到光纤通道(FC) SAN的卷。那么可能在FC SAN中的WWN没有划分正确，即计算节点连接到存储阵列的匹配。

```
ERROR nova.compute.manager [req-2ddd5297-e405-44ab-aed3-152cd2cfb8c2 admin demo|req-2ddd5297-e405-44ab-aed3-152cd2cfb8c2 admin demo] [instance: 60ebd6c7-c1e3-4bf0-8ef0-f07aa4c3d5f3|instance: 60ebd6c7-c1e3-4bf0-8ef0-f07aa4c3d5f3]
Failed to connect to volume 6f6a6a9c-dfcf-4c8d-b1a8-4445ff883200 while attaching at /dev/vdjTRACE nova.compute.manager [instance: 60ebd6c7-c1e3-4bf0-8ef0-f07aa4c3d5f3|instance: 60ebd6c7-c1e3-4bf0-8ef0-f07aa4c3d5f3]
Traceback (most recent call last):...f07aa4c3d5f3 ] ClientException: The server has either erred or is incapable of performing the requested operation.(HTTP 500)(Request-ID: req-71e5132b-21aa-46ee-b3cc-19b5b4ab2f00)
```

解决方案

网络管理员必须配置 FC SAN 划分给计算节点HBA卡正确的WWN(端口名称)。

无法为x86_64找到合适的模拟器

问题

假设当你创建一个虚拟机时，虚拟机的错误信息显示BUILD而不是ERROR状态。

解决方案

在运行KVM的主机中，执行cat /proc/cpuinfo。确保有vme and svm 指令集。

请遵照配置参考中的章节 [启用KVM](#)，来在BIOS中打开硬件虚拟化的支持。

不存在的主机

问题

此错误是由于OpenStack所使用的主机名和系统的名称不一致所导致。此错误会显示IQN，如果使用了iSCSI所期望的主机。

```
2013-04-19 04:02:02,336 2814 ERROR cinder.openstack.common.rpc.common [-] Returning exception Not found (HTTP 404)
NON_EXISTENT_HOST - HOST '10' was not found to caller.
```

解决方案

驱动使用的只是本地主机名，而不是完全合格域名(FQDN)的计算节点。例如，如果某计算节点的FQDN是myhost.example.com，仅使用myhost用作3PAR主机名即可，但是IP地址是不可以用于3PAR存储服务器的主机名的。

不存在的 VLUN

问题

如果3PAR主机存在且主机名也是按照OpenStack块存储驱动希望的那样配置的，但是在不同的域创建卷就会发生此错误。

```
HTTPNotFound: Not found (HTTP 404) NON_EXISTENT_VLUN - VLUN 'osv-DqT7CE3mSrWi4gZJmHAP-Q' was not found.
```

解决方案

hp3par_domain配置项不仅需要更新域为其当前的3PAR主机，还有3PAR主机需要迁入所创建的卷的域中。

第 7 章 网络

目录

网络介绍	180
插件配置	184
配置neutron代理	187
网络架构	194
为网络配置身份服务	196
高级配置选项	200
DHCP代理的扩展性和高可用性	201
使用网络	208
通过API扩展的高级功能	212
高级运维特性	222
认证和授权	224

学习OpenStack网络概念，架构，和neutron and nova命令行接口(CLI)命令的基本和高级用法。

网络介绍

网络服务，即 neutron，提供了一个 API，允许您在云中定义网络连接和地址。网络服务使运营商能够使用不同的技术来实现加强他们的云网络。网络服务也提供了一个 API 来配置和管理各种各样的网络服务，范围从 L3 转发和 NAT 到负载均衡、边缘防火墙和 IPsec VPN。

有关网络API抽象及其属性的细节描述，请参阅[OpenStack 网络 API v2.0 参考](#)。

网络应用程序接口

Networking 是一个虚拟网络服务，提供了一个强大的 API 来定义网络连接和其他服务设备上的 IP 地址，例如计算节点。

计算节点 API 有一个虚拟服务器抽象来描述计算资源。类似的，Networking API 也有虚拟网络、子网和端口抽象来描述网络资源。

表 7.1. 网络资源

资源	描述
网络	隔离的2层段，类似于物理网络的VLAN。
子网	v4 或 v6 IP地址及其关联的配置状态块。
端口	附加单个设备的连接点到一个虚拟网络中，例如虚拟服务器的 NIC。还需要描述相关的网络配置，例如在其端口上使用的 MAC 和 IP 地址。

要配置丰富的网络拓扑，您可以创建和配置网络和子网，并通知其他 OpenStack 服务，如计算服务，来附加虚拟设备到这些网络的端口上。

特别地，网络支持每个租户拥有多个私有网络并允许租户选择他们自己的 IP 地址方案，即使这些 IP 地址与其他租户所使用的地址重复。

网络服务：

- 启用高级云网络使用案例，如构建多层 web 应用并允许应用在不改变 IP 地址的情况下迁移到云上。
- 为云管理员定制网络产品提供灵活性。
- 允许开发者扩展 Networking API。随着时间的推移，扩展的功能成为了核心 Networking API 的一部份。

为网络API配置SSL支持

OpenStack网络支持网络API服务器的SSL。默认情况下，SSL是禁用的，但是你可以在文件 `neutron.conf` 中将之激活。

要配置SSL须设置的这些属性:

<code>use_ssl = True</code>	在网络API 服务器上打开SSL。
<code>ssl_cert_file = PATH_TO_CERTFILE</code>	当你安全的启动网络API服务器时使用证书文件。
<code>ssl_key_file = PATH_TO_KEYFILE</code>	当你安全的启动网络API服务器时使用私钥文件。
<code>ssl_ca_file = PATH_TO_CAFILE</code>	可选的。CA 证书文件在您安全地启动 Networking API 服务器时被使用。这个文件校验客户端的连接。当 API 客户端必需通过使用由信任 CA 分配的 SSL 证书在 API 服务器进行认证时，配置该选项。
<code>tcp_keepidle = 600</code>	为每个API 服务器socket 设置 TCP_KEEPIDLE 的秒值. 在 OSX上不支持.
<code>retry_until_window = 30</code>	保持重试监听的秒数。
<code>backlog = 4096</code>	配置套接字的backlog请求数。

负载均衡即服务(LBaaS)概览

负载均衡即服务 (LBaaS) 就是分发网络将来自外部的请求均衡的分配给实例。此分发确保负载可在意料之中且更加有效的利用好系统的资源。使用下面方法中的一种来实现分发来自外部的请求：

- 轮循 在多个实例之间均匀地分发请求
- 源IP 从一个独特的源IP地址的请求都一致指向同一个实例。
- 至少 分配到实例的请求，以最后活跃的连接数为准。
- 连接

表 7.2. 负载均衡即服务特性

特性	描述
监控	LBaaS提供了监控ping, TCP, HTTP 和 HTTPS GET 方法的能力。监控 实现了如何决定哪个可用的池的成员来响应请求。
管理	LBaaS可由多种工具来管理。 REST API可用于编程管理和脚本。普通用户可通过命令行 (neutron) 或OpenStack面板来管理负载均衡。
连接限制	连接限制可有效的组织入口流量。此功能能实现负载的控制，且可用于缓解DoS(拒绝服务)攻击。

特性	描述
会话持久化	LBaaS支持持久会话，以确保来自外部的请求可以在实例池中每次都路由到同一个实例。LBaaS支持基于cookies和来源IP地址的路由决策。

防火墙即服务(FWaaS)概览

防火墙即服务 (FWaaS) 插件增加了网络的防火墙功能。FWaaS使用iptables来对整个项目的网络路由的防火墙规则起作用。FWaaS支持一个防火墙规则和每个项目的逻辑防火墙实例。

安全组的操作在实例这一层，FWaaS的操作在neutron路由的流量过滤这层。

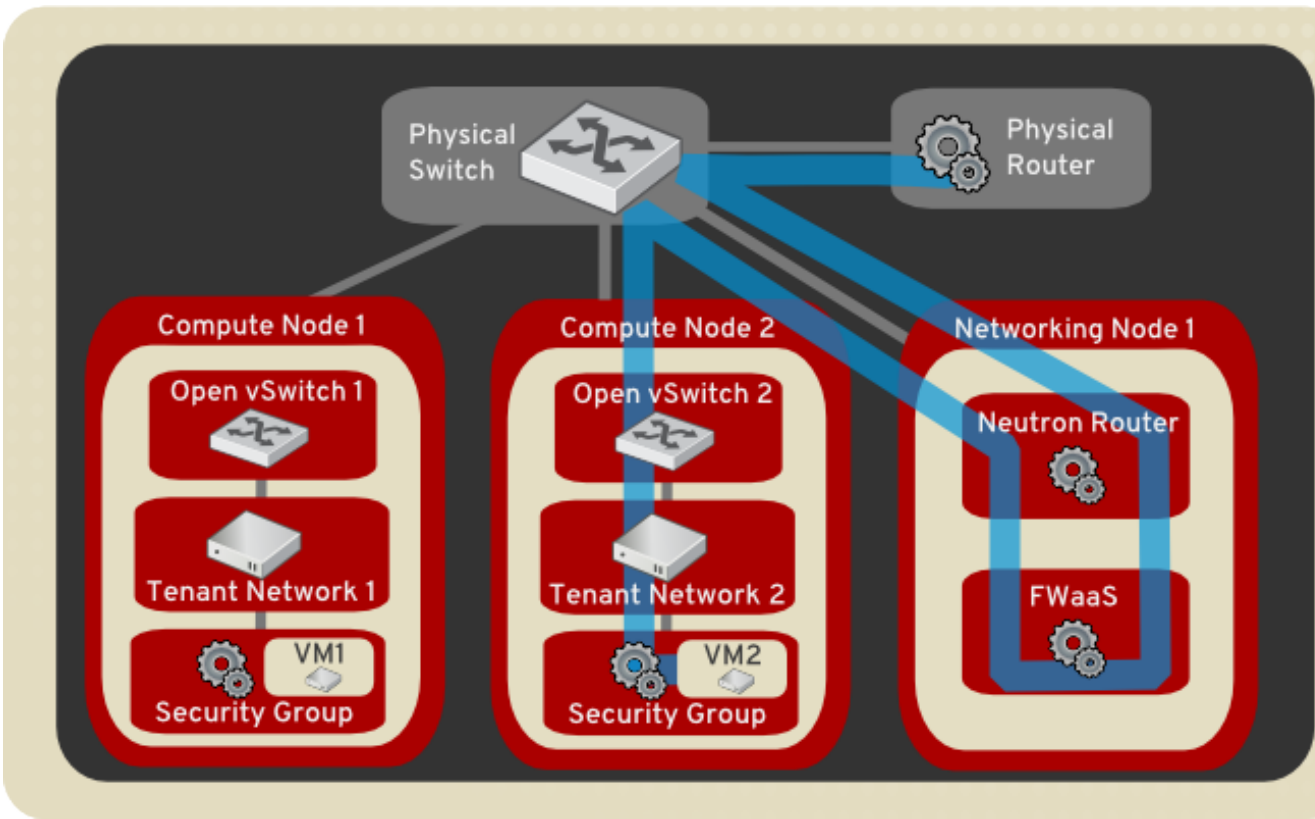


注意

防火墙即服务目前仍属技术预览；为测试的操作是不建议的。

此示例图显示了VM2 实例的入口和出口流量的流程：

图 7.1. FWaaS架构



使用FWaaS

在OpenStack图形面板中也有FWaaS的管理选项。

1. 在文件/etc/neutron/neutron.conf中激活FWaaS插件:


```
service_plugins = firewall
[service_providers]
...
service_provider = FIREWALL:Iptables:neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver:default

[fwaas]
driver = neutron_fwaas.services.firewall.drivers.linux.iptables_fwaas.IptablesFwaasDriver
enabled = True
```



注意

在Ubuntu下，没有使用/etc/neutron/neutron.conf，使用的/etc/neutron/fwaas_driver.ini文件，在此文件的[fwaas]一节做修改。

2. 在数据库中创建所要求的表：

```
# neutron-db-manage --service fwaas upgrade head
```

3. 激活选项在文件 /usr/share/openstack-dashboard/openstack_dashboard/local/local_settings.py中，它在控制器节点中可以找到：

```
OPENSTACK_NEUTRON_NETWORK = {
    ...
    'enable_firewall' = True,
    ...
}
```

4. 重启服务neutron-l3-agent 和 neutron-server 以应用设置。

配置防火墙即服务

创建防火墙规则，然后创建一策略将前者囊括，再创建一防火墙应用此策略。

1. 创建一个防火墙rule：

```
$ neutron firewall-rule-create --protocol {tcp|udp|icmp|any} --destination-port PORT_RANGE --action {allow|deny}
```

网络客户端会请求一协议值；如果此值时不可知的协议，那么使用any值。

2. 创建一个防火墙规则：

```
$ neutron firewall-policy-create --firewall-rules "FIREWALL_RULE_IDS_OR_NAMES" myfirewallpolicy
```

将防火墙规则的ID或名称用空格隔开。指定规则的顺序也非常的重要。

你可以创建一个没有任何规则的防火墙策略，稍后再添加规则，如下所示：

- 要添加多个用户，使用更新操作。
- 要添加单个的规则，使用insert-rule操作。

，更多细节，请参阅OpenStack 命令行接口参考的[网络命令行客户端](#)一章内容。



注意

FWaaS通常每个策略增加一条最低优先权的默认规则deny all，因此，默认情况下，没有任何规则的一个防火墙策略是屏蔽所有流量。

3. 创建一个防火墙：

```
$ neutron firewall-create FIREWALL_POLICY_UUID
```



注意

防火墙会保持 PENDING_CREATE 状态，直到创建了一个网络路由然后将之附加到网卡为止。

Allowed-address-pairs. Allowed-address-pairs 允许您指定 mac_address/ip_address(cidr) 对，不管子网如何，它们会传递到一个端口上。这启用了协议的使用，如 VRRP，它会在两台实例之间浮动一个 IP 地址来使得快速数据平行故障转移。



注意

目前为止，仅ML2, Open vSwitch, 和 VMware NSX 插件支持allowed-address-pairs 扩展。

基本的 allowed-address-pairs 操作.

- 基于指定的所允许的地址对来创建一个端口：

```
$ neutron port-create net1 --allowed-address-pairs type=dict list=true mac_address=MAC_ADDRESS,  
ip_address=IP_CIDR
```

- 通过添加允许的地址对来更新一个端口：

```
$ neutron port-update PORT_UUID --allowed-address-pairs type=dict list=true mac_address=MAC_ADDRESS,  
ip_address=IP_CIDR
```



注意

在早于Juno的版本中，OpenStack网络所表现的设置地址对是使用端口对应MAC地址和一个固定IP地址的端口。

插件配置

对于配置属性，请参阅配置参考中的[网络配置属性](#)。此章节解释了如何配置特定的插件。

配置 Big Switch (Floodlight REST Proxy) 插件

要和OpenStack网络一起使用REST代理插件

1. 编辑文件/etc/neutron/neutron.conf然后添加此行：

```
core_plugin = bigswitch
```

2. 在文件/etc/neutron/neutron.conf中，设置service_plugins属性：

```
service_plugins = neutron.plugins.bigswitch.l3_router_plugin.L3RestProxy
```

3. 编辑文件/etc/neutron/plugins/bigswitch/restproxy.ini，然后指定一个逗号隔离的controller_ip:port对的列表：

```
server = CONTROLLER_IP:PORT
```

对于数据库的配置，请参阅 [OpenStack 文档索引](#) 中的安装手册中的[安装网络服务](#)。（默认的连接是Ubuntu发行版。）

4. 重启neutron-server 以让设置生效：

```
# service neutron-server restart
```

配置 Brocade 插件

要和OpenStack网络一起使用Brocade的插件

1. 安装博科修改过的Python netconf客户端程序库(ncclient),其地址是<https://github.com/brocade/ncclient>:

```
$ git clone https://github.com/brocade/ncclient
```

以 root, 运行此命令:

```
# cd ncclient;python setup.py install
```

2. 编辑文件/etc/neutron/neutron.conf并设置如下属性：

```
core_plugin = brocade
```

3. 为Brocade插件编辑文件/etc/neutron/plugins/brocade/brocade.ini，然后指定管理员用户名，密码，以及Brocade交换机的IP地址：

```
[SWITCH]
username = ADMIN
password = 密码
address = SWITCH_MGMT_IP_ADDRESS
ostype = NOS
```

对于数据库的配置，请参阅[OpenStack 文档索引](#)的安装文档（默认是Ubuntu的版本）中[安装网络服务](#)一节。

4. 重启neutron-server 服务，以应用设置：

```
# service neutron-server restart
```

配置NSX-mh插件

配置OpenStack网络使用NXS多Hypervisor插件

此节是Vmware NSX-mh平台的说明，NSX-mh即是原来著名的Nicira NVP。

1. 安装NSX插件：

```
# apt-get install neutron-plugin-vmware
```

2. 编辑文件/etc/neutron/neutron.conf，然后设置这些行：

```
core_plugin = vmware
```

整合NSX-mh的neutron.conf 文件实例：

```
core_plugin = vmware
rabbit_host = 192.168.203.10
allow_overlapping_ips = True
```

3. 要为OpenStack网络配置NSX-mh控制器集群的话，找到/etc/neutron/plugins/vmware/nsx.ini文件中[default]一节，添加下面的项：

- 要建立和配置控制器集群的连接，你必须设置一些参数，包括NSX-mh API的端点，访问凭证，以及可选的指定设置HTTP超时，在连接失败的情况下的重定向或重试：

```
nsx_user = ADMIN_USER_NAME
nsx_password = NSX_USER_PASSWORD
http_timeout = HTTP_REQUEST_TIMEOUT # (seconds) default 75 seconds
retries = HTTP_REQUEST_RETRIES # default 2
redirects = HTTP_REQUEST_MAX_REDIRECTS # default 2
nsx_controllers = API_ENDPOINT_LIST # comma-separated list
```

为了确保正确的操作，nsx_user用户必须在NSX-mh平台上拥有管理员凭证。

控制节点的 API 端点由 IP 地址和控制节点的端口组成；如果您省略了端口，则会使用 443 端口。如果指定了多个 API 端点，它取决于用户来保证所有这些端点都属于同一个控制集群。OpenStack Networking VMware NSX-mh 插件没有对其进行检查，所以结果可能时不可预测的。

如果您指定了多个 API 端点，该插件会在各种 API 端点中关注负载均衡的请求。

- 当租户创建了一个网络，默认应使用 NSX-mh 的 UUID 传输域。您可以从 NSX-mh 管理者的 Transport Zones 页面中获取该值：

另外，传输区标识可以通过查寻 NSX-mh API 检索到：`/ws.v1/transport-zone`

```
default_tz_uuid = TRANSPORT_ZONE_UUID
```

- `default_l3_gw_service_uuid = GATEWAY_SERVICE_UUID`



警告

目前 Ubuntu 打包不会更新 neutron 初始化脚本来指向 NSX-mh 配置文件。因此，您必需手动更新 /etc/default/neutron-server，添加以下行：

```
NEUTRON_PLUGIN_CONFIG = /etc/neutron/plugins/vmware/nsx.ini
```

对于数据库的配置，请参阅Installation 指南中的 [安装网络服务](#)。

4. 重启neutron-server，以让设置生效：

```
# service neutron-server restart
```



警告

neutron NSX-mh 插件没有实现 Neutron 资源的初始化重新同步。因此当 Neutron 切换到 NSX-mh 插件时，资源可能已经在数据库中存在，不会在重启时在 NSX-mh 后端创建数据库。

文件nsx.ini的例子:

```
[DEFAULT]
default_tz_uuid = d3afb164-b263-4aaa-a3e4-48e0e09bb33c
default_l3_gw_service_uuid=5c8622cc-240a-40a1-9693-e6a5fca4e3cf
nsx_user=admin
nsx_password=changeme
nsx_controllers=10.127.0.100,10.127.0.200:8888
```



注意

要调试nsx.ini的配置问题，在运行 neutron-server的主机中执行这些命令:

```
# neutron-check-nsx-config PATH_TO_NSX.INI
```

此命令会测试neutron-server 是否可以登录到所有的NSX-mh控制器和SQL服务，以及所有的UUID值是否正确。

配置PLUMgrid插件

要和OpenStack网络一起使用PLUMgrid插件

1. 编辑文件/etc/neutron/neutron.conf，然后设置这些行：

```
core_plugin = plumgrid
```

2. 编辑文件/etc/neutron/plugins/plumgrid/plumgrid.ini中的[PLUMgridDirector]一节，然后指定PLUMgrid Director的IP地址，端口，管理员用户名，及其密码：

```
[PLUMgridDirector]
director_server = "PLUMgrid-director-ip-address"
director_server_port = "PLUMgrid-director-port"
username = "PLUMgrid-director-admin-username"
password = "PLUMgrid-director-admin-password"
```

对于数据库的配置，请参阅Installation 指南中的 [安装网络服务](#)。

3. 重启neutron-server 服务，以应用设置：

```
# service neutron-server restart
```

配置neutron代理

插件通常对特定软件有要求，它们必需运行在每个处理数据包的节点上。这包括了任何运行 nova-compute 的节点和运行专有 OpenStack Networking 服务代理的节点，这些服务如 neutron-dhcp-agent、neutron-l3-agent、neutron-metering-agent 或 neutron-lbaas-agent。

数据转发节点通常有一个包含管理网络上的 IP 地址的网络接口和其他数据网络上的接口。

本小节展示了如何安装和配置可用插件的子网，这可能包括了切换软件（如，Open vSwitch）以及用于与 neutron-server 进程通信的运行在数据中心的代理的安装。

配置数据转发节点

节点设置：NSX插件

如果你使用NSX插件，你必须在每个数据转发的节点安装Open v Switch。然而，无须再每个节点都安装额外的代理。



警告

运行一个与当前 NSX 控制器软件兼容的 Open vSwitch 版本是很关键的。不要使用 Ubuntu 默认安装的 Open vSwitch 版本。应使用 VMware 支持门户提供的符合您 NSX 控制器版本的 Open vSwitch 版本。

要设置每个节点的NSX插件

1. 确保每个数据转发节点拥有一个管理网络上的 IP 地址和一个用于隧道传输数据流量的“数据网络”上的 IP 地址。要了解详尽的转发节点配置，请阅读 NSX Administrator Guide。
2. 使用 NSX Administrator Guide 来通过 NSX 管理界面将节点添加为 hypervisor。即使您的转发节点上没有虚拟机且仅用于服务代理，如 neutron-dhcp-agent 或 neutron-lbaas-agent，它仍然可以作为 Hypervisor 添加到 NSX。
3. 完成 NSX Administrator Guide 的提示后，请为 NSX 管理界面中的该 Hypervisor 使用这个页面来确认节点正常连接到 NSX 控制器集群以及 NSX 控制器集群可以使用 br-int 整合桥接。

配置 DHCP 代理

DHCP 服务代理与所有现有插件是相互兼容的，且它对所有虚拟机需要通过 DHCP 自动获取 IP 地址的部署环境是必需的。

安装和配置DHCP代理

1. 您必需根据您的插件需要来配置主机运行 neutron-dhcp-agent 作为一个数据转发节点。详见“[配置neutron代理](#)”一节 [187]。

2. 安装DHCP代理

```
# apt-get install neutron-dhcp-agent
```

3. 最后，更新 /etc/neutron/dhcp_agent.ini 文件中所有取决于所使用的插件的选项。详见子小节。



重要

如果你重启了运行DHCP代理的节点，你必须在启动neutron-dhcp-agent服务之前运行命令neutron-ovs-cleanup。

在Red Hat，SUSE，和Ubuntu基于的系统中，neutron-ovs-cleanup 服务会自动运行命令neutron-ovs-cleanup。然而，在Debian系统中，你必须手动的运行此命令，或者是自己写个系统脚本，在启动的时候要在neutron-dhcp-agent 服务启动之前运行。

Networking dhcp-agent 可以使 [dnsmasq](#) 驱动，它支持有状态和无状态的 DHCPv6 的子网，可以以 `--ipv6_address_mode` 选项，可设置为 `dhcpv6-stateful` 或 `dhcpv6-stateless`，创建子网。

例如：

```
$ neutron subnet-create --ip-version 6 --ipv6_ra_mode dhcpv6-stateful --ipv6_address_mode dhcpv6-stateful NETWORK CIDR
```

```
$ neutron subnet-create --ip-version 6 --ipv6_ra_mode dhcpv6-stateless --ipv6_address_mode dhcpv6-stateless NETWORK CIDR
```

如果子网的网络没有 `dnsmasq` 启动，Networking 会在子网的 `qdhcp-XXX` 命名空间里的 `dhcp` 端口中启动一个新的进程。如果之前的 `dnsmasq` 进程已经启动，则会以新的配置重启 `dnsmasq`。

网络会更新 `dnsmasq` 进程，且当子网获得更新时重启它。



注意

dhcp代理要有IPv6模式的话，需要dnsmasq的版本至少是v2.63。

在一定的所配置好时间格式的时间后，如果代理不再使用，网络会从 DHCP 代理上解耦。您可以配置当代理不再提供服务或不再需要使用时，将 DHCP 代理自动地从网络上分离。

此特性能够让所有的插件支持DHCP的扩展。更多信息，请参阅在OpenStack配置参考中的 [DHCP 代理配置属性](#) 列表。

DHCP代理 设置: OVS 插件

这些DHCP代理属性均是OVS插件在 `/etc/neutron/dhcp_agent.ini` 文件中所需要的：

```
[DEFAULT]
enable_isolated_metadata = True
use_namespaces = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

DHCP代理 设置: NSX 插件

这些DHCP代理属性均是NSX插件在 `/etc/neutron/dhcp_agent.ini` 文件中所需要的：

```
[DEFAULT]
enable_metadata_network = True
enable_isolated_metadata = True
use_namespaces = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

配置3层代理

OpenStack Networking 服务有一个广泛使用的 API 扩展，允许管理员和租户创建路由来与 L2 网络和浮动 IP 相互连接，使端口对私有网络可以公开访问。

很多插件依赖于 L3 服务代理实现了 L3 的功能。然而，下列插件已经内置了 L3 的功能：

- Big Switch/Floodlight 插件，它们都同时支持开源的 [Floodlight](#) 控制器和商业的 Big Switch 控制器。



注意

只有商业版的BigSwitch控制器实现了3层点功能。当使用Floodlight作为你的OpenFlow控制器时，3层功能是不可用的。

- 插件IBM SDN-VE
- MidoNet 插件
- NSX 插件
- PLUMgrid 插件



警告

如果你使用了这些插件中的某一种，请不要配置或使用neutron-l3-agent。

为所有其它插件安装3层代理

1. 在网络节点安装二进制的 neutron-l3-agent:

```
# apt-get install neutron-l3-agent
```

2. 对于在外部网络运行 neutron-l3-agent 的上行节点，创建一个名为 "br-ex" 的桥接并在这个网卡上附加外部网络到这个桥接上。

例如，基于Open vSwitch的eth1网卡连接到外部网络，运行:

```
# ovs-vsctl add-br br-ex  
# ovs-vsctl add-port br-ex eth1
```

请勿在运行 neutron-l3-agent 的节点的连接到外部网络的网卡上手动配置 IP 地址。当然，您必需有一个外部网络的 IP 地址范围，让它可以被 OpenStack Networking 的上传到外部网络的路由所使用。这个范围必需足够大，以能够为环境中的每一个路由和浮动 IP 分配 IP 地址。

3. neutron-l3-agent 使用 Linux IP 栈和 iptables 来进行 L3 转发和 NAT。为了支持具有潜在重叠 IP 地址的多路由，neutron-l3-agent 默认只用 Linux 网络命名空间来提供隔离的转发上下文。因此，仅仅在节点上运行 ip addr list 或 ifconfig 命令是看不见路由的 IP 地址的。类似地，您不能直接 ping 固定 IP。

为了实现这些，您必需在一个路由的特定的网络命名空间里执行命令。命名空间名称为 "qrouter-ROUTER_UUID"。这些示例命令在 UUID 为 47af3868-0fa8-4447-85f6-1304de32153b 路由命名空间里运行：

```
# ip netns exec qrouter-47af3868-0fa8-4447-85f6-1304de32153b ip addr list
```

```
# ip netns exec qrouter-47af3868-0fa8-4447-85f6-1304de32153b ping FIXED_IP
```



注意

对于iproute版本为3.12.0或更高，网络命名空间默认配置为删除。此行为可通过更改DHCP和L3代理。各自的配置文件是/etc/neutron/dhcp_agent.ini 和 /etc/neutron/l3_agent.ini。

对 DHCP 命名空间配置关键字：`dhcp_delete_namespaces = True`。您可以将其设置为 `False`，防止命名空间在运行 DHCP 代理的主机上无法彻底删除。

对于 L3 命名空间，配置关键字：`router_delete_namespaces = True`。您可以将其设置为 `False`，以防命名空间无法在运行 L3 代理的节点上彻底删除。



重要

如果你要重启运行3层代理的节点，你必须在 `neutron-l3-agent` 服务启动之前运行命令 `neutron-ovs-cleanup`。

在基于 Red Hat，SUSE 以及 Ubuntu 的系统中，`neutron-ovs-cleanup` 服务自动运行命令 `neutron-ovs-cleanup`。然而，Debian 系统中，你必须手动的运行此命令，或者是你自己写个系统启动时的脚本，当然此脚本也要在 `neutron-l3-agent` 服务启动之前运行。

配置计量代理

Neutron 计量代理在 `neutron-l3-agent` 旁边。

要安装计量代理并配置节点

1. 通过运行以下内容来安装代理:

```
# apt-get install neutron-metering-agent
```

2. 如果你使用下列插件的某一种，你就需要基于这些内容来配置计量代理：

- 基于 OVS 的插件，诸如 OVS, NSX, NEC, BigSwitch/Floodlight:

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

- 使用 Linux 网桥的插件：

```
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
```

3. 要使用这些参考的实现，你必须设置:

```
driver = neutron.services.metering.drivers.iptables.iptables_driver.IptablesMeteringDriver
```

4. 在运行 `neutron-server` 的主机上的文件 `/etc/neutron/neutron.conf` 设置 `service_plugins` 属性：

```
service_plugins = metering
```

如果此属性早已经定义好，添加 `metering` 到列表，使用逗号来分隔。例如：

```
service_plugins = router,metering
```

配置负载均衡即服务(LBaaS)

基于 Open vSwitch 或 Linux 网桥插件来配置负载均衡即服务(LBaaS)。当启用基于 OVS 插件的 LBaaS 时需要 Open vSwitch LBaaS 驱动，这些插件包括 BigSwitch, Floodlight, NEC, and NSX。

基于Open vSwitch 或 Linux 网桥插件来配置LBaaS

1. 安装代理:

```
# apt-get install neutron-lbaas-agent haproxy
```

2. 在 /etc/neutron/neutron.conf 文件中使用 service_provider 属性，来启用 HAProxy 插件：

```
service_provider = LOADBALANCER:Haproxy:neutron_lbaas.services.loadbalancer.drivers.haproxy.  
plugin_driver.HaproxyOnHostPluginDriver:default
```



警告

在基于红帽的系统下，service_provider 属性在 /usr/share/neutron/neutron-dist.conf 文件中已经定义好了。不要再在 neutron.conf 中定义了，否则网络服务会重启失败。

3. 在 /etc/neutron/neutron.conf 文件中通过使用 service_plugins 属性来启用负载均衡插件：

```
service_plugins = lbaas
```

如果此属性已经定义过了，将 lbaas 添加到列表，使用逗号分隔。例如：

```
service_plugins = router,lbaas
```

4. 在 /etc/neutron/lbaas_agent.ini 文件中启用 HAProxy 负载均衡器：

```
device_driver = neutron_lbaas.services.loadbalancer.drivers.haproxy.namespace_driver.  
HaproxyNSDriver
```

5. 在文件 /etc/neutron/lbaas_agent.ini 中选择所需要的驱动：

激活 Open vSwitch 负载均衡即服务驱动:

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

或者，激活 Linux 网桥负载均衡即服务驱动:

```
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
```

6. 在数据库中创建所要求的表：

```
# neutron-db-manage --service lbaas upgrade head
```

7. 重启 neutron-server 和 neutron-lbaas-agent 服务以使设置生效。

8. 在面板的项目 选项卡下启用负载均衡。

在 local_settings 文件中修改 enable_lb 的属性为 True (发行版 Fedora, RHEL, 和 CentOS: /etc/openstack-dashboard/local_settings, 发行版 Ubuntu 和 Debian: /etc/openstack-dashboard/local_settings.py, 以及发行版 openSUSE 和 SLES: /srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py):

```
OPENSTACK_NEUTRON_NETWORK = {  
    'enable_lb': True,  
    ...  
}
```

重启web服务器以使设置生效。此时可以在图形面板中的项目中看到负载均衡器的管理项了。

配置Hyper-V 2层代理

在安装OpenStack 网络Hyper-V 2层代理到Hyper-V 计算节点之前，请确保是正确使用了[说明](#)来完成计算节点的配置的。

安装OpenStack 网络的Hyper-V代理且配置节点

1. 从仓库下载OpenStack 网络的代码:

```
> cd C: OpenStack
> git clone https://git.openstack.org/cgit/openstack/neutron
```

2. 安装OpenStack网络Hyper-V代理:

```
> cd C: OpenStack neutron
> python setup.py install
```

3. 复制文件 policy.json :

```
> xcopy C: OpenStack neutron etc policy.json C: etc
```

4. 创建文件C: etc neutron-hyperv-agent.conf，然后添加对应的配置属性，以及[Hyper-V 相关的属性](#)。这里是一个样例文件：

```
[DEFAULT]
verbose = true
control_exchange = neutron
policy_file = C: etc policy.json
rpc_backend = neutron.openstack.common.rpc.impl_kombu
rabbit_host = IP_ADDRESS
rabbit_port = 5672
rabbit_userid = guest
rabbit_password = <password>
logdir = C: OpenStack Log
logfile = neutron-hyperv-agent.log

[AGENT]
polling_interval = 2
physical_network_vswitch_mappings = *:YOUR_BRIDGE_NAME
enable_metrics_collection = true

[SECURITYGROUP]
firewall_driver = neutron.plugins.hyperv.agent.security_groups_driver.HyperVSecurityGroupsDriver
enable_security_group = true
```

5. 启动OpenStack网络Hyper-V代理:

```
> C: Python27 Scripts neutron-hyperv-agent.exe --config-file C: etc neutron-hyperv-agent.conf
```

在代理中的基本操作

此表展示了网络命令的例子，其可以让你完成代理的基本操作：

表 7.3. 网络代理的基本操作

操作	命令
列出所有可用代理。	<code>\$ neutron agent-list</code>
显示给点代理的信息。	<code>\$ neutron agent-show AGENT_ID</code>
为指定的代理更新管理员状态和描述。命令可以通过指定参数 <code>--admin-state-up</code> 来禁用和启用代理，参数的值设置为 <code>False</code> 或 <code>True</code> 。	<code>\$ neutron agent-update --admin-state-up False AGENT_ID</code>
删除给定的代理，要在删除之前先禁用。	<code>\$ neutron agent-delete AGENT_ID</code>

请参阅 [OpenStack 命令行接口参考](#) 来获取更多关于网络命令的信息。

网络架构

在部署网络之前，理解了网络服务是如何与OpenStack其它组件交互的，是非常有好处的。

概况

网络在OpenStack的模块化架构中是相对独立的组件。它的定位和OpenStack其它诸如计算，镜像服务，认证或面板是同一个维度的。和其它组件一样，网络通常也会部署服务到不同的主机中。

网络服务使用 `neutron-server` 守护进程来抛出网络API以及启用管理配置网络插件。典型地，插件需要为持久化的数据访问数据库(和其它OpenStack服务类似)。

如果您的部署环境使用了一个控制节点来运行集中的计算组件，那么您可以部署网络服务器到这台主机上。但是，Networking 是完全独立的，且可以部署在一台专用主机上。取决于您的配置，Networking 也可以包含下列代理：

表 7.4. 网络代理

代理	描述
插件代理 (<code>neutron-*agent</code>)	在每台 hypervisor 上运行本地 vSwitch 配置。运行的代理取决于您所使用的插件。某些插件不需要代理。
dhcp 代理 (<code>neutron-dhcp-agent</code>)	向租户网络提供 DHCP 服务。需要某些插件。
3层代理 (<code>neutron-l3-agent</code>)	提供 L3/NAT 转发以向虚拟机在租户网络中提供外部网络的访问。需要某些插件。
计量代理 (<code>neutron-metering-agent</code>)	为租户网络提供3层的流量计量。

这些代理通过 RPC (如，RabbitMQ 或 Qpid) 或标准 Networking API 与主 neutron 进程交互。此外，Networking 以多种方式整合了 OpenStack 组件：

- Networking 依赖于认证服务 (keystone) 对所有 API 请求进行认证和授权。
- 计算服务 (nova) 通过调用其标准 API 来与 Networking 进行交互。作为创建虚拟机的一部份，nova-compute 服务与 Networking API 通信，以将虚拟机上的每个虚拟网卡上插入一个特定的网络。
- 仪表盘 (horizon) 整合了 Networking API，允许管理员和租户用户通过基于 web 的 GUI 来创建和管理网络服务。

集成VMware NSX

OpenStack Networking 使用 NSX 插件来与已有 VMware vCenter 部署环境整合。在网络节点上安装了 NSX 插件后，它允许一个 NSX 控制器来集中管理配置设置并将它们推送到管理的网络节点上。当网络节点作为 hypervisor 添加到 NSX 控制器时，它们会被管理。

下图描绘了一些 VMware NSX 部署环境的例子。图一说明了不同计算节点上的虚拟机的流量，图二说明了同一台计算节点上的两台虚拟机之间的流量。注意替换 VMware NSX 插件和网络节点上的 neutron-server 服务。绿色的行表示 NSX 控制器和网络节点之间的管理关系。

图 7.2. VMware NSX 部署实例 — 两台计算节点

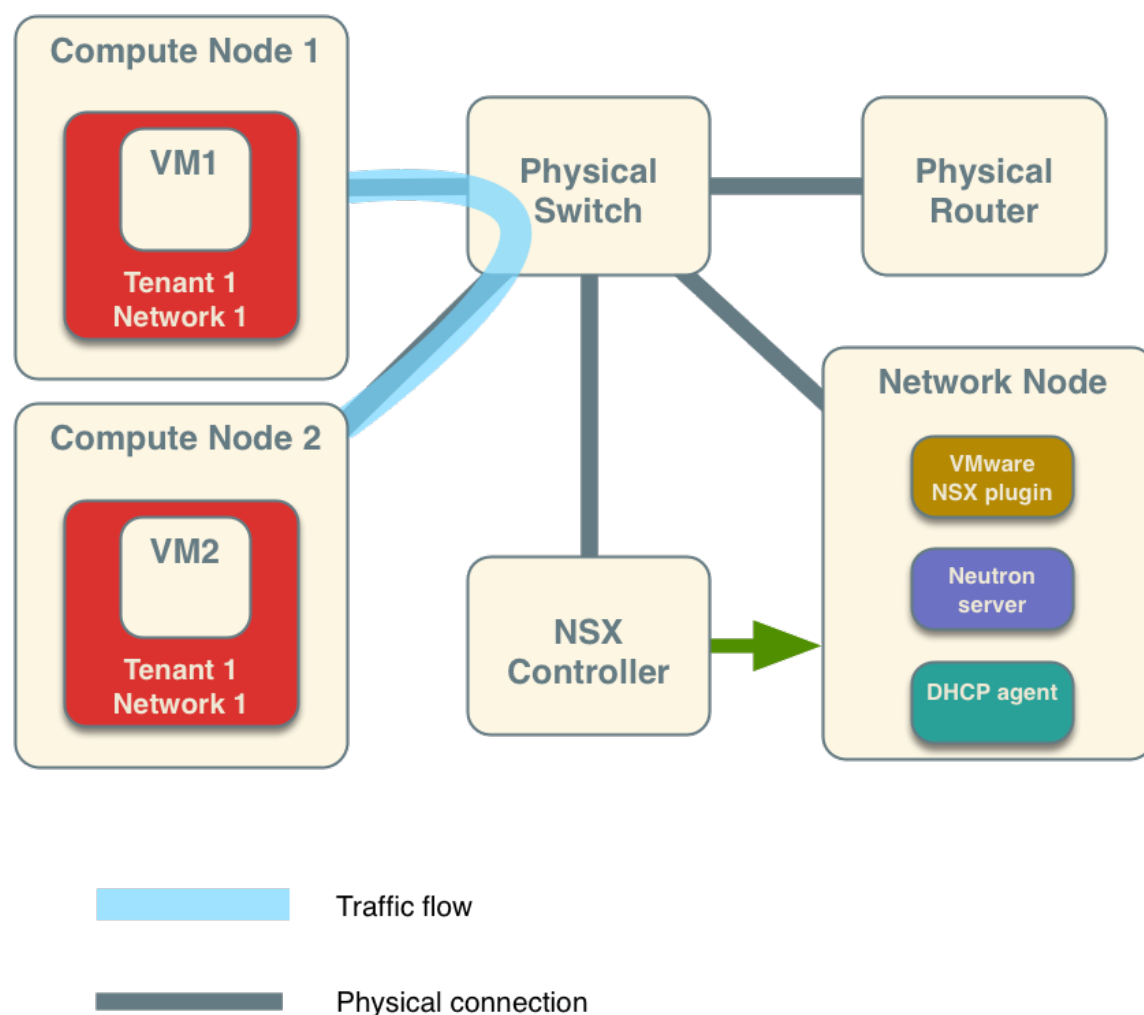
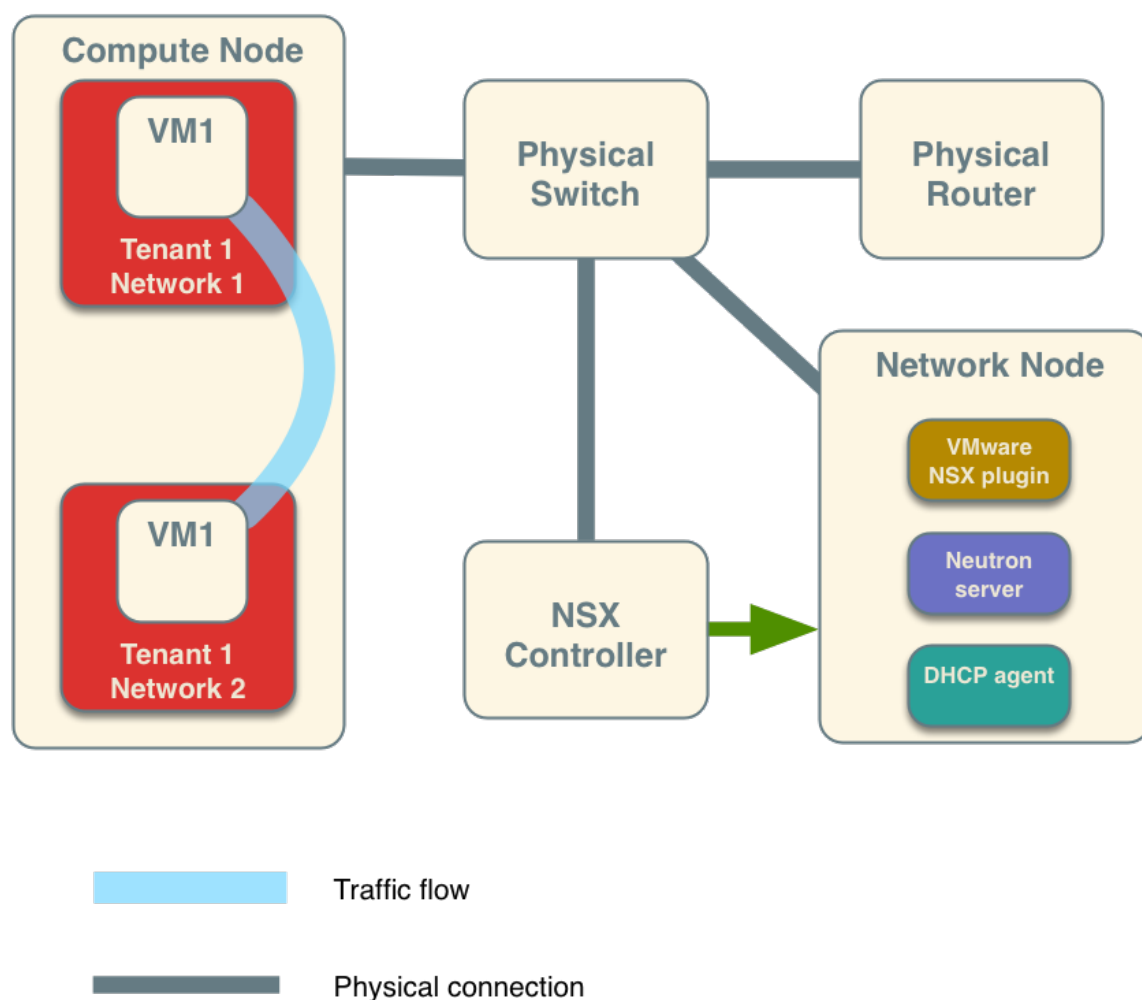


图 7.3. VMware NSX部署实例 – 单计算节点



为网络配置身份服务

为使用网络来配置身份服务

1. 创建get_id()函数

get_id()函数存放所创建对象的ID，以及避免了在后面步骤中需要复制和粘贴这些对象ID繁琐的过程：

a. 将下列函数添加到文件.bashrc中：

```
function get_id() {  
  echo `"$@" | awk '/ id / { print $4 }`  
}
```

b. Source .bashrc 文件。

```
$ source .bashrc
```

2. 创建网络服务入口

网络必须在计算服务中。创建服务：

```
$ NEUTRON_SERVICE_ID=$(get_id keystone service-create --name neutron --type network --description 'OpenStack Networking Service')
```

3. 创建网络服务端点入口

创建网络端点入口的方式取决于您是否使用 SQL 或模板目录驱动：

- 如果您使用的是 SQL driver，请以指定的 region (\$REGION)、网络服务器的 IP 地址 (\$IP) 和服务 ID (\$NEUTRON_SERVICE_ID，包含在前面的步骤中) 来执行下列命令：

```
$ keystone endpoint-create --region $REGION --service-id $NEUTRON_SERVICE_ID --publicurl 'http://$IP:9696/' --adminurl 'http://$IP:9696/' --internalurl 'http://$IP:9696/'
```

例如：

```
$ keystone endpoint-create --region myregion --service-id $NEUTRON_SERVICE_ID --publicurl "http://10.211.55.17:9696/" --adminurl "http://10.211.55.17:9696/" --internalurl "http://10.211.55.17:9696/"
```

- 如果您使用的是 template driver，请在您的计算服务模板文件 (default_catalog.templates) 中指定以下参数，包括 region (\$REGION) 和网络服务器的 IP 地址 (\$IP)。

```
catalog.$REGION.network.publicURL = http://$IP:9696
catalog.$REGION.network.adminURL = http://$IP:9696
catalog.$REGION.network.internalURL = http://$IP:9696
catalog.$REGION.network.name = Network Service
```

例如：

```
catalog.$Region.network.publicURL = http://10.211.55.17:9696
catalog.$Region.network.adminURL = http://10.211.55.17:9696
catalog.$Region.network.internalURL = http://10.211.55.17:9696
catalog.$Region.network.name = Network Service
```

4. 创建网络服务用户

您需要提供管理员用户的证书，计算服务和一些内部网络组件可能会用它来访问 Networking API。创建一个特定的 service 租户和该租户下的 neutron 用户，并给它分配 admin 角色。

a. 创建 admin 角色：

```
$ ADMIN_ROLE=$(get_id keystone role-create --name admin)
```

b. 创建 neutron 用户：

```
$ NEUTRON_USER=$(get_id keystone user-create --name neutron --pass "$NEUTRON_PASSWORD" --email demo@example.com --tenant-id service)
```

c. 创建 service 租户：

```
$ SERVICE_TENANT=$(get_id keystone tenant-create --name service --description "Services Tenant")
```

- d. 建立租户，用户和角色之间的关系：

```
$ keystone user-role-add --user_id $NEUTRON_USER --role_id $ADMIN_ROLE --tenant_id
$SERVICE_TENANT
```

关于如何创建服务的入口和用户更多信息，请根据你的发行版(docs.openstack.org)参阅OpenStack 安装指南。

计算

如果您使用的是网络节点，请不要运行计算节点上的 nova-network 服务(就像您在传统计算节点部署中所做的那样)。相反，计算节点代表了大多数 Networking 的网络相关的决定。计算节点代理面向租户的 API 调用，来为 Networking API 管理安全组和浮动 IP。然而，面向操作的工具，例如 nova-manage，没有被代理也不会被使用。



警告

当您配置了网络节点，您必需使用本指南。不要依靠计算网络文档或对计算节点的以前的经验。如果 nova 命令或关于网络的配置选项没有在本指南中提及，该命令很可能是不支持使用 Networking 的。特别是您不能使用 CLI 工具，如 nova-manage 和 nova 来用 Networking 管理网络或 IP 地址，包括固定 IP 和浮动 IP。



注意

在将它们使用为 Networking 之前，请卸载 nova-network 并重启那些已经运行了 nova-network 的物理节点。使用 Networking 时不经意地运行了 nova-network 的话会发生问题，它可能会使旧的 iptables 规则被之前运行的 nova-network 破坏掉。

为了保证计算节点与 Networking (而不是传统的 nova-network 机制) 正常工作，您必需调整 nova.conf 配置文件中的设置。

网络API和凭证配置

每次您在计算节点上提供或重新提供一台虚拟机，nova-* 服务会通过标准 API 与 Networking 进行会话。对于这个情况，您必需在 nova.conf 文件中设置下列元素(它们会被每个 nova-compute 和 nova-api 实例所使用)。

表 7.5. nova.conf API 和凭据设置

条目	配置
[DEFAULT] network_api_class	修改默认值为nova.network.neutronv2.api.API，来表明网络须使用传统的 nova-network 网络模式。
[neutron] url	为此部署的neutron-server实例更新主机名/IP和端口。
[neutron] auth_strategy	所有的生产环境请保留keystone 默认值。
[neutron] admin_tenant_name	更新上面章节的身份配置中所创建的租户服务名称。
[neutron] admin_username	更新上面章节中的身份配置中所创建的用户名。
[neutron] admin_password	更新上面章节中身份配置中所创建的用户密码。
[neutron] admin_auth_url	更新身份服务的IP和端口。这是身份服务(Keystone)管理API的服务IP和端口，并不是身份服务 API的IP和端口。

配置安全组

网络服务使用一个更灵活和比内置在计算节点上更强大的安全组的机制提供了安全组功能。因此，如果您使用的是 Networking，您应该禁用内置安全组并代理所有安全组到 Networking API 的调用。如果您没有禁用，安全策略会同时应用于服务并发生冲突。

要代理安全组的联网，在 nova.conf 中使用下面配置值：

表 7.6. nova.conf 安全组设置

条目	配置
firewall_driver	更新 nova.virt.firewall.NoopFirewallDriver，nova-compute 也就不要执行基于其自身的 iptables 过滤了。
security_group_api	更新到 neutron，所以全部的安全组请求都有网络服务代理了。

配置元数据

计算服务允许虚拟机通过一个到特定 169.254.169.254 地址的 web 请求来查寻虚拟机相关的元数据。Networking 支持代理这些请求给 nova-api，即使这些请求是从隔离的网络或从使用了重复 IP 地址的多网络发送过来的。

为了启用请求代理，您必需更新 nova.conf 中 [neutron] 小节的以下字段。

表 7.7. nova.conf 元数据设置

条目	配置
service_metadata_proxy	修改为 true，否则 nova-api 不会正常响应来自 neutron-metadata-agent 的请求。
metadata_proxy_shared_secret	修改 "password" 字符串的值。您还需要在 metadata_agent.ini 文件中配置相同的值，以认证元数据的请求。 在这两个文件中的空字符的默认值会允许使用元数据的功能，但如果有非信任的实体访问 nova-api 暴露的元数据 API，这样将是不安全的。



注意

防御起见，即使使用了 metadata_proxy_shared_secret，也建议您不要使用同一个用于租户的 nova-api 实例暴露了元数据。相反，您应该为仅在您管理网络上可用的元数据运行一个 nova-api 实例的专用设置。一个给定的 nova-api 实例是否暴露了元数据 API 是由 nova.conf 中的 enabled_apis 决定的。

nova.conf 实例(针对 nova-compute 和 nova-api)

对于上面的设置示例值，假设运行计算和网络服务的云控制器节点的 IP 地址是 192.168.1.2:

```
[DEFAULT]
security_group_api=neutron
network_api_class=nova.network.neutronv2.api.API
firewall_driver=nova.virt.firewall.NoopFirewallDriver

[neutron]
url=http://192.168.1.2:9696
```

```
auth_strategy=keystone
admin_tenant_name=service
admin_username=neutron
admin_password=password
admin_auth_url=http://192.168.1.2:35357/v2.0
service_metadata_proxy=true
metadata_proxy_shared_secret=foo
```

高级配置选项

此章阐释了各种系统组件的高级配置属性。例如，默认的配置属性是可正常工作的，但是用户要定制属性。在安装完成软件包后，`$NEUTRON_CONF_DIR` 就是 `/etc/neutron`。

3层计量代理

您可以运行一个 L3 计量代理以启用 layer-3 流量的计量。通常情况下，您需要在所有运行 L3 代理的节点上启动计量代理：

```
neutron-metering-agent --config-file NEUTRON_CONFIG_FILE --config-file L3_METERING_CONFIG_FILE
```

您必需配置一个匹配运行在服务中插件的驱动。驱动添加计量到路由接口上。

表 7.8. 设置

选项	值
Open vSwitch	
interface_driver (\$NEUTRON_CONF_DIR/ metering_agent.ini)	neutron.agent.linux.interface.OVSInterfaceDriver
Linux 网桥	
interface_driver (\$NEUTRON_CONF_DIR/ metering_agent.ini)	neutron.agent.linux.interface.BridgeInterfaceDriver

命名空间

计量代理和 L3 代理必需有相同的网络命名空间配置。



注意

如果 Linux 的安装不支持网络命名空间，您必需禁用 L3 计量配置文件中的网络命名空间。`use_namespaces` 选项的默认值是 `True`。

```
use_namespaces = False
```

3层计量驱动

您必需配置那些实现计量抽象的驱动。目前可用的唯一实现使用了 `iptables` 来计量。

```
driver = neutron.services.metering.drivers.iptables.iptables_driver.IptablesMeteringDriver
```

3层计量服务驱动

要启用 L3 计量，您必需设置运行 `systemitem class="service">neutron-server`

```
service_plugins = metering
```

DHCP代理的扩展性和高可用性

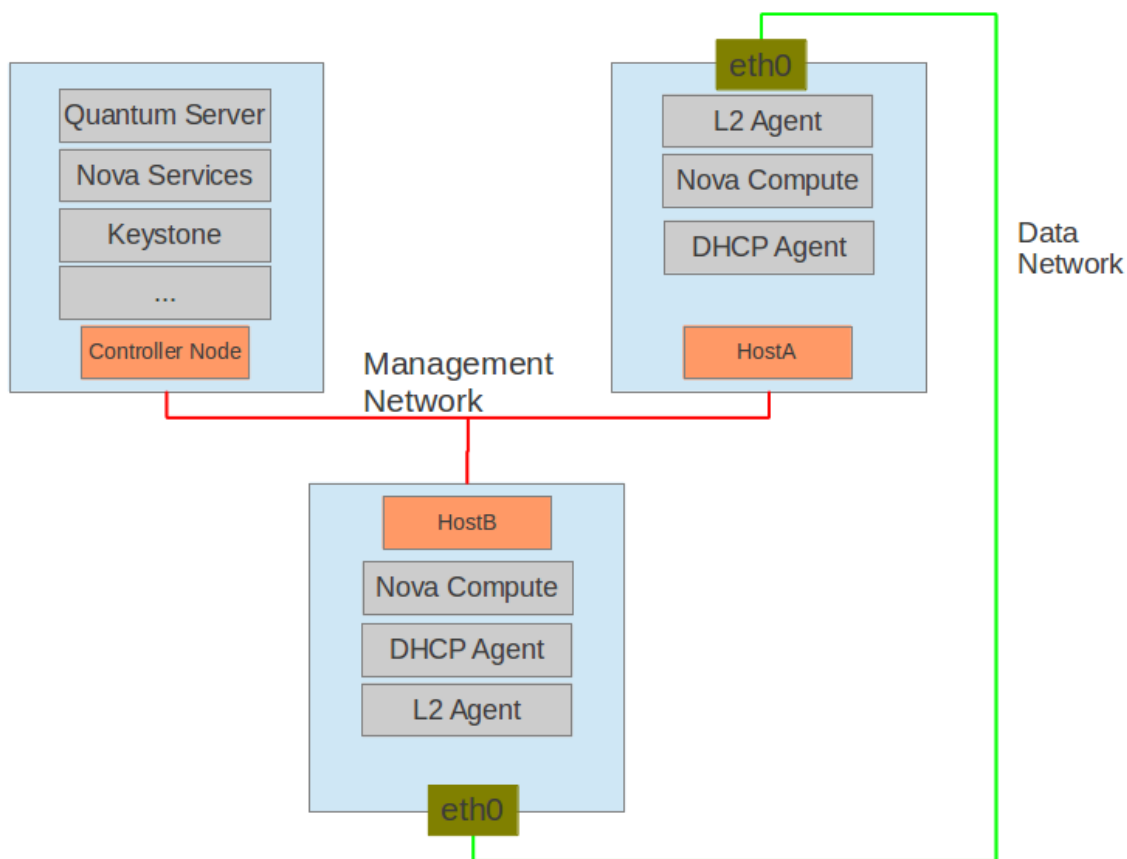
本小节描述了如何使用代理管理（别名 agent）和调度器（别名 agent_scheduler）扩展来实现 DHCP 的扩展性和 HA。



注意

使用neutron ext-list客户端命令来检查这些扩展是否已经启用：

```
$ neutron ext-list -c name -c alias
+-----+-----+
| alias   | name           |
+-----+-----+
| agent_scheduler | Agent Schedulers |
| binding      | Port Binding      |
| quotas       | Quota management support |
| agent        | agent            |
| provider     | Provider Network  |
| router       | Neutron L3 Router |
| lbaas        | LoadBalancing service |
| extraroute   | Neutron Extra Route |
+-----+-----+
```



步骤中有三台主机。

表 7.9. 功能

主机	描述
OpenStack 控制器—控制节点	部署虚拟机需要运行网络，认证和计算服务。节点必须拥有至少一块网络用来连接到管理网络。 注意，不可运行nova-network，因为其已被Neutron取代。
HostA	运行 nova-compute, Neutron L2 代理和 DHCP 代理
HostB	和 HostA 一样

配置

控制节点: Neutron 服务

1. Neutron 配置文件 /etc/neutron/neutron.conf:

```
[DEFAULT]
core_plugin = linuxbridge
rabbit_host = controlnode
allow_overlapping_ips = True
host = controlnode
agent_down_time = 5
```

2. 更新插件配置文件/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini:

```
[vlangs]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1000:2999
[database]
connection = mysql://root:root@127.0.0.1:3306/neutron_linux_bridge
retry_interval = 2
[linux_bridge]
physical_interface_mappings = physnet1:eth0
```

主机A和主机B: 2层代理

1. Neutron 配置文件 /etc/neutron/neutron.conf:

```
[DEFAULT]
rabbit_host = controlnode
rabbit_password = openstack
# host = HostB on hostb
host = HostA
```

2. 更新插件配置文件/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini:

```
[vlangs]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1000:2999
[database]
connection = mysql://root:root@127.0.0.1:3306/neutron_linux_bridge
retry_interval = 2
[linux_bridge]
physical_interface_mappings = physnet1:eth0
```

3. 更新nova配置文件 /etc/nova/nova.conf:

```
[DEFAULT]
network_api_class=nova.network.neutronv2.api.API
firewall_driver=nova.virt.firewall.NoopFirewallDriver

[neutron]
admin_username=neutron
admin_password=servicepassword
admin_auth_url=http://controlnode:35357/v2.0/
auth_strategy=keystone
admin_tenant_name=servicetenant
url=http://100.1.1.10:9696/
```

主机A和主机B: DHCP代理

- 更新DHCP配置文件 /etc/neutron/dhcp_agent.ini:

```
[DEFAULT]
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
```

在代理管理和调度扩展中的命令

以下命令需要运行的命令的租户是管理员角色。



注意

确保设置了下面的环境变量，这些都是多个客户端用于访问身份服务的。

```
export OS_USERNAME=admin
export OS_PASSWORD=adminpassword
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://controlnode:5000/v2.0/
```

设置

- 要完成实验，你需要虚拟机和一个neutron网络:

```
$ nova list
+-----+-----+-----+-----+
| ID              | Name      | Status | Networks |
+-----+-----+-----+-----+
| c394fcd0-0baa-43ae-a793-201815c3e8ce | myserver1 | ACTIVE | net1=10.0.1.3 |
| 2d604e05-9a6c-4ddb-9082-8a1fbdcc797d | myserver2 | ACTIVE | net1=10.0.1.4 |
| c7c0481c-3db8-4d7a-a948-60ce8211d585 | myserver3 | ACTIVE | net1=10.0.1.5 |
+-----+-----+-----+-----+

$ neutron net-list
+-----+-----+-----+-----+
| id              | name      | subnets |
+-----+-----+-----+-----+
| 89dca1c6-c7d4-4f7a-b730-549af0fb6e34 | net1      | f6c832e3-9968-46fd-8e45-d5cf646db9d1 |
+-----+-----+-----+-----+
```

在neutron部署中管理代理

每个代理都支持这些扩展，当neutron服务启动时会注册自己。

- 列出所有代理：

```
$ neutron agent-list
```

id	agent_type	host	alive	admin_state_up
1b69828d-6a9b-4826-87cd-1757f0e27f31	Linux bridge agent	HostA	:-)	True
a0c1c21c-d4f4-4577-9ec7-908f2d48622d	DHCP agent	HostA	:-)	True
ed96b856-ae0f-4d75-bb28-40a47ffd7695	Linux bridge agent	HostB	:-)	True
f28aa126-6edb-4ea5-a81e-8850876bc0a8	DHCP agent	HostB	:-)	True

输出显示了四个代理的信息。alive栏显示:-)，如果在neutron.conf中定义了agent_down_time属性，此即是期间由代理所报告的状态。另外，alive是xxx。

2. 驻扎在特定网络的DHCP代理列表

在一些部署中，一个DHCP代理无法满足掌控所有的网络数据。另外，当部署还算小的时候你就必须开始备份。相同的网络可以分配给多于一个的DHCP代理，一个DHCP代理也可以有多个网络。

驻扎在指定网络的DHCP代理列表：

```
$ neutron dhcp-agent-list-hosting-net net1
```

id	host	admin_state_up	alive
a0c1c21c-d4f4-4577-9ec7-908f2d48622d	HostA	True	:-)

3. 通过给定的DHCP代理列出所托管的网络。

此命令显示了那个网络时给定的DHCP代理所管理的。

```
$ neutron net-list-on-dhcp-agent a0c1c21c-d4f4-4577-9ec7-908f2d48622d
```

id	name	subnets
89dca1c6-c7d4-4f7a-b730-549af0fb6e34	net1	f6c832e3-9968-46fd-8e45-d5cf646db9d1 10.0.1.0/24

4. 显示代理细节。

agent-show 命令展示了一个指定代理的细节：

```
$ neutron agent-show a0c1c21c-d4f4-4577-9ec7-908f2d48622d
```

Field	Value
admin_state_up	True
agent_type	DHCP agent
alive	False
binary	neutron-dhcp-agent
configurations	{
	"subnets": 1,
	"use_namespaces": true,
	"dhcp_driver": "neutron.agent.linux.dhcp.Dnsmasq",
	"networks": 1,
	"dhcp_lease_time": 120,
	"ports": 3

```

|      | }      |
|created_at      | 2013-03-16T01:16:18.000000      |
|description      |      |
|heartbeat_timestamp | 2013-03-17T01:37:22.000000      |
|host      | HostA      |
|id      | 58f4ce07-6789-4bb3-aa42-ed3779db2b03      |
|started_at      | 2013-03-16T06:48:39.000000      |
|topic      | dhcp_agent      |
+-----+-----+

```

在这个输出中，`heartbeat_timestamp` 是 `neutron` 服务器的时间。您不需要为该扩展的正确执行而将所有的代理与这个时间同步。`configurations` 描述了代理或执行数据的静态配置。这个代理是一个 DHCP 代理，且它托管一个网络、一个子网和三个端口。

不同类型的代理会显示不同的细节。下面的输出是Linux网桥代理的信息：

```

$ neutron agent-show ed96b856-ae0f-4d75-bb28-40a47ffd7695
+-----+-----+
|Field      |Value      |
+-----+-----+
|admin_state_up | True      |
|binary      | neutron-linuxbridge-agent      |
|configurations | {         |
|             | "physnet1": "eth0",         |
|             | "devices": "4"         |
|             | }         |
|created_at      | 2013-03-16T01:49:52.000000      |
|description      |      |
|disabled      | False      |
|group      | agent      |
|heartbeat_timestamp | 2013-03-16T01:59:45.000000      |
|host      | HostB      |
|id      | ed96b856-ae0f-4d75-bb28-40a47ffd7695 |
|topic      | N/A      |
|started_at      | 2013-03-16T06:48:39.000000      |
|type      | Linux bridge agent      |
+-----+-----+

```

该输出显示了 `bridge-mapping` 和在这个 L2 代理上的虚拟网络设备的数目。

管理网络DHCP代理的任务

现在您已经执行了 `net-list-on-dhcp-agent` 和 `dhcp-agent-list-hosting-net` 命令，您可以进行添加一个网络到 DHCP 代理以及从中移除一个网络的操作。

1. 默认调度。

当您以一个端口创建了一个网络，您可以将它调度到一个激活的 DHCP 代理中。如果有很多激活的代理正在运行，则随机选择一个。您可以以与日后的 `nova-schedule` 相同的方式设计更复杂的调度算法。

```

$ neutron net-create net2
$ neutron subnet-create net2 9.0.1.0/24 --name subnet2
$ neutron port-create net2
$ neutron dhcp-agent-list-hosting-net net2
+-----+-----+-----+-----+
|id      |host |admin_state_up |alive |
+-----+-----+-----+-----+
|a0c1c21c-d4f4-4577-9ec7-908f2d48622d |HostA |True      |:-) |

```

它被分配给 HostA 的 DHCP 代理。如果您想通过 `dnsmasq` 命令验证其行为，您必需为该网络创建一个子网，因为 DHCP 代理仅在有 DHCP 的情况下启动 `dnsmasq` 服务。

2. 分配网络给指定的DHCP代理。

要添加另外的DHCP代理给托管的网络，运行此命令：

```
$ neutron dhcp-agent-network-add f28aa126-6edb-4ea5-a81e-8850876bc0a8 net2
Added network net2 to dhcp agent
$ neutron dhcp-agent-list-hosting-net net2
```

id	host	admin_state_up	alive
a0c1c21c-d4f4-4577-9ec7-908f2d48622d	HostA	True	:-)
f28aa126-6edb-4ea5-a81e-8850876bc0a8	HostB	True	:-)

两个DHCP代理均托管在net2网络。

3. 从指定到DHCP代理删除一个网络。

此命令是前一个到伙伴命令。为HostA从DHCP代理删除net2：

```
$ neutron dhcp-agent-network-remove a0c1c21c-d4f4-4577-9ec7-908f2d48622d net2
Removed network net2 to dhcp agent
$ neutron dhcp-agent-list-hosting-net net2
```

id	host	admin_state_up	alive
f28aa126-6edb-4ea5-a81e-8850876bc0a8	HostB	True	:-)

你可以看到只有HostB的DHCP代理托管了net2网络。

DHCP代理的高可用

在 net2 上启动一台虚拟机。让两个 DHCP 代理托管 net2。使代理依次故障，以查看虚拟机是否仍然获得了想要的 IP。

1. 在net2中启动一个虚拟机。

```
$ neutron net-list
```

id	name	subnets
89dca1c6-c7d4-4f7a-b730-549af0fb6e34	net1	f6c832e3-9968-46fd-8e45-d5cf646db9d1 10.0.1.0/24
9b96b14f-71b8-4918-90aa-c5d705606b1a	net2	6979b71a-0ae8-448c-aa87-65f68eedcaaa 9.0.1.0/24

```
$ nova boot --image tty --flavor 1 myserver4
--nic net-id=9b96b14f-71b8-4918-90aa-c5d705606b1a
$ nova list
```

ID	Name	Status	Networks
c394fcd0-0baa-43ae-a793-201815c3e8ce	myserver1	ACTIVE	net1=10.0.1.3
2d604e05-9a6c-4ddb-9082-8a1fbdcc797d	myserver2	ACTIVE	net1=10.0.1.4
c7c0481c-3db8-4d7a-a948-60ce8211d585	myserver3	ACTIVE	net1=10.0.1.5


```
| f62f4731-5591-46b1-9d74-f0c901de567f | myserver4 | ACTIVE | net2=9.0.1.2 |
+-----+-----+-----+-----+
```

2. 确保'net2'上有全部的DHCP代理。

使用前面的命令来分配网络给代理。

```
$ neutron dhcp-agent-list-hosting-net net2
+-----+-----+-----+-----+
| id                | host | admin_state_up | alive |
+-----+-----+-----+-----+
| a0c1c21c-d4f4-4577-9ec7-908f2d48622d | HostA | True           | :- ) |
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | HostB | True           | :- ) |
+-----+-----+-----+-----+
```

测试高可用

1. 登录到myserver4虚拟机，然后运行udhcpd, dhclient或者其它DHCP客户单。
2. 在HostA中停止DHCP代理。另外要停止neutron-dhcp-agent，你必须停止 dnsmasq进程。
3. 在虚拟机中运行DHCP客户单，以检查其能够获得意料中的IP。
4. 同样在主机B中停止DHCP代理。
5. 在VM中运行udhcpd；它无法获得想要的IP。
6. 在主机B中启动DHCP代理。虚拟机再次获得了想要的IP。

禁用和删除一个代理

如果计划升级系统硬件或软件，管理员可能想要禁用代理。一些支持调度的代理也支持禁用和启用代理，例如 L3 和 DHCP 代理。代理禁用后，调度器不会再调度新的资源给代理。禁用代理后，您可以安全地将代理移除。请在您删除代理之前移除代理上的资源。

- 要运行下面的命令，你必须在主机A中停止DHCP代理。

```
$ neutron agent-update --admin-state-up False a0c1c21c-d4f4-4577-9ec7-908f2d48622d
$ neutron agent-list
+-----+-----+-----+-----+
| id                | agent_type | host | alive | admin_state_up |
+-----+-----+-----+-----+
| 1b69828d-6a9b-4826-87cd-1757f0e27f31 | Linux bridge agent | HostA | :- ) | True           |
| a0c1c21c-d4f4-4577-9ec7-908f2d48622d | DHCP agent      | HostA | :- ) | False          |
| ed96b856-ae0f-4d75-bb28-40a47ffd7695 | Linux bridge agent | HostB | :- ) | True           |
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | DHCP agent      | HostB | :- ) | True           |
+-----+-----+-----+-----+
$ neutron agent-delete a0c1c21c-d4f4-4577-9ec7-908f2d48622d
Deleted agent: a0c1c21c-d4f4-4577-9ec7-908f2d48622d
$ neutron agent-list
+-----+-----+-----+-----+
| id                | agent_type | host | alive | admin_state_up |
+-----+-----+-----+-----+
| 1b69828d-6a9b-4826-87cd-1757f0e27f31 | Linux bridge agent | HostA | :- ) | True           |
| ed96b856-ae0f-4d75-bb28-40a47ffd7695 | Linux bridge agent | HostB | :- ) | True           |
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | DHCP agent      | HostB | :- ) | True           |
+-----+-----+-----+-----+
```

丢失后，若重启DHCP代理的话，它会再次回到代理列表中。

使用网络

你可以通过使用service 命令来管理OpenStack网络服务。例如:

```
# service neutron-server stop
# service neutron-server status
# service neutron-server start
# service neutron-server restart
```

日志文件都在目录/var/log/neutron 下。

配置文件都在 /etc/neutron目录下。

云管理员和租户可以使用 OpenStack 网络来构建丰富的网络拓扑。云管理员可以代替租户创建网络连接。

核心网络API特性

安装和配置了网络服务之后，租户和管理员可以通过直接使用 Networking API 或 neutron 命令行接口 (CLI) 来执行 create-read-update-delete (CRUD) API 网络操作。neutron CLI 是一个 Networking API 的封装。每个 Networking API 调用都有一个对应的 neutron 命令。

命令行包含有多个属性，关于细节，请参阅[OpenStack 最终用户指南](#)。

基本网络操作

要学习关于通过使用neutron命令行接口(CLI)实现的高级功能，请阅读 [OpenStack 最终用户指南](#)中的网络部分。

此表展示了一个neutron的例子，其可以让你完成网络的基本操作：

表 7.10. 基本网络操作

操作	命令
创建一个网络。	\$ neutron net-create net1
创建一个分配给net1的子网。	\$ neutron subnet-create net1 10.0.0.0/24
列出指定租户的端口。	\$ neutron port-list
为指定租户列出端口以及显示id, fixed_ips, 和 device_owner 列。	\$ neutron port-list -c id -c fixed_ips -c device_owner
显示一个指定端口的信息。	\$ neutron port-show PORT_ID



注意

device_owner的地方描述了端口的属主。一个端口谁的 device_owner以下面开始

- network由网络服务所创建。
- compute 由计算服务所创建。

管理操作

云管理员可以通过在命令行中指定身份 `tenant_id` 来运行租户行为的任何 `neutron` 命令，如下面所示：

```
$ neutron net-create --tenant-id TENANT_ID NETWORK_NAME
```

例如：

```
$ neutron net-create --tenant-id 5e4bbe24b67a4410bc4d9fae29ec394e net1
```



注意

在身份服务中要查看所有租户的ID，以身份服务管理员用户运行下面的命令：

```
$ keystone tenant-list
```

高级网络操作

此表展示了网络命令的例子，可以完成高级的网络操作：

表 7.11. 高级网络操作

操作	命令
创建一个所有租户可使用的网络。	<code>\$ neutron net:create --shared public-net</code>
为一个给定网络IP地址创建一个子网。	<code>\$ neutron subnet:create --gateway 10.0.0.254 net1 10.0.0.0/24</code>
创建一个没有网关IP地址的子网。	<code>\$ neutron subnet:create --no-gateway net1 10.0.0.0/24</code>
创建一个禁用DHCP的子网。	<code>\$ neutron subnet:create net1 10.0.0.0/24 --enable-dhcp False</code>
为指定的一组主机路由创建一个子网。	<code>\$ neutron subnet:create test-net1 40.0.0.0/24 --host-routes type=dict list=true destination=40.0.1.0/24,nexthop=40.0.0.2</code>
为指定的一组dns域名服务创建一个子网。	<code>\$ neutron subnet:create test-net1 40.0.0.0/24 --dns-nameservers list=true 8.8.4.4 8.8.8.8</code>
展示所分配动网络所有的端口和IP。	<code>\$ neutron port-list --network_id NET_ID</code>

使用有网络的计算服务

基本的计算和网络操作

此表展示了neutron和nova命令的例子，其可完成基本的虚拟机网络操作：

表 7.12. 基本的计算和网络操作

动作	命令
检查可用网络。	<code>\$ neutron net-list</code>
在选择的网络中启动一个单一网卡的虚拟机。	<code>\$ nova boot --image 镜像 --flavor FLAVOR --nic net-id=NET_ID VM_NAME</code>
基于device_id来搜索端口匹配到计算实例的UUID。请参考 创建和删除虚拟机 [211] 。	<code>\$ neutron port-list --device_id VM_ID</code>
搜索端口，但是仅展示mac_address的端口。	<code>\$ neutron port-list --field mac_address --device_id VM_ID</code>
临时禁用端口的来源流量。	<code>\$ neutron port-update PORT_ID --admin_state_up False</code>



注意

device_id也可以是逻辑路由ID。



创建和删除虚拟机

- 当启动一个虚拟机时，网络的端口对应到虚拟机的网卡会自动创建，且会分配默认的安全组。可以配置[安全组规则](#)从而让用户可以访问虚拟机。
- 当删除一台虚拟机时，此虚拟机所对应动网络端口也会自动的删除。

高级虚拟机创建操作

此表展示了nova和neutron的命令，其可完成高级的虚拟机创建操作：

表 7.13. 高级虚拟机创建操作

操作	命令
启动一个多网卡的虚拟机。	<code>\$ nova boot --image 镜像 --flavor FLAVOR --nic net-id=NET1-ID --nic net-id=NET2-ID VM_NAME</code>
启动一个指定IP地址的虚拟机。注意在此场景下你不可以使用参数 <code>--num-instances</code> 。	<code>\$ nova boot --image 镜像 --flavor FLAVOR --nic net-id=NET-ID,v4-fixed-ip=IP-ADDR VM_NAME</code>
启动一台连接到可访问提交请求的租户的所有网络的虚拟机（不使用 <code>--nic</code> 选项）。	<code>\$ nova boot --image 镜像 --flavor FLAVOR VM_NAME</code>



注意

发行版厂商通常提供的云镜像仅激活一块网络。当你启动多块网卡时，你必须在镜像中配置所添加的网卡，否则哪些网卡是无法访问的。

下面的Debian/Ubuntu的实例说明了如何设置网卡，在实例中的/etc/network/interfaces 文件中设置。必须应用这些配置到镜像。

```
# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet dhcp
```

在虚拟机激活ping和SSH(安全组)

配置安全组规则取决于你使用的插件类型。如果你使用的插件有：

- 要实现网络安全组，你可以通过使用命令 `neutron security-group-rule-create` 直接配置安全组规则。下面的例子启用了 ping 和 ssh 访问你的虚拟机。

```
$ neutron security-group-rule-create --protocol icmp
--direction ingress default
```

```
$ neutron security-group-rule-create --protocol tcp --port-range-min 22
--port-range-max 22 --direction ingress default
```

- 要是不实现网络安全组，你可以通过使用 `nova secgroup-add-rule` 或 `euca-authorize` 命令来配置安全组规则。这些 nova 命令启用 ping 和 ssh 来访问你的虚拟机。

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```



注意

如果您的插件实现了网络服务安全组，您也可以通过设置 `nova.conf` 文件中的 `security_group_api = neutron` 来使用计算服务安全组。当您设置了这些选项后，所有计算服务安全组的命令会被网络服务所代理。

通过API扩展的高级功能

一些插件实现了可以提供类似于 `nova-network` 中可用功能的 API 扩展：OpenStack 社区可能会对这些插件感兴趣。

供应商网络

网络可以被分类为租户网络或供应商网络。租户网络由普通用户创建且关于他们如何在物理上实现的细节对这些用户是隐藏的。供应商网络是以拥有管理员权限的证书创建的，指定了网络物理实现的细节，通常与一些数据中心的现有网络相匹配。

供应商网络允许云管理员创建 `Networking` 网络，它会直接映射到数据中心的物理网络中。这通常用于给租户直接访问到可以到达因特网的公共网络。它也可能用于与网络中的 VLANs 整合，它已经有一个定义好的含义（如，允许一台“市场”部门的虚拟机放置到同一个 VLAN 作为通一个数据中心的裸金属市场主机）。

供应商扩展允许管理员明确地管理 `Networking` 虚拟网络和底层物理机制如 VLAN、隧道之间的关系。如果支持该扩展，`Networking` 的具有管理员权限的客户端用户可以看到所有虚拟网络上的附加的供应商属性并可以指定这些属性来创建供应商网络。

供应商扩展是由 Open vSwitch 和 Linux 桥接插件支持的。这些插件的配置需要对该扩展非常精通。

术语

在供应商扩展和支持供应商扩展的插件配置中使用了一系列术语：

表 7.14. 提供者扩展的术语

术语	描述
虚拟网络	Networking L2 网络 (由 UUID 和可选名称标识) 的端口可以作为 vNIC 附加到计算实例和各种 Networking 代理中。Open vSwitch 和 Linux Bridge 插件均支持一些不同的机制来实现虚拟网络。
物理网络	网络相互连接虚拟化主机 (例如计算节点) 和其他网络资源。每个物理网络应该支持多个虚拟网络。供应商扩展和插件配置使用简单的字符串名称来识别物理网络。
租户网络	租户或管理员创建的一个虚拟网络。该网络的物理细节是不会暴露给租户的。
提供者网络	虚拟网络管理创建来映射到数据中心里的特定网络中，特别是允许直接访问该网络上的非 OpenStack 资源。租户可以有访问供应商网络的权限。
VLAN 网络	作为特定的包含特定 VID 字段值的 IEEE 802.1Q 头的物理网络的包实现的一个虚拟网络。VLAN 网络共享同一个物理网络，它在 L2 中是相互隔离的，并甚至可能有重复的 IP 地址。每个不同的支持 VLAN 网络的物理网络被认为是分离的 VLAN 躯干，包含一个分离的 VID 值空间。有效的 VID 值是 1 到 4094。
扁平化网络	作为特定的包含 IEEE 802.1Q 头的物理网络的包实现的一个虚拟网络。每个物理网络最多可以实现一个扁平化网络。
本地网络	允许在每台主机中但不通过网络进行通信的一个虚拟网络。本地网络主要用于单节点的测试场景，但也可以有其他的用途。
GRE 网络	作为使用 GRE 封装的网络包实现的一个虚拟网络。GRE 网络也指隧道。GRE 隧道包是由主机的 IP 路由表转发的，所以 GRE 网络与 Networking 特定的物理网络无关。
虚拟扩展 LAN (VXLAN) 网络	VXLAN 是一个被提出的用于运行一个覆盖已有 3 层架构网络的封装协议。覆盖网络是构建在已有网络 2 层和 3 层拓扑之上的虚拟网络，以支持干性计算架构。

ML2、Open vSwitch 和 Linux Bridge 插件支持 VLAN 网络、扁平化网络和本地网络。目前只有 ML2 和 Open vSwitch 插件支持 GRE 和 VXLAN 网络，提供了所需的存在于主机中的 Linux 内核、Open vSwitch 和 iproute2 包的特性。

供应商属性

供应商扩展以这些属性扩展了 Networking 网络资源：

表 7.15. 供应商网络属性

属性名称	类型	默认值	描述
provider:network_type	字符串	无	虚拟网络所实现的物理机制。可能的值为 flat、vlan、本地、gre 和 vxlan，对应于上述定义的扁平化网络、VLAN 网络、本地网络、GRE 网络和 VXLAN 网络。所有供应商网络的类型都可以由管理员创建，而租户网络可以根据插件的配置作为 vlan、gre、vxlan 或本地网络类型实现。
provider:physical_network	字符串	如果一个物理网络的名称已经配置为"default"，而且如果 provider:network_type 是flat 或 vlan，那么使用多就是"default"。	用于实现扁平化和 VLAN 网络的虚拟网络所基于的物理网络的名称。不适用于本地 或 gre 网络类型。
provider:segmentation_id	整型	无	对于 VLAN 网络来说，是实现了虚拟网络的物理网络上的 VLAN VID。有效的 VLAN VID 从 1 到

属性名称	类型	默认值	描述
			4094。对于 GRE 网络来说，是隧道 ID。有效的隧道 ID 是 32 位的无符号整数。不适用于 flat 或本地网络类型。

要浏览或设置供应商扩展属性，客户端必需在 Networking 策略配置中授权 `extension:provider_network:view` 和 `extension:provider_network:set` 操作。默认的配置对 `admin` 角色均授权了这两种操作。已经授权的客户端或具有管理员权限的用户可以通过调用 Networking API 浏览或设置供应商扩展的属性。关于策略配置的详情请阅读 [“认证和授权”一节 \[224\]](#)。

3层路由和网络地址转换

Networking API 提供了抽象的用于实现 L2 网络的技术中分离出来的 L2 网段。Networking 包括了一个提供抽象 L3 路由的 API 扩展，API 用户可以动态地提供和配置。这些 Networking 路由可以连接到多个 L2 Networking 网络，而且还可以提供一个连接一个或更多私有 L2 网络到共享的外部网络上的网关。例如，一个公有网络对因特网的访问。请阅读 [OpenStack Configuration Reference](#) 以了解更多关于部署 Networking L3 路由的通用模块的详细信息。

L3 路由提供了基本的网关端口上的 NAT 功能，上行流量从路由到外部网络。这默认转发了 SNAT 的所有流量，并支持浮动 IP，可以创建一个静态的从外部网络上的公有 IP 到附加到该路由上的其中一个子网上的私有 IP 的一对一的映射。这使租户可以选择性地将私有网络上的虚拟机暴露到其他公有网络上的主机上（以及暴露到所有因特网里的主机上）。您可以根据需要分配并在一个端口到另一个端口上映射浮动 IP。

基本的L3操作

扩展网络对所有用户均可见。但是，默认的规则设置是仅管理员用户可以创建、更新和删除扩展网络。

此表展示了一个neutron的例子，其可以让你完成3层的基本操作：

表 7.16. 基本的L3操作

操作	命令
创建外部网络。	# neutron net-create public --router:external True \$ neutron subnet-create public 172.16.1.0/24
列出外部网络。	\$ neutron net-list --router:external True
创建一个仅内部使用的私密连接到多个 L2 网络的路由。	\$ neutron net-create net1 \$ neutron subnet-create net1 10.0.0.0/24 \$ neutron net-create net2 \$ neutron subnet-create net2 10.0.1.0/24 \$ neutron router-create router1 \$ neutron router-interface-add router1 SUBNET1_UUID \$ neutron router-interface-add router1 SUBNET2_UUID
连接一个路由到外部网络上，这使得路由将作为一个外部连接的 NAT 网关。	\$ neutron router-gateway-set router1 EXT_NET_ID 路由获取一个包含子网的 gateway_ip 地址的接口，且这个接口被附加到与该子网相关的 L2 Networking 网络的端口上。该路由也获得一个到特定外部网络的网关接口。这提供了到外部网络的 SNAT 连接，并支持分配到该外部网络的浮动 IP。通常一个外部网络会映射到一个供应商的网络中
路由列表。	\$ neutron router-list
展示特定的路由的信息。	\$ neutron router-show ROUTER_ID
为一个路由展示所有的内部网卡。	\$ neutron router-port-list ROUTER_ID \$ neutron router-port-list ROUTER_NAME
识别表示浮动 IP 所映射到的虚拟机网卡的 PORT_ID。	\$ neutron port-list -c id -c fixed_ips --device_id INSTANCE_ID 这个端口必需在一个附加到一个通过创建浮动 IP 上行到外部网络的路由的 Networking 子网上。从概念上讲，这是由于路由必需能够 Destination NAT (DNAT)，重写由浮动 IP 地址（从外网上的子网中选择的）到内部固定 IP（从绑定到路由上的私有子网中选择的）的包。
创建一个浮动IP地址然后将之分配给一个端口。	\$ neutron floatingip-create EXT_NET_ID \$ neutron floatingip-associate FLOATING_IP_ID INTERNAL_VM_PORT_ID
一步创建浮动IP地址并将之分配给端口。	\$ neutron floatingip-create --port_id INTERNAL_VM_PORT_ID EXT_NET_ID
列出浮动IP。	\$ neutron floatingip-list
基于指定的虚拟机端口发现浮动IP。	\$ neutron floatingip-list --port_id ZZZ
去掉浮动IP地址的关联。	\$ neutron floatingip-disassociate FLOATING_IP_ID
删除浮动IP地址。	\$ neutron floatingip-delete FLOATING_IP_ID
清除网关。	\$ neutron router-gateway-clear router1
从路由中删除网卡。	\$ neutron router-interface-delete router1 SUBNET_ID
删除路由	\$ neutron router-delete router1

安全组

安全组和安全组规则允许管理员和租户可以指定流量类型和允许通过端口的方向 (ingress/egress)。安全组是一个安全组规则的容器。

当在 Networking 中创建了一个端口，它将与安全组关联。如果没有指定安全组，端口会关联到 'default' 安全组。默认情况下，这个安全组会丢掉所有入口的流量并允许所有出口的流量。如果要修改其行为，可以添加规则到这个安全组中。

要使用计算服务的安全组 API 或使用计算服务来在指定安全组上编排实例端口的创建，您必需完成额外的配置。您需要配置每个运行 nova-compute and nova-api 的节点上的 /etc/nova/nova.conf 文件并设置 security_group_api=neutron 选项。当您执行了这些修改，请重启 nova-api 和 nova-compute 服务以加载这些修改。然后，您就可以同时使用计算服务和 OpenStack 的网络安全组 API 了。



注意

- 要使用包含 Netwokring 的计算服务安全组 API，Networking 插件需要实现安全组的 API。目前下列插件实现了这些：ML2、Open vSwitch、Linux Bridge、NEC 和 VMware NSX。
- 您需要在插件/代理的配置文件的 securitygroup 小节中设置正确的防火墙驱动。一些插件和代理，如 Linux Bridge Agent 和 Open vSwitch Agent，默认使用非操作的驱动，会导致安全组不能正常工作。
- 如果通过计算服务使用安全组 API，安全组会应用到实例的所有端口上。导致这样的原因是计算服务安全组 API 是基于实例的，而非基于 Networking 端口的。

基本安全组操作

此表展示了neutron命令完成基本安全组操作的实例：

表 7.17. 基本安全组操作

操作	命令
为我们web服务器创建一个安全组。	<code>\$ neutron security-group-create webservers --description "security group for webservers"</code>
列出安全组。	<code>\$ neutron security-group-list</code>
创建一个允许80端口近来的安全组规则。	<code>\$ neutron security-group-rule-create --direction ingress --protocol tcp --port_range_min 80 --port_range_max 80 SECURITY_GROUP_UUID</code>
列出安全组规则。	<code>\$ neutron security-group-rule-list</code>
删除一个安全组规则。	<code>\$ neutron security-group-rule-delete SECURITY_GROUP_RULE_UUID</code>
删除一个安全组。	<code>\$ neutron security-group-delete SECURITY_GROUP_UUID</code>
创建一个端口关联两个安全组。	<code>\$ neutron port-create --security-group SECURITY_GROUP_ID1 --security-group SECURITY_GROUP_ID2 NETWORK_ID</code>
从一个端口删除安全组。	<code>\$ neutron port-update --no-security-groups PORT_ID</code>

负载均衡即服务的基本操作



注意

负载均衡即服务(LBaaS) API 规定和配置负载均衡器。这里参考的实现是基于 HAProxy 软件负载均衡器。

此列表展示了 neutron 命令的实例，此例可让你完成基本的 LBaaS 操作：

- 通过使用指定的供应商来创建一个负载均衡池。

--provider 是一个可选的参数。如果没有使用，会以默认供应商的 LBaaS 服务创建池。您应该在 neutron.conf 文件的 [service_providers] 小节中配置默认的供应商。如果没有为 LBaaS 指定默认的供应商，在创建池时，--provider 是必需指定的。

```
$ neutron lb-pool-create --lb-method ROUND_ROBIN --name mypool --protocol HTTP --subnet-id SUBNET_UUID --provider PROVIDER_NAME
```

- 在池中分配两个 web 服务。

```
$ neutron lb-member-create --address WEBSERVER1_IP --protocol-port 80 mypool
$ neutron lb-member-create --address WEBSERVER2_IP --protocol-port 80 mypool
```

- 创建一个健康监控，它用来检查以确保我们的实例仍然是运行在指定的协议—端口之上的。

```
$ neutron lb-healthmonitor-create --delay 3 --type HTTP --max-retries 3 --timeout 3
```

- 给池分配一个健康检测。

```
$ neutron lb-healthmonitor-associate HEALTHMONITOR_UUID mypool
```

- 创建一个虚拟 IP (VIP) 地址，当通过负载均衡访问时，会直接请求到池成员之一。

```
$ neutron lb-vip-create --name myvip --protocol-port 80 --protocol HTTP --subnet-id SUBNET_UUID mypool
```

特定插件扩展

每个供应商可以选择实现核心 API 的额外 API 扩展。本小节将描述插件的扩展。

VMware NSX 扩展

这些章节解释了NSX插件扩展。

VMware NSX QoS 扩展

VMware NSX QoS 扩展速率限制网络端口来保证每个端口具有特定数量的带宽。这个扩展默认情况下只对具有管理员角色的租户是可访问的，但可以通过 `policy.json` 文件进行配置。要使用这个扩展，需要创建一个队列并指定最大/最小带宽速率 (kbps)，还可以选择性地设置 QoS Marking 和 DSCP 值 (如果您的网络光纤使用了这些值来进行转发决定的话)。一旦创建了，您可以关联一个队列到网络中。然后，当端口在该网络创建爱你后，它们会自动地被创建，并关联到相关网络的特定队列大小中。由于每个网络上的端口的一个队列大小可能不是优化的，那么当创建爱你端口来扩展队列时，nova 类型的 `'rxtx_factor'` 缩放选项会从计算服务传递过来。

最后，如果您想为带宽数目设置一个特定的基准的 QoS 策略，单个端口可以使用 (除非网络队列指定了所创建的网络端口) 一个在 Networking 中创建的队列，它之后会引起端口创建关联到一个该大小乘以 `rxtx` 扩展元素的队列。请注意在网络或默认队列指定之后，队列会被添加到随后创建但没有添加到已有端口的端口中。

VMware NSX Qos的基本操作

此表展示了一个neutron的例子，其可以让你完成查询操作：

表 7.18. VMware NSX Qos的基本操作

操作	命令
创建QoS队列(仅管理员)。	<code>\$ neutron queue-create --min 10 --max 1000 myqueue</code>
将队列和网络关联。	<code>\$ neutron net-create network --queue_id QUEUE_ID</code>
创建一个默认的系统队列。	<code>\$ neutron queue-create --default True --min 10 --max 2000 default</code>
列出QoS队列。	<code>\$ neutron queue-list</code>
删除一个QoS队列。	<code>\$ neutron queue-delete QUEUE_ID_OR_NAME</code>

VMware NSX 提供商网络扩展

在NSX平台下可以有不同的提供网络的实现方法。

FLAT 和 VLAN 网络类型使用桥接传输连接器。这些网络类型允许大规模数目的端口附加。为了处理增加的扩展，NSX 插件可以回调一个 NSX 逻辑开关链的单个 OpenStack 网络。您可以指定这个链上的每个逻辑开关的最大端口数目，可以在 `max_lp_per_bridged_ls` 参数中指定，它的默认值是 5,000。

该参数的建议值与 NSX 运行在后端的版本变化，如下表所示。

表 7.19. max_lp_per_bridged_ls 的建议值

NSX 版本	建议值
2.x	64

NSX 版本	建议值
3.0.x	5,000
3.1.x	5,000
3.2.x	10,000

除了这些网络类型，NSX 插件也支持一个特定的 l3_ext 网络类型，它映射外部网络到指定的 NSX 网关服务中，正如下一个小节所讨论的那样。

VMware NSX 3层扩展

NSX 通过一般配置在 OpenStack 上不同频道信号传输的网关服务暴露其 L3 容量。要使用 NSX 的 L3 容量，首先需要在 NSX 管理者中创建 L3 网关服务。接着，在 /etc/neutron/plugins/vmware/nsx.ini 中将 default_l3_gw_service_uuid 设置为该值。默认情况下，路由是映射到这个网关服务上的。

VMware NSX 3层扩展操作

创建外部网络，且将之映射到指定的NSX网关服务：

```
$ neutron net-create public --router:external True --provider:network_type l3_ext
--provider:physical_network L3_GATEWAY_SERVICE_UUID
```

从一个NSX网关服务终止一个指定的VLAN流量：

```
$ neutron net-create public --router:external True --provider:network_type l3_ext
--provider:physical_network L3_GATEWAY_SERVICE_UUID --provider:segmentation_id VLAN_ID
```

在VMwareNSX插件中操作状态同步

以 Havana 发行版启动，VMware NSX 插件提供了一个异步机制来从 NSX 后端中获取 neutron 资源操作的状态；它可以应用于 network、port 和 router 资源。

后端周期性投票并检索每个每个资源的状态；然后，Networking 数据库中的资源状态会在状态改变时更新。由于现在可操作的状态是异步检索的，GET 操作的性能会有一定的提高。

数据从后端检索分块以避免昂贵的 API 请求；这是利用 NSX API 响应分页功能实现的。最小的块大小可以通过配置选项指定；实际的块大小会根据这些来动态地决定：检索的总资源数目、两个同步任务执行之间的时间间隔、对 NSX 后端的两个连续请求之间的最小延时。

可操作的状态同步可以通过该表格中的配置选项进行调整或禁用；值得一提的是，在大多数情况下，默认值已经可以很好地工作。

表 7.20. NSX 插件中调整可操作的状态同步的配置选项

选项名称	组	默认值	类型和约束	注记
state_sync_interval	nsx_sync	10秒	整型；没有约束。	运行两个同步任务之间的时间间隔，以秒计算。如果同步任务的执行花费了超过 state_sync_interval 秒的时间，该任务的新实例会在其他任务执行完成时马上启动。可以设置该选项的值为 0 来禁用同步任务。
max_random_sync_delay	nsx_delay	0秒	整型。不可执行	当不为 0 时，在下一个块执行之前，会在 0 和 max_random_sync_delay 之间选择一个随机延时添加其中。
min_sync_req_delay	nsx_delay	1秒	整型。不可执行	该选项的值可以根据 NSX 控制器实测的负荷进行调整。较低的值会导致更快的同步，但可能会增加控制集群上的负荷。
min_chunk_size	nsx_sync	500 资源	整型；没有约束。	每个同步块从后端检索的资源的最小值。同步块的期望数值通过给定 state_sync_interval 和

选项名称	组	默认值	类型和约束	注记
				<code>min_sync_req_delay</code> 之间的比率来设置。如果资源的总数目超过 <code>min_chunk_size</code> 必需从一个块中以当前块的数目获取的资源，这个块的大小可能会增加。
<code>always_read_status</code>	<code>status</code>	<code>False</code>	布尔；没有约束。	如果启用了这个选项，可操作的状态将总是从 NSX 后端的每个 GET 请求中获取的。在这个情况下，禁用同步任务是明智之举。

如果运行多个 OpenStack Networking 服务器实例，那么状态同步任务不应该在每个节点上运行；这样会发送不必要的流量到 NSX 后端中进行不必要的数据库操作。在一个节点上专门设置 `state_sync_interval` 配置选项为一个非零值，以诊断后端状态的同步。

Networking API 请求中的 `fields=status` 参数总是触发一个明确的 NSX 后端查询，即使您启用了异步状态同步。例如，`GET /v2.0/networks/NET_ID?fields=status&fields=name`。

Big Switch 插件扩展

此节阐释了 Big Switch neutron 插件的扩展。

Big Switch 路由规则

Big Switch 允许添加路由规则到每个租户路由中。这些规则可以用于增强路由策略，例如拒绝子网之间的流量或拒绝到达外网的流量。通过在路由级别进行增强，网络分段策略可以通过不同安全组的虚拟机得到增强。

路由规则属性

每个租户路由有一套路由规则与它关联。每个路由规则在这个表中有一些属性。路由规则和它们的属性可以使用 `neutron router-update` 命令来设置，通过 `horizon` 接口或 Networking API 均可。

表 7.21. Big Switch 路由规则属性

属性名称	必须	输入类型	描述
源	是	一合法段CIDR，或者其中是'any'或'external'任何一个的关键字	一个网络，它的数据包的源 IP 必需匹配所应用的规则
目的	是	一合法段CIDR，或者其中是'any'或'external'任何一个的关键字	一个网络，它的数据包的目的 IP 必需匹配所应用的规则
动作	是	'permit' 或 'deny'	决定匹配的包是否允许通过路由
下一跳	不	下一跳 IP 地址的 plus-separated (+) 列表。例如，1.1.1.1+1.1.1.2。	覆盖了默认虚拟路由，用于处理匹配规则的包的流量

处理规则的顺序

路由规则的顺序是没有影响的。在源地址和目的地址上使用最长的前缀匹配来评估重叠的规则。源地址首先匹配，所以它总是比目的地址具有更高的优先级。换句话说，最长前缀匹配仅在有多个规则匹配同一个源地址时，使用在目的地址上。

Big Switch 路由规则操作

路由规则以 OpenStack Networking 中的一个路由更新操作来进行配置。更新会覆盖所有之前设置的规则，所以规则必需在同一时间一次性设置。

以规则更新路由，默认允许流量，但阻止从外网到达 10.10.10.0/24 子网的流量：

```
$ neutron router-update ROUTER_UUID --router_rules type=dict list=true
source=any,destination=any,action=permit
source=external,destination=10.10.10.0/24,action=deny
```

为特定的子网指定候补的下一跳地址：

```
$ neutron router-update ROUTER_UUID --router_rules type=dict list=true
source=any,destination=any,action=permit
source=10.10.10.0/24,destination=any,action=permit,nexthops=10.10.10.254+10.10.10.253
```

阻止两个子网之间的流量，允许其他流量：

```
$ neutron router-update ROUTER_UUID --router_rules type=dict list=true
source=any,destination=any,action=permit
source=10.10.10.0/24,destination=10.20.20.20/24,action=deny
```

3层计量

L3 计量 API 扩展允许管理员配置 IP 范围并给它分配一个特定的标签，以测量通过一个虚拟路由的流量。

L3 计量扩展是从技术上分离的，实现了测量功能。两个抽象已经添加：其中一个是可以包含计量规则的计量标签。因为计量标签是和租户关联的，这个租户下所有的虚拟路由都会关联这个标签。

基本的3层计量操作

只有管理员可以管理3层计量的标签和规则。

此表展示了命令neutron的实例，其可以让你完成基本的3层计量操作：

表 7.22. 基本的L3操作

操作	命令
创建一个计量标签。	<code>\$ neutron meter-label-create LABEL1 --description "DESCRIPTION_LABEL1"</code>
列出计量标签。	<code>\$ neutron meter-label-list</code>
展示一个指定标签的信息。	<code>\$ neutron meter-label-show LABEL_UUID</code> <code>\$ neutron meter-label-show LABEL1</code>
删除一个计量标签。	<code>\$ neutron meter-label-delete LABEL_UUID</code> <code>\$ neutron meter-label-delete LABEL1</code>
创建一个计量规则。	<code>\$ neutron meter-label-rule-create LABEL_UUID CIDR --direction DIRECTION --excluded</code> 例如： <code>\$ neutron meter-label-rule-create label1 10.0.0.0/24 --direction ingress</code> <code>\$ neutron meter-label-rule-create label1 20.0.0.0/24 --excluded</code>
列出所有计量的所有标签规则。	<code>\$ neutron meter-label-rule-list</code>
展示一个指定标签规则的信息。	<code>\$ neutron meter-label-rule-show RULE_UUID</code>
删除一个计量标签规则。	<code>\$ neutron meter-label-rule-delete RULE_UUID</code>
列出所创建的计量标签规则的值。	<code>\$ ceilometer sample-list -m SNMP_MEASUREMENT</code> 例如： <code>\$ ceilometer sample-list -m hardware.network.bandwidth.bytes</code> <code>\$ ceilometer sample-list -m hardware.network.incoming.bytes</code>

操作	命令
	\$ ceilometer sample-list -m hardware,network,outgoing,bytes
	\$ ceilometer sample-list -m hardware,network,outgoing,errors

高级运维特性

日志设置

网络服务组件使用 Python 日志模块来进行日志记录。日志记录配置可以在 `neutron.conf` 或以命令行选项的方式提供。命令选项会覆盖 `neutron.conf` 中的配置。

要配置网络组件的日志，使用下面方法中的一种即可：

- 在日志配置文件中提供日志的设置。
- 请参阅[Python logging how-to](#) 学习更多关于日志的内容。
- 在`neutron.conf`中设置日志

```
[DEFAULT]
# Default log level is WARNING
# Show debugging output in logs (sets DEBUG log level output)
# debug = False

# Show more verbose log output (sets INFO log level output) if debug is False
# verbose = False

# log_format = %(asctime)s %(levelname)s [%(name)s] %(message)s
# log_date_format = %Y-%m-%d %H:%M:%S

# use_syslog = False
# syslog_log_facility = LOG_USER

# if use_syslog is False, we can set log_file and log_dir.
# if use_syslog is False and we do not set log_file,
# the log will be printed to stdout.
# log_file =
# log_dir =
```

通知

当网络资源，如网络、子网或端口被创建、更新或删除时，会有消息提醒。

通知选项

要支持 DHCP 代理，需要设置 `rpc_notifier` 驱动。要配置消息提醒，请修改 `neutron.conf` 中的消息提醒选项：

```
# Driver or drivers to handle sending notifications. (multi
# valued)
#notification_driver=

# AMQP topic used for OpenStack notifications. (list value)
# Deprecated group/name - [rpc_notifier2]/topics
notification_topics = notifications
```


设置实例

日志和远程过程调用

该选项会配置网络服务器通过日志和 RPC 发送消息提醒。日志记录选项在 OpenStack Configuration Reference 中有介绍。RPC 消息提醒会转到 'notifications.info' 对列绑定到一个由 neutron.conf 中的 'control_exchange' 定义的话题交换中。

```
# ===== Notification System Options =====

# Notifications can be sent when network/subnet/port are create, updated or deleted.
# There are three methods of sending notifications: logging (via the
# log_file directive), rpc (via a message queue) and
# noop (no notifications sent, the default)

# Notification_driver can be defined multiple times
# Do nothing driver
# notification_driver = neutron.openstack.common.notifier.no_op_notifier
# Logging driver
notification_driver = neutron.openstack.common.notifier.log_notifier
# RPC driver
notification_driver = neutron.openstack.common.notifier.rpc_notifier

# default_notification_level is used to form actual topic names or to set logging level
default_notification_level = INFO

# default_publisher_id is a part of the notification payload
# host = myhost.com
# default_publisher_id = $host

# Defined in rpc_notifier for rpc way, can be comma-separated values.
# The actual topic names will be %s,%(default_notification_level)s
notification_topics = notifications

# Options defined in oslo.messaging
#

# The default exchange under which topics are scoped, May be
# overridden by an exchange name specified in the
# transport_url option. (string value)
#control_exchange=openstack
```

多RPC主题

这些选项会配置网络服务器发送消息提醒到多个 RPC 话题中。RPC 消息转到 'notifications_one.info' 和 'notifications_two.info' 对列绑定到一个由 neutron.conf 中的 'control_exchange' 定义的话题交换中。

```
# ===== Notification System Options =====

# Notifications can be sent when network/subnet/port are create, updated or deleted.
# There are three methods of sending notifications: logging (via the
# log_file directive), rpc (via a message queue) and
# noop (no notifications sent, the default)

# Notification_driver can be defined multiple times
# Do nothing driver
# notification_driver = neutron.openstack.common.notifier.no_op_notifier
# Logging driver
```

```
# notification_driver = neutron.openstack.common.notifier.log_notifier
# RPC driver
notification_driver = neutron.openstack.common.notifier.rpc_notifier

# default_notification_level is used to form actual topic names or to set logging level
default_notification_level = INFO

# default_publisher_id is a part of the notification payload
# host = myhost.com
# default_publisher_id = $host

# Defined in rpc_notifier for rpc way, can be comma-separated values.
# The actual topic names will be %s.%(default_notification_level)s
notification_topics = notifications_one,notifications_two
```

认证和授权

网络使用认证服务作为默认的认证服务。当认证服务启用，提交请求到网络服务的用户必需在 X-Auth-Token 请求头中提供一个认证令牌。用户可以通过认证服务的端点进行认证后获得该令牌。要了解更多关于认证服务的信息，请阅读 [OpenStack Identity Service API v2.0 Reference](#)。启用了认证服务后，在创建请求时不会要求强制性地指定资源的租户 ID，因为租户 ID 可以从认证令牌中得到。



注意

默认的授权配置仅允许有管理权限的用户代表一个不同的租户来创建资源。网络服务使用从认证服务检索的信息来授权用户请求。网络会服务处理两种授权策略：

- Operation-based 策略指定了特定操作的访问标准，可能包括通过特定属性进行细粒度控制；
- Resource-based 策略根据对资源设置的权限来指定了是否有权限访问特定资源（当前仅对网络资源可用）。网络服务中加强的实际授权策略在不同的部署环境中可能是各种各样的。

策略引擎从 policy.json 文件中读取入口。该文件的实际位置在不同的发行版中可能会不同。当系统运行时，入口可能会更新，且不需要服务重启。每次策略文件更新，策略会自动重新加载。目前，更新策略的唯一方法是修改策略文件。在本小节，policy 和 rule 术语指的是在策略文件中的特定方法。在规则和策略之间没有语法的不同。策略是直接匹配网络策略引擎的东西，规则是策略中的一个进行评价的元素。例如，create_subnet: ["admin_or_network_owner"]、create_subnet 是一个策略，而 admin_or_network_owner 是一个规则。

当策略匹配到 Networking API 操作或在给定的操作中使用了一个特定属性，策略由网络策略引擎触发。例如，create_subnet 策略会在 POST /v2.0/subnets 请求发送到网络服务器时被触发；另一方面，create_network:shared 会在 shared 属性在一个 POST /v2.0/networks 请求中明确指定（并设置为一个非默认值）时被触发。值得一提的是，策略可以关联到特定的 API 扩展；例如，如果由提供者网络扩展定义的属性在一个 API 请求中被指定时，extension:provider_network:set 会被触发。

权策略可以由一个或多个规则组成。如果指定了多个规则，那么当任意一个规则评价成功，则策略评价成功；如果 API 操作匹配了多个策略，那么所有的策略必需评价成功。而且，授权规则是递归的。一旦规则匹配，规则会被解析为另一个规则，直到到达最终的规则。

网络的策略引擎目前定义了以下类型的终端规则：

- 当提交请求的用户拥有指定的角色，那么 Role-based rules 会评估成功。例如，如果提交请求的用户是一位管理员，那么 "role:admin" 会成功。
- 如果当前请求的特定资源的字段匹配了一个特定值，Field-based rules 会评估成功。例如，如果 network 资源的 shared 设置为 true，那么 "field:networks:shared=True" 会成功。
- Generic rules 将资源中的属性与用户的安全证书中的额外属性进行比较，如果比较成功，则评估成功。例如，如果资源中的租户标识与提交请求的用户的租户标识相同，则 "tenant_id:%(tenant_id)s" 会成功。

此截取的内容来自默认的政策.json文件：

```
{
  "admin_or_owner": [
    [
      "role:admin"
    ],
    [
      "tenant_id:%(tenant_id)s"
    ]
  ],
  "admin_or_network_owner": [
    [
      "role:admin"
    ],
    [
      "tenant_id:%(network_tenant_id)s"
    ]
  ],
  "admin_only": [
    [
      "role:admin"
    ]
  ],
  "regular_user": [],
  "shared": [
    [
      "field:networks:shared=True"
    ]
  ],
  "default": [
    [
      "rule:admin_or_owner"
    ]
  ],
  "create_subnet": [
    [
      "rule:admin_or_network_owner"
    ]
  ],
  "get_subnet": [
    [
      "rule:admin_or_owner"
    ],
    [
      "rule:shared"
    ]
  ]
}
```

```
]
],
"update_subnet": [
  [
    "rule:admin_or_network_owner"
  ]
],
"delete_subnet": [
  [
    "rule:admin_or_network_owner"
  ]
],
"create_network": [],
"get_network": [
  [
    "rule:admin_or_owner"
  ],
  [
    ③
    "rule:shared"
  ]
],
"create_network:shared": [
  [
    "rule:admin_only"
  ]
],
  ④
"update_network": [
  [
    "rule:admin_or_owner"
  ]
],
"delete_network": [
  [
    "rule:admin_or_owner"
  ]
],
"create_port": [],
"create_port:mac_address": [
  [
    "rule:admin_or_network_owner"
  ]
],
"create_port:fixed_ips": [
  [
    ⑤
    "rule:admin_or_network_owner"
  ]
],
"get_port": [
  [
    "rule:admin_or_owner"
  ]
],
"update_port": [
  [
    "rule:admin_or_owner"
  ]
],
"delete_port": [
  [
    "rule:admin_or_owner"
  ]
]
```

```
]
]
}
```

- ❶ 如果当前用户为管理员或请求中特定资源的拥有者（租户标识相同），则规则评估成功。
- ❷ 如果一个 API 操作没有匹配到 `policy.json` 中的任何策略，默认策略会进行评价。
- ❸ 如果 `admin_or_owner` 或 `shared` 之一评价成功，则策略会评价成功。
- ❹ 这一政策限制了管理员只对网络的 `shared` 属性进行操作的功能。
- ❺ 该策略限制了仅有管理员和附加端口的网络的拥有者对端口可以进行 `mac_address` 属性操作的功能。

在一些情况下，一些操作会限制为仅有管理员可以进行。这个示例向您显示了如何修改一个策略文件来授权租户能够定义网络、查看他们的资源以及授权管理员用户执行其他所有的操作：

```
{
  "admin_or_owner": [["role:admin"], ["tenant_id:%(tenant_id)s"]],
  "admin_only": [["role:admin"]], "regular_user": [],
  "default": [["rule:admin_only"]],
  "create_subnet": [["rule:admin_only"]],
  "get_subnet": [["rule:admin_or_owner"]],
  "update_subnet": [["rule:admin_only"]],
  "delete_subnet": [["rule:admin_only"]],
  "create_network": [],
  "get_network": [["rule:admin_or_owner"]],
  "create_network:shared": [["rule:admin_only"]],
  "update_network": [["rule:admin_or_owner"]],
  "delete_network": [["rule:admin_or_owner"]],
  "create_port": [["rule:admin_only"]],
  "get_port": [["rule:admin_or_owner"]],
  "update_port": [["rule:admin_only"]],
  "delete_port": [["rule:admin_only"]]
}
```

第 8 章 Telemetry

目录

介绍	228
系统架构	228
数据收集	231
数据恢复	243
警报	249
测量	254
事件	267
Telemetry 故障排查	269
Telemetry 最佳实践	270

在OpenStack中，Telemetry模块提供的是计量服务。

介绍

即使是在云工业中，供应商也必需使用一个多步骤的过程来进行付费。在云环境中对使用量进行付费需要的步骤为计量、评估和支付。由于供应商的对于一个共享的解决方案太过于具体，评估和支付方案无法设计为一个通用的模块来满足所有。向用户提供包含对云服务的测量需要满足云计算中对“measured service”的定义。

Telemetry 模块最初是设计于支持 OpenStack 云资源的付费系统的。这个项目仅仅覆盖了付费所需过程中的计量部分。这个模块收集关于系统的信息，并将其以一个简单的格式保存，以提供任意需要付费的数据。

除了系统系统的度量之外，Telemetry 模块也会获取在 OpenStack 系统中执行的各种操作所触发的事件消息。这个数据会获取为 Events 并与计量数据一起保存。

计量的列表会持续增长，这使得将通过 Telemetry 收集的数据用于各种目的成为可能，而不仅仅是付费。例如，Orchestration 模块中的自动扩展特性会通过警告这个模块的配置来触发，然后获取 Telemetry 中的消息。

本文档中的章节包含了 Telemetry 的架构和使用信息。第一个部分包含了对在一个典型的 OpenStack 部署环境中所使用的系统架构的简单的概括。第二个部分描述了数据收集机制。您也可以阅读关于报警的内容来了解报警的定义是如何传递给 Telemetry 的，以及当发生报警时可能发生什么操作。最后一个部分包含了一个故障排除指南，它会提及一些错误的解决方案和针对问题的可能解决方案。

您可以以三种不同的方式获取收集的样例：通过 REST API、通过命令行接口或通过 OpenStack 仪表盘上的计量标签进行操作。

系统架构

Telemetry模块使用了基于代理对架构。几个模块结合承担收集数据,样本存储在一个数据库,或者提供一个API服务来处理传入的请求等的责任。

Telemetry模块由下列代理和服务所构建：

ceilometer-api	聚合计量数据来消费(如计费引擎，分析工具等等)。
ceilometer-polling	通过使用以不同的命名空间注册的轮询插件(pollsters)来轮询不同类型的计量数据。
ceilometer-agent-central	轮询诸如计算服务和镜像服务等其它等OpenStack服务的公共的RESTful API，为了保持以存在资源的标签，通过使用在中心轮询命名空间中注册的轮询插件(pollsters)来实现。
ceilometer-agent-compute	通过使用在计算轮询命名空间中注册的轮询插件(pollsters)，轮询本地的hypervisor或libvirt守护进程来为本地实例收集性能数据，消息，然后发送数据到AMQP。
ceilometer-agent-ipmi	通过使用在IPMI轮询命名空间中注册的轮询插件(pollsters)，轮询拥有IPMI支持的本地节点，是为了收集IPMI传感器数据和Intel节点管理器的数据。
ceilometer-agent-notification	其它OpenStack服务所消费动AMQP消息。
ceilometer-collector	接收来自代理的AMQP通知，然后分发这些数据到对应的数据存储。
ceilometer-alarm-evaluator	决定当警报触发由于相关统计趋势超过阈值超过滑动时间窗口。
ceilometer-alarm-notifier	发起警告操作，例如调用一个带有警告状态转换描述的webhook。



注意

从Kilo发布后，服务 ceilometer-polling就可用了。

除ceilometer-agent-compute和ceilometer-agent-ipmi服务之外，所有的其它服务都运行在一个或多个控制器节点中。

Telemetry的架构高度依赖于AMQP服务，既有消费来自OpenStack服务的事件，又有其内部的通信。

支持的数据库

其它Telemetry关键的组件就是数据库了，存放事件，样例，警告定义和警告的地方。



注意

配置多个后端数据库是为了分别存储事件、样例和警告。

后端所支持的数据库列表：

- [ElasticSearch](#) (仅事件)
- [MongoDB](#)
- [MySQL](#)
- [PostgreSQL](#)
- [HBase](#)
- [DB2](#)

支持的 Hypervisor

Telemetry模块收集关于虚拟机的信息，其要求运行在计算节点上的Hypervisor保持紧密联系。

所支持的Hypervisor列表:

- 以下hypervisor是通过[Libvirt](#)来支持的：
 - [基于内核的虚拟机\(KVM\)](#)
 - [快速模拟器 \(QEMU\)](#)
 - [Linux容器\(LXC\)](#)
 - [用户模式Linux \(UML\)](#)



注意

关于libvirt所支持的Hypervisor的细节，请参阅[Libvirt API 支持矩阵](#)。

- [Hyper-V](#)
- [XEN](#)
- [VMWare vSphere](#)

支持的网络服务

Telemetry是可以从OpenStack网络以及扩展网络服务获取信息的：

- OpenStack 网络:
 - 基本网络计量
 - 防火墙即服务(FWaaS) 计量
 - 负载均衡即服务(LBaaS) 计量
 - VPN 即服务 (VPNaaS) 计量

- SDN 控制器计量:
- [OpenDaylight](#)
- [OpenContrail](#)

用户，角色和租户

此OpenStack的模块使用OpenStack身份服务来验证和授权给用户。需要的属性配置在OpenStack 配置参考的章节 [Telemetry](#) 均有谈到。

在系统中基本会用到两类角色，分别是'admin' 和 'non-admin'。在处理每个API请求之前要进行认证。所返回动数据取决于请求者自身的角色。

警告定义的创建也非常的依赖用户的角色，这些角色会启动一些动作。关于掌控警告的更多细节，可在本书中的 [“警报”一节 \[249\]](#)中找到。

数据收集

在OpenStack中Telemetry主要责任是收集关于系统的信息，用于计费系统或者使用分析工具解读。而以前的聚焦的是所收集的数据，计数的部分可用于计费，但是范围不断在扩大。

收集的数据可以样本或事件的形式存储到被支持的数据库中，这些数据库有 [“支持的数据库”一节 \[229\]](#)。

样本需要多种来源，那么Telemetry就需要配置多种方法来收集数据。

可用的数据收集机制是：

通知	处理来自其它OpenStack服务的通知，通过消费来自其它所配置动消息队列系统的消息。
轮询	使用SNMP直接从Hypervisor或主机获取信息，又或者是使用其它OpenStack服务的API。
RESTful 应用程序接口	通过Telemetry的RESTful API 发送实例。

通知

在OpenStack中，所有服务所发送的通知都是关于所执行动操作或者是系统的状态。一些通知所携带的信息是可以被计量的，比如由OpenStack计算服务所创建的虚拟机实例的CPU时间。

Telemetry模块拥有一单独的代理，其是负责消费通知的，即通知代理。此组件是负责消费来自消息总线，以及转换通知为事件，以及测量样本。

不同的 OpenStack 服务发出一些关于各种在平时操作过程中的系统中的事件类型的通知。不是所有这些通知都由 Telemetry 模块消费，因为其目的仅仅是获得可付费的事件和通知，可用于监控或分析的目的。通知代理通过事件类型进行过滤，其中类型是由每个通知消息包含的。下面的表格包含了每个 OpenStack 服务通过 Telemetry 传输到样品的的事件类型。

表 8.1. OpenStack服务消费的事件类型

OpenStack 服务	事件类型	提示
OpenStack Compute	scheduler,run_instance,scheduled scheduler,select_destinations compute,instance,*	更多计算通知列表的细节，请参考 系统使用数据的维基页面 。
裸金属服务	hardware,ipmi,*	
OpenStack镜像服务	image,update image,upload image,delete image,send	配置镜像服务的需求可在OpenStack安装指南中的 为 Telemetry 一节 配置镜像服务 一节找到。
OpenStack 网络	floatingip,create,end floatingip,update,* floatingip,exists network,create,end network,update,* network,exists port,create,end port,update,* port,exists router,create,end router,update,* router,exists subnet,create,end subnet,update,* subnet,exists l3,meter	
编排模块	orchestration,stack,create,end orchestration,stack,update,end orchestration,stack,delete,end orchestration,stack,resume,end orchestration,stack,suspend,end	
OpenStack块存储	volume,exists volume,create,* volume,delete,* volume,update,* volume,resize,* volume,attach,* volume,detach,*	块存储服务配置需求可在OpenStack安装指南章节中的 为 Telemetry 一节 添加块存储服务代理 中找到。

OpenStack 服务	事件类型	提示
	snapshot.exists	
	snapshot.create.*	
	snapshot.delete.*	
	snapshot.update.*	



注意

一些服务需要额外的配置方可在消息队列上使用正确的交换控制发送通知。这些配置需要参考上面的表格，每个OpenStack服务都可能需要不同的设置。



注意

当在文件ceilometer.conf中store_events的属性设置为True时，通知代理需要访问数据库才能正常工作。

OpenStack对象存储服务的中间件

对象存储集的统计需要在对象存储代理的后端安装额外的中间件。此额外的组件发送的通知包括面向数据流的计量，也就是storage.objects.(incoming|outgoing).bytes值。这些计量的列表都在“[OpenStack对象存储](#)”一节 [261]中有列出，所标记的notification为起始。

如何安装此中间件的说明可以在OpenStack I安装指南的章节 [配置对象存储服务的Telemetry](#)中找到。

Telemetry 中间件

Telemetry在OpenStack中为每个API端点提供了统计HTTP请求和响应计数的能力。这是通过为每个标记为 audit.http.request, audit.http.response, http.request 或 http.response的事件存储一份样本而实现定。

建议这些通知作为事件消费，而不是样品，以更好地索引合适的值，并避免 Metering 数据库中的大量负载。如果选择了该项，当服务配置为发送 http.* 通知，那么 Telemetry 可以以样品来消费这些事件。

轮询

Telemetry模块用于存储所述基础设施的复杂的图像。此目标需要额外的由每个服务所发布的事件和通知的信息。其中一些信息时不会直接发送的，例如虚拟机实例的资源使用。

因此 Telemetry 使用了另一个方法来收集这些数据，它通过选择基础设施，包括不同 OpenStack 服务的 API 和其他的断言，如 hypervisors。后者的情况需要与计算主机进行更亲密的交互。为了解决这个问题，Telemetry 使用一个基于代理的架构，以满足对数据采集的需求。

轮询的机制支持三种类型的代理，有计算代理，中心代理，以及IPMI代理。透过现象看本质，所有类型的轮询代理都是同一个ceilometer-polling代理，但是它们从不同的命名空间加载了不同的轮询插件(pollsters)来收集数据。下面的几个小节给出了这些组件的架构和配置细节的进一步信息。

运行 ceilometer-agent-compute和：

```
$ ceilometer-polling --polling-namespaces compute
```

是一样的，运行ceilometer-agent-central和

```
$ ceilometer-polling --polling-namespaces central
```

是一样的，运行ceilometer-agent-ipmi和

```
$ ceilometer-polling --polling-namespaces ipmi
```

是一样的

另外加载所有在指定命名空间中注册的轮询插件，ceilometer-polling 代理通过使用pollster-list 属性也可指定加载轮询的插件：

```
$ ceilometer-polling --polling-namespaces central  
--pollster-list image image.size storage.*
```



注意

若使用了属性pollster-list，高可用部署是不被支持的。



注意

从Kilo发布后，服务 ceilometer-polling就可用了。

中心代理

正如此代理的名称所展示的，在Telemetry架构中是一个中心组件。此代理为轮询公共REST API负责获取额外的信息的，尤其指得是哪些OpenStack资源还没有准备好发送通知，以及通过SNMP来轮询硬件资源的。

此代理可以获取下面的服务：

- OpenStack 网络
- OpenStack对象存储
- OpenStack块存储
- 通过SNMP来获取硬件资源
- 电能消耗计量是通过 [Kwapi](#)框架实现的

要安装和配置此服务，参考 OpenStack 安装指南的章节 [安装Telemetry 模块](#)。

中心代理不需要直接的数据库连接。此代理所收集的样本都会通过AMQP发送到收集器服务或者是任何外部的服务，这些服务均是后端配置有数据库的负责持久化的数据。

计算代理

此代理负责OpenStack部署内部单个的计算节点中的虚拟机实例的使用数据资源收集。此机制需要和hypervisor近距离的互动，因此单独对代理类型符合这样的相关的计量收集，其须驻扎在主机中在本地获取这些信息。

计算代理实例已经安装到每台计算节点，安装说明可在OpenStack 安装指南中的 [为 Telemetry安装计算代理](#) 章节中找到。

和中心代理一样，此组件也不需要直接连接到数据库。样本均是通过AMQP发送到收集器的。

所支持的hypervisor列表可在“[支持的 Hypervisor](#)”一节 [230]中找到。计算代理使用安装在计算主机中hypervisor的API。因此，每个虚拟化后端所支持的计量是不一样的，正如每个检查工具所提供的计量的不同集合。

所收集的计量列表可在“[OpenStack Compute](#)”一节 [255]中找到。支持一栏所提供的信息是由Telemetry模块所支持的每个hypervisor的可用的计量。



注意

Telemetry支持Libvirt，Libvirt是在hypervisor之上的服务。

中心和计算代理服务支持高可用部署

中心代理和计算代理可同时运行在高可用的部署中，这意味着这些服务的多个实例可以并行的运行，基于这些运行着的实例的负载分区。

TooZ库提供服务实例组内部的协调。它为上述几个后端提供一个API，这些后端是用于构建分布式应用的。

TooZ支持 [各种驱动](#)，包括以下的后端解决方案：

- [Zookeeper](#). TooZ项目建议的解决方案。
- [Redis](#). TooZ项目建议的解决方案。
- [Memcached](#) 建议用于测试。
- 你必须为Telemetry服务的高可用部署配置所支持的TooZ驱动。

关于为中心代理和计算代理在配置文件ceilometer.conf中设置所需要的属性的更多信息，请参阅OpenStack 配置参考中的[coordination](#)一节。



注意

若不设置 backend_url属性的话，中心代理和计算代理二者之间只有一个实例服务能正常的运行。

实例检查的可用性是由心跳消息所提供。当实例的连接丢失时，负载将会在下一个轮询周期被仍然保持连接的实例重新分配。



注意

Memcached使用timeout值，对于Telemetry来说此值的设置必须高于heartbeat的值。

为了向后兼容以及支持已有的部署环境，中心代理配置也支持使用不同的配置文件来对并行运行的该类型的服务实例进行分类。要启用这个配置，请为 OpenStack Configuration Reference 中 [central](#) 小节的 partitioning_group_prefix 选项设置值。



警告

对于有同一个 `partitioning_group_prefix` 的每个中心代理池的子组，一个不相关的计量子集需要被轮询，否则样品可能丢失或重复。计量列表轮询可以在 `/etc/ceilometer/pipeline.yaml` 配置文件中设置。要了解关于管道的更多信息，请阅读“[数据收集和处理](#)”一节 [237]。

要启用计算代理基于负载分区的并行运行多个实例，`workload_partitioning` 属性需要设置为 `True`，此属性的配置在配置文件 `ceilometer.conf` 中的 [compute](#) 一节。

IPMI 代理

此代理负责收集一个 OpenStack 部署内的单个计算节点的 IPMI 传感器数据和 Intel 节点管理器数据。此代理需要 IPMI 节点安装有 `ipmitool`，这是各种 Linux 发行版用于控制 IPMI 通用的工具。

IPMI 代理实例须被安装到每台支持 IPMI 的计算节点中，除非此台计算节点是由裸机服务所管理，且裸机服务中 `conductor.send_sensor_data` 的属性是设置为 `true` 的。即使是没有 IPMI 或 Intel 节点管理器的计算节点安装此代理也是没什么坏处的，代理会检查硬件，如果没有可用的 IPMI 或 Intel 节点管理器，会返回空数据。这里的建议让你仅为有 IPMI 能力的节点安装 IPMI 代理是出于性能上的考虑。

正如中心代理一样，此组件也不需要直接访问数据库。样例会通过 AMQP 发送给收集者。

计量收集列表可在“[裸金属服务](#)”一节 [258] 中找到。



注意

不要在一个计算节点中同时部署 IPMI 代理和裸金属服务。如果设置了 `conductor.send_sensor_data`，那么就会出现重复的 IPMI 传感器样例。

发送样例给 Telemetry

在 Telemetry 模块中大部分的数据收集都是自动化的，Telemetry 提供通过 REST API 来提交样本的能力，这样就允许用户发送定制的样本到此模块中。

此属性让发送任何类型的样本成为可能，而且还无须写任何额外的代码或者是对配置作变更。

发送给 Telemetry 的样本还不限于实际存在的计量，有一种可能性，通过填写 POST 请求的所有必填字段，以提供数据的任何新的客户定义计数器。

如果样本对应的现有的计量，那么诸如 `meter-type` 的字段以及计量名称须正确的匹配。

使用命令行客户端发送样本所需要的字段：

- 相应资源的 ID。(`--resource-id`)
- `meter.`(`--meter-name`) 名称
- `meter.`(`--meter-type`) 类型

预定义的计量类型：

- Gauge
- Delta
- 累积
- 计量单元。(--meter-unit)
- 样例值。(--sample-volume)

要使用命令行客户端发送样本给Telemetry，需要调用下面的命令：

```
$ ceilometer sample-create -r 37128ad6-daaa-4d22-9509-b7e1c6b08697
-m memory_usage --meter-type gauge --meter-unit MB --sample-volume 48
```

Property	Value
message_id	6118820c-2137-11e4-a429-08002715c7fb
name	memory_usage
project_id	e34eaa91d52a4402b4cb8bc9bbd308c1
resource_id	37128ad6-daaa-4d22-9509-b7e1c6b08697
resource_metadata	{}
source	e34eaa91d52a4402b4cb8bc9bbd308c1:openstack
timestamp	2014-08-11T09:10:46.358926
type	gauge
unit	MB
user_id	679b0499e7a34ccb9d90b64208401f8e
volume	48.0

数据收集和处理

数据收集和处理机制称之为管道。管道，在配置这个层次，描述数据的来源和相应的转化和发布的数据池之间的耦合。

来源是数据的生产者：样本或事件。事实上，它是pollster或事件掌控者发送数据点的集合，用来匹配计量和事件类型的集合。

每个源的配置囊括名称匹配，polling间隔决定，属性资源列举或发现，以及映射到一个或多个的发布池。

数据收集可以用于不同的目的，这会影响到它发布通知的频率。通常，一个计量是以付费目的发布的，需要每 30 分钟更新一次，而相同的计量可能需要每分钟进行性能调整。



警告

应该避免快速轮询的节奏，因为它会在段时间内产生一个庞大数目的数据，这对 Telemetry 和底层数据库后端都有性能上的消极影响。因此我们强烈建议您不要使用小粒度值，如 10 秒。

池，换句话说，就是数据的消费，为从相关到源发送的转换和发布数据提供逻辑支撑。

在效果上，一个接收器描述一个链的处理程序。链以零个或更多的转换器开始，以一个或多个发布结束。链中的第一个转换器从相应的资源传递数据，进行一些操作，如得到变化率、执行单位转换或聚合，在传递之前，下一步所修改的数据在“发布者”一节 [248] 中有描述。

管道配置

默认的管道多配置，是存放在多个配置文件中的，叫做pipeline.yaml 以及 event_pipeline.yaml, 接下来是ceilometer.conf 文件，事件管道和计量管道的配置文件可通过分别设置pipeline_cfg_file 和 event_pipeline_cfg_file属性完成，至于属性的列表可参考OpenStack 配置参考中[Description of configuration options for api table](#) 一节的内容。多个管道可以定义到同一个配置文件。

计量管道的定义类似如下：

```
---
sources:
  - name: 'source name'
    interval: 'how often should the samples be injected into the pipeline'
    meters:
      - 'meter filter'
    resources:
      - 'list of resource URLs'
    sinks
      - 'sink name'
sinks:
  - name: 'sink name'
    transformers: 'definition of transformers'
    publishers:
      - 'list of publishers'
```

在源的章节中间隔参数须被定义为以秒计算。它决定了样本注入到管道的轮询频率，样本的生产是在代理的直接控制之下的。

有多种方法来为管道源定义计量列表。合法计量列表可以在“[测量](#)”一节 [254]中找到。有可能定义所有的计量，又或者是包括/不包括计量，其中源应该操作：

- 要包括所有计量，使用*通配符。只选择你打算使用的计量是非常明智的，从而避免计量数据库收到洪水般的访问。
- 要定义计量的列表，全部使用下面两个：
 - 要定义包括计量的列表，使用meter_name语法。
 - 要定义不包含计量的列表，使用!meter_name语法。
- 对于计量来说，已经变化为通过复杂的名称字段来标识，使用通配符来选择全部，例如，对于 "instance:ml.tiny", 使用 "instance:*"。



注意

请注意我们在管道之间没有重复的检查，如果您添加了一个计量到多个管道中，那么它将认为是故意重复的，且可能会根据指定的接收器保存多次。

上述定义的方法可以用于下面对组合：

- 仅使用通配符。
- 使用包含计量的列表。
- 使用不包括计量的列表。

- 使用不包括计量的通配符。



注意

上述的变化至少有一个应该包含在计量小节中。包括和排除计量不能共存于同一管道中。通配符和包括的计量不能共同存在于同一管道定义小节中。

管道源的可选资源一节允许资源URL的统计列表为轮询而配置。

管道池的转换器一节提供了增加转换器定义的可能性。可用的转换器有：

表 8.2. 可用转换器的列表

转换器的名称	配置参考名称
累加器	累加器
聚合	聚合
算术	算术
变化率	rate_of_change
单元变换	unit_conversion

发布者一节包含了发布者列表，哪里是在转换后样本数据须发送的地方。

同样地，事件管道定义类似如下：

```
---
sources:
- name: 'source name'
  events:
  - 'event filter'
  sinks
  - 'sink name'
sinks:
- name: 'sink name'
  publishers:
  - 'list of publishers'
```

事件过滤使用相同的过滤逻辑作为计量管道。

转换器

转换器的定义可包含以下领域：

名 转换器的名称

参 转换器的参数。

参数小节可以包含转换器特定的字段，如在比率变化的情况下，资源和目标字段的不同子字段，这取决于转换器的实现。

转换器变化率

如果是转换器创建了cpu_util计量，定义如下所示：

```
transformers:
- name: "rate_of_change"
  parameters:
    target:
      name: "cpu_util"
      unit: "%"
      type: "gauge"
      scale: "100,0 / (10**9 * (resource_metadata.cpu_number or 1))"
```

转换器生成的变化率是来自于 cpu 计数器的样品值的 cpu_util 计量，它表示累计的 CPU 时间，以纳秒计算。上述转换器的定义（为纳秒和多 CPU）决定了一个扩展元素，这将在转换从 cpu 计量的顺序值中得到一个以 '%' 为单位的样品序列之前生效。

对于磁盘I/O率的定义，它是由转换器变化率所生成的：

```
transformers:
- name: "rate_of_change"
  parameters:
    source:
      map_from:
        name: "disk .(read|write) .(bytes|requests)"
        unit: "(B|request)"
    target:
      map_to:
        name: "disk. 1. 2.rate"
        unit: " 1/s"
      type: "gauge"
```

单元转换变形器

转换器应用一个单位转换。它使用计量的卷并将它乘以一个给定的 'scale' 表达式。也支持 map_from 和 map_to，与转换器变化率相同。

简单配置：

```
transformers:
- name: "unit_conversion"
  parameters:
    target:
      name: "disk.kilobytes"
      unit: "KB"
      scale: "1,0 / 1024,0"
```

基于map_from 和 map_to：

```
transformers:
- name: "unit_conversion"
  parameters:
    source:
      map_from:
        name: "disk .(read|write) .bytes"
    target:
      map_to:
        name: "disk. 1.kilobytes"
      scale: "1,0 / 1024,0"
      unit: "KB"
```

聚合变形器

合计了到来的直到有足够样品的转换器已经在路上或已经到达超时。

可以通过`retention_time`属性指定超时。如果我们打算在设置了一些聚合的样本后刷新聚合的话，我们可以指定大小参数。

所创建的样品的卷是进入转换器的样品卷的总数。样品可以以 `project_id`、`user_id` 和 `resource_metadata` 属性进行聚集。要以所选择的属性聚集，请在配置中指定它们，并设置属性值以使用新的样品（第一个会使用第一个样品的属性，最后一个会使用最后一个样品的属性，并丢弃不用的属性）。

通过`resource_metadata`来聚合60s的样本，且保持最后接收到的样本`resource_metadata`：

```
transformers:
- name: "aggregator"
  parameters:
    retention_time: 60
    resource_metadata: last
```

要通过 `user_id` 和 `resource_metadata` 使每 15 个样品进行聚合，保留第一个接收的样品的 `user_id` 并丢弃 `resource_metadata`：

```
transformers:
- name: "aggregator"
  parameters:
    size: 15
    user_id: first
    resource_metadata: drop
```

累加变形器

这个转换器简单地缓存了样品，直到足够的样品到达，然后将它们一次性放入管道中。

```
transformers:
- name: "accumulator"
  parameters:
    size: 15
```

多个计量算术转换器

该转换器允许我们在一个或多个计量和/或它们的元数据上进行算术计算，例如：

```
memory_util = 100 * memory.usage / memory
```

一个新的样品以转换器配置的 `target` 小节中描述的属性被创建。这个样品的卷是所提供的表达式的结果。计算在样品上执行，来自于相同的资源。



注意

计算限于相同间隔的计量。

配置实例：

```
transformers:
- name: "arithmetic"
  parameters:
    target:
      name: "memory_util"
      unit: "%"
      type: "gauge"
      expr: "100 * $(memory.usage) / $(memory)"
```

为了演示使用元数据，这里是一个简单的计量，显示每个core的平均CPU执行时间：

```
transformers:
- name: "arithmetic"
  parameters:
    target:
      name: "avg_cpu_per_core"
      unit: "ns"
      type: "cumulative"
      expr: "$ (cpu) / ($(cpu).resource_metadata.cpu_number or 1)"
```



注意

表达式求值正常处理NaN和异常，在此情况下，它不会创建一个新的样本，而是会记录一个警告。

块存储审计脚本的设置用以获取事件

如果你打算按需收集OpenStack块存储事件，你可以使用OpenStack块存储的cinder-volume-usage-audit。此脚本在你安装了OpenStack块存储时就可用了，所以你无须任何设置即可使用它，而且访问数据还不需要认证。要使用它，你必须按照下面的格式来运行命令：

```
$ cinder-volume-usage-audit
--start_time='YYYY-MM-DD HH:MM:SS' --end_time='YYYY-MM-DD HH:MM:SS' --send_actions
```

这个脚本输出创建、删除了什么卷或快照，或在一个给定时间内是否存在，以及一些关于这些卷或快照的信息。关于卷和快照存在及大小的信息保存在 Telemetry 模块中。这个数据也作为事件保存，这是建议的用法，因为它提供了更好的数据索引。

通过计划任务使用此脚本，你可以周期性的得到通知，例如，每5分钟。

```
*/5 * * * * /path/to/cinder-volume-usage-audit --send_actions
```

存储样本

Telemetry 模块有一个单独的服务，负责保留来自分析者或作为通知接收的数据。该数据可以保存在一个文件或数据库后端中，因为支持的数据库列表可以在 [“支持的数据库”一节 \[229\]](#) 中找到。该数据也可以通过 HTTP 调度发送到一个外部数据存储中。

ceilometer-collector 服务将数据作为消息从配置的 AMQP 服务消息总线中接收。它对这些数据不做任何修改，发送到配置的目标。该服务必须运行在一台可以访问已配置的调度程序的主机上。



注意

Telemetry可同时配置多个分发器。

同时可运行多个ceilometer-collector进程。每个 collector也支持启动多个线程。在配置文件ceilometer.conf [collector 一节](#)有对collector_workers配置属性作修改的说明。



注意

从Juno版本发布后，不建议在后端数据库使用了PostgreSQL而在每个collector进程使用多个线程。

数据库分发器

如果数据库调度器作为数据存储进行配置，您可以选择为样品设置 `time_to_live` 选项 (ttl)。默认情况下，样品活动的事件值设置为 -1，意为它们将在数据库中永远保留。

生存时间值是按秒来指定的。每个样本都有一时间戳，ttl的值说明当从样本读取时间戳后过去了秒数大于ttl时，样本就会从数据库中被删除。举例来说，如果生存时间的设置为600的话，所有的大于600秒的样本都会被从数据库中清理掉。

确保数据库本身支持TTL过期。假如没有也没有关系，你可以使用一个命令行脚本 `ceilometer-expirer` 达到同样的目的。你可以在计划任务中运行它，计划任务就可以帮助你的数据库保持一致性的状态。

配置后端的情况不同而支持的级别也不同：

表 8.3. 后端数据库所支持的存活时间

数据库	TTL所支持的值	提示
MongoDB	是	MongoDB本身就支持TTL，用于删除那些早于配置的ttl值的样本。
后端基于SQL	是	<code>ceilometer-expirer</code> 被用来删除数据库中的样本及其相关的数据。
HBase	不	Telemetry 的 HBase所支持的并不包括本地TTL和 <code>ceilometer-expirer</code> 。
DB2 NoSQL	不	DB2 NoSQL 既不支持本地TTL也不支持 <code>ceilometer-expirer</code> 。

HTTP 分发器

Telemetry模块支持发送样本到外部的HTTP目标。样本无需任何的变更就可被发出。要设置这些属性作为收集器的目录，在配置文件`filename>ceilometer.conf`

文件分发器

你可以通过在配置文件`ceilometer.conf`中设置`dispatcher` 属性来将样本存储到一个文件。对于配置属性列表，请参阅OpenStack 配置指南中的[dispatcher_file](#) 一节。

数据恢复

Telemetry模块提供了持久数据可以被访问的多种机制。正如在“[系统架构](#)”一节 [228] 和在“[数据收集](#)”一节 [231] 所描述的，收集的信息可以存储在后端一个或多个数据库，这些数据库是隐藏在Telemetry RESTful API后的。



注意

强烈建议不要直接访问数据库以及读取或修改其中的任何数据。API层隐藏了所有的变化，在实际的数据库模式，并提供一个标准接口，以暴露的样例，告警等等。

Telemetry v2 API

Telemetry模块提供了RESTful API，从这里可以获取到所收集到的样本和所有相关的信息，诸如计量列表，警告定义等等。

Telemetry API的URL可以从OpenStack认证服务所提供的服务目录获取得到，OpenStack认证服务在安装的时候定义的。API的访问需要一个合法的令牌以及对应的权限可访问的数据，正如在“[用户，角色和租户](#)”一节 [231]中所描述的。

关于可用的API端点进一步的信息，可在 [Telemetry API 参考](#)中找到。

队列

API还提供了一些额外的功能，如查询所收集数据的集合。对于样本和警告API的端点来说，无论是样本还是复杂的查询均可用，而对于其他端点仅支持简单的查询。

验证完查询的参数后，如果后端是数据库的话，那么就在数据库端再来完成其他的处理，这样是为了达到更好的性能表现。

简单队列

许多API端点接受查询过滤器参数，这应该是数据结构包括以下项的列表：

- 域
- 操作
- 值
- 类型

不管端点应用了何种过滤器，它总是针对 [样本类型](#)的字段。

许多API端点字段都接受短名称，而不是参考中所定义的。API将会在内部进行改造，人后返回 [API reference](#)中所列出的字段。字段如下：

- project_id: 项目
- resource_id: 资源
- user_id: 用户

当过滤器参数包括了多个上述形势的约束，很明显它们之间的逻辑关系是AND。

复杂队列

复杂查询的过滤器表达式是在Sample, Alarm 以及AlarmChange类型字段上操作。下面对照的操作都是被支持的:

- =
- !=
- <
- <=
- >
- >=

下列逻辑操作者会被使用：

- 与
- 或
- 非



注意

not 操作在 MongoDB 和基于 SQLAlchemy 的数据库引擎中有不同的行为。如果 not 操作在一个不存在的元数据字段上应用，那么结果是取决于数据库引擎的。在 MongoDB 中，当 not 操作评估每一个所给定的字段不存在的样品为真，它会返回每一个样品。另一方面，由于底层的 join 操作，基于 SQL 的数据库引擎会返回一个空结果。

复杂查询支持指定一个 orderby 表达的列表。这意味着查询的结果可以根据这个列表中提供的字段名称进行排序。如果定义了排序的多个关键字，这些会以特定的排序顺序生效。第二个表达会在组中生效，其值与第一个表达式是相同的。顺序可以是升序或降序。

使用属性 limit 来界定返回项的数量。

filter, orderby 以及 limit 区域都是可选项。



注意

相对于简单的查询，复杂的查询可通过分离的 API 端点来实现。更多信息，请参阅 [Telemetry v2 Web API 参考](#)。

统计数值

样本数据可以用于各种各样的多种目的，比如计费或调优。外部的系统通常使用的数据的形式是状态的聚合。Telemetry API 提供了几个内置函数做了一些基本的计算，这些都无须额外的编码。

Telemetry 支持下面的统计和汇总功能：

平均 每个时期的样例平均值。

基数 在指定为该聚合函数的参数的关键各个时期数不同的值。所支持的参数值有：

- project_id
- resource_id
- user_id



注意

属性 aggregate.param 是必须的。

计数 在每个时期的例子数量。

最大 在每个时期的最大例子数量值。

最小 在每个时期的最少例子数量。

stddev 在每个周期中的样例值的标准偏差。

总计 每个周期的样例值的总和。

简单查询和统计功能可以在一单个的API请求中一起使用：

Telemetry 命令行客户端和SDK

Telemetry模块提供了命令行客户端，其可收集一些诸如警告定义等可用的数据，且可获取选项。客户端使用Telemetry RESTful API是为了执行所请求的操作。

要想使用命令ceilometer，软件包python-ceilometerclient需要被安装且要争取的配置。关于安装过程的细节，请参阅OpenStack 安装指南的 [Telemetry 一节](#)。



注意

Telemetry模块抓取用户可见的资源使用数据。因此，数据库不会包括任何不存在这些资源的数据，比如OpenStack镜像服务中的虚拟机镜像。

和其它OpenStack命令行客户端一样，ceilometer客户端使用OpenStack身份来做认证。正确的凭证和--auth_url参数须通过命令行参数或环境变量来定义好。

此章提供了一些不是很完整的实例。这些命令用于实例来验证Telemetry的安装。

要获取收集计量的列表，必须使用下面的命令：

```
$ ceilometer meter-list
```

Name	Type	Unit	Resource ID	User ID	Project ID
cpu	cumulative	ns	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
cpu	cumulative	ns	c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
cpu_util	gauge	%	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
cpu_util	gauge	%	c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk_device.read.bytes	cumulative	B	bb52e52b-1e42-4751-b3ac-45c52d83ba07-hdd	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk_device.read.bytes	cumulative	B	bb52e52b-1e42-4751-b3ac-45c52d83ba07-vda	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk_device.read.bytes	cumulative	B	c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b-hdd	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk_device.read.bytes	cumulative	B	c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b-vda	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
...					

ceilometer命令运行需要admin权限，这意味着可以访问数据库中所有的数据。关于访问权限更多信息请参阅“[用户、角色和租户](#)”一节 [231]。正如在上面的例子中所看到，在系统中存在两个虚拟机实例，虚拟机实例相关的计量在结果列表的顶端。这些存在的计量并不表示是在实例运行期间的请求。结果包含了当前所收集的每个资源的计量，基于计量名称的升序排列。

所收集到每个计量的样本都存在于计量的列表中，除非是实例不再运行或从OpenStack计算的数据库中删除。如果一个实例已经不在，且在配置文件ceilometer.conf中设置了time_to_live值，那么在每个满周期会将一组样本删除掉。当最后一个样本被删除时，数据库通过运行ceilometer-expirer来进行清理，且计量不会存在于上面的列表中。关于过期处理的更多信息请参阅“[存储样本](#)”一节 [242]。

Telemetry API在计量端点之上支持简单的查询。查询功能须遵循下面的语法：

```
--query <field1>Xoperator1Xvalue1>:...<field_n>Xoperator_nXvalue_n>
```

下面命令在请求一虚拟机实例时需要被调用：

```
$ ceilometer meter-list --query resource=bb52e52b-1e42-4751-b3ac-45c52d83ba07
```

Name	Type	Unit	Resource ID	User ID	Project ID
cpu	cumulative	ns	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f

cpu_util	gauge	%	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk.ephemeral.size	gauge	GB	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk.read.bytes	cumulative	B	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk.read.bytes.rate	gauge	B/s	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk.read.requests	cumulative	request	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk.read.requests.rate	gauge	request/s	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk.root.size	gauge	GB	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk.write.bytes	cumulative	B	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk.write.bytes.rate	gauge	B/s	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk.write.requests	cumulative	request	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
disk.write.requests.rate	gauge	request/s	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
instance	gauge	instance	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
instance.m1.tiny	gauge	instance	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
memory	gauge	MB	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f
vcpus	gauge	vcpu	bb52e52b-1e42-4751-b3ac-45c52d83ba07	b6e62aad26174382bc3781c12fe413c8	cbfa8e3dfab64a27a87c8e24ecd5c60f

正如上面所描述的，可以检索整个样本集存储为计量或者是通过可用的查询类型过滤结果。请求cpu计量的所有样本且没有任何额外的过滤看起来如下这样：

```
$ ceilometer sample-list --meter cpu
```

Resource ID	Meter	Type	Volume	Unit	Timestamp
c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b	cpu	cumulative	5,4863e+11	ns	2014-08-31T11:17:03
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	5,7848e+11	ns	2014-08-31T11:17:03
c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b	cpu	cumulative	5,4811e+11	ns	2014-08-31T11:07:05
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	5,7797e+11	ns	2014-08-31T11:07:05
c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b	cpu	cumulative	5,3589e+11	ns	2014-08-31T10:27:19
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	5,6397e+11	ns	2014-08-31T10:27:19

所请求的结果集包含的样本，是按照时间戳降序排列的实例。

简单查询也是可以获取到所收集样本的一个子集的。下面命令的执行是请求一个虚拟机实例的cpu样本：

```
$ ceilometer sample-list --meter cpu --query resource=bb52e52b-1e42-4751-b3ac-45c52d83ba07
```

Resource ID	Name	Type	Volume	Unit	Timestamp
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	5,7906e+11	ns	2014-08-31T11:27:08
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	5,7848e+11	ns	2014-08-31T11:17:03
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	5,7797e+11	ns	2014-08-31T11:07:05
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	5,6397e+11	ns	2014-08-31T10:27:19
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	5,6207e+11	ns	2014-08-31T10:17:03
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	5,3831e+11	ns	2014-08-31T08:41:57

正如上面输出所看到的，结果集包含的样本只有两个实例中的一个。

命令`ceilometer query-samples` 被用于执行复杂查询。此命令接受下面参数：

- `--filter` 对于查询包含过滤表达式形式是: `{complex_op: [{simple_op: {field_name: value}}]}`。
- `--orderby` 包含排序表达式的列表形式是: `[{field_name: direction}, {field_name: direction}]`。
- `--limit` 指定返回样例的最大值。

，更多关于复杂查询的信息，请参阅[“复杂队列”一节 \[244\]](#)。

作为复杂查询功能提供了使用复杂操作的可能性，让获取给定虚拟机实例样本的子集称为可能。请求前6个样本的cpu 和 `disk.read.bytes` 计量，下面的命令须被调用：

```
$ ceilometer query-samples --filter '["and": [{"resource": "bb52e52b-1e42-4751-b3ac-45c52d83ba07"}], [{"or": [{"counter_name": "cpu"}], [{"counter_name": "disk.read.bytes"}]}] --orderby [{"timestamp": "asc"}] --limit 6
```

Resource ID	Meter	Type	Volume	Unit	Timestamp
bb52e52b-1e42-4751-b3ac-45c52d83ba07	disk.read.bytes	cumulative	385334,0	B	2014-08-30T13:00:46
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	1,2132e+11	ns	2014-08-30T13:00:47
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	1,4295e+11	ns	2014-08-30T13:10:51
bb52e52b-1e42-4751-b3ac-45c52d83ba07	disk.read.bytes	cumulative	601438,0	B	2014-08-30T13:10:51
bb52e52b-1e42-4751-b3ac-45c52d83ba07	disk.read.bytes	cumulative	601438,0	B	2014-08-30T13:20:33
bb52e52b-1e42-4751-b3ac-45c52d83ba07	cpu	cumulative	1,4795e+11	ns	2014-08-30T13:20:34

Telemetry python 绑定

命令行客户端库提供了Python的绑定，为的是在python程序中直接使用Telemetry的Python API。

设置客户端的第一步是基于正确的凭证创建客户端实例：

```
>>> import ceilometerclient.client
>>> cclient = ceilometerclient.client.get_client(VERSION, username=用户名, password=密码,
tenant_name=PROJECT_NAME, auth_url=AUTH_URL)
```

，参数VERSION可以是1 或 2，指定使用那个版本的API。

方法调用看起来像下面：

```
>>> cclient.meters.list()
[<Meter ...>, ...]

>>> cclient.samples.list()
[<Sample ...>, ...]
```

关于python-ceilometerclient软件包的更多细节，请参阅 [OpenStack Ceilometer API的Python 绑定](#) 参考。

发布者

Telemetry模块提供了用来转发所收集的数据到 ceilometer-collector或者是外部的系统的集中传输的方法。消费的这些数据有很大的不同，就像监控系统，数据丢失是可以接受的，就像计费系统，需要稳定的数据传输。Telemetry提供的方法来满足上述两类系统的需求，如下面所描述的。

发布者组件通过消息总线将数据持久的保存在存储中，或者是发送它们到外部的系统。一个链可以包含多个发布者。

要解决上述的问题，在Telemetry模块内部的每个数据点都可配置多个分发器的概念，允许同样的技术计量或事件多次到多个目的的分发，每个都可能使用不同的传输。

发布者可以在每个管道的publishers一节中指定(关于管道的更多细节，请参阅 [“数据收集和處理”一节 \[237\]](#))，管道被定义在文件中调用[pipeline.yaml](#)。

以下发布者类型是被支持的：

- | | |
|-----|--|
| 通知 | 它可以被指定打形式 <code>notifier://?option1=value1&option2=value2</code> 。它使用 <code>oslo.messaging</code> 通过AMQP来发送数据。此是发布建议的方法。 |
| rpc | 它可以指定形式为 <code>rpc://?option1=value1&option2=value2</code> ，它发出了有损AMQP计量数据，此方法是同步的且可能会遇到性能问题。 |
| udp | 它以 <code>udp://<host>:<port>/</code> 的形式指定。它通过UDP发送计量数据。 |
| 文件 | 它可以以 <code>file://path?option1=value1&option2=value2</code> 的形式来指定。此发布者记录计量数据到一个文件。 |



注意

如果文件名称和路径均没有制定的话，此发布者就不会记录任何的计量，相反会为Telemetry纪录警告信息。

kafka 可以指定形式为 `kafka://kafka_broker_ip:kafka_broker_port?topic=kafka_topic&option1=value1`。此发布者发送计量信息到Kafka broker。



注意

如果主题的参数丢失，此分发器会发出以主题名称为ceilometer的计量数据。当没有指定端口号时，此分发器会使用9092作为broker的端口。

下列属性可用于 `rpc` 和 `notifier`。策略属性可被kafka发布者使用：

<code>per_meter_topic</code>	它的值为1，它用于在额外的 <code>metering_topic.sample_name</code> 主题队列分发样本，而不是使用默认的 <code>metering_topic</code> 队列。
<code>policy</code>	它是用于在某些情况下的行为的配置，当分发器发送样本失败时，可能预先定义的值是如下：
默认值	用于等待和屏蔽，直到样例发送完成。
丢弃	用于丢弃那些发送失败到样例。
队列	用于创建一个基于内存的队列，而且会在下个发布样本期间重试在队列中发送样本(队列的长度可以被配置为 <code>max_queue_length</code> ，默认值为1024)。

以下属性用于file发布：

<code>max_bytes</code>	当此属性大于零时，它会引起溢出。当预定的大小被超过时，文件会关闭，然后新的文件会被默默的打开用于输出。如果此值为零，则不会产生溢出。
<code>backup_count</code>	如果此值不为零，扩展名将被附加到旧的日志的文件名，以'.1', '.2', 增长，直到达到指定的值。包括的最新的数 据 会写入该文件，是没有任何扩展名的。

默认的分发器在没有任何额外属性指定时就是通知。在`/etc/ceilometer/pipeline.yaml`文件中的 `publishers`一节的样例看起来如下的样子：

```
publishers:
- udp://10.0.0.2:1234
- rpc://?per_meter_topic=1
- notifier://?policy=drop&max_queue_length=512
```

警报

警告提供了运行在OpenStack中用户导向的资源监控即服务。此类型的监控确保你可以通过编排模块自动的缩小和扩展实例组，但是你也可以使用警告用于你云资源的健康的通用目的。

这些警告有下面的三态模式：

好	规则管理警报已被评估为False。
警报	规则管理警报已被评估为True。
数据不足	在评估期间没有足够的数据点来决定警告的状态，意义不大。

预警定义

警报的定义提供了一个当状态改变发生时的管理规则，以及在其上采取的行动。这些规则的性质取决于警报类型。

预警阈值规则

对于常规的基于阈值的警告，状态转换是由下面管辖的：

- 用一个比较操作符的静态阈值，例如大于或小于。
- 统计选择来汇总数据。
- 滑动时间窗口要表明的是，你打算看最近过去多长的一段时间。

组合的规则预警

Telemetry 模块也支持元警报的概念，它通过一个逻辑操作 (AND 或 OR) 聚合了当前底层基于结合警报的状态。

预警规模

一个关键的相关的概念是 **dimensioning** 概念，定义了一组进入警报评估的匹配计量。回调每一资源实例计量，所以在最简单的情况下，警报可能被定义在应用于所有对特定用户可见的资源的特定计量上。然而，更有用的是您可以明确地选择指定感兴趣的特定资源来设置警报。

在极端情况下，您可能设置了一个小尺寸的警报，这个选择只有一个单个的目标 (由资源 ID 标识)。在其他极端情况下，您可能在统计数据的聚合上设置了一个非常大的尺寸。例如，所有从特定镜像启动的实例或匹配用户元数据的实例 (后者是 Orchestration 模块对自动伸缩组的标识)。

告警评估

警告由alarm-evaluator服务周期性的进行评估，默认是每分钟一次。

警报行为

任何个体的警报状态转换 (转换为 ok、alarm 或 insufficient data) 可能有一个或多个与之相关的操作。这些操作有效地发送了一个信号给发生状态转换的消费者，并提供一些额外的上下文。这包括了新的和之前的状态，用一些理性的数据来描述对该阈值的配置，涉及到数据点的数目和最近的数据。状态转换是由 alarm-evaluator 检测的，然而 alarm-notifier 影响了实际的通知操作。

Webhooks

这些是被 Telemetry 警报所使用的 de facto 消息类型，且简单地包含了一个发送到端点的以包含状态变化描述的编码为 JSON 格式的请求主体的 HTTP POST 请求。

日志行为

这是一个轻量级的 webhook 的替代，即状态转换是由 alarm-notifier 简单记录的，且主要是用于测试目的。

负载分区

警告评估处理使用的是和中心以及计算代理一样的负载分区机制。TooZ 库提供了服务实例组内部的协调。关于此方法的更多信息，请参阅“[中心和计算代理服务支持高可用部署](#)”一节 [235]。

要使用此负载分区的解决方案，将 evaluation_service 属性设置为 default。更多信息，请参阅[OpenStack 配置参考](#)中的警告一节。

使用告警

ceilometer 命令行提供了创建和操作警告的简单实例。

告警创建

一个创建基于阈值的警告案例，基于特定实例的 CPU 利用率上限：

```
$ ceilometer alarm-threshold-create --name cpu_hi
--description 'instance running hot'
--meter-name cpu_util --threshold 70.0
--comparison-operator gt --statistic avg
--period 600 --evaluation-periods 3
--alarm-action 'log://'
--query resource_id=INSTANCE_ID
```

当一个单独实例的平均 CPU 利用率在三次连续的 10 分钟时间超过 70%，会创建一个警报。这个情况下的通知仅仅是一个日志消息，尽管它可以选择作为一个 webhook URL。



注意

对于和一个独立的项目相关联的警告来说，警告名称必须是唯一的。

The cloud administrator can limit the maximum resulting actions for three different states, and the ability for a normal user to create log:// and test:// notifiers is disabled. This prevents unintentional consumption of disk and memory resources by the Telemetry service.

在这个示例中，警报所评估的平滑的时间窗口是 30 分钟。这个窗口并不是夹在 wall-clock 时间界限之间，相反，它在每个评估周期中停留在当前的时间上，并在每个评估周期到来之前持续缓慢地前进（默认情况下，每分钟都会发生）。

在这个案例中，周期长度设置为 600s，以反映相关计量收集的开箱默认节奏。这个周期匹配说明了一个重要的一般原则，以记得进行警报：



注意

报警周期应为对应于目标计量的管道中设置的一个整数(1 或更大) 间隔。

另外，由于实际保存在计量中的数据点频率和用于比较警报阈值的状态查询之间的不匹配，警报会趋向进入或脱离 `insufficient data` 状态。如果需要一个更短的警报时间，那么相应的间隔应该在 `pipeline.yaml` 文件中进行调整。

其它显目的警告属性应在创建时设置，或者是通过稍后的更新来设置，这包括：

状态	初始化警告状态(默认为insufficient data)。
描述	警告的自由文本描述(默认是警告规则的简介)。
激活	如果为True的话，就表明了为此警告启用了评估和行为(默认即为True)。
repeat-actions	如果为True的话，当警告仍旧在目标状态中时须重复发送通知(默认是False)。
ok-action	当警告状态转换为ok时所调用的动作。
insufficient-data-action	当警告状态转换为insufficient data时调用的动作。
time-constraint	用来限制警告的评估，如一天的某些时间，或一周的某几天(表示为cron表达式加可选的时区)。

一个创建组合警告的例子，基于两个底层报警的组合状态：

```
$ ceilometer alarm-combination-create --name meta
--alarm_ids ALARM_ID1
--alarm_ids ALARM_ID2
--operator or
--alarm-action 'http://example.org/notify'
```

当这两个底层的警告其中一个转换警告状态时，此创建的警告就会发出。此情景下的通知是一个webhook的调用。使用 `and` 和 `or` 的方法可将任意数量的底层警告组合起来。

告警恢复

你可以通过下面操作显示所有的警告(为了简短省略了一些属性):

```
$ ceilometer alarm-list
+-----+-----+-----+-----+
| Alarm ID | Name | State | Alarm condition |
+-----+-----+-----+-----+
| ALARM_ID | cpu_hi | insufficient data | cpu_util > 70.0 during 3 x 600s |
+-----+-----+-----+-----+
```

在此种情况下，是以insufficient data来报告状态的，其表明：

- 关于此实例最近一段时间的评估窗口的计量还没有集合(例如一个新类型的实例)
- 或者是, 它表明拥有警告的实例对于用户/租户不可见

- 又或者是,自从警告创建后,警告评估周期还没有开始(默认情况下,警告的评估每分钟一次)。



注意

警告的可见性取决于与发出查询的用户相关联的角色和项目：

- 管理员用户可以看到全部的 警告，并不管属主
- 非管理员用户只能看到和它的项目相关等警告(在OpenStack中按照正常的租户隔离)

告警更新

一旦警告的状态稳定下来了,我们就可决定低于70%的设置,此情况下的阈值(或大多数其它警告属性)可以按如下来更新：

```
$ ceilometer alarm-update --threshold 75 ALARM_ID
```

变更将会在下一个评估周期生效,而评估周期默认是每分钟进行一次。

多数但警告属性可通过此方法来修改,但是还是有更为便利的捷径用来获取和设置警告状态：

```
$ ceilometer alarm-state-get ALARM_ID
$ ceilometer alarm-state-set --state ok -a ALARM_ID
```

随着时间的推移,警告的状态也许会经常的变更,尤其是如果阈值被选择为接近的统计信息的趋势值。你可通过审计API来跟随警告生命周期的整个历史：

```
$ ceilometer alarm-history ALARM_ID
+-----+-----+-----+
| Type      | Timestamp | Detail                                     |
+-----+-----+-----+
| creation  | time0     | name: cpu_hi                             |
|           |           | description: instance running hot       |
|           |           | type: threshold                         |
|           |           | rule: cpu_util > 70.0 during 3 x 600s   |
| state transition | time1     | state: ok                               |
| rule change  | time2     | rule: cpu_util > 75.0 during 3 x 600s   |
+-----+-----+-----+
```

警告解除

警告若不再需要的话可以禁用掉,所以它不再有活动的评估：

```
$ ceilometer alarm-update --enabled False -a ALARM_ID
```

或者是永久的删除(不可逆步骤):

```
$ ceilometer alarm-delete ALARM_ID
```



注意

默认情况,警告的历史都会退休到已删除中。

测量

Telemetry模块在OpenStack部署中收集计量。本节讲述了关于计量格式以及其来源，甚至包括可用计量的列表的一些小结。

telemetry通过轮询基础设施元素来收集计量，而且也消费由其它OpenStack服务所发送过来到通知。关于轮询机制和通知的更多信息，请参阅“[数据收集](#)”一节 [231]。由很多计量是由轮询和消费来收集的，下表所列出的都是每个计量的出处。



注意

你也许需要配置Telemetry或其它OpenStack服务，以满足你需要收集所有的样本。关于配置需求的进一步信息，请参阅OpenStack 安装指南中的 [Telemetry 章节](#)。也可参考 [手动安装Telemetry](#) 的描述。

Telemetry 使用下列计量类型：

表 8.4. Telemetry计量类型

类型	描述
累积	随时间而增加(实例的小时)
Delta	随时间到变更(带宽)
Gauge	离散的项目(浮动IP，镜像上传)和波动值(磁盘 I/O)

Telemetry提供了存储样例元数据的可能性。此元数据可以扩展，为OpenStack计算和OpenStack对象存储。

为了给OpenStack计算服务添加额外的元数据信息，你两个属性可选。第一个是在你启动一个新的实例时指定它们。额外的信息将会以形式为resource_metadata.user_metadata.*的样本被存储。新字段须通过使用前缀metering.来定义。修改过的启动命令类似如下：

```
$ nova boot --meta metering.custom_metadata=a_value my_vm
```

另外一个属性是设置 reserved_metadata_keys 到元数据键，元数据键是你想列入到实例相关样本的resource_metadata，而样本就是OpenStack计算所收集的。此属性的配置在配置文件ceilometer.conf的DEFAULT一节。

你也可以指定报头，其值将与OpenStack对象存储样本数据一起存储。额外的信息也是存储在resource_metadata之下。新字段的格式是 resource_metadata.http_header_\$name，\$name是报头的名称，且使用_替代了 -。

为了指定新的头，你需要在swift目录下的proxy-server.conf文件中的[filter:ceilometer]一节设置 metadata_headers 属性。你可以使用此额外的数据来为实例区分内部和外部用户。

测量该列表由通过Telemetry轮询或发出通知，这个模块消耗的服务组合。



注意

Telemetry模块支持存储通知为事件。此功能是后来加上的，因此计量列表仍然包含已存在的类型和其它事件相关项。正确的做法是使用配置Telemetry使用事件存储和关闭收集事件相关的计量。关于事件的更多信息请参阅Telemetry文档中的[Events](#) 一节。关于如何打开和关闭计量的更多信息请参阅“[管道配置](#)”一节 [238]。也请注意目前还不能够将已经存在的事件类型样本迁移为事件存储。

OpenStack Compute

以下所收集的是OpenStack 计算的计量：

表 8.5. OpenStack计算服务计量

名称	类型	单元	资源	起源	支持	提示
在Icehoust版本或更早版本加入的计量						
云主机	Gauge	云主机	实例ID	通知，整理 测验结果	Libvirt, Hyper-V, vSphere	存在的实例
实例:<类型>	Gauge	云主机	实例ID	通知，整理 测验结果	Libvirt, Hyper-V, vSphere	已存在的实例<类型> (OpenStack 类型).
内存	Gauge	MB	实例ID	通知	Libvirt, Hyper-V	为实例所分配的内存大小值。
memory.usage	Gauge	MB	实例ID	测验结果	vSphere	实例实际使用的内存值 占其总的分配值的比例。
CPU	累积	纳秒	实例ID	测验结果	Libvirt, Hyper-V	已用的CPU时间。
cpu 利用率	Gauge	%	实例ID	测验结果	Libvirt, Hyper-V, vSphere	平均CPU使用
虚拟CPU	Gauge	虚拟CPU	实例ID	通知	Libvirt, Hyper-V	分配给实例的虚拟CPU 数量。
disk.read.requests	累积	请求	实例ID	测验结果	Libvirt, Hyper-V	读请求的数目
disk.read.requests.rate	Gauge	每秒请求	实例ID	测验结果	Libvirt, Hyper-V, vSphere	读请求的平均速率
disk.write.requests	累积	请求	实例ID	测验结果	Libvirt, Hyper-V	写请求的数目
disk.write.requests.rate	Gauge	每秒请求	实例ID	测验结果	Libvirt, Hyper-V, vSphere	写请求的平均速率
disk.read.bytes	累积	B	实例ID	测验结果	Libvirt, Hyper-V	读容量
disk.read.bytes.rate	Gauge	B/s	实例ID	测验结果	Libvirt, Hyper-V, vSphere	读平均速率
disk.write.bytes	累积	B	实例ID	测验结果	Libvirt, Hyper-V	写容量
disk.write.bytes.rate	Gauge	B/s	实例ID	测验结果	Libvirt, Hyper-V, vSphere	写时平均速率。
disk.root.size	Gauge	GB	实例ID	通知	Libvirt, Hyper-V	根磁盘大小
disk.ephemeral.size	Gauge	GB	实例ID	通知	Libvirt, Hyper-V	临时磁盘大小
network.incoming.bytes	累积	B	接口 ID	测验结果	Libvirt, Hyper-V	流入字节数目
network.incoming.bytes.rate	Gauge	B/s	接口 ID	测验结果	Libvirt, Hyper-V, vSphere	平均流入字节速率。

名称	类型	单元	资源	起源	支持	提示
network,outgoing.bytes	累积	B	接口 ID	测验结果	Libvirt, Hyper-V	流出字节数目
network,outgoing.bytes.rate	Gauge	B/s	接口 ID	测验结果	Libvirt, Hyper-V, vSphere	平均流出字节速率。
network,incoming.packets	累积	包	接口 ID	测验结果	Libvirt, Hyper-V	流入包数量。
network,incoming.packets.rate	Gauge	packet/s	接口 ID	测验结果	Libvirt, Hyper-V, vSphere	平均流入包的速率。
network,outgoing.packets	累积	包	接口 ID	测验结果	Libvirt, Hyper-V	流出包的数量。
network,outgoing.packets.rate	Gauge	packet/s	接口 ID	测验结果	Libvirt, Hyper-V, vSphere	平均流出包的速率。
在Juno发行增加了计量或支持的hypervisor变更了						
云主机	Gauge	云主机	实例ID	通知·整理 测验结果	Libvirt, Hyper-V, vSphere, XenAPI	存在的实例
实例:<类型>	Gauge	云主机	实例ID	通知·整理 测验结果	Libvirt, Hyper-V, vSphere, XenAPI	已存在的实例<类型> (OpenStack 类型).
memory.usage	Gauge	MB	实例ID	测验结果	vSphere, XenAPI	实例实际使用的内存值 占其总的分配值的比例。
cpu 利用率	Gauge	%	实例ID	测验结果	Libvirt, Hyper-V, vSphere, XenAPI	平均CPU使用
disk.read.bytes.rate	Gauge	B/s	实例ID	测验结果	Libvirt, Hyper-V, vSphere, XenAPI	读平均速率
disk.write.bytes.rate	Gauge	B/s	实例ID	测验结果	Libvirt, Hyper-V, vSphere, XenAPI	写时平均速率。
disk.device.read.requests	累积	请求	磁盘 ID	测验结果	Libvirt, Hyper-V	读请求的数目
disk.device.read.requests.rate	Gauge	每秒请求	磁盘 ID	测验结果	Libvirt, Hyper-V, vSphere	读请求的平均速率
disk.device.write.requests	累积	请求	磁盘 ID	测验结果	Libvirt, Hyper-V	写请求的数目
disk.device.write.requests.rate	Gauge	每秒请求	磁盘 ID	测验结果	Libvirt, Hyper-V, vSphere	写请求的平均速率
disk.device.read.bytes	累积	B	磁盘 ID	测验结果	Libvirt, Hyper-V	读容量
disk.device.read.bytes.rate	Gauge	B/s	磁盘 ID	测验结果	Libvirt, Hyper-V, vSphere	读平均速率
disk.device.write.bytes	累积	B	磁盘 ID	测验结果	Libvirt, Hyper-V	写容量

名称	类型	单元	资源	起源	支持	提示
disk.device.write.bytes.rate	Gauge	B/s	磁盘 ID	测验结果	Libvirt, Hyper-V, vSphere	写时平均速率。
network.incoming.bytes.rate	Gauge	B/s	接口 ID	测验结果	Libvirt, Hyper-V, vSphere, XenAPI	平均流入字节速率。
network.outgoing.bytes.rate	Gauge	B/s	接口 ID	测验结果	Libvirt, Hyper-V, vSphere, XenAPI	平均流出字节速率。
network.incoming.packets.rate	Gauge	packet/s	接口 ID	测验结果	Libvirt, Hyper-V, vSphere, XenAPI	平均流入包的速率。
network.outgoing.packets.rate	Gauge	packet/s	接口 ID	测验结果	Libvirt, Hyper-V, vSphere, XenAPI	平均流出包的速率。
在Kilo版本中新增或变化的hypervisor计量						
memory.usage	Gauge	MB	实例ID	测验结果	Libvirt, Hyper-V, vSphere, XenAPI	实例实际使用的内存值占其总的分配值的比例。
memory.resident	Gauge	MB	实例ID	测验结果	Libvirt	在物理服务器中实例使用的内存值。
disk.latency	Gauge	ms	实例ID	测验结果	Hyper-V	平均磁盘延时。
disk.iops	Gauge	count/s	实例ID	测验结果	Hyper-V	磁盘的平均IOPS。
disk.device.latency	Gauge	ms	磁盘 ID	测验结果	Hyper-V	每个设备的平均磁盘延时。
disk.device.iops	Gauge	count/s	磁盘 ID	测验结果	Hyper-V	每个设备的平均磁盘 iops。
disk.capacity	Gauge	B	实例ID	测验结果	Libvirt	实例可以看到的磁盘数。
disk.allocation	Gauge	B	实例ID	测验结果	Libvirt	主机上实例占有的磁盘数。
disk.usage	Gauge	B	实例ID	测验结果	Libvirt	主机上镜像容器的物理 byte 大小。
disk.device.capacity	Gauge	B	磁盘 ID	测验结果	Libvirt	实例实例可以看到的磁盘设备数。
disk.device.allocation	Gauge	B	磁盘 ID	测验结果	Libvirt	主机上由实例占用的磁盘设备数。
disk.device.usage	Gauge	B	磁盘 ID	测验结果	Libvirt	主机上每个设备的镜像容器的物理 byte 大小。

Telemetry模块支持通过使用转换器来创建新的计量。关于转换器的更多细节请参阅[“转换器”一节 \[239\]](#)。其中一些计量是由libvirt和Hyper-V生成的，也有一些是由其它计量生成的。计量列表是通过使用来自上面表格中的rate_of_change 转换器所创建的，如下：

- cpu 利用率
- disk.read.requests.rate
- disk.write.requests.rate
- disk.read.bytes.rate

- disk.write.bytes.rate
- disk.device.read.requests.rate
- disk.device.write.requests.rate
- disk.device.read.bytes.rate
- disk.device.write.bytes.rate
- network.incoming.bytes.rate
- network.outgoing.bytes.rate
- network.incoming.packets.rate
- network.outgoing.packets.rate



注意

要启用libvirt对memory.usage的支持，你需要安装libvirt的版本为1.1.1+,QEMU的版本1.5+，而且你还需要在镜像中安装合适的气球驱动，此适用于尤其是Windows客户操作系统，现在多数的Linux发行版已经内置了气球驱动。Telemetry若没有气球驱动的话是无法获取到memory.usage样本的。

OpenStack计算服务能够从计算主机中收集到CPU相关的计量。当然你需要在配置文件nova.conf 中将compute_monitors 属性设置为ComputeDriverCPUMonitor。计算配置的更多信息请参阅OpenStack 配置参考中的 [Compute 一章](#)。

以下是为OpenStack计算所收集的主机相关的计量:

表 8.6. OpenStack 计算主机计量

名称	类型	单元	资源	起源	提示
在Icehoust版本或更早版本加入的计量					
compute.node.cpu.frequency	Gauge	MHz	主机 ID	通知	CPU 主频。
compute.node.cpu.kernel.time	累积	纳秒	主机 ID	通知	CPU 内核时间。
compute.node.cpu.idle.time	累积	纳秒	主机 ID	通知	CPU 空闲时间。
compute.node.cpu.user.time	累积	纳秒	主机 ID	通知	CPU 用户模式时间。
compute.node.cpu.iowait.time	累积	纳秒	主机 ID	通知	CPU I/O 等待时间。
compute.node.cpu.kernel.percent	Gauge	%	主机 ID	通知	CPU 内核百分比。
compute.node.cpu.idle.percent	Gauge	%	主机 ID	通知	CPU 空闲百分比。
compute.node.cpu.user.percent	Gauge	%	主机 ID	通知	CPU 用户模式百分比。
compute.node.cpu.iowait.percent	Gauge	%	主机 ID	通知	CPU I/O 等待百分比。
compute.node.cpu.percent	Gauge	%	主机 ID	通知	CPU 利用率。

裸金属服务

Telemetry捕获了由裸机服务发出的事件。事件的来源是IPMI传感器，IPMI传感器收集来自主机的数据。



注意

默认情况下，裸机服务中是没有传感器数据的。要启用计量和配置次模块来发送关于测量值的话，请参阅 [安装指南](#) 中的裸机服务章节。

以下是为裸金属服务所记录的计量：

表 8.7. OpenStack裸金属模块的计量

名称	类型	单元	资源	起源	提示
在Juno版本加入计量					
hardware.ipmi.fan	Gauge	RPM	风扇传感器	通知	风扇每分钟的转数 (RPM)。
hardware.ipmi.temperature	Gauge	C	温度传感器	通知	从传感器读取温度。
hardware.ipmi.current	Gauge	W	当前传感器	通知	从传感器读取当前。
hardware.ipmi.voltage	Gauge	V	电压传感器	通知	从传感器读取电压。

基于IPMI对计量

另外一种采集IPMI基本数据的方法是从裸机服务组件中将IPMI传感器分离出来使用。一些计量作为 [“裸金属服务”一节 \[258\]](#) 可以获取到，除了原来的以Pollster替代的Notification。

你需要部署在每个有IPMI的节点中部署ceilometer-agent-ipmi 以来获取其本地的传感器数据。关于IPMI代理的更多信息，请参阅 [“IPMI 代理”一节 \[236\]](#)。



警告

为了避免重复计量数据和对IPMI卡不必要的负载，不要将IPMI代理部署到由裸机服务所管理的节点，且要保持配置文件ironic.conf中将conductor.send_sensor_data 属性设置为False。

除了通用IPMI传感器数据，以下英特尔节点管理器的计量从平台上来记录：

表 8.8. 基于IPMI对计量

名称	类型	单元	资源	起源	提示
在Juno版本加入计量					
hardware.ipmi.node.power	Gauge	W	主机 ID	测验结果	当前系统的电源。
hardware.ipmi.node.temperature	Gauge	C	主机 ID	测验结果	当前系统的温度。
Kilo版本中加入的计量					
hardware.ipmi.node.inlet_temperature	Gauge	C	主机 ID	测验结果	系统入口温度。
hardware.ipmi.node.outlet_temperature	Gauge	C	主机 ID	测验结果	系统出口温度。
hardware.ipmi.node.airflow	Gauge	CFM	主机 ID	测验结果	该系统的体积气流，表现为CFM的1/10。
hardware.ipmi.node.cups	Gauge	CUPS	主机 ID	测验结果	系统CUPS(每秒的计算使用)的索引数据。
hardware.ipmi.node.cpu_util	Gauge	%	主机 ID	测验结果	系统CPU CUPS的使用。
hardware.ipmi.node.mem_util	Gauge	%	主机 ID	测验结果	系统内存CUPS使用。
hardware.ipmi.node.io_util	Gauge	%	主机 ID	测验结果	系统IO CUPS的使用。
在Kilo发行时重命名的计量					
原名			新名		

名称	类型	单元	资源	起源	提示
hardware.ipmi.node.temperature			hardware.ipmi.node.inlet_temperature		

基于SNMP的计量

Telemetry支持采集基于SNMP的常见主机计量。为了能够收集到数据，你需要在每个目标节点上运行snmpd。

以下可用的计量是关于使用SNMP的主机：

表 8.9. 基于SNMP的计量

名称	类型	单元	资源	起源	提示
Kilo版本中加入的计量					
hardware.cpu.load,1min	Gauge	处理器	主机 ID	测验结果	过去1分钟的CPU负载。
hardware.cpu.load,5min	Gauge	处理器	主机 ID	测验结果	过去5分钟的CPU负载。
hardware.cpu.load,10min	Gauge	处理器	主机 ID	测验结果	过去10分钟的CPU负载。
hardware.disk.size,total	Gauge	B	磁盘 ID	测验结果	总的磁盘大小。
hardware.disk.size,used	Gauge	B	磁盘 ID	测验结果	使用的磁盘大小。
hardware.memory.total	Gauge	B	主机 ID	测验结果	总的物理内存大小。
hardware.memory.used	Gauge	B	主机 ID	测验结果	已使用的物理内存大小。
hardware.memory.buffer	Gauge	B	主机 ID	测验结果	物理内存缓冲大小。
hardware.memory.cached	Gauge	B	主机 ID	测验结果	物理内存缓存大小。
hardware.memory.swap,total	Gauge	B	主机 ID	测验结果	总的交换空间大小。
hardware.memory.swap,avail	Gauge	B	主机 ID	测验结果	可用的交换空间大小。
hardware.network.incoming.bytes	累积	B	接口 ID	测验结果	网卡接收到的Byte数。
hardware.network.outgoing.bytes	累积	B	接口 ID	测验结果	由网卡发送的Byte数。
hardware.network.outgoing.errors	累积	包	接口 ID	测验结果	网卡发送错误。
hardware.network.ip.incoming.datagrams	累积	数据报文	主机 ID	测验结果	接收到数据报文数。
hardware.network.ip.outgoing.datagrams	累积	数据报文	主机 ID	测验结果	发送的数据报文数。
hardware.system_stats.io.incoming.blocks	累积	块	主机 ID	测验结果	块设备收到的块的数量合计。
hardware.system_stats.io.outgoing.blocks	累积	块	主机 ID	测验结果	块设备发送的块的数量合计。
hardware.system_stats.cpu.idle	Gauge	%	主机 ID	测验结果	CPU 空闲百分比。

OpenStack镜像服务

以下是为OpenStack镜像服务收集的计量：

表 8.10. OpenStack镜像服务计量

名称	类型	单元	资源	起源	提示
在Icehoust版本或更早版本加入的计量					
镜像	Gauge	镜像	镜像ID	通知·整理 测验结果	已存在的镜像。
镜像大小	Gauge	镜像	镜像ID	通知·整理 测验结果	已上传的镜像大小。
image.update	Delta	镜像	镜像ID	通知	镜像更新次数
image.upload	Delta	镜像	镜像ID	通知	镜像上载次数
image.delete	Delta	镜像	镜像ID	通知	镜像删除次数
image.download	Delta	B	镜像ID	通知	镜像已下载。
image.serve	Delta	B	镜像ID	通知	镜像已使用

OpenStack块存储

以下是为OpenStack块存储所收集的计量：

表 8.11. OpenStack块存储计量

名称	类型	单元	资源	起源	提示
在Icehoust版本或更早版本加入的计量					
卷	Gauge	卷	卷 ID	通知	已存在的卷
volume.size	Gauge	GB	卷 ID	通知	卷的大小。
在Juno版本加入计量					
快照	Gauge	快照	快照 ID	通知	已存在的快照。
snapshot.size	Gauge	GB	快照 ID	通知	快照的大小。
Kilo版本中加入的计量					
volume.create,(start end)	Delta	卷	卷 ID	通知	创建卷。
volume.delete,(start end)	Delta	卷	卷 ID	通知	缺失的卷。
volume.update,(start end)	Delta	卷	卷 ID	通知	更新卷的名称或描述。
volume.resize,(start end)	Delta	卷	卷 ID	通知	更新卷的大小。
volume.attach,(start end)	Delta	卷	卷 ID	通知	挂接卷到某个实例。
volume.detach,(start end)	Delta	卷	卷 ID	通知	从某个实例取消挂接卷。
snapshot.create,(start end)	Delta	快照	快照 ID	通知	快照的创建。
snapshot.delete,(start end)	Delta	快照	快照 ID	通知	确实的快照。

OpenStack对象存储

以下是为OpenStack对象存储所收集的计量：

表 8.12. OpenStack对象存储计量

名称	类型	单元	资源	起源	提示
在Icehoust版本或更早版本加入的计量					
storage.objects	Gauge	对象	存储ID	测验结果	对象数目
storage.objects.size	Gauge	B	存储ID	测验结果	存储对象总大小
storage.objects.containers	Gauge	容器	存储ID	测验结果	容器数目

名称	类型	单元	资源	起源	提示
storage.objects.incoming.bytes	Delta	B	存储ID	通知	流入字节数目
storage.objects.outgoing.bytes	Delta	B	存储ID	通知	流出字节数目
storage.api.request	Delta	请求	存储ID	通知	OpenStack对象存储的API请求数。
storage.containers.objects	Gauge	对象	存储 ID/容器	测验结果	容器中的对象数量
storage.containers.objects.size	Gauge	B	存储 ID/容器	测验结果	在容器中所存放的对象总大小。

Ceph 对象存储

要从Ceph中收集计量，你需要按照 [安装手册](#) 中所描述的来安装和配置Ceph对象网关 (radosgw)。你还需要启用 [usage logging](#) 来从Ceph获得相关的计量。你还需要admin用户来配置users, buckets, metadata 以及 usagecaps。

为了能够从Telemetry访问Ceph，你需要在ceilometer.conf配置文件中为radosgw指定service group，以及上面提到的admin 用户的access_key 和 secret_key。

以下是为Ceph对象存储收集的计量信息：

表 8.13. Ceph 对象存储的计量

名称	类型	单元	资源	起源	提示
Kilo版本中加入的计量					
radosgw.objects	Gauge	对象	存储ID	测验结果	对象数目
radosgw.objects.size	Gauge	B	存储ID	测验结果	存储对象总大小
radosgw.objects.containers	Gauge	容器	存储ID	测验结果	容器数目
radosgw.api.request	Gauge	请求	存储ID	测验结果	Ceph 对象网关的API请求数目 (radosgw)。
radosgw.containers.objects	Gauge	对象	存储 ID/容器	测验结果	容器中的对象数量
radosgw.containers.objects.size	Gauge	B	存储 ID/容器	测验结果	在容器中所存放的对象总大小。



注意

usage相关的信息在上载或下载之后可能不能正确的更新，因为Ceph对象网关需要时间来更新用法属性。对于实例来说，默认的配置需要大约30分钟来省城用法日志。

OpenStack 身份

以下是为OpenStack认证所收集的计量：

表 8.14. OpenStack 身份计量

名称	类型	单元	资源	起源	提示
在Juno版本加入计量					
identity.authenticate.success	Delta	用户	用户ID	通知	用户验证成功。
identity.authenticate.pending	Delta	用户	用户ID	通知	用户等待身份验证。
identity.authenticate.failure	Delta	用户	用户ID	通知	用户验证失败。
identity.user.created	Delta	用户	用户ID	通知	用户已创建。
identity.user.deleted	Delta	用户	用户ID	通知	用户已删除。

名称	类型	单元	资源	起源	提示
identity.user.updated	Delta	用户	用户ID	通知	用户已更新。
identity.group.created	Delta	组	组ID	通知	组已创建。
identity.group.deleted	Delta	组	组ID	通知	组已删除。
identity.group.updated	Delta	组	组ID	通知	组已更新。
identity.role.created	Delta	角色	角色ID	通知	角色已创建。
identity.role.deleted	Delta	角色	角色ID	通知	角色已删除。
identity.role.updated	Delta	角色	角色ID	通知	角色已更新。
identity.project.created	Delta	项目	项目ID	通知	项目已创建。
identity.project.deleted	Delta	项目	项目ID	通知	项目已删除。
identity.project.updated	Delta	项目	项目ID	通知	项目已更新。
identity.trust.created	Delta	信任	信任ID	通知	信任已创建。
identity.trust.deleted	Delta	信任	信任ID	通知	信任已删除。
Kilo版本中加入的计量					
identity.role_assignment.created	Delta	角色分配	角色ID	通知	角色已经从目标的扮演者中添加。
identity.role_assignment.deleted	Delta	角色分配	角色ID	通知	角色已经从目标的扮演者中删除。

OpenStack 网络

以下是为OpenStack网络所收集的计量：

表 8.15. OpenStack 网络计量

名称	类型	单元	资源	起源	提示
在Icehoust版本或更早版本加入的计量					
网络	Gauge	网络	网络ID	通知	存在的网络。
network.create	Delta	网络	网络ID	通知	该网络的创建请求
network.update	Delta	网络	网络ID	通知	该网络的更新请求
子网	Gauge	子网	子网ID	通知	存在的子网
subnet.create	Delta	子网	子网ID	通知	该子网的创建请求
subnet.update	Delta	子网	子网ID	通知	该子网的更新请求
端口	Gauge	端口	端口ID	通知	现有端口。
port.create	Delta	端口	端口ID	通知	该端口的创建请求
port.update	Delta	端口	端口ID	通知	该端口的更新请求
路由	Gauge	路由	路由id	通知	存在的路由
router.create	Delta	路由	路由id	通知	该路由的创建请求
router.update	Delta	路由	路由id	通知	该路由的更新请求
ip.floating	Gauge	ip	ip的 ID	通知，整理 测验结果	存在的IP。
ip.floating.create	Delta	ip	ip的 ID	通知	该IP的创建请求。
ip.floating.update	Delta	ip	ip的 ID	通知	为此IP更新请求。
带宽	Delta	B	标签 ID	通知	通过此3层计量标签的字节。

SDN 控制器

以下是为SDN所收集的计量：

表 8.16. SDN 计量

名称	类型	单元	资源	起源	提示
在Icehoust版本或更早版本加入的计量					
交换机	Gauge	交换机	交换机 ID	测验结果	已存在的交换机。
switch.port	Gauge	端口	交换机 ID	测验结果	现有端口。
switch.port.receive.packets	累积	包	交换机 ID	测验结果	在端口接收到的包。
switch.port.transmit.packets	累积	包	交换机 ID	测验结果	在端口传输的包。
switch.port.receive.bytes	累积	B	交换机 ID	测验结果	在端口接收到的Byte数。
switch.port.transmit.bytes	累积	B	交换机 ID	测验结果	在端口传输的Byte数。
switch.port.receive.drops	累积	包	交换机 ID	测验结果	在端口丢弃接收的。
switch.port.transmit.drops	累积	包	交换机 ID	测验结果	在端口丢弃的传输量。
switch.port.receive.errors	累积	包	交换机 ID	测验结果	在端口接收的错误。
switch.port.transmit.errors	累积	包	交换机 ID	测验结果	在端口的传输错误。
switch.port.receive.frame_error	累积	包	交换机 ID	测验结果	端口接收的帧偏移错误。
switch.port.receive.overrun_error	累积	包	交换机 ID	测验结果	端口上接收到的溢出错误。
switch.port.receive.crc_error	累积	包	交换机 ID	测验结果	在端口接收到的CRC错误。
switch.port.collision.count	累积	计数	交换机 ID	测验结果	端口上的碰撞。
switch.table	Gauge	表格	交换机 ID	测验结果	持续时间表。
switch.table.active.entries	Gauge	条目	交换机 ID	测验结果	表中活跃的条目。
switch.table.lookup.packets	Gauge	包	交换机 ID	测验结果	为表寻找包。
switch.table.matched.packets	Gauge	包	交换机 ID	测验结果	和表中匹配的包。
switch.flow	Gauge	流	交换机 ID	测验结果	流的持续时间。
switch.flow.duration.seconds	Gauge	秒	交换机 ID	测验结果	以秒计算的流的持续时间。
switch.flow.duration.nanoseconds	Gauge	纳秒	交换机 ID	测验结果	以纳秒算的流的持续时间。
switch.flow.packets	累积	包	交换机 ID	测验结果	接收到的包。
switch.flow.bytes	累积	B	交换机 ID	测验结果	接收到的Byte数。

这些对于基于OpenFlow的交换机来说都可用。为了启用这些计量，每个驱动都需要进行相应的配置。

负载均衡即服务(LBaaS)

以下是为LBaaS所收集的计量：

表 8.17. 负载均衡即服务计量

名称	类型	单元	资源	起源	提示
在Juno版本加入计量					
network.services.lb.pool	Gauge	池	池 ID	通知，整理 测验结果	存在的负载均衡池。
network.services.lb.vip	Gauge	虚拟IP	虚拟IP ID	通知，整理 测验结果	已存在的负载均衡虚拟IP。
network.services.lb.member	Gauge	成员	成员ID	通知，整理 测验结果	已存在的负载均衡成员。
network.services.lb.health_monitor	Gauge	health_monitor	监控ID	通知，整理 测验结果	已存在的负载均衡健康探针。
network.services.lb.total.connections	累积	连接	池 ID	测验结果	在一个负载均衡中的总连接。

名称	类型	单元	资源	起源	提示
network.services.lb.active.connections	Gauge	连接	池 ID	测验结果	在一个负载均衡中的活动中的连接。
network.services.lb.incoming.bytes	累积	B	池 ID	测验结果	流入字节数目
network.services.lb.outgoing.bytes	累积	B	池 ID	测验结果	流出字节数目
Kilo版本中加入的计量					
network.services.lb.pool.create	Delta	池	池 ID	通知	负载均衡池已经创建。
network.services.lb.pool.update	Delta	池	池 ID	通知	负载均衡池已经更新。
network.services.lb.vip.create	Delta	虚拟IP	虚拟IP ID	通知	负载均衡虚拟IP已经创建。
network.services.lb.vip.update	Delta	虚拟IP	虚拟IP ID	通知	负载均衡虚拟IP已经更新。
network.services.lb.member.create	Delta	成员	成员ID	通知	负载均衡成员已经创建。
network.services.lb.member.update	Delta	成员	成员ID	通知	成员已更新。
network.services.lb.health_monitor.create	Delta	health_monitor	监控ID	通知	已创建负载健康均衡探针。
network.services.lb.health_monitor.update	Delta	health_monitor	监控ID	通知	负载均衡健康探针已经更新。

VPN 即服务 (VPNaaS)

以下是为VPNaaS所收集的计量：

表 8.18. VPN即服务计量

名称	类型	单元	资源	起源	提示
在Juno版本加入计量					
network.services.vpn	Gauge	vpnservice	vpn ID	通知，整理 测验结果	存在的VPN
network.services.vpn.connections	Gauge	ipsec_site_connection	连接ID	通知，整理 测验结果	存在的IPSec 连接。
Kilo版本中加入的计量					
network.services.vpn.create	Delta	vpnservice	vpn ID	通知	VPN 已经创建。
network.services.vpn.update	Delta	vpnservice	vpn ID	通知	VPN已经更新。
network.services.vpn.connections.create	Delta	ipsec_site_connection	连接ID	通知	IPSec 连接已经创建。
network.services.vpn.connections.update	Delta	ipsec_site_connection	连接ID	通知	IPSec 连接已经更新。
network.services.vpn.ipsecpolicy	Gauge	ipsecpolicy	ipsecpolicy ID	通知，整理 测验结果	存在的IPSec 规则。
network.services.vpn.ipsecpolicy.create	Delta	ipsecpolicy	ipsecpolicy ID	通知	IPSec 规则已经创建。
network.services.vpn.ipsecpolicy.update	Delta	ipsecpolicy	ipsecpolicy ID	通知	IPSec 规则已经更新。
network.services.vpn.ikepolicy	Gauge	ikepolicy	ikepolicy ID	通知，整理 测验结果	已存在的Ike 规则。
network.services.vpn.ikepolicy.create	Delta	ikepolicy	ikepolicy ID	通知	Ike 规则已经创建。
network.services.vpn.ikepolicy.update	Delta	ikepolicy	ikepolicy ID	通知	Ike 规则已经更新。

防火墙即服务(FWaaS)

以下是为FWaaS所收集的计量：

表 8.19. 防火墙即服务计量

名称	类型	单元	资源	起源	提示
在Juno版本加入计量					
network.services.firewall	Gauge	防火墙	防火墙ID	通知，整理 测验结果	已存在的防火墙。
network.services.firewall.policy	Gauge	防火墙策略	防火墙ID	通知，整理 测验结果	已存在的防火墙策略。
Kilo版本中加入的计量					
network.services.firewall.create	Delta	防火墙	防火墙ID	通知	防火墙已经创建。
network.services.firewall.update	Delta	防火墙	防火墙ID	通知	防火墙已经更新。
network.services.firewall.policy.create	Delta	防火墙策略	策略ID	通知	防火墙策略已经创建。
network.services.firewall.policy.update	Delta	防火墙策略	策略ID	通知	防火墙策略已更新。
network.services.firewall.rule	Gauge	firewall_rules	规则ID	通知	已存在的防火墙规则。
network.services.firewall.rule.create	Delta	firewall_rules	规则ID	通知	防火墙规则已经创建。
network.services.firewall.rule.update	Delta	firewall_rules	规则ID	通知	防火墙规则已经更新。

编排模块

以下是为编排模块所收集的计量：

表 8.20. 编排模块的计量

名称	类型	单元	资源	起源	提示
在Icehoust版本或更早版本加入的计量					
stack.create	Delta	栈	栈 ID	通知	成功的创建了栈。
stack.update	Delta	栈	栈 ID	通知	成功的更新了栈。
stack.delete	Delta	栈	栈 ID	通知	栈已成功的删除。
stack.resume	Delta	栈	栈 ID	通知	栈已成功的恢复。
stack.suspend	Delta	栈	栈 ID	通知	栈已成功的挂起。

OpenStack的数据处理服务

以下是为OpenStack的数据处理所收集的计量：

表 8.21. OpenStack的数据处理服务计量

名称	类型	单元	资源	起源	提示
在Juno版本加入计量					
cluster.create	Delta	集群	集群ID	通知	集群已成功创建。
cluster.update	Delta	集群	集群ID	通知	集群已成功更新。

名称	类型	单元	资源	起源	提示
cluster.delete	Delta	集群	集群ID	通知	集群已成功删除。

键值存储模块

以下是为键值存储模块所收集的计量：

表 8.22. 键值存储模块计量

名称	类型	单元	资源	起源	提示
Kilo版本中加入的计量					
magnetodb.table.create	Gauge	表格	表 ID	通知	成功创建表。
magnetodb.table.delete	Gauge	表格	表 ID	通知	成功的删除了表。
magnetodb.table.index.count	Gauge	索引	表 ID	通知	在表中所创建的索引数目。

电能

以下是可用的电能相关的计量：

表 8.23. 电表

名称	类型	单元	资源	起源	提示
在Icehoust版本或更早版本加入的计量					
电能	累积	kWh	探测器 ID	测验结果	能量。
电力	Gauge	W	探测器 ID	测验结果	电力消耗。

事件

除了计量之外，Telemetry模块在OpenStack环境中收集所触发动事件。此节提供了Telemetry模块的事件格式的简要描述。

一个样品表示了一个单个的、一段时间序列内的数字数据点，而时间是一个更广泛的概念，表示了一个时间点的资源状态。该状态可通过各种数据类型进行描述，包括非数字的数据，如实例的类型。通常，事件表示了 OpenStack 系统中所发生的任意操作。

事件配置

要在Telemetry模块中启用创建和存储事件需要设置store_events 的属性为True。更多的配置属性，请参阅[OpenStack 配置参考](#)中的事件一节。



注意

如果启用了 Telemetry 模块中的事件，将 disable_non_metric_meters 设置为 True 是比较明智的做法。Telemetry 模块以前代表事件作为计量数据，如果事件和非度量的计量都启用了，它会创建重复的数据。

事件结构

由Telemetry模块捕获的事件由五个关键属性：

event_type	一个点串定义发生的事件，如compute.instance.resize.start”。
message_id	事件的UUID。
生成的	当系统中由事件发生时的时间戳。
特定	描述事件的一个关键值对的扁平化映射。事件的特征包括了大多数事件的细节。特征类型，可以是字符串、整型、浮点型或日期时间。
raw	主要是审计的目的，全部的事件消息都被存储(未索引的)，为了将来的评估。

事件索引

OpenStack 里通知的一般哲学是发出任何和所有可能需要的数据，然后让消费者过滤掉他们不感兴趣的内容。为了使处理更加简单有效，通知作为事件在 Ceilometer 中保存和处理。通知的有效载荷被转换为关键值对的扁平化集合，其中负载可以是任意复杂的 JSON 数据结构。这个转换是由配置文件指定的。



注意

该事件格式是为了高效的处理和查询。完全的将通知保存是为了审计的目的，可通过配置store_raw属性来启用将通知全部的存起来。

事件转变

从通知到事件的转变是由一个配置文件所驱动的，此配置文件在ceilometer.conf配置文件中由definitions_cfg_file 所定义。

这包括了如何在通知内容中映射字段到特征的描述，而且有可选的插件来做任何的程序翻译(字符串切片，强制用例)。

每个event_type都定义了通知到事件的映射，它可以是通配符。如果对应的字段在已存在的且不为空的通知中，特征会添加到事件中。



注意

包含在 Telemetry 模块中的默认定义文件包括了一个已知通知和有用的特征的列表。所提供的映射可以被修改，以根据用户需求来包含更多或更少的数据。

如果没有预先设置定义文件，将会记录一个警告信息，但它会认为设置了一个空的定义文件。默认情况下，任意没有在定义文件中有相应事件定义的通知会被转换为一个最小特征的事件集合。这可以通过设置 ceilometer.conf 文件中的 drop_unmatched_notifications 选项来修改。如果该项被设置为 True，所有没有映射的通知将被丢弃。

如果通知有相关的数据，特征的基本配置(均为 TEXT 类型)会被添加到所有的事件中，这些相关数据为：service(消息的发布者)、tenant_id 和 request_id。它们不需要在事件定义中指定，是自动添加的，但它们的定义会被给定的 event_type 覆盖。

事件定义格式

事件定义文件是 YAML 格式的。它由一个映射事件定义的列表组成。顺序是很重要的，定义列表会被倒序扫描以寻找一个匹配通知的 event_type 的定义。该定义将用于生成事件。执

行倒序的是因为它通常希望有一个更通用的通配符定义（如 `compute.instance.*`），和所有这些事件的一系列共同特征、一些更具体的包含上述特征以及更多的事件定义。

每个事件的定义都映射到2个键：

event_type 这是一个该定义会处理的 `event_types` 的列表（或字符串，可以认为是只有1个元素的列表）。它们与 `unix` 的 `shell glob` 语法是通用的。一个排除的列表（以 `!` 开头）会排除所有列表中匹配的类型。如果只使用了排除列表，定义会匹配所有没有与排除列表中匹配的内容。

特定 此是一个映射，键是特征的名称，值是特征的定义。

每个特点的定义会映射到下面的值：

域 在通知中的您所希望提取的这个特性的路径规范字段（`fields`）。可以编写规范以匹配多个可能的字段。默认值是第一个该字段。路径可以以点的语法指定（`payload.host`）。方括号语法（`payload[host]`）也是支持的。在其他情况下，如果您所寻找的字段的关键字包含了指定的字符，如 `.`，它将需要被引用（以双引号或单引号引用）：`payload.image_meta.'org.openstack_1__architecture'`。用于字段规范的语法是 [JSONPath](#) 的变异语法。

类型 （可选）此特征的数据类型。合法的属性是：`text`, `int`, `float`, 和 `datetime`。若没有指定默认是 `text`。

插件 （可选）用于在通知字段中的值进行简单的编程转换。

Telemetry 故障排查

登录到Telemetry

Telemetry模块拥有和其它OpenStack服务相似的日志设置。目录的记录，日志的格式和日志级别均有多个可供修改的属性。

在`ceilometer.conf`可修改日志的设置。配置属性的列表在OpenStack 配置参考中的 [Telemetry 一节](#) 中的日志配置属性表格中有列出。

默认日志信息的标准输出用的是`stderr`。它可以变更到日志文件或系统日志中。默认的设置`debug` 和 `verbose` 属性都为`false`，默认日志级别相应的模块可在上面参考表格中找到。

建议启动服务的顺序

若你看到过 [Bug 1355809](#)，就会明白错误的服务启动顺序甚至会导致数据丢失。

当服务第一次启动时，或者是消息队列服务重启，`ceilometer-collector`服务创建连接以及加入或重新加入到所配置的交换机会花费一段时间。因此，如果 `ceilometer-agent-compute`, `ceilometer-agent-central` 和 `ceilometer-agent-notification`服务都先于 `ceilometer-collector`启动的话，在连接到消息队列服务时，这些服务发送的信息就会丢失。

当轮询的间隔设置的是相对短的周期时，此问题出现的可能性还是很高的。为了避免这样的情形发生，建议的服务顺序是在消息队列启动后再启动或重启`ceilometer-collector` 服务。

然后再启动或重启所有其它的Telemetry服务，而且 `ceilometer-agent-compute` 须排到最后启动，因为此组件发送计量信息是为了发送样本给收集者。

通知代理

在Icehouse版本中，OpenStack引入了一个新的服务，那就是负责消费来自其它OpenStack服务的通知。

如果`ceilometer-agent-notification` 服务没有安装和启动的话，来自事件的起始样本就不会生成。发生了缺少基于样本的事件，第一要检查的是此服务的状态和Telemetry的日志文件。

计量列表的起源是来自通知，请参阅 [Telemetry 测量参考](#)。

建议使用 `auth_url`

当使用Telemetry的命令行客户端时，设置了凭证和`os_auth_url` 是为了提供客户端再次向OpenStack身份服务认证的可能性。关于意境提供的凭证更多细节，请参阅 [Python API的Telemetry参考](#)。

由OpenStack身份服务所提供的服务目录包括了可用的URL，这些URL是用于认证的。URL包含了不同的端口，基于该给定的URL类型有`public`, `internal` 或 `admin`。

OpenStack身份变更API的版本从v2到v3。`adminURL`端点仅支持v3版本(通过端口35357来提供)，而另外两个版本均同时支持。

Telemetry的命令行客户端没有更改OpenStack身份 API的v3版本。如果`adminURL`是使用了`os_auth_url`，`ceilometer`命令的话，结果就会出现下面的错误信息：

```
$ ceilometer meter-list
Unable to determine the Keystone version to authenticate with using the given
auth_url: http://10.0.2.15:35357/v2.0
```

因此当在命令行或使用环境变量中指定`os_auth_url`参数时，使用 `internalURL` 或者 `publicURL`。

更多细节请参考 [Bug 1351841](#)的报告。

Telemetry 最佳实践

当部署和配置Telemetry服务时，以下是几个推荐的最佳实践。最佳实践分为数据收集和数据存储。

数据收集

1. Telemetry模块会持续的收集日益增长的数据集。不是所有的数据都会有相应的云管理员来监控的。
 - 根据你的需要，你可以编辑配置文件`pipeline.yaml`，来选定计量的数量而无须顾及其它。

- 默认情况下，Telemetry服务每10分钟轮询服务的API。你可以通过编辑配置文件pipeline.yaml按照每个计量来更改轮询间隔。



警告

如果轮询的间隔太短的话，它会引起存储数据的增加以及服务API的压力。

- 展开配置，以更好地控制不同计量的间隔。



注意

更多信息，请查阅 [管道配置选项](#)。

2. 如果你使用的是Kilo版本的Telemetry，你可以启用抖动支持来延迟或调整轮询请求。抖动是增加了轮询代理发送请求给服务的API的随机延迟。要启用抖动，在配置文件ceilometer.conf中设置shuffle_time_before_polling_task为大于0的整数。
3. 如果你使用的是Juno或起后续版本，根据所要轮询的资源数量，来决定增加额外的中心和计算代理。代理的设计是支持横向扩展的。



注意

更多信息，请参阅 [中心和计算代理的高可用部署](#)。

4. 如果你使用的是Juno或是其后续版本，使用notifier://发布器而不是 rpc://，因为RPC会增加一定水平的开销。



注意

RPC的更多信息，请参阅 [RPC 之上的信息](#)。

数据存储

1. 我们建议你避免开放式的查询。为了获得更好的性能，你可以使用合理的时间范围或者是其它获取测量的查询条件。

举例来说，此开放式的查询会返回不可预料的数据量：

```
$ceilometer sample-list --meter cpu -q 'resource_id=INSTANCE_ID_1'
```

而此良好的查询会返回更多合理的数据量，而且更加的快：

```
$ceilometer sample-list --meter cpu -q 'resource_id=
INSTANCE_ID_1;timestamp>2015-05-01T00:00:00;timestamp<2015-06-01T00:00:00'
```

2. 你可以在API之后安装 mod_wsgi，因为它提供了更多的设置来调整，正如WSGIDaemon下的 threads 和 processes。



注意

关于如何配置mod_wsgi的更多信息，请参阅 [安装文档](#)。

3. 由Telemetry项目所提供的收集服务并不想成为一个归档服务。设置生存时间(TTL)值给过期数据和数据库的最小规模。如果你想长时间的保存你的数据，应考虑将它们存储在Telemetry之外的数据仓库中。



注意

更多关于如何设置TTL的信息，请参考 [TTL 支持的各种后端](#)。

4. 我们建议在Juno之前的版本后端不要使用SQLAlchemy，因为它包含无关的关系来处理废弃的数据模型。而这会导致极度低下的查询性能。
5. 我们建议你不要让控制器和MongoDB运行在同一台服务器中。将它们保持分离对于快速存储的优化会带来更佳的性能。我们也建议为MongoDB节点配置大量的内存。



注意

更多关于你需要多大内存的信息，请参考 [MongoDB FAQ](#)。

6. 在MongoDB中使用副本设置。副本设置通过自动的失效切换来实现高可用。如果主节点宕机了，MongoDB会选举出第二个节点来替代主节点，集群整体的功能不受影响。

更多关于replica设置的信息，请参考 [MongoDB replica 设置文档](#)。

7. 使用MongoDB的分区。分区可帮助跨多台机器的存储数据记录，且MongoDB可满足数据日渐增长的需求。

关于分片的更多信息，请参考 [MongoDB 分片文档](#)。

第 9 章 数据库

目录

介绍	273
创建一个 datastore	273
配置一集群	276

数据库服务模块提供的是数据库管理。

介绍

数据库服务提供可扩展性和可靠的云部署关系型和非关系性数据库引擎的功能。用户可以快速和轻松使用数据库的特性而无须掌控复杂的管理任务，云用户和数据库管理员可以按需部署和管理多个数据库实例。

数据库服务在高性能层次上提供了资源的隔离，以及自动化了复杂的管理任务，诸如部署、配置、打补丁、备份、恢复以及监控。

创建一个 datastore

一个管理员用户可以为各式各样的数据库创建数据存储。

此节假设你还没有一个MySQL数据存储，然后向你展示了如何创建一个MySQL数据存储，且基于MySQL 5.5版本。

要创建一个datastore

1. 创建一个trove 镜像

为您要使用的类型的数据库创建一个镜像，例如，MySQL、MongoDB、Cassandra等。

这个镜像必需安装了 trove guest agent，并且需要有 trove-guestagent.conf 配置文件的配置，连接到您的 OpenStack 环境中。要正确地配置 trove-guestagent.conf 文件，请在您要构建镜像的实例虚拟机上在文件trove-guestagent.conf 中添加下面几行内容：

```
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
nova_proxy_admin_user = admin
nova_proxy_admin_pass = ADMIN_PASS
nova_proxy_admin_tenant_name = service
trove_auth_url = http://控制器:35357/v2.0
```

此例假设你已经创建了一个名为 mysql-5.5.qcow2的Mysql 5.5的镜像。

2. 向镜像服务注册镜像

用户需要将客户镜像注册到镜像服务。

在此例中，你使用glance image-create命令来注册镜像 mysql-5.5.qcow2。

```
$ glance image-create --name mysql-5.5 --disk-format qcow2 --container-format bare --is-public True <
mysql-5.5.qcow2
+-----+
| Property | Value |
+-----+
| checksum | d41d8cd98f00b204e9800998ecf8427e |
| container_format | bare |
| created_at | 2014-05-23T21:01:18 |
| deleted | False |
| deleted_at | None |
| disk_format | qcow2 |
| id | bb75f870-0c33-4907-8467-1367f8cb15b6 |
| is_public | True |
| min_disk | 0 |
| min_ram | 0 |
| name | mysql-5.5 |
| owner | 1448da1223124bb291f5ae8e9af4270d |
| protected | False |
| size | 0 |
| status | active |
| updated_at | 2014-05-23T21:01:22 |
| virtual_size | None |
+-----+
```

3. 创建 datastore

创建数据存储会使用新的镜像，要完成这步，使用trove-manage命令datastore_update。

此例中使用了下列参数：

参数	描述	在此例中：
配置文件	所使用的配置文件。	--config-file=/etc/trove/trove.conf
名称	为你打算使用的数据存储起一个名称。	mysql
默认版本	你可以为一个数据存储挂接过个版本/镜像。举例来说，你拥有MySQL 5.5版本和5.6版本。你可以指定其中一个版本为默认，当用户对版本没有特别的需求的时候系统来使用。	"" 在此处，你无须指定一个默认版本，将之留为空白即可。

例子：

```
$ trove-manage --config-file=/etc/trove/trove.conf datastore_update mysql ""
```

4. 为新的数据存储添加一个版本

现在你拥有了一个mysql数据存储，你可以使用trove-manage命令datastore_version_update 为其添加一个版本，版本表明了使用那个guest镜像。

此例中使用了下列参数：

参数	描述	在此例中：
配置文件	所使用的配置文件。	--config-file=/etc/trove/trove.conf

参数	描述	在此例中：
数据存储	你刚刚通过trove-manage datastore_update所创建的数据存储的名称。	mysql
版本名	你添加的数据存储的版本名。	mysql-5.5
数据存储管理器	用于此版本的数据存储管理器。典型的情况是，数据存储管理器依赖于具体的数据库，有下面的字符串的某一个来鉴别： <ul style="list-style-type: none"> • mysql • redis • mongodb • cassandra • couchbase • percona 	mysql
glance ID	ID是你刚刚为认证服务添加的guest镜像ID。你可以使用glance命令 image-show IMAGE_NAME 来获取此ID。	bb75f870-0c33-4907-8467-1367f8cb15b6
软件包	如果你打算为你创建的此版本的数据存储的每个guest添加额外的软件包，你可以将软件包的名称列到此处：	"" 在此例中，guest镜像已经包含了所有需要的软件包，所以将这些参数留为空白。
激活	设置此为1或0： <ul style="list-style-type: none"> • 1 = 激活 • 0 = 禁用 	1

例子：

```
$ trove-manage --config-file=/etc/trove/trove.conf datastore_version_update
mysql mysql-5.5 mysql GLANCE_ID "" 1
```

可选。设置你的新版本的数据存储作为默认版本。要完成此步骤，再次使用命令 trove-manage datastore_update ,这次你为刚刚创建的数据存储指定了版本。

```
$ trove-manage --config-file=/etc/trove/trove.conf datastore_update mysql "mysql-5.5"
```

5. 为配置组加载验证规则



仅应用MySQL和Percona数据存储

- 如果你刚刚创建了一个MySQL或Percona数据存储，那么你需要加载对应的验证规则，正如在此步骤中所描述的。
- 如果你刚创建了一个非MySQL或Percona数据存储，请忽略这步。

背景。你可以使用配置组来管理数据库的配置任务。配置组可以让你设置配置参数，基于块，一个或多个数据库。

当你设置一个配置组时会用到trove configuration-create命令，此命令会比较你设置的配置值和存储在validation-rules.json文件中的配置值，从而达到验证的效果。

操作系统	地址是validation-rules.json	注记
Ubuntu 14.04	/usr/lib/python2.7/ dist-packages/trove/ templates/DATASTORE_NAME	DASTORE_NAME既可以是MySQL数据库的名称，也可以是Percona数据库的名称。mysql 和 percona通常也是如此。
RHEL 7, CentOS 7, Fedora 20, 以及 Fedora 21	/usr/lib/python2.7/ site-packages/trove/ templates/DATASTORE_NAME	DASTORE_NAME既可以是MySQL数据库的名称，也可以是Percona数据库的名称。mysql 和 percona通常也是如此。

因此，作为创建一个数据存储的一部分，你需要加载validation-rules.json文件，使用trove-manage命令db_load_datastore_config_parameters，此命令后跟下面这些参数：

- 数据存储名称
- 数据存储版本
- 文件validation-rules.json的完整路径

此例是在Ubuntu 14.04下为一个MySQL数据存储加载了文件 validation-rules.json:

```
$ trove-manage db_load_datastore_config_parameters mysql "mysql-5.5"
/usr/lib/python2.7/dist-packages/trove/templates/mysql/validation-rules.json
```

6. 验证数据存储

要验证你的新的数据存储和版本，从在你的系统中列出数据存储开始:

```
$ trove datastore-list
+-----+-----+
|      id      |  name  |
+-----+-----+
| 10000000-0000-0000-0000-000000000001 | Legacy MySQL |
| e5dc1da3-f080-4589-a4c2-ef7928f969a |  mysql  |
+-----+-----+
```

拿到 mysql 数据存储的ID，然后将之传送到命令 datastore-version-list 中：

```
$ trove datastore-version-list DATASTORE_ID
+-----+-----+
|      id      |  name  |
+-----+-----+
| 36a6306b-efd8-4d83-9b75-8b30dd756381 | mysql-5.5 |
+-----+-----+
```

配置一集群

一个管理员用户可以为 MongoDB 集群配置各种参数。

查询路由和配置服务器

背景。每个集群包括至少一个查询路由其和一个配置服务器。查询路由其和配置服务器要计入配额。当您删除一个集群，系统会删除相关的查寻路由其和配置服务器。

配置. 默认情况下，系统会为每个集群创建一个查寻路由其和一个配置服务器。您可以通过编辑 `/etc/trove/trove.conf` 文件来修改这个默认值。则写设置都在文件的 `[mongodb]` 小节中：

设置	有效值是:
<code>num_config_servers_per_cluster</code>	1或者3
<code>num_query_routers_per_cluster</code>	1或者3

第 10 章 编排

目录

介绍	278
编排认证模式	278
栈的域用户	280

Orchestration 是一个编排引擎，它提供了基于类似于代码的文本文件模板启动多个混合云应用程序的可能。一个原生的 Heat Orchestration Template (HOT) 格式正在进化，但它也尝试提供与 AWS CloudFormation 模板格式的兼容性，这样很多已有的 CloudFormation 模板就能在 OpenStack 中启动了。

介绍

Orchestration 是一个编排云的工具，它可以自动配置和部署栈中的资源。这样的部署可以变得很简单 -- 就像在 Ubuntu 中以 SQL 后端部署 WordPress 那样。而且它们也可以非常复杂，像启动一组自动扩展的云主机：基于 Telemetry 模块的实时 CPU 加载信息启动和停止。

编排栈被定义为模板，它们是非程序性文件，以资源、参数、输入、约束和依赖描述任务。Orchestration 模块最初被引入，它与 AWS CloudFormation 模板一同工作，使用 JSON 格式。

目前，编排也可以执行HOT(Heat 编排模版)模版，以YAML写就：一种简洁、松散的类似 Python/Ruby 类型结构格式(冒号，返回，缩进)，所以其很容易写就，解析，查找，生成工具，以及基于源码管理系统的维护。

Orchestration 可以通过 CLI 访问，也可以使用 RESTful 查询。Orchestration 模块提供 OpenStack-native REST API 和 CloudFormation-compatible Query API。为了通过 web 接口从模板启动栈，Orchestration 也与 OpenStack 仪表盘整合了。

更多如何使用编排模块命令行的细节请参阅 [OpenStack 命令行接口参考](#)

编排认证模式

Orchestration 认证模块定义了编排模块在所谓的延时操作中使用认证请求进行认证的过程。该操作的典型例子是 autoscaling 组在 heat 请求另一个组件 (nova、neutron 或其他) 来扩展 (或缩小) autoscaling 组的容量时进行更新。

目前，编排提供了两种类型的认证模式：

- 密码认证
- 基于OpenStack身份信任的认证。

密码认证

密码认证是初始的认证模式，编排模块是支持这种模式的。此类型的认证需要用于通过密码来编排。编排模块在数据库中存储着加密过的密码以及使用它来延迟操作。

下列步骤是密码认证的执行过程：

1. 用户请求创建栈，需提供一个令牌和用户名/密码(你使用python-heatclient或OpenStack图形面板亦需要令牌)。
2. 如果栈包含了任何标记为需要延迟的操作的资源，如果没有提供用户名/密码，编排引擎将会验证失败。
3. 用户名/密码 均加密过并存储在编排的数据库中。
4. 栈创建完成。
5. 在后续阶段，编排获取凭证而且请求另外的代表用户的令牌，令牌时没有限制范围的，为所有的栈所有者的角色提供访问权限。

keystone 信任认证

从IceHouse发布后，OpenStack的身份信任是一种新引进的认证方法。

信任是OpenStack认证的扩展，其提供了委托的办法，并通过OpenStack的身份可选模拟。关键的术语是trustor (委托人)和trustee (所接受委托的用户)。

要创建一个信任，trustor(在此情况下用户在编排模块中创建栈)提供OpenStack信任的下面信息：

- trustee的ID(你打算给委托给谁,在此情况下的编排服务用户)。
- 该角色被委派(通过配置heat.conf，但是它需要包括能够代表用户执行的延期操作的任何角色，例如为响应自动扩展事件而启动一个OpenStack计算实例)。
- 是否启用模拟。

OpenStack 认证提供一个 trust_id，它可以被信任人(且 仅能是 信任人)获取一个 trust scoped token。如果在创建时勾选了的话，这个令牌在是有限制范围的，这样信任人对授予了角色具有有限的访问权，以及信任用户的有效模拟。更多信息请阅读在认证管理小节。

下列步骤是信任认证的执行过程：

1. 用户通过API请求来创建一个栈(仅需要令牌)。
2. Orchestration 使用令牌来创建一个栈拥有者(委托人)和栈服务用户(受托人)之间的委托，代表一个在 heat 配置文件中的 trusts_delegated_roles 列表里的特定的角色(或多个角色)。默认情况下，heat 从委托人到受托人都设置了所有的可用角色。开发者们可能修改了这个列表来显示本地 RBAC 策略，如确保 heat 进程在模拟一个栈拥有者时仅能访问那些期望的服务。
3. 编排组件会加密trust id 并存放在编排数据库中。
4. 当需要进行一个延期的操作时，Orchestration 检索 trust id，然后请求一个委托范围的令牌，这个令牌允许服务用户在延期操作过程中扮演栈拥有者，例如启动一些需要栈拥有者支持的 OpenStack 计算实例来负责 AutoScaling 事件。

认证模式配置

在Kilo之前编排模块默认启用的认证模式是密码认证模式。Kilo版本则默认启用使用了信任认证模式。

要启用密码认证模式，在heat.conf中如下修改：

```
deferred_auth_method=password
```

要启用信任认证模式，在heat.conf中作如下修改：

```
deferred_auth_method=trusts
```

为了在授权过程中指定可以授权给受托人的委托人角色，需要在 heat.conf 中指定 trusts_delegated_roles 参数。如果 trusts_delegated_roles 参数没有定义，那么所有的委托人角色都可以授权给受托人。请注意委托授权角色应该在使用 Orchestration 模块之前，预先在 OpenStack 认证服务中配置好。

栈的域用户

编排的栈域用户允许heat在虚拟机启动后的内部进行认证，其执行下面操作：

- 提供元数据给实例内部的代理，此代理调用变更并应用元数据的配置表达式给实例。
- 发现完成一些动作的信号，典型的在虚拟机启动后配置软件(因为OpenStack计算服务在生成时会尽快的置虚拟机的状态为“活动”，而不是全部的编配都配置完毕)。
- 提供应用级别的状态或者实例内部的计量。例如，允许自动扩展的动作的执行，以响应一些性能的测量和服务质量。

编排提供了启用所有这些的API，但是所有的这些API均需要认证。例如，访问运行中的实例的代理的凭证。heat-cnftools 代理使用签名的请求，其需要由OpenStack身份服务所创建的ec2密钥对，此密钥对也用于编排的cloudformation和cloudwatch兼容的API的签名请求，由编排通过签名验证来进行授权(使用OpenStack身份服务ec2tokens扩展)。栈域用户允许封装所有的栈定义的用户(所创建的用户作为结果包含在了编排的模版中)，在不同的域中。其是特别创建的来包含仅和编排栈相关的内容。创建的用户是domain admin,那么编排使用此用户来管理栈用户域中的用户的生命周期。

栈的域用户配置

要配置栈域用户，要执行下面步骤：

1. 创建了一个特别的OpenStack身份服务域。例如，其中一个叫做heat，且在heat.conf中stack_user_domain属性设置了ID。
2. 用户有足够的权限来创建和删除项目，且创建了heat域中的用户。
3. 对于域管理用户的用户名和密码的设置在heat.conf (stack_domain_admin 和 stack_domain_admin_password)中。因为 heat_domain_admin仅给heat的管理权限。

要完成栈域用户的配置，你必须完成下面的步骤：

1. 创建域

\$OS_TOKEN 指的是令牌。例如，服务管理员令牌或其他一些有足够角色来创建用户和域的用户的有效令牌。\$KS_ENDPOINT_V3 指的是 v3 OpenStack 认证端点(如 http://keystone_address:5000/v3，其中的 keystone_address 是 OpenStack 认证服务的 IP 地址或可解析的域名)。

```
$ openstack --os-token $OS_TOKEN --os-url=$KS_ENDPOINT_V3 --os-identity-api-version=3 domain create heat --description "Owns users and projects created by heat"
```

此命令返回域ID，且简称为如下\$HEAT_DOMAIN_ID。

此命令返回域ID，且简称为如下\$HEAT_DOMAIN_ID。

2. 创建用户

```
$ openstack --os-token $OS_TOKEN --os-url=$KS_ENDPOINT_V3 --os-identity-api-version=3 user create --password $PASSWORD --domain $HEAT_DOMAIN_ID heat_domain_admin --description "Manages users and projects created by heat"
```

此命令返回用户ID，且简称为如下\$DOMAIN_ADMIN_ID。

3. 让用户成为源管理员:

```
$ openstack --os-token $OS_TOKEN --os-url=$KS_ENDPOINT_V3 --os-identity-api-version=3 role add --user $DOMAIN_ADMIN_ID --domain $HEAT_DOMAIN_ID admin
```

然后你需要添加域ID，用户名和密码，从这些步骤到heat.conf:

```
stack_domain_admin_password = 密码
stack_domain_admin = heat_domain_admin
stack_user_domain = 返回的域id是上面所创建的域
```

使用 workflow

下列步骤在栈创建期间会被执行：

1. 编排服务在“heat”域中创建了一个新的“栈域项目”，如果栈包含了任何需要创建“栈域用户”的话。
2. 所有需要用户的资源，Orchestration 会在 “stack domain project” 中创建用户，这个项目会在 heat 数据库中域 heat stack 关联，但它是完全分离并域 stack 项目本身（创建在 stack 域中的用户仍然会分配 heat_stack_user 角色，因此 API 显示它们通过 policy.json 访问有限的资源。详情请阅读 OpenStack 认证文档以获取更多相关信息）是不相关的（从认证观点上看）。
3. 当 API 请求被执行，Heat Orchestration 会进行一个内部的查询，并根据 policy.json 的限制，允许给定的栈可以被栈的拥有者的项目（默认到栈的 API 路径）和 栈的域项目 从数据库查询到特定的栈的详细信息。

要澄清的最后一点，这意味着现在有两条路径，可以通过 Orchestration API 来相同数据的检索。resource-metadata 的例子如下：

```
GET v1/{stack_owner_project_id}/stacks/{stack_name}/{stack_id}/resources/{resource_name}/metadata
```

或：

```
GET v1/{stack_domain_project_id}/stacks/{stack_name}/{stack_id}/resources/{resource_name}/metadata
```

栈属主会用到形式（通过 heat resource-metadata {stack_name} {resource_name}），而且实例中的任何代理稍后都会用到。

附录 A. 社区支持

目录

文档	282
问答论坛	283
OpenStack 邮件列表	283
OpenStack 维基百科	283
Launchpad的Bug区	283
The OpenStack 在线聊天室频道	284
文档反馈	285
OpenStack分发包	285

以下可用的资源是帮助用户运行和使用OpenStack。OpenStack社区会经常性的改进和增加OpenStack的主要特性，如果用户有问题，请不要在提问题方面犹豫。使用下面列出的资源，以获得OpenStack社区的支持，也能得到一些安装/使用时一些解决问题的思路和方法。

文档

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

要为文档那个提供反馈，请使用<openstack-docs@lists.openstack.org>邮件列表加入我们，可在网站[OpenStack 文档邮件列表](#)找到，或者[报告bug](#)。

以下书籍解释了如何安装一个基于OpenStack云及其相关的组件

- [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 \(LTS\)](#)

以下书籍解释了如何配置和运行一个基于OpenStack的云：

- [架构设计指南](#)
- [云计算平台管理员手册](#)
- [配置参考手册](#)
- [实战指南](#)
- [网络指南](#)
- [高可用指南](#)
- [安全指南](#)
- [虚拟机镜像指南](#)

以下书籍解释了如何使用OpenStack图形界面和命令行客户端：

- [应用程序接口快速入门](#)
- [用户指南](#)
- [管理员手册](#)
- [命令行参考](#)

下面文档提供了OpenStack 应用程序接口的参考和向导：

- [OpenStack应用程序接口完全参考\(HTML\)](#)
- [OpenStack应用程序接口完全参考\(PDF\)](#)

The [Training Guides](#) offer software training for cloud administration and management.

问答论坛

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 邮件列表

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>.

OpenStack 维基百科

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.

Launchpad的Bug区

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.

一些小贴士：

- 提供清晰、简洁的语法。
- 尽可能提供详细的细节描述。将命令行的输出或者trace输出粘贴出来，如果是截图请贴链接，以及其他任何有用的信息。
- 确保包含了软件和包的版本信息，尤其是使用的正在开发中的分支，诸如“Juno release” vs `git commit bc79c3ecc55929bac585d04a03475b72e06a3208`，这样的描述。
- 任何特别的部署信息都是有用的。例如用户使用的是Ubuntu 14.04，或者多节点安装。

以下列出Launchpad Bug区：

- [Bugs: OpenStack 块存储 \(cinder\)](#)
- [Bugs: OpenStack 计算 \(nova\)](#)
- [Bugs: OpenStack 仪表盘 \(horizon\)](#)
- [Bugs: OpenStack 认证 \(keystone\)](#)
- [Bugs: OpenStack Image service \(glance\)](#)
- [Bugs: OpenStack 网络 \(neutron\)](#)
- [Bugs: OpenStack 对象存储 \(swift\)](#)
- [Bugs: 裸金属服务 \(ironic\)](#)
- [Bugs: 数据处理服务 \(sahara\)](#)
- [Bugs: Database service \(trove\)](#)
- [Bugs: 编排 \(heat\)](#)
- [Bugs: 计量 \(ceilometer\)](#)
- [Bugs: 消息服务 \(zaqar\)](#)
- [Bugs: OpenStack 应用程序接口文档 \(developer.openstack.org\)](#)
- [Bugs: OpenStack 文档 \(docs.openstack.org\)](#)

The OpenStack 在线聊天室频道

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>.

文档反馈

要为文档提供反馈，请使用<openstack-docs@lists.openstack.org>邮件列表加入我们，可在网站[OpenStack 文档邮件列表](#)找到，或者[报告bug](#)。

OpenStack分发包

以下是Linux发行版针对OpenStack的社区支持：

- Debian: <https://wiki.debian.org/OpenStack>
- CentOS, Fedora, 以及 Red Hat Enterprise Linux: <https://www.rdoproject.org/>
- openSUSE 和 SUSE Linux Enterprise Server: <https://en.opensuse.org/Portal:OpenStack>
- Ubuntu: [ubuntu官方服务器团队之OpenStack云](#)