

Red Hat Enterprise Linux 7, CentOS 7

Kilo (March 30, 2015)

DRAFT
Kilo



Red Hat Enterprise Linux 7, CentOS 7, and Fedora 20 的 OpenStack 安装手册

Kilo (2015-03-30)

版权 © 2012-2015 OpenStack基金会 All rights reserved.

摘要

OpenStack® 系统由一些您所分开安装的关键项目组成。这些项目会根据您的云的需要共同运作。这些项目包括 Compute、Identity Service、Networking、Image Service、Block Storage、Object Storage、Telemetry、Orchestration 和 Database。您可以分开安装这些项目中的任何一个，并将他们配置成单节点或作为连接的实体。本指南会通过使用 Fedora 20 的包和 Red Hat Enterprise Linux 7 及其衍生的 EPEL 仓库进行安装。配置选项和示例配置文件的解释包含其中。

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.

目录

前言	vii
约定	vii
Document change history	vii
1. 构架	1
概况	1
概念架构	2
架构样例	2
2. 基本环境	11
在你开始之前	1
安全	12
网络	13
网络时间协议(NTP)	24
OpenStack包	25
Database	26
Messaging server	27
3. 添加身份认证服务	29
OpenStack的身份概念	29
安装和配置	30
创建租户、用户和角色	32
创建服务实体和 API 端点	35
验证操作	36
创建 OpenStack 客户端环境脚本	38
4. 添加镜像服务	39
OpenStack镜像服务	39
安装和配置	40
验证操作	43
5. 添加 Compute 服务	45
OpenStack 计算服务	45
安装配置控制节点服务器	48
安装和配置计算节点	51
验证操作	53
6. 添加网络组件	55
OpenStack 网络(neutron)	55
传统联网方式(nova-network)	77
下一步	79
7. 添加仪表盘(dashboard)	80
系统需求	80
安装和配置	81
验证操作	82
下一步	82
8. 添加块设备存储服务	83
OpenStack块存储	83
安装配置控制节点服务器	83
安装并配置一个存储节点	86
验证操作	89
下一步	91
9. 添加对象存储	92
OpenStack对象存储	92

安装并配置控制器节点	92
安装和配置存储节点	95
创建初始化的 rings	99
完成安装	103
验证操作	104
下一步	104
10. 添加 Orchestration 模块	105
Orchestration模块概念	105
安装和配置	105
验证操作	109
下一步	110
11. 添加 Telemetry 模块	111
Telemetry模块	111
安装配置控制节点服务器	112
Install the Compute agent for Telemetry	115
Configure the Image Service for Telemetry	117
Add the Block Storage service agent for Telemetry	117
Configure the Object Storage service for Telemetry	117
验证 Telemetry 的安装	118
下一步	119
12. 添加数据库服务	120
数据库服务概览	120
安装数据库服务	121
验证数据服务的安装	124
13. 添加数据处理服务	125
数据处理服务	125
安装数据处理服务	125
验证数据处理服务的安装	127
14. 启动一个实例	128
使用 OpenStack 网络 (neutron) 启动一台实例	128
使用传统网络 (nova-network) 启动一个实例	134
A. 保留的用户ID	140
B. 社区支持	141
文档	141
问答论坛	142
OpenStack 邮件列表	142
OpenStack 维基百科	142
Launchpad的Bug区	143
The OpenStack 在线聊天室频道	144
文档反馈	144
OpenStack分发包	144
术语表	145

表格清单

1.1. OpenStack 服务	1
2.1. Passwords	12
A.1. 保留的用户ID	140

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



敬告

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 \$ 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。

Document change history

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
October 15, 2014	<ul style="list-style-type: none"> 对于 Juno 的发布版本，本指南包含了一些更新：以一般的配置文件修改替换了 openstack-config 命令。规范化单个消息队列系统 (RabbitMQ)。关联通用 SQL 数据库，在合适的地方启用 MySQL 或 MariaDB。以 identity_uri 替换 auth_port 和 auth_protocol，以 auth_uri 替换 auth_host。多个修改的一致性。已经将其更新为 Juno 和更新的发布版本。
June 3, 2014	<ul style="list-style-type: none"> 开始使用 Juno 文档。
April 16, 2014	<ul style="list-style-type: none"> 更新 Icehouse，重新进行 Networking 安装，以将 ML2 作为插件使用，增加新的 Database Service 安装的章节，改进了基本的配置。
October 25, 2013	<ul style="list-style-type: none"> 增加最初的Debian支持。
October 17, 2013	<ul style="list-style-type: none"> Havana发行版。
October 16, 2013	<ul style="list-style-type: none"> 增加SUSE 系统企业版支持
October 8, 2013	<ul style="list-style-type: none"> 为Havana版本完成改编。
September 9, 2013	<ul style="list-style-type: none"> 为openSUSE建造

Revision Date	Summary of Changes
August 1, 2013	• Fixes to Object Storage verification steps. Fix bug 1207347 .
July 25, 2013	• Adds creation of cinder user and addition to the service tenant. Fix bug 1205057 .
May 8, 2013	• 为了统一更新了书的标题。
May 2, 2013	• 更新了封面并修正了附录中的一些小错误。

概况

概况	1
概念架构	2
架构样例	2

概况

OpenStack 通过各种各样的补充服务提供基础设施即服务 (IaaS) 的解决方案。每个服务都提供了应用程序接口 (API)，这样有利于集成。下面表格中列出了 OpenStack 的服务列表：

表 1.1. OpenStack 服务

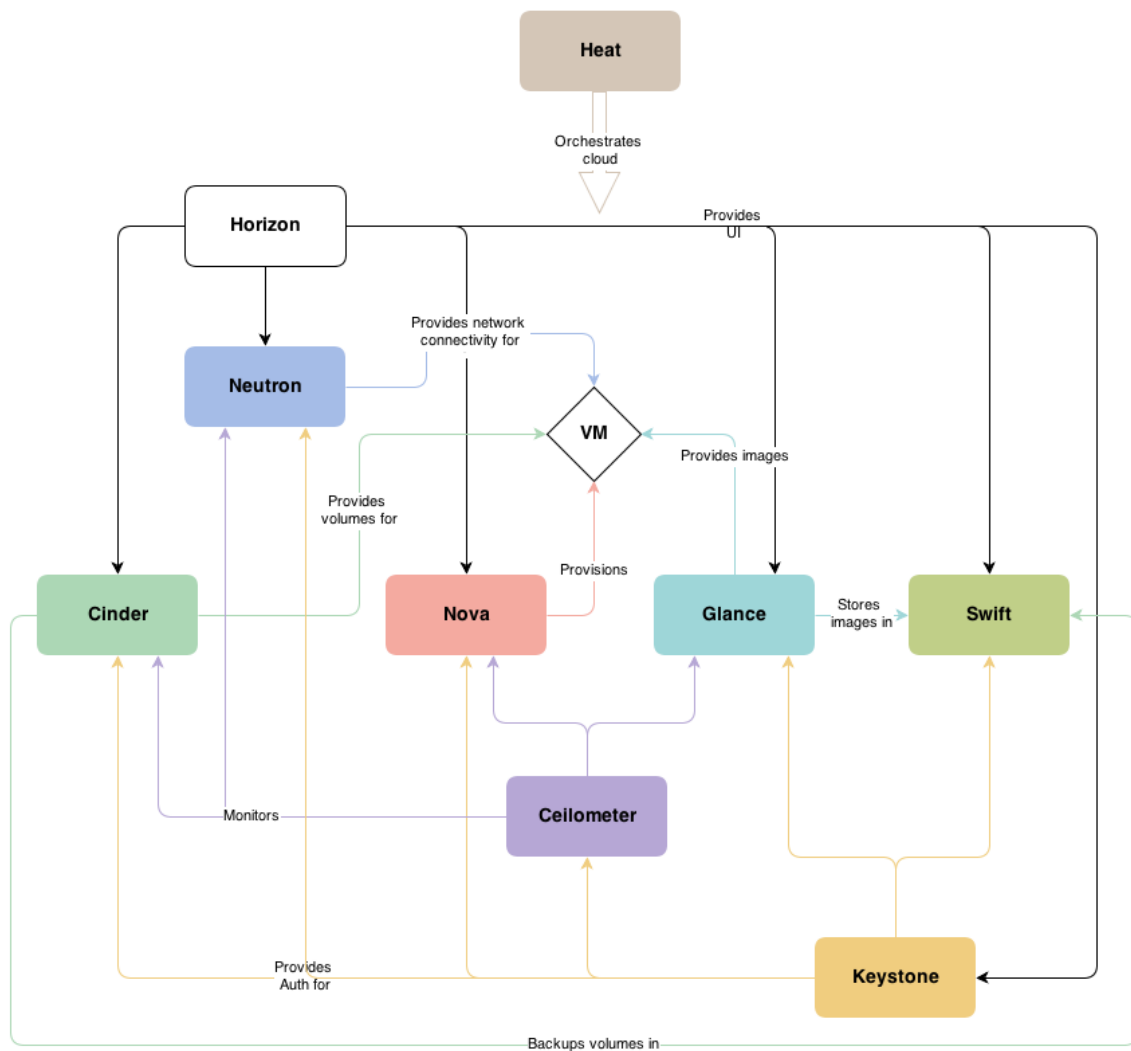
服务	项目名称	描述
Dashboard	控制面板	提供了一个基于web的自服务门户，与OpenStack底层服务交互，诸如启动一个实例，分配IP地址以及配置访问控制。
Compute	Nova	在OpenStack环境中计算实例的生命周期管理。按需响应包括生成、调度、回收虚拟机等操作。
网络	Neutron	确保为其它OpenStack服务提供网络连接即服务，比如OpenStack计算。为用户提供API定义网络和使用。基于插件的架构其支持众多的网络提供商和技术。
存储		
Object Storage	Swift	通过一个 RESTful,基于HTTP的应用程序接口存储和任意检索的非结构化数据对象。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。在此种方式下，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		
编配	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	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。

本指南描述如何在一个功能测试环境中部署这些服务，而且，举个例子来说，可以教会您如何构建一个生产环境。但实际上，您应该使用自动化工具，如 Ansible、Chef 和 Puppet 等来部署和管理生产环境。

概念架构

启动一台虚拟机或示例会包含很多服务之间的交互。下面的图展示了一个普通的 OpenStack 环境的概念架构。

图 1.1. 概念架构



架构样例

OpenStack 是高度可配置的，以满足各种计算、网络和存储选项的需要。本指南使您能够选择使用内核的组合以及可选的服务来开启自己的 OpenStack 冒险之旅。本指南使用以下架构样例：

- 包含 OpenStack 网络 (neutron) 的三个节点架构以及可选的块存储和对象存储服务节点：
- 控制节点服务器运行身份认证服务、镜像服务，管理部分计算和网络服务，运行网络插件以及仪表板。它还包括一些支持服务，例如 SQL 数据库、消息队列和网络时间协议 (NTP)。

可选地，可以在控制节点运行块设备存储、对象存储、Orchestration、Telemetry、数据库和数据处理服务的一部分。这些组件可以为您的环境提供额外的功能。

- 网络节点运行 Networking 插件和一些提供租户网络的代理，并提供 switching、routing、NAT 和 DHCP 服务。这个节点也处理租户虚拟机实例的外部 (Internet) 的连接。
- 控制节点运行 Compute 的 hypervisor 部分，这个部分将操作 tenant virtual machines 或实例。默认情况下，Compute 使用 KVM 作为 hypervisor。计算节点也可以运行 Networking 插件和代理，它们连接租户网络到虚拟机上，并提供防火墙 (security groups) 服务。您可以运行不止一台计算节点。

可选地，可以在计算节点运行 Telemetry 代理来收集 metrics。而且，它可以在一个隔离的存储网络上设置第三个网络接口，以增加存储服务性能。

- 该可选的块存储节点包含磁盘，块存储服务会向租户虚拟机实例提供这些磁盘。您可以运行多个该节点。

可选地，可以在块设备存储节点运行 Telemetry 来收集 metrics。并且，它可以在一个隔离的存储网络上设置第二个网络接口，来增加存储服务的性能。

- 该可选的对象存储节点包含磁盘，对象存储服务使用这些磁盘来存储帐户、容器和对象。您可以运行多个该节点。但是在最小架构样例中需要两个节点。

可选地，这些节点可以在一个隔离的存储网络上设置第二个网络接口，来增加存储服务的性能。



注意

如果您完成了这个架构，请跳过 [第 6 章 添加网络组件 \[55\]](#) 的“传统联网方式(nova-network)”一节 [77]。可选的服务可能需要额外的节点或已有节点上的额外资源。

图 1.2. Minimal architecture example with OpenStack Networking (neutron)—Hardware requirements

Minimal Architecture Example - Hardware Requirements

OpenStack Networking (neutron)

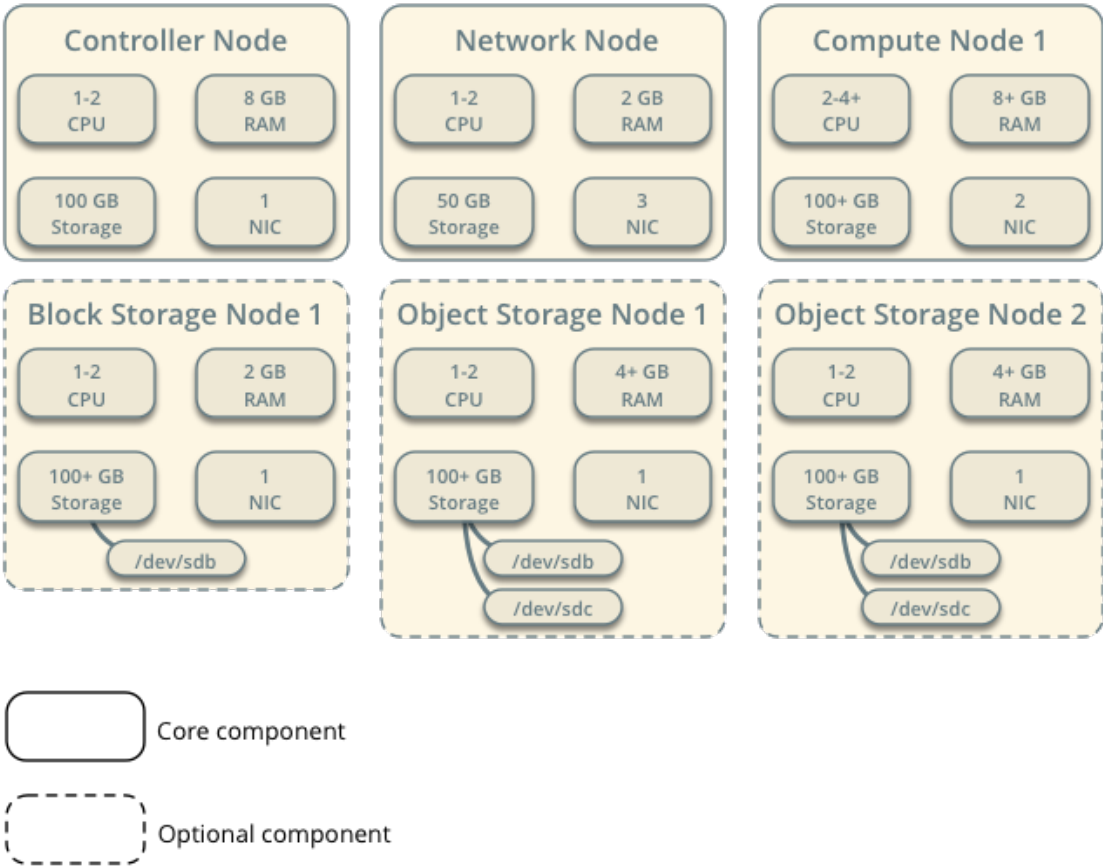
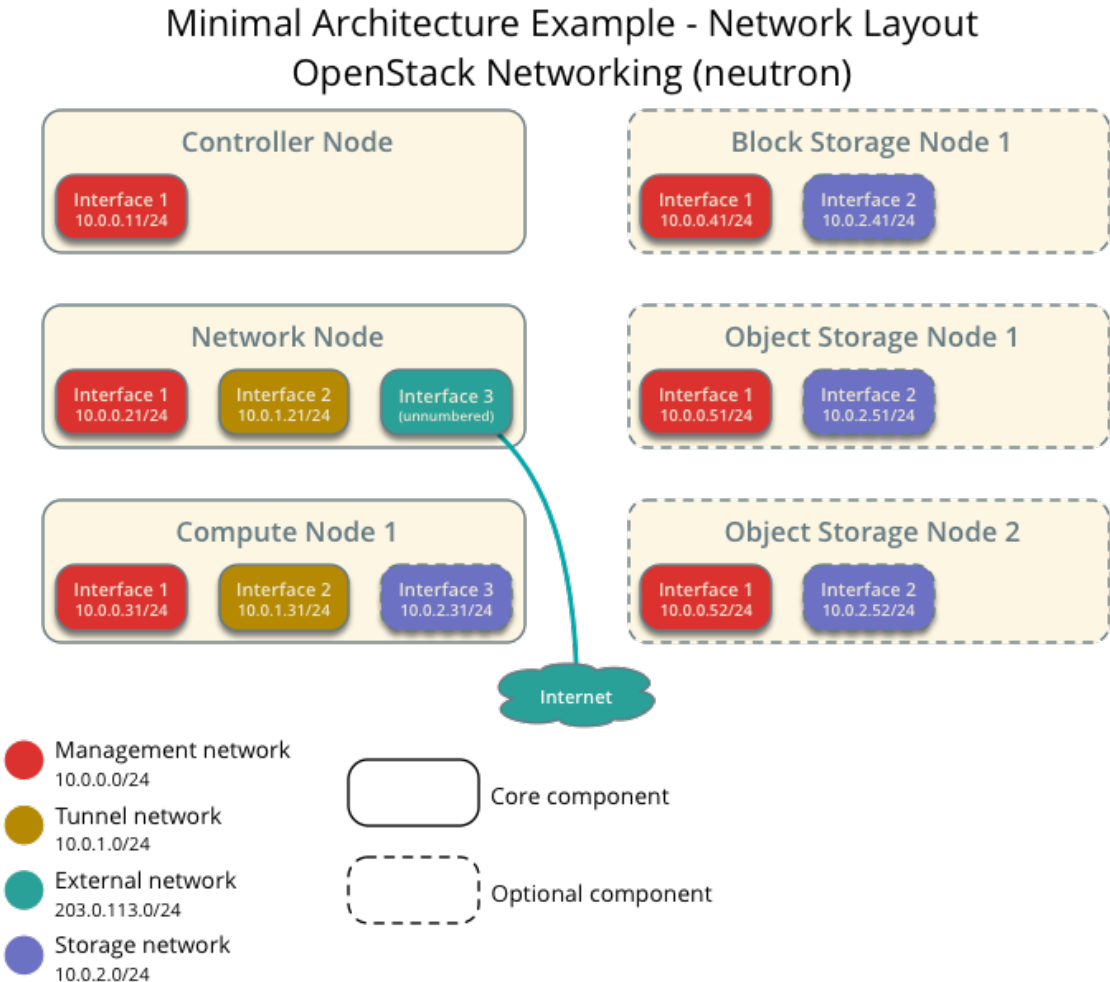


图 1.3. Minimal architecture example with OpenStack Networking (neutron)—Network layout



DRAFT



- DRAI

T-Ki

O - DR

- 可选地，可以在块设备存储节点运行 Telemetry 来收集 metrics。并且，它可以在一个隔离的存储网络上设置第二个网络接口，来增加存储服务的性能。

- 可选地，这些节点可以在一个隔离的存储网络上设置第二个网络接口，来增加存储服务的性能。

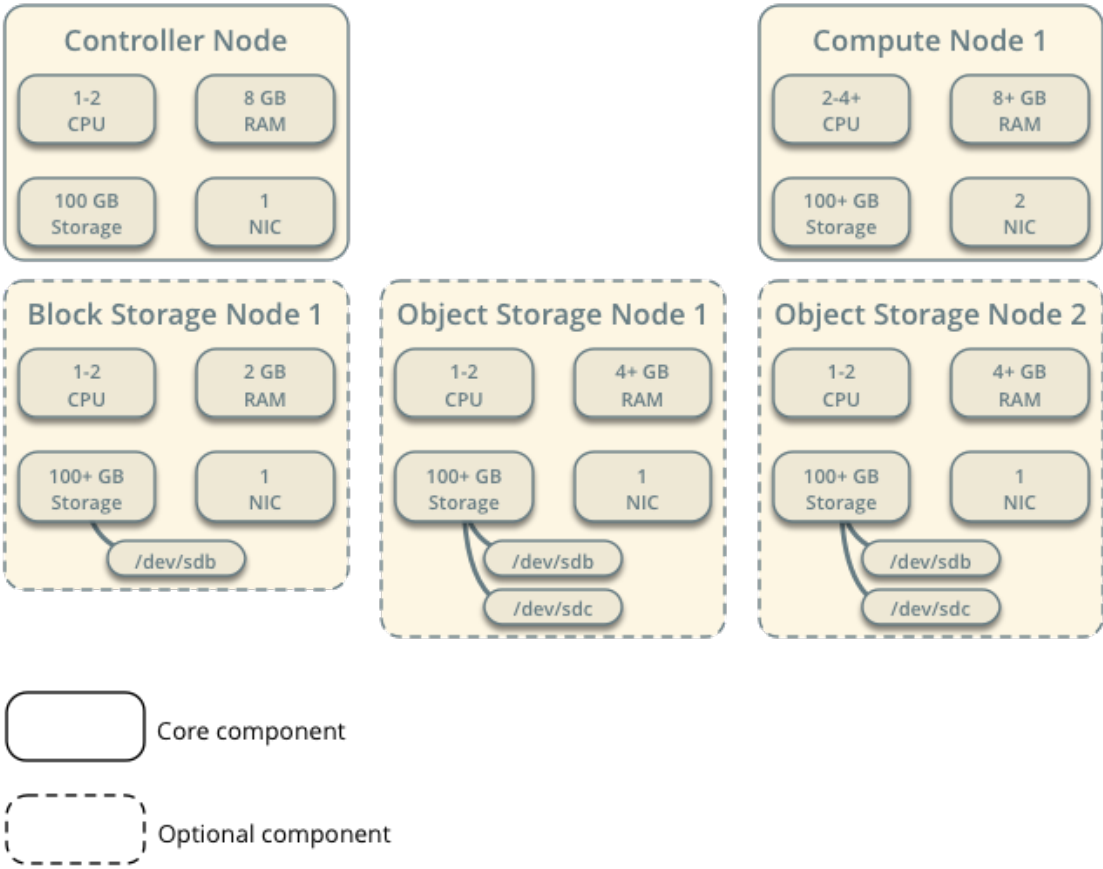


如果您完成了这个架构，请跳过 [第 6 章 添加网络组件 \[55\]](#) 的“[OpenStack 网络 \(neutron\)](#)”一节 [55]。为了使用可选的服务，您可能需要搭建额外的节点，如后面章节所描述。

图 1.5. Minimal architecture example with legacy networking (nova-network)—Hardware requirements

Minimal Architecture Example - Hardware Requirements

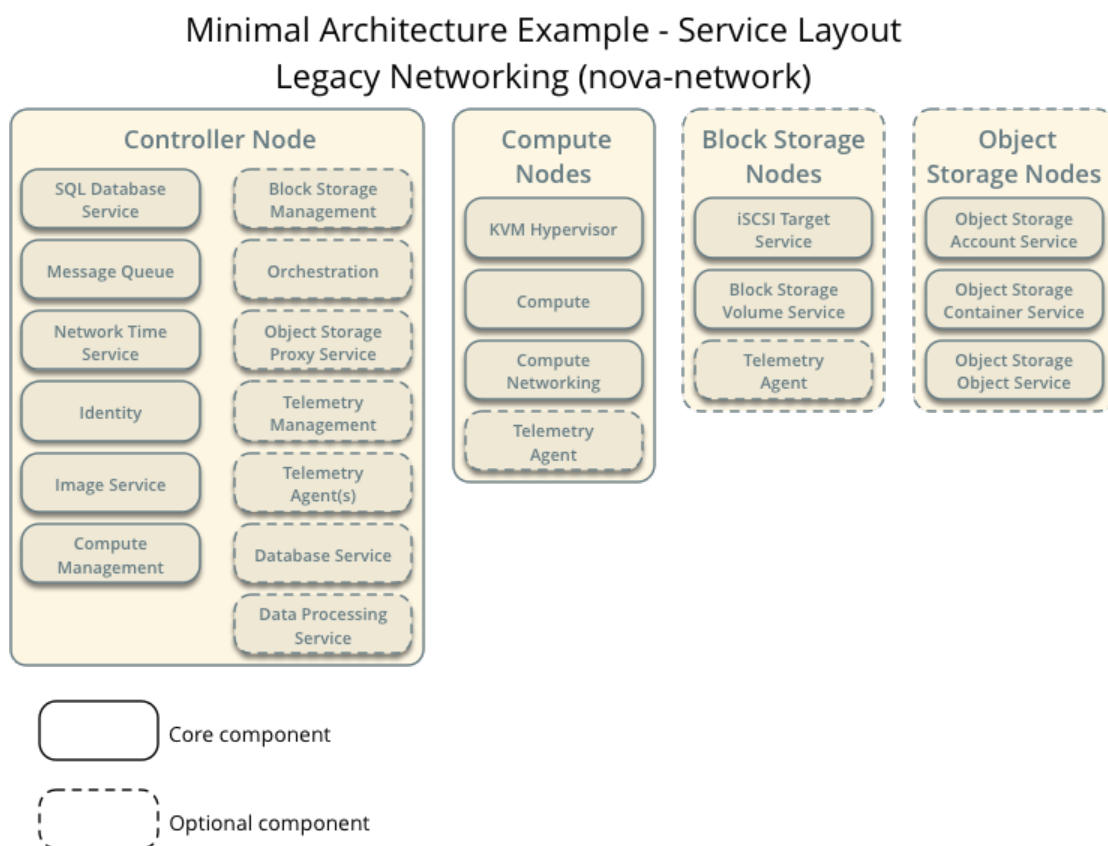
Legacy Networking (nova-network)



DRAFT



图 1.7. Minimal architecture example with legacy networking (nova-network)—Service layout



- DRAFT

- Kilc



T-Ki.

O - DR

ET - K

- Lot - D11

AET -



F

安全

To ease the installation process, this guide only covers password security where applicable. You can create secure passwords manually, generate them using a tool such as [pwgen](#), or by running the following command:

对于 OpenStack 的服务，本指南使用 SERVICE_PASS 来关联服务帐户的密码，用 SERVICE_DBPASS 来关联数据库的密码。

The following table provides a list of services that require passwords and their associated references in the guide:

表 2.1. Passwords

12

Password name	Description
ADMIN_PASS	Password of user admin
GLANCE_DBPASS	Database password for Image Service
GLANCE_PASS	Password of Image Service user glance
NOVA_DBPASS	Database password for Compute service
NOVA_PASS	Password of Compute service user nova
DASH_DBPASS	Database password for the dashboard
CINDER_DBPASS	Database password for the Block Storage service
CINDER_PASS	Password of Block Storage service user cinder
NEUTRON_DBPASS	Database password for the Networking service
NEUTRON_PASS	Password of Networking service user neutron
HEAT_DBPASS	Database password for the Orchestration service
HEAT_PASS	Password of Orchestration service user heat
CEILOMETER_DBPASS	Database password for the Telemetry service
CEILOMETER_PASS	Password of Telemetry service user ceilometer
TROVE_DBPASS	Database password of Database service
TROVE_PASS	Password of Database Service user trove

OpenStack and supporting services require administrative privileges during installation and operation. In some cases, services perform modifications to the host that can interfere with deployment automation tools such as Ansible, Chef, and Puppet. For example, some OpenStack services add a root wrapper to sudo that can interfere with security policies. See the [Cloud Administrator Guide](#) for more information. Also, the Networking service assumes default values for kernel network parameters and modifies firewall rules. To avoid most issues during your initial installation, we recommend using a stock deployment of a supported distribution on your hosts. However, if you choose to automate deployment of your hosts, review the configuration and policies applied to them before proceeding further.

网络

After installing the operating system on each node for the architecture that you choose to deploy, you must configure the network interfaces. We recommend that you disable any automated network management tools and manually edit the appropriate configuration files for your distribution. For more information on how to configure networking on your distribution, see the [documentation](#).

All nodes require Internet access for administrative purposes such as package installation, security updates, DNS, and NTP. In most cases, nodes should obtain Internet access through the management network interface. To highlight the importance of network separation, the example architectures use **private address space** for the management network and assume that network infrastructure provides Internet access via NAT. To illustrate the flexibility of IaaS, the example architectures use public IP address space for the external network and assume that network infrastructure provides direct Internet access to instances in your OpenStack environment. In environments with only one block of public IP address space, both the management and external networks must ultimately obtain Internet access using it. For simplicity, the diagrams in this guide only show Internet access for OpenStack services.



注意

Your distribution enables a restrictive firewall by default. During the installation process, certain steps will fail unless you alter or disable the firewall. For more information about securing your environment, refer to the [OpenStack Security Guide](#).

为架构样例中的 [OpenStack Networking \(neutron\)](#) 或 [legacy networking \(nova-network\)](#) 进行网络配置。

OpenStack 网络 (neutron)

使用 OpenStack 网络 (neutron) 的架构样例中，需要一个控制节点、一个网络节点以及至少一个计算节点。控制节点一个在管理网络上的网络接口。网络节点在包含一个在管理网络上的网络接口，一个在实例隧道网络上，一个在外部网络上。计算节点包含一个在管理网络上的网络接口和一个在实例隧道网络上的接口。

该样例架构假设使用以下网络：

- 管理使用 10.0.0.0/24 帶有网关 10.0.0.1



注意

这个网络需要一个网关以为所有节点提供内部的管理目的的访问，例如包的安装、安全更新、DNS 以及 NTP。

- 实例通道使用 10.0.1.0/24 无网关



注意

这个网络不需要网关，因为它只发生在您的 OpenStack 环境中的网络节点和计算节点之间的会话中。

- 外部通道使用 203.0.113.0/24 帶有网关 203.0.113.1



注意

这个网络需要一个网关来提供在环境中内部实例的访问。

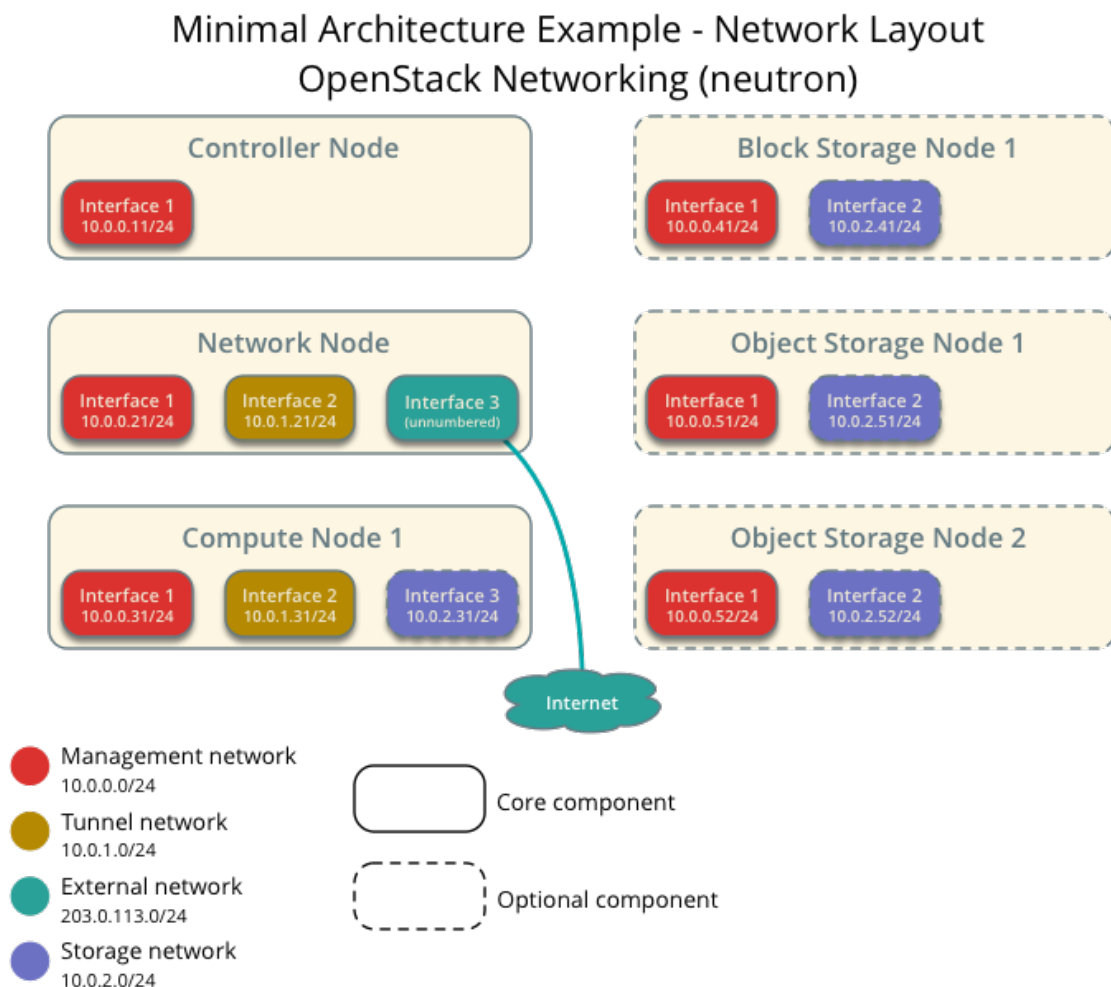
您可以修改这些范围和网关来以您的特定网络设施进行工作。



注意

网络接口名称在不同发布版本上各异。通常，接口使用“eth”并跟着一个顺序值。为了涵盖各种差异，本指南简单的指代第一个接口为最小数值的接口，第二个接口为中间数值的接口，第三个接口为最高数值的接口。

图 2.1. Minimal architecture example with OpenStack Networking (neutron)—Network layout



除非您打算使用该架构样例中提供的准确配置，否则您必须在本过程中修改网络以匹配您的环境。并且，每个节点除了 IP 地址之外，还必须能够解析其他节点的名称。例如，`controller` 这个名称必须解析为 `10.0.0.11`，即控制节点上的管理网络接口的 IP 地址。



敬告

重新配置网络接口会中断网络连接。我们建议使用本地终端会话来进行这个过程。

控制节点服务器

配置网络：

1. 将第一个接口配置为管理网络接口：

IP 地址: 10.0.0.11

子网掩码: 255.255.255.0 (or /24)

默认网关: 10.0.0.1

2. 重启系统以激活修改。

设置名称解析：

1. 将节点的主机名设置为 `controller`。
2. 修改文件 `/etc/hosts` 并包含如下内容：

```
# controller
10.0.0.11    controller

# network
10.0.0.21    network

# compute1
10.0.0.31    compute1
```

网络节点

配置网络：

1. 将第一个接口配置为管理网络接口：

IP地址: 10.0.0.21

子网掩码: 255.255.255.0 (or /24)

默认网关: 10.0.0.1

2. 将第二个接口配置为实例的隧道网络接口：

IP地址: 10.0.1.21

子网掩码: 255.255.255.0 (or /24)

3. 外部网络接口使用特殊的配置，不分配 IP 地址。将第三个接口配置为外部网络接口：

将其中的 INTERFACE NAME 替换为实际的接口名称。例如，eth2 或 ens256。

- 修改配置文件 `/etc/sysconfig/network-scripts/ifcfg-INTERFACE_NAME` 包含如下内容：

不要修改 HWADDR 和 UUID 关键字的内容。

```
DEVICE=INTERFACE_NAME
TYPE=Ethernet
ONBOOT="yes"
BOOTPROTO="none"
```

4. 重启系统以激活修改。

设置名称解析：

1. 将该节点的主机名设置为 `network`。
2. 修改文件 `/etc/hosts` 并包含如下内容：

```
#network
10.0.0.21    network

#controller
10.0.0.11    controller

#compute1
10.0.0.31    compute1
```

计算节点

配置网络：

1. 将第一个接口配置为管理网络接口：

IP 地址：10.0.0.31

子网掩码: 255.255.255.0 (or /24)

默认网关: 10.0.0.1



注意

另外的计算节点应使用 10.0.0.32、10.0.0.33 等等。

2. 将第二个接口配置为实例的隧道网络接口：

IP地址: 10.0.1.31

子网掩码: 255.255.255.0 (or /24)



注意

附加的计算节点需要使用10.0.1.32, 10.0.1.33等等。

- ### 3. 重启系统以激活修改。

设置名称解析：

1. 将该节点的主机名设置为 `computer1`。
2. 修改文件 `/etc/hosts` 并包含如下内容：

```
#compute1
10.0.0.31    compute1

#controller
10.0.0.11    controller

#network
10.0.0.21    network
```

验证连通性

我们建议您在继续之前，验证到 Internet 和各个节点之间的连通性。

1. 从控制节点，ping 互联网上的一个站点：

```
# ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data:
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

2. 从控制节点，ping 管理接口位于网络节点：

```
# ping -c 4 网络
PING network (10.0.0.21) 56(84) bytes of data.
64 bytes from network (10.0.0.21): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from network (10.0.0.21): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from network (10.0.0.21): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from network (10.0.0.21): icmp_seq=4 ttl=64 time=0.202 ms

--- network ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

3. 从控制节点，ping 管理接口位于计算节点：

```
# ping -c 4 computel
PING computel (10.0.0.31) 56(84) bytes of data.
64 bytes from computel (10.0.0.31): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from computel (10.0.0.31): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from computel (10.0.0.31): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from computel (10.0.0.31): icmp_seq=4 ttl=64 time=0.202 ms

--- network ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

4. 从网络节点，ping 互联网上的一个站点：

```
# ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data:
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

5. 从网络节点，ping 管理接口位于控制节点：

```
# ping -c 4 控制器
PING controller (10.0.0.11) 56(84) bytes of data.
```

```
64 bytes from controller (10.0.0.11): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from controller (10.0.0.11): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from controller (10.0.0.11): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from controller (10.0.0.11): icmp_seq=4 ttl=64 time=0.202 ms

--- controller ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

6. 从网络节点，ping 实例通道接口位于计算节点：

```
# ping -c 4 10.0.1.31
PING 10.0.1.31 (10.0.1.31) 56(84) bytes of data.
64 bytes from 10.0.1.31 (10.0.1.31): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from 10.0.1.31 (10.0.1.31): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from 10.0.1.31 (10.0.1.31): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from 10.0.1.31 (10.0.1.31): icmp_seq=4 ttl=64 time=0.202 ms

--- 10.0.1.31 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

7. 从计算节点，ping 互联网上的一个站点：

```
# ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data:
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18,3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17,5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17,5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17,4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

8. 从计算节点，ping 管理接口位于控制节点：

```
# ping -c 4 控制器
PING controller (10.0.0.11) 56(84) bytes of data.
64 bytes from controller (10.0.0.11): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from controller (10.0.0.11): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from controller (10.0.0.11): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from controller (10.0.0.11): icmp_seq=4 ttl=64 time=0.202 ms

--- controller ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

9. 从计算节点， ping 实例通道接口位于网络节点：

```
# ping -c 4 10.0.1.21
PING 10.0.1.21 (10.0.1.21) 56(84) bytes of data.
64 bytes from 10.0.1.21 (10.0.1.21): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from 10.0.1.21 (10.0.1.21): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from 10.0.1.21 (10.0.1.21): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from 10.0.1.21 (10.0.1.21): icmp_seq=4 ttl=64 time=0.202 ms

--- 10.0.1.21 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

传统联网方式(nova-network)

使用传统网络 (nova-network) 的架构样例需要一个控制节点和至少一个计算节点。控制节点包含一个管理网络上的网络接口。计算节点包含一个管理网络上的网络接口和一个外部网络上的接口。

该样例架构假设使用以下网络：

- 管理使用 10.0.0.0/24 帶有网关 10.0.0.1



注意

这个网络需要一个网关以为所有节点提供内部的管理目的的访问，例如包的安装、安全更新、DNS 以及 NTP。

- 外部通道使用 203.0.113.0/24 帶有网关 203.0.113.1



注意

这个网络需要一个网关来提供在环境中内部实例的访问。

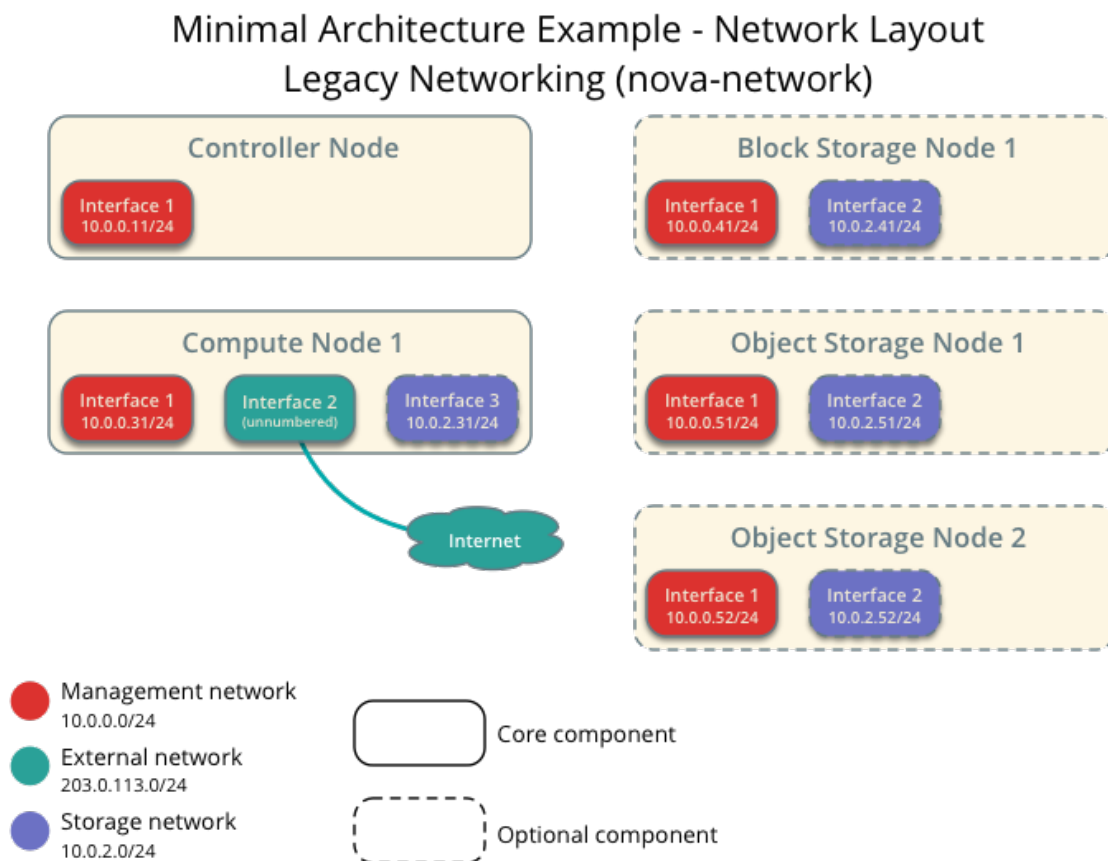
您可以修改这些范围和网关来以您的特定网络设施进行工作。



注意

网络接口由发行版的不同而有各种名称。传统上，接口使用“eth”加上一个数字序列命名。为了覆盖到所有不同的名称，本指南简单地将数字最小的接口引用为第一个接口，第二个接口则为更大数字的接口。

图 2.2. Minimal architecture example with legacy networking (nova-network)—Network layout



除非您打算使用该架构样例中提供的准确配置，否则您必须在本过程中修改网络以匹配您的环境。并且，每个节点除了 IP 地址之外，还必须能够解析其他节点的名称。例如，`controller` 这个名称必须解析为 `10.0.0.11`，即控制节点上的管理网络接口的 IP 地址。



敬告

重新配置网络接口会中断网络连接。我们建议使用本地终端会话来进行这个过程。

控制节点服务器

配置网络：

1. 将第一个接口配置为管理网络接口：

IP 地址: 10.0.0.11

子网掩码: 255.255.255.0 (or /24)

默认网关: 10.0.0.1

2. 重启系统以激活修改。

设置名称解析：

1. 将节点的主机名设置为 controller。
2. 修改文件 `/etc/hosts` 并包含如下内容：

```
# controller
10.0.0.11    controller

# compute1
10.0.0.31    compute1
```

计算节点

配置网络：

1. 将第一个接口配置为管理网络接口：

IP 地址：10.0.0.31

子网掩码: 255.255.255.0 (or /24)

默认网关: 10.0.0.1



注意

另外的计算节点应使用 10.0.0.32、10.0.0.33 等等。

2. 外部网络接口使用一个特定的配置，不分配 IP 地址给这个接口。将第二个接口配置为外部网络接口：

将其中的 INTERFACE NAME 替换为实际的接口名称。例如，eth1 或 ens224。

- 修改配置文件 `/etc/sysconfig/network-scripts/ifcfg-INTERFACE_NAME` 包含如下内容：

不要修改 HWADDR 和 UUID 关键字的内容。

```
DEVICE=INTERFACE_NAME
TYPE=Ethernet
ONBOOT="yes"
BOOTPROTO="none"
```

3. 重启系统以激活修改。

设置名称解析：

1. 将该节点的主机名设置为 `computer1`。
2. 修改文件 `/etc/hosts` 并包含如下内容：

```
# computel
10.0.0.31    computel

# controller
10.0.0.11    controller
```

验证连通性

我们建议您在继续之前，验证到 Internet 和各个节点之间的连通性。

1. 从控制节点，ping 互联网上的一个站点：

```
# ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data:
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

2. 从控制节点，ping 管理接口位于计算节点：

```
# ping -c 4 computel
PING computel (10.0.0.31) 56(84) bytes of data.
64 bytes from computel (10.0.0.31): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from computel (10.0.0.31): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from computel (10.0.0.31): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from computel (10.0.0.31): icmp_seq=4 ttl=64 time=0.202 ms

--- computel ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 0.202/0.217/0.263/0.030 ms
```

3. 从计算节点，ping 互联网上的一个站点：

```
# ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data:
64 bytes from 174.143.194.225: icmp_seq=1 ttl=54 time=18.3 ms
64 bytes from 174.143.194.225: icmp_seq=2 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=3 ttl=54 time=17.5 ms
64 bytes from 174.143.194.225: icmp_seq=4 ttl=54 time=17.4 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3022ms
rtt min/avg/max/mdev = 17.489/17.715/18.346/0.364 ms
```

4. 从计算节点，ping 管理接口位于控制节点：

```
# ping -c 4 控制器
PING controller (10.0.0.11) 56(84) bytes of data.
64 bytes from controller (10.0.0.11): icmp_seq=1 ttl=64 time=0.263 ms
64 bytes from controller (10.0.0.11): icmp_seq=2 ttl=64 time=0.202 ms
64 bytes from controller (10.0.0.11): icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from controller (10.0.0.11): icmp_seq=4 ttl=64 time=0.202 ms

--- controller ping statistics ---
```

Kilo

DRAFT

5 - Kilc

O - DRA

AFT - F

- Kilo -

DRAFT

T - Kil

- 10 - DR

RAFT -

Kilo



5 - Kilc

AFT - F

- DRAFT

T - Kil

10 - DR

RAFT -

- Kilo

DRAFT

5 - Kilc

- AFT - F

Kilo -

DRAFT

- T - Kil

10 - DR


```
# systemctl start ntpd.service
```

验证操作

我们建议您在继续进一步的操作之前验证 NTP 的同步。有些节点，特别是哪些引用了控制节点的，需要花费一些时间去同步。

1. 在控制节点上执行这个命令：

#ntpq -c peers	remote	refid	st	t	when	poll	reach	delay	offset	jitter
*ntp-server1	192.0.2.11		2	u	169	1024	377	1.901	-0.611	5.483
+ntp-server2	192.0.2.12		2	u	887	1024	377	0.922	-0.246	2.864

remote 这一列的内容应该填写一个或多个 NTP 服务器的主机名或 IP 地址。



注意

refid 这一列的内容一般关联到上游服务器的 IP 地址。

2. 在控制节点上执行这个命令：

```
# ntpq -c assoc
ind assid status  conf reach auth condition  last_event cnt
=====
1 20487 961a  yes  yes  none sys.peer  sys_peer 1
2 20488 941a  yes  yes  none candidate  svs_peer 1
```

condition 这一列的内容应该填写至少一台服务器的 sys.peer。

3. 在所有节点上执行这个命令：

#ntpq -c peers	remote	refid	st	t	when	poll	reach	delay	offset	jitter
*controller	192.0.2.21		3	u	47	64	37	0.308	-0.251	0.079

remote 这一列的内容应该填写控制节点的主机名。



注意

refid 这一列的内容一般关联到上游服务器的 IP 地址。

4. 在所有节点上执行这个命令：

```
#ntpq -c assoc
ind assid status conf reach auth condition last_event cnt
=====
1 21181 963a yes yes none sys,peer sys peer 3
```

condition 这一列的内容应该填写 sys.peer。

OpenStack包

由于不同的发布日程，发行版发布 OpenStack 的包作为发行版的一部分，或使用其他方式。请在所有节点上执行这些程序。



注意

禁用或移除所有自动更新的服务，因为它们会影响到您的 OpenStack 环境。

配置前的准备

1. 安装 yum-plugin-priorities 包，以启用仓库中相对优先级的分配：

```
# yum install yum-plugin-priorities
```

2. On RHEL and CentOS, enable the [EPEL](#) repository:

```
# yum install http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-5.noarch.rpm
```



注意

Fedora不需要这个仓库

3. 在RHEL中，使用订阅管理器启用extras仓库：

```
# subscription-manager repos --enable=rhel-7-server-extras-rpms
```



注意

CentOS和Fedora不需要这个仓库。

启用 OpenStack 仓库

- 安装包 `rdo-release-juno` 以启用 RDO 仓库：

```
# yum install http://rdo.fedorapeople.org/openstack-juno/rdo-release-juno.rpm
```

完成安装

- 更新系统中的这些软件包：

```
# yum upgrade
```



注意

如果更新了一个新内核，重启系统来使用新内核。

Database

Most OpenStack services use an SQL database to store information. The database typically runs on the controller node. The procedures in this guide use MariaDB or MySQL depending on the distribution. OpenStack services also support other SQL databases including [PostgreSQL](#).

安装并配置数据库服务

1. 安装软件包：

To configure the message broker service

1. Start the message broker service and configure it to start when the system boots:

```
# systemctl enable rabbitmq-server.service
# systemctl start rabbitmq-server.service
```

2. The message broker creates a default account that uses `guest` for the username and password. To simplify installation of your test environment, we recommend that you use this account, but change the password for it.

Run the following command:

将其中的 RABBIT_PASS 替换为一个合适的密码。

```
# rabbitmqctl change_password guest RABBIT_PASS
Changing password for user "guest" ...
...done.
```

You must configure the `rabbit_password` key in the configuration file for each OpenStack service that uses the message broker.



注意

For production environments, you should create a unique account with a suitable password. For more information on securing the message broker, see the [documentation](#).

If you decide to create a unique account with a suitable password for your test environment, you must configure the `rabbit_userid` and `rabbit_password` keys in the configuration file of each OpenStack service that uses the message broker.

3. For RabbitMQ version 3.3.0 or newer, you must enable remote access for the guest account.

- a. Check the RabbitMQ version:

```
# rabbitmqctl status | grep rabbit
Status of node 'rabbit@controller' ...
{running_applications,[{rabbit,"RabbitMQ","3.4.2"},
```

- b. If necessary, edit the `/etc/rabbitmq/rabbitmq.config` file and configure `loopback_users` to reference an empty list:

```
[{rabbit, [{loopback users, []}]}].
```



注意

Contents of the original file might vary depending on the source of the RabbitMQ package. In some cases, you might need to create this file.

- c. Restart the message broker service:

```
# systemctl restart rabbitmq-server.service
```

第3章 添加身份认证服务

目录

OpenStack的身份概念	29
安装和配置	30
创建租户、用户和角色	32
创建服务实体和 API 端点	35
验证操作	36
创建 OpenStack 客户端环境脚本	38

OpenStack的身份概念

OpenStack Identity Service 实现下列功能:

- 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	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。
	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。
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，诸如计算服务(nova),对象服务(swift),或镜像服务(glance)。它提供一个或多个端点来让用户可以访问资源和执行操作。

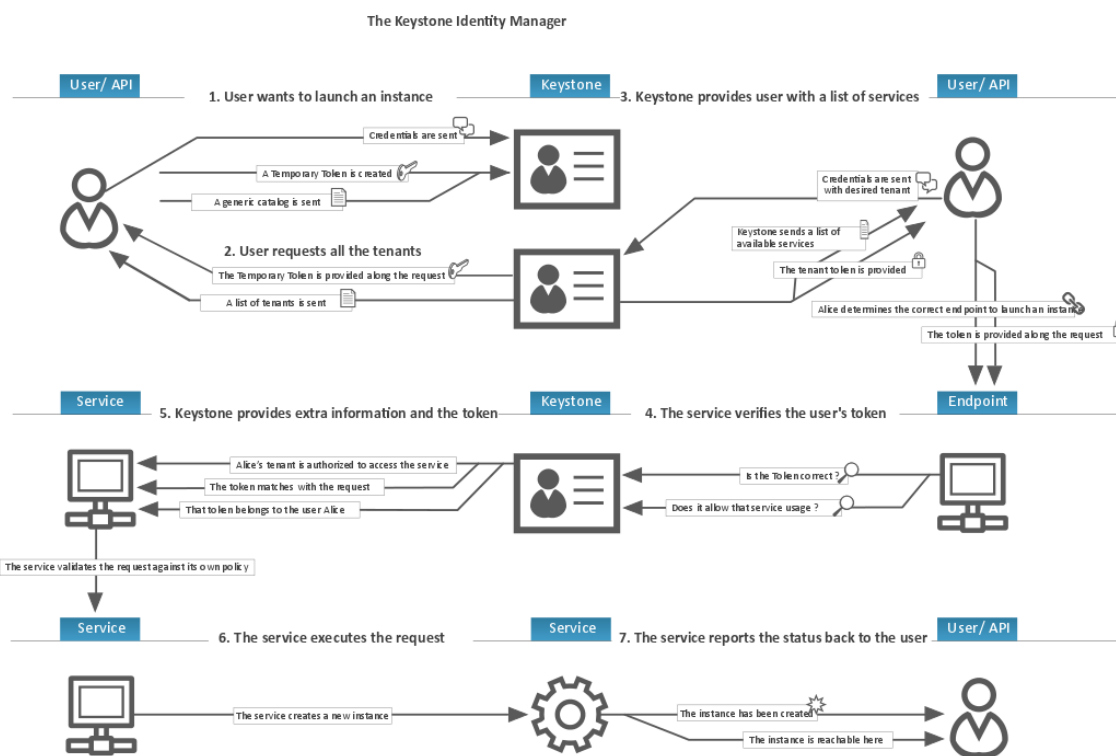
端点 一个当用户访问服务时用到的可访问的网络地址，通常是一个URL地址。如果用户为模板而使用扩展，一个端点可以被创建，它表示模板是为所有可用的跨region的可消费的服务。

角色 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。

在身份服务中，一个令牌所携带用户信息包含了角色列表。服务在被调用时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。

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:



安装和配置

这一章描述了怎样在控制结点安装和配置Openstack身份认证服务

配置前的准备

在你配置 OpenStack 身份认证服务前，你必须创建一个数据库和管理员令牌。

1. 完成下面的步骤以创建数据库：
 - a. 以 root 用户身份通过数据库客户端连接到数据库服务：

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

- b. 创建 keystone 数据库：

```
CREATE DATABASE keystone;
```

- c. 为 keystone 数据库赋予恰当的访问权限：

```
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost'  
  IDENTIFIED BY 'KEYSTONE_DBPASS';  
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%'  
  IDENTIFIED BY 'KEYSTONE_DBPASS';
```

将 KEYSTONE DBPASS 替换为实际的密码。

- d. 退出数据库客户端。

2. 生成一个随机值在初始的配置中作为管理员的令牌。

```
$ openssl rand -hex 10
```

安装和配置组件

1. 运行以下命令来安装包。

```
# yum install openstack-keystone python-keystoneclient
```

2. 编辑 `/etc/keystone/keystone.conf`，并完成下列操作：

- a. 在 [DEFAULT] 段中，对管理员令牌进行配置。

```
[DEFAULT]
...
admin_token = ADMIN_TOKEN
```

将 ADMIN TOKEN 替换为上一步中生成的随机字符串。

- b. 在[database]段，配置数据库访问相关参数：

```
[database]
...
connection=mysql://keystone:KEYSTONE_DBPASS@控制器/keystone
```

将 KEYSTONE DBPASS 替换为实际的数据库用户的密码。

- c. 在[token]部分，设置 UUID 令牌提供者和 SQL 驱动。

```
[token]
...
provider = keystone.token.providers.uuid.Provider
driver = keystone.token.persistence.backends.sql.Token
```

- d. 在[revoke]部分，配置 SQL 的撤回驱动：

```
[revoke]
...
driver = keystone.contrib.revoke.backends.sql.Revoke
```

- e. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

3. 生成通用的证书和私钥文件，并修改这些文件的访问权限：

```
#keystone-manage pki_setup --keystone-user keystone --keystone-group keystone
#chown -R keystone:keystone /var/log/keystone
#chown -R keystone:keystone /etc/keystone/ssl
#chmod -R o-rwx /etc/keystone/ssl
```

- #### 4. 初始化身份认证服务的数据库：

```
# su -s /bin/sh -c "keystone-manage db_sync" keystone
```

完成安装

1. 启动身份认证服务，并将该服务配为系统启动时自动启动：

```
# systemctl enable openstack-keystone.service
# systemctl start openstack-keystone.service
```

- 身份认证服务默认的情况会永久保存已经过期的身份凭证。过期凭证的累积会增加数据库的体积甚至减低整个服务的性能，特别是在那些资源紧缺的环境中。

建议使用 cron 配置计划任务每小时定时清理过期的身份凭证：

```
# (crontab -l -u keystone 2>&1 | grep -q token_flush) ||
echo 'hourly /usr/bin/keystone-manage token_flush >/var/log/keystone/keystone-tokenflush.log
2>&1'
>> /var/spool/cron/keystone
```

创建租户、用户和角色

安装了身份认证服务后，为您的环境创建 tenants (项目)、users 和 roles。在您执行 keystone 命令之前，必须使用您“[安装和配置](#)”一节 [30] 中创建的临时的管理员令牌并手动配置身份认证服务的位置（入口点）。

您可以使用 `--os-token` 参数将管理员令牌传递给 `keystone` 命令，或设置临时的 `OS_SERVICE_TOKEN` 环境变量。类似的，您也可以使用 `--os-endpoint` 选项将身份认证服务的位置传递给 `keystone` 命令，或设置临时的 `OS_SERVICE_ENDPOINT` 环境变量。本指南使用环境变量以缩短命令行的长度。

For more information, see the [Operations Guide - Managing Project and Users](#).

配置前的准备

1. 配置管理员令牌：

```
$ export OS_SERVICE_TOKEN=ADMIN_TOKEN
```

将其中的 ADMIN_TOKEN 替换为您在 “[安装和配置](#)” 一节 [30] 中生成的管理员令牌。例如：

```
$ export OS_SERVICE_TOKEN=294a4c8a8a475f9b9836
```


- GT - K

- DRA

- Kilc

- DRAFT

- Kilo

RAFT

ilo - I

C - DR

T-Ki

- DRAFT

-Kilo

E

E

- E

E

E

- E

E



E

E


```
--service-id $(keystone service-list | awk '/ identity / {print $2}')
--publicurl http://控制器:5000/v2.0
--internalurl http://控制器:5000/v2.0
--adminurl http://控制器:35357/v2.0
--region regionOne
```

Property	Value
adminurl	http://controller:35357/v2.0
id	11f9c625a3b94a3f8e66bf4e5de2679f
internalurl	http://controller:5000/v2.0
publicurl	http://controller:5000/v2.0
region	regionOne
service_id	15c11a23667e427e91bc31335b45f4bd



注意

这个命令需要关联您在前一个步骤中所创建的服务的 ID。



注意

每一个您添加到 OpenStack 环境中的服务需要添加一些信息，如 API 端点，到身份认证服务中。本指南涵盖服务安装的部分包括一些步骤，以添加合适的信息到身份认证服务中。

验证操作

这个部分描述如何验证身份认证服务的操作。

- 取消 OS SERVICE TOKEN 和 OS SERVICE ENDPOINT 临时环境变量的设置：

```
$ unset OS_SERVICE_TOKEN OS_SERVICE_ENDPOINT
```

2. 使用admin 租户和用户，需要一个认证的令牌：

```
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS  
--os-auth-url http://controller:35357/v2.0 token-get
```

Property	Value
expires	2014-10-10T12:50:12Z
id	8963eb5ccd864769a894ec316ef8f7d4
tenant_id	6f4c1e4cbfef4d5a8a1345882fbca110
user_id	ea8c352d253443118041c9c8b8416040

将其中的 `ADMIN_PASS` 替换为您在身份认证服务中为 `admin` 用户设置的密码。如果包含特殊字符，您可能需要使用单引号 (') 将密码引用。

3. 使用 `admin` 租户和用户，列出租户以验证 `admin` 租户和用户能够执行只有管理员才能执行的命令行，且身份认证服务包含您在“[创建租户、用户和角色](#)”一节 [32] 中创建的租户：

```
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS  
--os-auth-url http://controller:35357/v2.0 tenant-list
```

id	name	enabled
6f4c1e4cbfef4d5a8a1345882fbca110	admin	True
4aa51bb942be4dd0ac0555d7591f80a6	demo	True
6b69202e1bf846a4ae50d65bc4789122	service	True



注意

由于 OpenStack 是动态生成 ID 的，您看到的值会与这个示例的输出不同。

4. 使用 `admin` 租户和用户，列出用户以验证身份认证服务包含您在“[创建租户、用户和角色](#)”一节 [32] 中创建的用户：

```
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS
--os-auth-url http://controller:35357/v2.0 user-list
```

id	name	enabled	email
ea8c352d253443118041c9c8b8416040	admin	True	admin@example.com
7004dfa0dda84d63aef81cf7f100af01	demo	True	demo@example.com

5. 使用 `admin` 租户和用户，列出角色以验证 身份认证服务包含您在 “创建租户、用户和角色” 一节 [32] 中创建的角色：

```
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS
--os-auth-url http://controller:35357/v2.0 role-list
```

id	name
9fe2ff9ee4384b1894a90878d3e92bab	_member_
bff3a6083b714fa29c9344bf8930d199	admin

6. 使用 demo 租户和用户，需要一个认证的令牌：

```
$ keystone --os-tenant-name demo --os-username demo --os-password DEMO_PASS
--os-auth-url http://controller:35357/v2.0 token-get
```

Property	Value
expires	2014-10-10T12:51:33Z
id	1b87ceae9e08411ba4a16e4dada04802
tenant_id	4aa51bb942be4dd0ac0555d7591f80a6
user_id	7004dfa0dda84d63aef81cf7f100af01

将其中的 DEMO PASS 替换为您在身份认证服务中为 demo 用户所设置的密码。

- 使用 `demo` 租户和用户，尝试列出用户以验证您不能执行只有管理员才能执行的命令行：

```
$ keystone --os-tenant-name demo --os-username demo --os-password DEMO_PASS  
--os-auth-url http://controller:35357/v2.0 user-list  
You are not authorized to perform the requested action, admin required. (HTTP 403)
```


第4章 添加镜像服务

目录

OpenStack 镜像服务	39
安装和配置	40
验证操作	43

OpenStack 的镜像服务 (glance) 允许用户发现、注册和恢复虚拟机镜像。它提供了一个 REST API, 允许您查询虚拟机镜像的 metadata 并恢复一个实际的镜像。您可以存储虚拟机镜像通过不同位置的镜像服务使其可用, 就像 OpenStack 对象存储那样从简单的文件系统到对象存储系统。



重要

简单来说，本指南描述了配置镜像服务以使用 file 后台，能够上传并存储在一个托管镜像服务的控制节点的目录中。默认情况下，这个目录是 `/var/lib/glance/images/`。

继续进行之前，确认控制节点的该目录有至少几千兆字节的可用空间。

For information on requirements for other back ends, see [Configuration Reference](#).

OpenStack镜像服务

正如在图 1.1 “概念架构” [2]所展现的那样，OpenStack 镜像服务是基础设施服务(IaaS)的中心，它接收请求磁盘或镜像的API、来自最终用户的或OpenStack 计算服务组件的镜像元数据。它支持在各种仓库类型存放磁盘或服务器镜像，仓库类型包括OpenStack 对象存储。

多种定期的进程运行在OpenStack镜像服务之上以支持缓存。复制服务通过集群确保一致性和可用性，其他的定期的进程包括审计、更新和回收。

OpenStack镜像服务包括下面组件:

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

registry是一个私有的内部服务，这意味着仅为OpenStack镜像服务所使用。不要将之开放给用户。

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。

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -



AFT

RA

- 1

Kilc

O - DRAFT - E

- K

- DRAFT

DRAFT - Kilo

10

- 1

15

- R.

- 1

- Kil

五

- DR

将其中的 GLANCE PASS 替换为您在身份认证服务中为 glance 用户所设置的密码。



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。

- c. 在[glance_store]部分，设置本地文件系统的存储和镜像文件的位置：

```
[glance_store]
...
default_store = file
filesystem store datadir = /var/lib/glance/images/
```

- d. 在 [DEFAULT] 部分，配置 `noop` 消息驱动以禁用消息，因为它们只与可选的 Telemetry 服务有关：

```
[DEFAULT]
...
notification_driver = noop
```

Telemetry 章节提供了一个启用消息机制的镜像服务配置。

- e. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

3. 修改配置文件 `/etc/glance/glance-registry.conf` 并完成以下操作：

- a. 在[database]段，配置数据库访问相关参数：

```
[database]
...
connection = mysql://glance:GLANCE_DBPASS@控制器/glance
```

将其中的 GLANCE_DBPASS 替换为您为镜像服务数据库所设置的密码。

- b. 在[keystone authtoken]和[paste_deploy]部分，配置身份认证服务的访问：

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

[paste_deploy]
...
flavor = keystone
```

将其中的 GLANCE PASS 替换为您在身份认证服务中为 glance 用户所设置的密码。



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。

- c. 在 [DEFAULT] 部分，配置 `noop` 消息驱动以禁用消息，因为它们只与可选的 Telemetry 服务有关：

```
[DEFAULT]
...
notification_driver = noop
```

Telemetry 章节提供了一个启用消息机制的镜像服务配置。

- d. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

4. 写入镜像服务数据库：

```
# su -s /bin/sh -c "glance-manage db sync" glance
```

完成安装

- 启动镜像服务并将其配置为随系统启动：

```
# systemctl enable openstack-glance-api.service openstack-glance-registry.service
# systemctl start openstack-glance-api.service openstack-glance-registry.service
```

验证操作

This section describes how to verify operation of the Image Service using [CirrOS](#), a small Linux image that helps you test your OpenStack deployment.

For more information about how to download and build images, see [OpenStack Virtual Machine Image Guide](#). For information about how to manage images, see the [OpenStack User Guide](#).

1. Create and change into a temporary local directory:

```
$ mkdir /tmp/images
```

- 10

Ki

- # I

第5章 添加 Compute 服务

目录

OpenStack 计算服务	45
安装配置控制节点服务器	48
安装和配置计算节点	51
验证操作	53

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计算服务的认证由和OpenStack认证服务交互来完成，磁盘和服务器镜像由OpenStack镜像服务交互来完成，以及OpenStack仪表盘是为用户和管理员提供的操作界面。镜像的访问通过项目和用户来限制，配额由每个项目来限制(例如，实例数量)。OpenStack计算通过标准的硬件来做横向扩展，下载镜像以启动实例。

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Hypervisor的API来创建和销毁虚拟机实例。例如：

- XenServer/XCP 的 XenAPI
- KVM 或 QEMU 的 libvirt
- VMware 的 VMwareAPI

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。

Mediates interactions between the nova-compute service and the database. It eliminates direct accesses to the cloud database made by the nova-compute service. The nova-conductor module scales horizontally. However, do not deploy it on nodes where the nova-compute service runs. For more information, see [A new Nova service: nova-conductor](#).

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。

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，执行的任務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-xvpncproxy. 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-xvpncproxy 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。

镜像管理(EC2 场景)

nova-objectstore 守护进程	一个S3接口，用于注册镜像，使用的是OpenStack 镜像服务，这个是由euca2ools来支持的，euca2ools使用 S3 语言来和nova-objectstore通话，然后，nova-objectstore将之翻译向镜像服务发出请求。
euca2ools 客户端	A set of command-line interpreter commands for managing cloud resources. Although it is not an OpenStack module, you can configure nova-api to support this EC2 interface. For more information, see the Eucalyptus 3.4 Documentation .

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 A central hub for passing messages between daemons. Usually implemented with [RabbitMQ](#), but can be implemented with an AMQP message queue, such as [Apache Qpid](#) or [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：
 - 可用实例类型
 - 使用中的实例
 - 可用网络

- E

DRAFT

DR

O

K

T

- DRA

01

- K

I

- A1

10 - D

K

- 15

- RA

1

- 10

- K

Kilo - DRAFT

15

- RA


```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将 RABBIT_PASS 替换为 RabbitMQ 服务中 guest 用户的密码。

- c. 在[DEFAULT]和[keystone_authtoken]部分，配置身份认证服务的访问：

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

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

将其中的 NOVA_PASS 替换为您之前在身份认证服务中为 nova 用户设置的密码。



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。

- d. 在 [DEFAULT] 部分，配置 `my_ip` 选项以使用控制节点上的管理网络接口的 IP 地址：

```
[DEFAULT]
...
my_ip = 10.0.0.11
```

- e. 在 [DEFAULT] 部分，配置 VNC 代理以使用控制节点的管理网络接口的 IP 地址。

```
[DEFAULT]
...
vncserver_listen = 10.0.0.11
vncserver_proxyclient_address = 10.0.0.11
```

- f. 在[glance]部分，配置镜像服务的位置：

```
[glance]
...
host = 控制器
```

- g. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

- ### 3. 同步Compute 数据库：

```
# su -s /bin/sh -c "nova-manage db sync" nova
```

完成安装

- 启动 Compute 服务并将其设置为随系统启动：

```
# systemctl enable openstack-nova-api.service openstack-nova-cert.service  
openstack-nova-consoleauth.service openstack-nova-scheduler.service  
openstack-nova-conductor.service openstack-nova-novncproxy.service  
# systemctl start openstack-nova-api.service openstack-nova-cert.service  
openstack-nova-consoleauth.service openstack-nova-scheduler.service  
openstack-nova-conductor.service openstack-nova-novncproxy.service
```

安装和配置计算节点

这个部分描述如何在一个计算节点上安装和配置 Compute 服务。这个服务支持一些 hypervisors 来部署实例 或 虚拟机。简单来说，这个配置使用在计算节点上扩展 KVM 的 QEMU hypervisor，支持虚拟机的硬件加速。在旧的硬件上，这个配置使用通用的 QEMU hypervisor。您可以根据这些说明进行轻微的修改，以横向扩展环境的额外计算节点。



注意

这个部分假设您是根据本指南中的指示一步一步配置的第一台计算节点。如果您像配置额外的计算节点，以与[架构样例](#)部分类似的方式准备好计算节点，使用与您环境相同的网络服务。对于两者的网络服务，依据 [NTP 配置](#)和 [OpenStack packages](#) 的指示进行。对于 OpenStack 网络 (neutron)，也可以依据 [OpenStack Networking compute node](#) 的指示进行。对于传统网络 (nova-network)，也可以依据 [legacy networking compute node](#) 的指示进行。每个额外的计算节点都需要唯一的 IP 地址。

安装和配置 Compute hypervisor 组件

1. 安装软件包：

```
# yum install openstack-nova-compute sysfsutils
```

2. 修改配置文件 `/etc/nova/nova.conf` 并完成如下操作：

- a. 在[DEFAULT]段，对 RabbitMQ 消息代理相关参数进行配置：

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将 RABBIT_PASS 替换为 RabbitMQ 服务中 guest 用户的密码。

- b. 在[DEFAULT]和[keystone authtoken]部分，配置身份认证服务的访问：

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

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

将其中的 NOVA_PASS 替换为您之前在身份认证服务中为 nova 用户设置的密码。



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。

- c. 在[DEFAULT]部分，设置my_ip选项：

```
[DEFAULT]
...
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
```

将其中的 `MANAGEMENT_INTERFACE_IP_ADDRESS` 替换为计算节点上的管理网络接口的 IP 地址，典型的例子是[架构样例](#)中第一台节点的 `10.0.0.31` 地址。

- d. 在 [DEFAULT] 部分，启用并配置远程控制台的访问：

```
[DEFAULT]
...
vnc_enabled = True
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = MANAGEMENT_INTERFACE_IP_ADDRESS
novncproxy base url = http://控制器:6080/vnc_auto.html
```

服务器组件监听所有的 IP 地址，而代理组件仅仅监听计算节点管理网络接口的 IP 地址。基本的 URL 指示您可以使用 web 浏览器访问位于该计算节点上实例的远程控制台的位置。

将其中的 `MANAGEMENT_INTERFACE_IP_ADDRESS` 替换为计算节点上的管理网络接口的 IP 地址，典型的例子是[架构样例](#)中第一台节点的 `10.0.0.31` 地址。



注意

如果 web 浏览器访问位于一台无法解析 controller 主机名的主机上的控制台，您必须将 controller 替换为控制节点管理网络接口的 IP 地址。

- e. 在[glance]部分，配置镜像服务的位置：

```
[glance]
...
host = 控制器
```

- f. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

完成安装

1. 确定您的计算节点是否支持虚拟机的硬件加速。

```
$ egrep -c '(vmx|svm)' /proc/cpuinfo
```

如果这个命令返回 1 或更大的值，说明您的计算节点支持硬件加速，一般不需要进行额外的配置。

如果这个命令返回的是 0，说明您的计算节点不支持硬件加速，您必须设置 libvirt 使用 QEMU 而不能使用 KVM。

- 根据以下内容修改 `/etc/nova/nova.conf` 的 `[libvirt]` 部分：

```
[libvirt]
...
virt type = qemu
```

2. 启动计算服务及其依赖，并将其配置为随系统自动启动：

```
# systemctl enable libvirtd.service openstack-nova-compute.service
# systemctl start libvirtd.service openstack-nova-compute.service
```

验证操作

This section describes how to verify operation of the Compute service.



注意

在控制节点上执行这些命令。

1. 导入 admin 身份凭证以执行管理员用户专有的命令：

```
$ source admin-openrc.sh
```

2. 列出服务组件以验证是否每个进程都成功启动：

```
$ nova service-list
```

	Id	Binary	Host	Zone	Status	State	Updated_at	Disabled Reason
1		nova-conductor	controller	internal	enabled	up	2014-09-16T23:54:02.000000	-
2		nova-consoleauth	controller	internal	enabled	up	2014-09-16T23:54:04.000000	-
3		nova-scheduler	controller	internal	enabled	up	2014-09-16T23:54:07.000000	-
4		nova-cert	controller	internal	enabled	up	2014-09-16T23:54:00.000000	-
5		nova-compute	compute1	nova	enabled	up	2014-09-16T23:54:06.000000	-



注意

This output should indicate four components enabled on the controller node one component enabled on the compute node.

3. List images in the Image Service catalog to verify connectivity with the Identity service and Image Service:

```
$ nova image-list
```

ID	Name	Status	Server
acafc7c0-40aa-4026-9673-b879898e1fc2	cirros-0.3.3-x86_64	ACTIVE	

任何给定的Networking设置至少有一个外部网络。不像其他的网络，外部网络不仅仅是一个虚拟定义的网络。相反，它代表了一种OpenStack安装之外的能从物理的，外部的网络访问的视图。外部网络上的IP地址能被任何物理接入外面网络的人所访问。因为外部网络仅仅代表了进入外面网络的一个视图，网络上的DHCP是关闭的。

如果外网需要访问虚拟机，或者相反，网络中的路由器就是必须要使用的。每个路由器配有一个网关，可以连接到网络和接口，这些接口又连接着子网。如同实体路由器一样，子网中的机器可以访问连接到同一个路由器的子网中的其它机器，机器可以通过该路由器的网关访问外网。

Networking也支持安全组。安全组允许管理员分组定义防火墙规则。一个虚拟机可以属于一个或多个安全组，Networking针对这个虚拟机，应用这些安全组里的规则来阻塞或者开启端口，端口范围，或通讯类型。

安装配置控制节点服务器

配置 OpenStack 网络 (neutron) 服务之前，您必须创建一个数据库、服务证书和 API 端点。

- ```
GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'localhost'
 IDENTIFIED BY 'NEUTRON_DBPASS';
GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@%'
 IDENTIFIED BY 'NEUTRON_DBPASS';
```

56



- d. 退出数据库客户端。
2. 导入 admin 身份凭证以执行管理员用户专有的命令：

```
$ source admin-openrc.sh
```

3. 创建服务证书，完成如下步骤：

- a. 创建 neutron 用户：

```
$ keystone user-create --name neutron --pass NEUTRON_PASS
```

| Property | Value                            |
|----------|----------------------------------|
| email    |                                  |
| enabled  | True                             |
| id       | 7fd67878dcd04d0393469ef825a7e005 |
| name     | neutron                          |
| username | neutron                          |

将其中的 NEUTRON PASS 替换为一个合适的密码。

- b. 为 neutron 用户添加 admin 角色：

```
$ keystone user-role-add --user neutron --tenant service --role admin
```



## 注意

这个命令执行后没有输出。

- c. 创建 neutron 服务实体：

```
$ keystone service-create --name neutron --type network
--description "OpenStack Networking"
```

| Property    | Value                            |
|-------------|----------------------------------|
| description | OpenStack Networking             |
| enabled     | True                             |
| id          | 6369ddaf99a447f3a0d41dac5e342161 |
| name        | neutron                          |
| type        | network                          |

- #### 4. 创建网络服务的 API 端点：

```
$ keystone endpoint-create
--service-id $(keystone service-list | awk '/ network / {print $2}')
--publicurl http://控制器:9696
--adminurl http://控制器:9696
--internalurl http://控制器:9696
--region regionOne
```

| Property | Value                            |
|----------|----------------------------------|
| adminurl | http://controller:9696           |
| id       | fa18b41938a94bf6b35e2c152063ee21 |

|             |                                  |
|-------------|----------------------------------|
| internalurl | http://controller:9696           |
| publicurl   | http://controller:9696           |
| region      | regionOne                        |
| service_id  | 6369ddaf99a447f3a0d41dac5e342161 |

- # yum install openstack-neutron openstack-neutron-ml2 python-neutronclient which

Networking 服务器组件的配置包括数据库、认证机制、消息 broker、拓扑变化通知和插件。

- 修改配置文件 `/etc/neutron/neutron.conf` 并完成以下操作：
  - a. 在 `[database]` 段，配置数据库访问相关参数：

```
[database]
...
connection = mysql://neutron:NEUTRON_DBPASS@控制器/neutron
```

将其中的 NEUTRON DBPASS 替换为您为数据库所设置的密码。

- b. 在[DEFAULT]段，对 RabbitMQ 消息代理相关参数进行配置：

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将 RABBIT\_PASS 替换为 RabbitMQ 服务中 guest 用户的密码。

- c. 在 [DEFAULT] 和 [keystone authtoken] 部分，配置身份认证服务的访问：

```
[DEFAULT]
...
auth_strategy = keystone
[keystone_authtoken]
...
auth_uri = http://控制器:5000/v2.0
identity_uri = http://控制器:35357
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

将其中的 NEUTRON\_PASS 替换为您在身份认证服务中为 neutron 用户所设置的密码。



## 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。

- DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo

AFT - Kilo - DRAFT - Kilo

- Kilo - D

DRAFT - Kilo - DRAFT - Kilo - DRAFT -

- Kilo - DR.



DRAFT

DRAFT - Ki

T - Kilo

- RAFT

Kilo -

## RAFT

Kilo -

- Kilo

```
[ml2]
...
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = openvswitch
```



警告

一旦您配置好了 ML2 插件，要注意如果禁用一个网络类型，然后再重新启用它，会导致数据库的不一致问题。

- b. 在[ml2 type gre]部分，配置隧道标识符(id)的范围：

```
[ml2_type_gre]
...
tunnel id ranges = 1:1000
```

- c. 在[securitygroup]部分，启用安全组，启用 ipset 并配置 OVS iptables 防火墙驱动：

```
[securitygroup]
...
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

## 配置 Compute 以使用 Networking

默认情况下，发行版的包会配置 Compute 使用传统网络。您必需重新配置 Compute 来通过 Networking 来管理网络。

- 修改控制节点上的配置文件 `/etc/nova/nova.conf`，并完成以下操作：

- a. 在[DEFAULT]部分，配置APIs和drivers：

```
[DEFAULT]
...
network_api_class = nova.network.neutronv2.api.API
security_group_api = neutron
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSSInterfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
```



## 注意

默认情况下，Compute 使用内部的防火墙服务。由于 Networking 包含了一个防火墙服务，您必须使用 `nova.virt.firewall.NoopFirewallDriver` 防火墙驱动来禁用 Compute 的防火墙服务。

- b. 在[neutron]部分，配置访问的参数：

```
[neutron]
...
url = http://控制器:9696
auth_strategy = keystone
admin_auth_url = http://控制器:35357/v2.0
admin_tenant_name = service
admin_username = neutron
admin_password = NEUTRON PASS
```

将其中的 NEUTRON\_PASS 替换为您在身份认证服务中为 neutron 用户所设置的密码。

## 完成安装

1. Networking 服务初始化脚本需要一个象征性的链接将 `/etc/neutron/plugin.ini` 指向 ML2 插件的配置文件 `/etc/neutron/plugins/ml2/ml2_conf.ini`。如果这个象征性的链接不存在，请用以下命令创建它：

```
ln -s /etc/neutron/plugins/ml2/ml2_conf.ini /etc/neutron/plugin.ini
```

- ## 2. 同步数据库：

```
su -s /bin/sh -c "neutron-db-manage --config-file /etc/neutron/neutron.conf
--config-file /etc/neutron/plugins/ml2/ml2_conf.ini upgrade jun0" neutron
```



## 注意

数据库的同步发生在 Networking 之后，因为脚本需要完成服务器和插件的配置文件。

- ### 3. 重启 Compute 服务：

```
systemctl restart openstack-nova-api.service openstack-nova-scheduler.service
openstack-nova-conductor.service
```

4. 启动 Networking 服务并将其配置为随系统启动：

```
systemctl enable neutron-server.service
systemctl start neutron-server.service
```

## 验证操作



## 注意

在控制节点上执行这些命令。

1. 导入 admin 身份凭证以执行管理员用户专有的命令：

```
$ source admin-openrc.sh
```

2. 列出加载的扩展，以验证是否成功启动了一个 neutron-server 进程：

| \$ neutron ext-list |      |
|---------------------|------|
| alias               | name |

|                       |                                               |  |
|-----------------------|-----------------------------------------------|--|
| security-group        | security-group                                |  |
| l3_agent_scheduler    | L3 Agent Scheduler                            |  |
| ext-gw-mode           | Neutron L3 Configurable external gateway mode |  |
| binding               | Port Binding                                  |  |
| provider              | Provider Network                              |  |
| agent                 | agent                                         |  |
| quotas                | Quota management support                      |  |
| dhcp_agent_scheduler  | DHCP Agent Scheduler                          |  |
| l3-ha                 | HA Router extension                           |  |
| multi-provider        | Multi Provider Network                        |  |
| external-net          | Neutron external network                      |  |
| router                | Neutron L3 Router                             |  |
| allowed-address-pairs | Allowed Address Pairs                         |  |
| extraroute            | Neutron Extra Route                           |  |
| extra_dhcp_opt        | Neutron Extra DHCP opts                       |  |
| dvr                   | Distributed Virtual Router                    |  |

## 安装和配置网络节点

网络节点主要为虚拟网络处理内部和外部路由及 DHCP 服务。

## 配置前的准备

在安装和配置 OpenStack 网络之前，您必须配置某些内核网络参数。

1. 修改配置文件 `/etc/sysctl.conf` 以将下列参数包含其中：

```
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

2. 使修改生效：

```
sysctl -p
```

## 安装网络组件

- `# yum install openstack-neutron openstack-neutron-ml2 openstack-neutron-openvswitch`

## 配置网络的通用组件

网络通用组件的配置包括认证机制、消息代理和插件。

- 修改配置文件 `/etc/neutron/neutron.conf` 并完成以下操作：
  - a. 在 `[database]` 部分，注释所有 `connection` 选项，因为网络节点不会直接访问数据库。
  - b. 在 `[DEFAULT]` 段，对 RabbitMQ 消息代理相关参数进行配置：

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将 RABBIT\_PASS 替换为 RabbitMQ 服务中 guest 用户的密码。

- c. 在 [DEFAULT] 和 [keystone authtoken] 部分，配置身份认证服务的访问：

```
[DEFAULT]
...
auth_strategy = keystone
[keystone_authtoken]
...
auth_uri = http://控制器:5000/v2.0
identity_uri = http://控制器:35357
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON PASS
```

将其中的 NEUTRON\_PASS 替换为您在身份认证服务中为 neutron 用户所设置的密码。



## 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。

- d. 在 [DEFAULT] 部分，启用 Modular Layer 2 (ML2) 插件、router 服务和 overlapping IP 地址：

```
[DEFAULT]
...
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

- e. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

## 配置 Modular Layer 2 (ML2) 插件

ML2 插件使用 Open vSwitch (OVS) 机制 (代理) 来为实例构建虚拟网络框架。

- 修改配置文件 `/etc/neutron/plugins/ml2/ml2_conf.ini` 并完成以下操作：
  - a. 在 `[ml2]` 部分，启用 `flat` 和 `generic routing encapsulation (GRE)` 网络类型驱动、`GRE 租户网络`和 `OVS 机制驱动`：

```
[ml2]
...
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = openvswitch
```

- b. 在[ml2 type flat]部分，配置外部供应商的网络：

```
[ml2_type_flat]
...
flat_networks = external
```

- c. 在[ml2\_type\_gre]部分，配置隧道标识符(id)的范围：

```
[ml2_type_gre]
...
tunnel_id_ranges = 1:1000
```

- d. 在 [securitygroup] 部分，启用安全组，启用 ipset 并配置 OVS iptables 防火墙驱动：

```
[securitygroup]
...
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

- e. 在 [ovs] 部分，启用 tunnels，配置本地 tunnel 端点，并映射外部供应商网络到 br-ex 外部网络桥接上：

```
[ovs]
...
local_ip = INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS
enable_tunneling = True
bridge_mappings = external:br-ex
```

将其中的 `INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS` 替换为网络节点上实例隧道网络接口的 IP 地址。

- f. 在[agent]部分，启用 GRE 隧道：

```
[agent]
...
tunnel_types = gre
```

## 配置 Layer-3 (L3) 代理

Layer-3 (L3) 代理为虚拟网络提供路由服务。

- 修改配置文件 `/etc/neutron/l3_agent.ini` 并完成以下操作：
  - a. 在 `[DEFAULT]` 部分，配置驱动，启用 `network namespaces`，配置外部网络桥接并启用删除废弃的路由命名空间：

```
[DEFAULT]
...
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True
external_network_bridge = br-ex
router_delete_namespaces = True
```

- b. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：



```
[DEFAULT]
...
verbose = True
```

## 配置 DHCP 代理

DHCP 代理为实例提供 DHCP 服务。

1. 修改配置文件 `/etc/neutron/dhcp_agent.ini` 并完成以下操作：
  - a. 在 `[DEFAULT]` 部分，配置驱动，启用命名空间并启用删除废弃 DHCP 命名空间的功能：

```
[DEFAULT]
...
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
use_namespaces = True
dhcp_delete_namespaces = True
```

- b. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

2. (可选)

类似于GRE的隧道协议包含有额外的数据包头，这些数据包头增加了开销，减少了有效内容或是用户数据的可用空间。在不了解虚拟网络架构的情况下，实例尝试用默认的以太网maximum transmission unit (MTU)1500字节来发送数据包。Internet protocol (IP)网络利用 path MTU discovery (PMTUD) 机制来探测和调整数据包的大小。但是有些操作系统或者是网络阻塞、缺乏对PMTUD的支持等原因会造成性能下降或是连接错误。

理想情况下，你可以通过在包含有租户虚拟网络的物理网络上开启jumbo frames来避免这些问题。巨型帧(Jumbo frames)支持最大接近9000字节的MTU，它抵消了虚拟网络上GRE开销的影响。但是，很多网络设备缺乏对于巨型帧的支持，Openstack的管理员也经常缺乏对网络架构的控制。考虑到后续的复杂性，也可以选择降低带来GRE开销的实例的MTU大小来避免MTU的问题。确定恰当的MTU值通常需要实验，但是大多数环境下1454字节都可以工作。你可以配置给实例分配IP地址的DHCP服务器来同时调整MTU。



注意

一些云镜像会忽略 DHCP MTU 选项，在这种情况下，您要配置其使用 metadata、一个脚本或其他合适方法。

- a. 修改配置文件 `/etc/neutron/dhcp_agent.ini` 并完成以下操作：
  - 在 `[DEFAULT]` 部分，启用 `dnsmasq` 配置文件：

```
[DEFAULT]
...
dnsmasq_config_file = /etc/neutron/dnsmasq-neutron.conf
```

- b. 创建并修改文件 `/etc/neutron/dnsmasq-neutron.conf`，完成以下操作：

- 启用 DHCP MTU 选项 (26) 并配置为 1454 字节：

```
dhcp-option-force=26,1454
```

- c. 杀死所有存在的 dnsmasq 进程：

```
pkill dnsmasq
```

## 配置 metadata 代理

metadata agent 提供一些配置信息，如实例的凭据。

1. 修改配置文件 `/etc/neutron/metadata_agent.ini` 并完成以下操作：

- a. 在[DEFAULT]部分，配置访问参数：

```
[DEFAULT]
...
auth_url = http://控制器:5000/v2.0
auth_region = regionOne
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

将其中的 NEUTRON\_PASS 替换为您在身份认证服务中为 neutron 用户所设置的密码。

- b. 在 [DEFAULT] 部分，配置 metadata 主机：

```
[DEFAULT]
...
nova metadata ip = 控制器
```



```
[DEFAULT]
...
metadata proxy shared secret = METADATA SECRET
```

- d. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

2. 在控制节点上，修改配置文件 `/etc/nova/nova.conf` 并完成以下操作：

- 在[neutron]部分，启用 metadata 代理并配置 secret：

```
[neutron]
...
service_metadata_proxy = True
metadata_proxy_shared_secret = METADATA_SECRET
```

将其中的 METADATA SECRET 替换为您为 metadata 代理所设置的 secret。

3. 在控制节点上，重启 Compute API 服务：

```
systemctl restart openstack-nova-api.service
```

## 配置 Open vSwitch (OVS) 服务

OVS 服务为实例提供了底层的虚拟网络框架。整合的桥梁 `br-int` 处理内部实例网络在 OVS 中的传输。外部桥梁 `br-ex` 处理外部实例网络在 OVS 中的传输。外部桥梁需要一个在物理外部网络接口上的端口来为实例提供外部网络的访问。本质上，这个端口连接了您环境中虚拟的和物理的外部网络。

1. 启动 OVS 服务并将其配置为随系统启动：

```
systemctl enable openvswitch.service
systemctl start openvswitch.service
```

- ## 2. 添加外部桥接：

```
ovs-vsctl add-br br-ex
```

3. 给一个连接到物理外部网络接口的外部桥接添加端口：

将其中的 `INTERFACE_NAME` 替换为实际的接口名称。例如，`eth2` 或 `ens256`。

```
ovs-vsctl add-port br-ex INTERFACE_NAME
```



## 注意

根据您的网络接口驱动，您可能需要禁用 generic receive offload (GRO) 来实现您实例和外部网络之间的合适吞吐量。

测试环境时，在外部网络接口上暂时地禁用 GRO：

```
ethtool -K INTERFACE NAME gro off
```

## 完成安装

1. Networking 服务初始化脚本需要一个象征性的链接将 `/etc/neutron/plugin.ini` 指向 ML2 插件的配置文件 `/etc/neutron/plugins/ml2/ml2_conf.ini`。如果这个象征性的链接不存在，请用以下命令创建它：

```
ln -s /etc/neutron/plugins/ml2/ml2_conf.ini /etc/neutron/plugin.ini
```

由于一个包的缺陷，Open vSwitch 代理初始化脚本会寻找 Open vSwitch 插件的配置文件，而不是指向 ML2 配置文件的象征性链接 `/etc/neutron/plugin.ini`。执行以下命令来解决这个问题：

```
cp /usr/lib/systemd/system/neutron-openvswitch-agent.service
/usr/lib/systemd/system/neutron-openvswitch-agent.service.orig
sed -i 's,plugins/openvswitch/ovs_neutron_plugin.ini,plugin.ini,g'
/usr/lib/systemd/system/neutron-openvswitch-agent.service
```

2. 启动 Networking 服务并将其配置为随系统启动：

```
systemctl enable neutron-openvswitch-agent.service neutron-l3-agent.service
neutron-dhcp-agent.service neutron-metadata-agent.service
neutron-ovs-cleanup.service
```

```
systemctl start neutron-openvswitch-agent.service neutron-l3-agent.service
neutron-dhcp-agent.service neutron-metadata-agent.service
```



## 注意

请勿直接地启动 neutron-ovs-cleanup 服务。

## 验证操作



## 注意

在控制节点上执行这些命令。

1. 导入 admin 身份凭证以执行管理员用户专有的命令：

```
$ source admin-openrc.sh
```

2. 列出代理以验证启动 neutron 代理是否成功：

## 安装和配置计算节点

计算节点处理实例的连接和安全组。

## 配置前的准备

在安装和配置 OpenStack 网络之前，您必须配置某些内核网络参数。

1. 修改配置文件 `/etc/sysctl.conf` 以将下列参数包含其中：

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

2. 使修改生效：

```
sysctl -p
```

## 安装网络组件

- `# yum install openstack-neutron-ml2 openstack-neutron-openvswitch`

## 配置网络的通用组件

网络通用组件的配置包括认证机制、消息代理和插件。

- 修改配置文件 `/etc/neutron/neutron.conf` 并完成以下操作：
  - a. 在 `[database]` 部分，注释所有 `connection` 选项，因为计算节点不会直接访问数据库。
  - b. 在 `[DEFAULT]` 段，对 RabbitMQ 消息代理相关参数进行配置：

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将 RABBIT\_PASS 替换为 RabbitMQ 服务中 guest 用户的密码。

- c. 在 [DEFAULT] 和 [keystone authtoken] 部分，配置身份认证服务的访问：

```
[DEFAULT]
...
auth_strategy = keystone
[keystone_authtoken]
...
auth_uri = http://控制器:5000/v2.0
identity_uri = http://控制器:35357
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON PASS
```

将其中的 NEUTRON\_PASS 替换为您在身份认证服务中为 neutron 用户所设置的密码。



## 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。

- d. 在 [DEFAULT] 部分，启用 Modular Layer 2 (ML2) 插件、router 服务和 overlapping IP 地址：

```
[DEFAULT]
...
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = True
```

- e. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

## 配置 Modular Layer 2 (ML2) 插件

ML2 插件使用 Open vSwitch (OVS) 机制 (代理) 来为实例构建虚拟网络框架。

- 修改配置文件 `/etc/neutron/plugins/ml2/ml2_conf.ini` 并完成以下操作：

- a. 在 [ml2] 部分，启用 flat 和 generic routing encapsulation (GRE) 网络类型驱动、GRE 租户网络和 OVS 机制驱动：

```
[ml2]
...
type_drivers = flat,gre
tenant_network_types = gre
mechanism_drivers = openvswitch
```

- b. 在 [ml2\_type\_gre] 部分，配置隧道标识符 (id) 的范围：

```
[ml2_type_gre]
...
tunnel_id_ranges = 1:1000
```

- c. 在[securitygroup]部分，启用安全组，启用 ipset 并配置 OVS iptables 防火墙驱动：

```
[securitygroup]
...
enable_security_group = True
enable_ipset = True
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```

- d. 在 [ovs] 部分，启用隧道并配置本地隧道的端点：

```
[ovs]
...
local_ip = INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS
enable_tunneling = True
```

将其中的 `INSTANCE_TUNNELS_INTERFACE_IP_ADDRESS` 替换为计算节点上的实例隧道网络接口的 IP 地址。

- e. 在[agent]部分，启用 GRE 隧道：

```
[agent]
...
tunnel types = gre
```

## 配置 Open vSwitch (OVS) 服务

OVS 服务为实例提供了底层的虚拟网络框架。

- 启动 OVS 服务并将其配置为随系统启动：

```
systemctl enable openvswitch.service
systemctl start openvswitch.service
```

## 配置 Compute 以使用 Networking

默认情况下，发行版的包会配置 Compute 使用传统网络。您必需重新配置 Compute 来通过 Networking 来管理网络。

- 修改配置文件 `/etc/nova/nova.conf` 并完成如下操作：

- a. 在[DEFAULT]部分，配置APIs和drivers：

```
[DEFAULT]
...
network_api_class = nova.network.neutronv2.api.API
security_group_api = neutron
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSIfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
```



## 注意

默认情况下，Compute 使用内部的防火墙服务。由于 Networking 包含了一个防火墙服务，您必须使用 `nova.virt.firewall.NoopFirewallDriver` 防火墙驱动来禁用 Compute 的防火墙服务。

- b. 在[neutron]部分，配置访问的参数：

```
[neutron]
...
url = http://控制器:9696
auth_strategy = keystone
admin_auth_url = http://控制器:35357/v2.0
admin_tenant_name = service
admin_username = neutron
admin_password = NEUTRON_PASS
```

将其中的 `NEUTRON_PASS` 替换为您在身份认证服务中为 `neutron` 用户所设置的密码。

## 完成安装

1. Networking 服务初始化脚本需要一个象征性的链接将 `/etc/neutron/plugin.ini` 指向 ML2 插件的配置文件 `/etc/neutron/plugins/ml2/ml2_conf.ini`。如果这个象征性的链接不存在，请用以下命令创建它：

```
ln -s /etc/neutron/plugins/ml2/ml2_conf.ini /etc/neutron/plugin.ini
```

由于一个包的缺陷，Open vSwitch 代理初始化脚本会寻找 Open vSwitch 插件的配置文件，而不是指向 ML2 配置文件的象征性链接 `/etc/neutron/plugin.ini`。执行以下命令来解决这个问题：

```
cp /usr/lib/systemd/system/neutron-openvswitch-agent.service
/usr/lib/systemd/system/neutron-openvswitch-agent.service.orig
sed -i 's,plugins/openvswitch/ovs_neutron_plugin.ini,plugin.ini,g'
/usr/lib/systemd/system/neutron-openvswitch-agent.service
```

- ## 2. 重启计算服务：

```
systemctl restart openstack-nova-compute.service
```

3. 启动 Open vSwitch (OVS) 代理并将其配置为随系统启动：

```
systemctl enable neutron-openvswitch-agent.service
systemctl start neutron-openvswitch-agent.service
```

## E



E

- E

E

- E

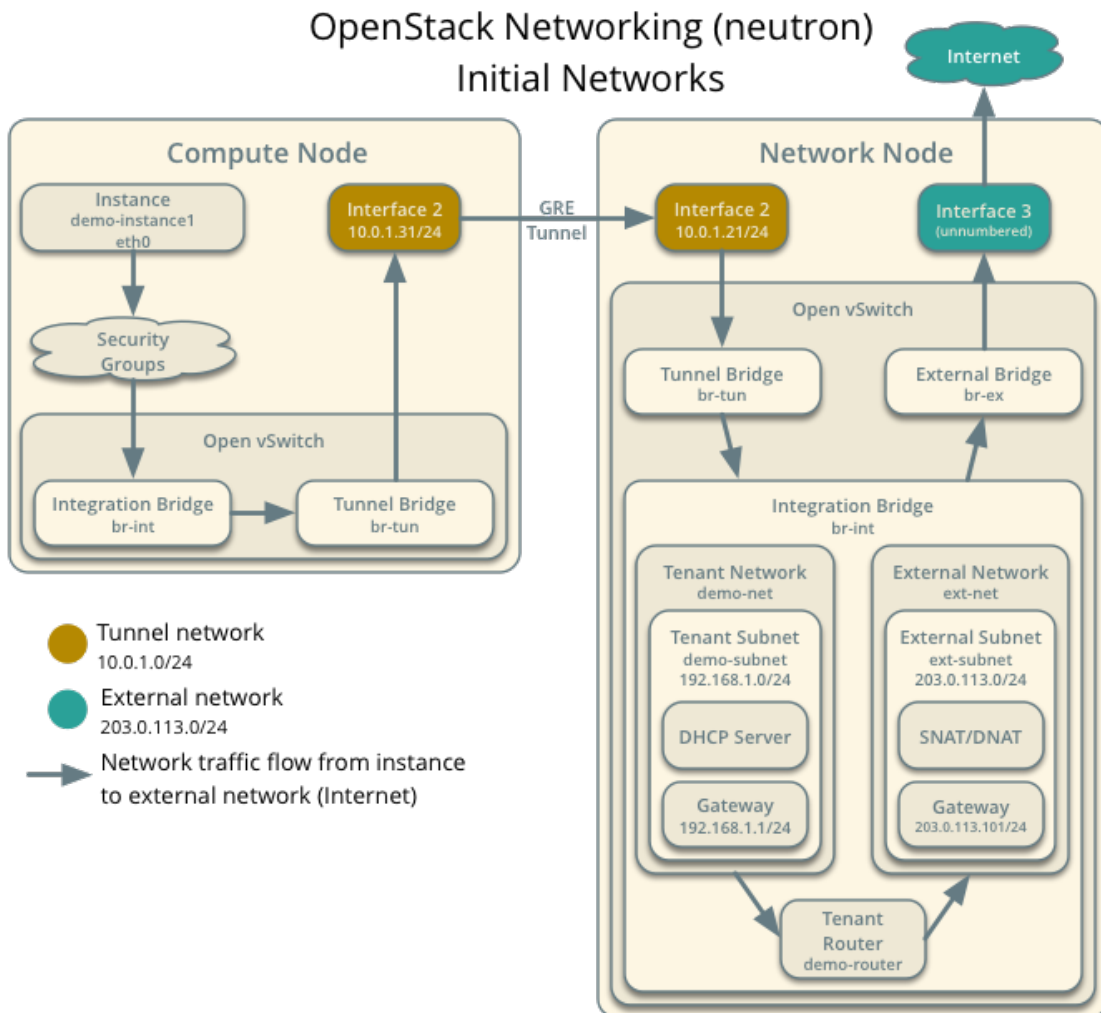
E

## E

E



图 6.1. 初始网络



## 外部网络

外部网络为您的实例分配互联网连接。该网络一般仅允许通过使用网络地址转换 (NAT) 的实例访问 Internet。您可以通过一个浮动 IP 地址和合适的安全组规则来启用 Internet 的访问到个别实例。**admin** 租户拥有这个网络，因为它为多个租户提供了外部网络的访问。



## 注意

在控制节点上执行这些命令。

## 创建外部网络

1. 导入 admin 身份凭证以执行管理员用户专有的命令：

```
$ source admin-openrc.sh
```

- ## 2. 创建网络：

```
$ neutron net-create ext-net --router:external True
--provider:physical_network external --provider:network_type flat
Created a new network:
+-----+-----+
| Field | Value |
+-----+-----+
admin_state_up	True
id	893aebb9-1c1e-48be-8908-6b947f3237b3
name	ext-net
provider:network_type	flat
provider:physical_network	external
provider:segmentation_id	
router:external	True
shared	False
status	ACTIVE
subnets	
tenant_id	54cd044c64d5408b83f843d63624e0d8
+-----+-----+
```

如同一个物理网络，一个虚拟网络需要为其分配一个子网。外部网络通过网络节点上的外部接口，分享与物理网络关联的相同的子网和网关。您需要为路由和浮动IP来指定一个此子网上的独有段，以免与外部网络上的其他设备相冲突。

## 在外部网络上创建子网

- 创建子网：

```
$ neutron subnet-create ext-net --name ext-subnet
--allocation-pool start=FLOATING_IP_START,end=FLOATING_IP_END
--disable-dhcp --gateway EXTERNAL_NETWORK_GATEWAY EXTERNAL_NETWORK_CIDR
```

使用您想用来分配给浮动IP的第一个和最后一个IP地址来替换FLOATING\_IP\_START和FLOATING\_IP\_END。用和物理网络相关联的子网替换EXTERNAL\_NETWORK\_CIDR。用和物理网络相关联的网关替换EXTERNAL\_NETWORK\_GATEWAY 常见的是 ".1" IP 地址。您应该关闭此子网上的DHCP，因为实例不直接连接到外部网络，并且浮动IP需要手工分配。

例如，使用 203.0.113.0/24 帶有浮动IP地址 203.0.113.101 到 203.0.113.200：

```
$ neutron subnet-create ext-net --name ext-subnet
--allocation-pool start=203.0.113.101,end=203.0.113.200
--disable-dhcp --gateway 203.0.113.1 203.0.113.0/24
Created a new subnet:
```

| Field             | Value                                                |
|-------------------|------------------------------------------------------|
| allocation_pools  | { "start": "203.0.113.101", "end": "203.0.113.200" } |
| cidr              | 203.0.113.0/24                                       |
| dns_nameservers   |                                                      |
| enable_dhcp       | False                                                |
| gateway_ip        | 203.0.113.1                                          |
| host_routes       |                                                      |
| id                | 9159f0dc-2b63-41cf-bd7a-289309da1391                 |
| ip_version        | 4                                                    |
| ipv6_address_mode |                                                      |
| ipv6_ra_mode      |                                                      |
| name              | ext-subnet                                           |
| network_id        | 893aebb9-1c1e-48be-8908-6b947f3237b3                 |
| tenant_id         | 54cd044c64d45408b83f843d63624e0d8                    |

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT

Kilo



T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

RA

## DR

- 10

ET



il

I

## D

- ilc

E

ilc

ilc

ilc

|                   |                                      |  |
|-------------------|--------------------------------------|--|
| cidr              | 192.168.1.0/24                       |  |
| dns_nameservers   |                                      |  |
| enable_dhcp       | True                                 |  |
| gateway_ip        | 192.168.1.1                          |  |
| host_routes       |                                      |  |
| id                | 69d38773-794a-4e49-b887-6de6734e792d |  |
| ip_version        | 4                                    |  |
| ipv6_address_mode |                                      |  |
| ipv6_ra_mode      |                                      |  |
| name              | demo-subnet                          |  |
| network_id        | ac108952-6096-4243-adf4-bb6615b3de28 |  |
| tenant_id         | cdef0071a0194d19ac6bb63802dc9bae     |  |

虚拟路由在两个或多个虚拟网络之间传递网络数据。每个路由需要一个或多个接口和/或提供到指定网络连接的网关。在如下部分，您将创建一个路由然后将您的租户网络和外部网络连接到的其上。

在租户网络上创建路由并将外部网络和租户网络附加给它

### 1. 创建路由：

```
$ neutron router-create demo-router
Created a new router:
+-----+-----+
|Field |Value |
+-----+-----+
admin_state_up	True
external_gateway_info	
id	635660ae-a254-4feb-8993-295aa9ec6418
name	demo-router
routes	
status	ACTIVE
tenant_id	cdef0071a0194d19ac6bb63802dc9bae
+-----+-----+
```

## 2. 附加路由给 demo 租户子网：

```
$ neutron router-interface-add demo-router demo-subnet
Added interface bla894fd-ae8-475c-9262-4342afdc1b58 to router demo-router.
```

3. 通过将路由设置为网关来将路由附加给外部网络：

```
$ neutron router-gateway-set demo-router ext-net
Set gateway for router demo-router
```

## 验证连通性

我们建议您在继续进行前验证网络连通性和解决其他任何问题。沿袭外部网络子网使用203.0.113.0/24的例子，租户路由网关应该占用了浮动IP地址范围内的最小IP地址，203.0.113.101。如果您正确的配置了您的外部物理网络和虚拟网络，您应该能够从您的外部物理网络上的任意主机ping这个IP地址。



## 注意

如果您在虚拟机上配置您的OpenStack节点，您必须配置管理程序以允许外部网络上的混杂模式。

## 验证网络的连通性

- Ping 租户路由网关：

```
$ ping -c 4 203.0.113.101
PING 203.0.113.101 (203.0.113.101) 56(84) bytes of data.
64 bytes from 203.0.113.101: icmp_req=1 ttl=64 time=0.619 ms
64 bytes from 203.0.113.101: icmp_req=2 ttl=64 time=0.189 ms
64 bytes from 203.0.113.101: icmp_req=3 ttl=64 time=0.165 ms
64 bytes from 203.0.113.101: icmp_req=4 ttl=64 time=0.216 ms

--- 203.0.113.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.165/0.297/0.619/0.187 ms
```

## 传统联网方式(nova-network)

## 配置控制节点

传统网络主要涉及到计算节点。但是，您必须配置控制节点以使用传统网络。

## 配置传统网络

1. 修改配置文件 `/etc/nova/nova.conf` 并完成如下操作：

- 在[DEFAULT]部分，配置网络和安全组的 API：

```
[DEFAULT]
...
network_api_class = nova.network.api.API
security_group_api = nova
```

- ## 2. 重启 Compute 服务：

```
systemctl restart openstack-nova-api.service openstack-nova-scheduler.service
openstack-nova-conductor.service
```

## 配置计算节点

这个部分覆盖了一个简单的扁平网络的部署，它通过 DHCP 为您的实例提供 IP 地址。如果您的环境包含多个计算节点，**multi-host** 的特性通过在计算节点上扩展网络功能提供冗余。

## 安装传统网络的组件

- `# yum install openstack-nova-network openstack-nova-api`

## 配置传统网络

1. 修改配置文件 `/etc/nova/nova.conf` 并完成如下操作：

- 在 [DEFAULT] 部分，设置网络参数：

[DEFAULT]  
...



## T - Kilo - DF

T - Kilo -

T - Kilo

## T-Ki

11





For details about browsers that support noVNC, see <https://github.com/kanaka/noVNC/blob/master/README.md>, and <https://github.com/kanaka/noVNC/wiki/Browser-support>, respectively.

## 安装和配置

这个部分将描述如何在控制节点上安装和配置仪表板。

Before you proceed, verify that your system meets the requirements in “系统需求” 一节 [80]. Also, the dashboard relies on functional core services including Identity, Image Service, Compute, and either Networking (neutron) or legacy networking (nova-network). Environments with stand-alone services such as Object Storage cannot use the dashboard. For more information, see the [developer documentation](#).

## 安装仪表板组件

- 安装软件包：

```
yum install openstack-dashboard httpd mod_wsgi memcached python-memcached
```

## 配置仪表板

- 修改配置文件 `/etc/openstack-dashboard/local_settings` 并完成以下操作：

- a. 在控制节点上配置仪表板以使用 OpenStack 服务：

OPENSTACK\_HOST = "控制器"

- b. 允许所有主机访问仪表盘：

```
ALLOWED_HOSTS = ['*']
```

- c. 配置 memcached 会话存储服务：

```
CACHES = {
 'default': {
 'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
 'LOCATION': '127.0.0.1:11211',
 }
}
```



## 注意

将其他的会话存储服务配置注释。

- d. 可以选择性地配置时区：

TIME\_ZONE = "TIME\_ZONE"

Replace TIME\_ZONE with an appropriate time zone identifier. For more information, see the [list of time zones](#).

## 完成安装

1. 在 RHEL 和 CentOS 中，配置 SELinux 以允许 web 服务器链接到 OpenStack 服务：

```
setsebool -P httpd_can_network_connect on
```

2. 由于一个包的 bug，仪表板的 CSS 会加载失败。可以执行以下命令来解决这个问题：

```
chown -R apache:apache /usr/share/openstack-dashboard/static
```

For more information, see the [bug report](#).

3. 启动 web 服务器和会话存储服务，并将其配置为随系统启动：

```
systemctl enable httpd.service memcached.service
systemctl start httpd.service memcached.service
```

## 验证操作

这个部分将描述如何进行仪表板的验证操作。

1. 使用 web 浏览器访问仪表板: <http://controller/dashboard>。
2. 使用 admin 或 demo 用户证书进行认证。

## 下一步

现在您的 OpenStack 环境已经包含了仪表板。您可以 [创建实例](#) 或根据下面的章节添加更多的服务到您的环境中。

安装和配置好仪表板后，您可以完成以下任务：

- Customize your dashboard. See section [Customize the dashboard](#) in the [OpenStack Cloud Administrator Guide](#) for information on setting up colors, logos, and site titles.
- Set up session storage. See section [Set up session storage for the dashboard](#) in the [OpenStack Cloud Administrator Guide](#) for information on user session data.

## 第 8 章 添加块设备存储服务

# 目录

|                    |    |
|--------------------|----|
| OpenStack块存储 ..... | 83 |
| 安装配置控制节点服务器 .....  | 83 |
| 安装并配置一个存储节点 .....  | 86 |
| 验证操作 .....         | 89 |
| 下一步 .....          | 91 |

The OpenStack Block Storage service provides block storage devices to guest instances. The method in which the storage is provisioned and consumed is determined by the Block Storage driver, or drivers in the case of a multi-backend configuration. There are a variety of drivers that are available: NAS/SAN, NFS, iSCSI, Ceph, and more. The Block Storage API and scheduler services typically run on the controller nodes. Depending upon the drivers used, the volume service can run on controllers, compute nodes, or standalone storage nodes. For more information, see the [Configuration Reference](#).



注意

本章节省略了备份管理，因为它是基于对象存储服务的。

## 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。                                                                                   |
| 消息队列                 | 在块存储的进程之间路由信息。                                                                                                          |

## 安装配置控制节点服务器

这个部分描述如何在控制节点上安装和配置块设备存储服务，即 cinder。这个服务需要至少一个额外的存储节点，以向实例提供卷。

## 配置前的准备

在您安装和配置块设备存储服务之前，您必须创建数据库、服务证书和 API 入口点。

- DRAFT

- Kilc

- DRA]

T-Ki.

- O - DR

ET - K

Lot - D11

- AET -

- ilo - [

F

F

F

F

- F

F



F

F

- F

F

F

|       |                                  |  |  |
|-------|----------------------------------|--|--|
| id    | 16e038e449c94b40868277f1d801edb5 |  |  |
| name  | cinderv2                         |  |  |
| type  | volumev2                         |  |  |
| ----- |                                  |  |  |

#### 4. 创建块存储服务的 API 入口点：

```
$ keystone endpoint-create
--service-id $(keystone service-list | awk ' / volume2 / {print $2}')
--publicurl http://控制器:8776/v2/(tenant_id)s
--internalurl http://控制器:8776/v2/(tenant_id)s
--adminurl http://控制器:8776/v2/(tenant_id)s
--region regionOne
```

| Property    | Value                                  |
|-------------|----------------------------------------|
| adminurl    | http://controller:8776/v2/(tenant_id)s |
| id          | 097b4a6fc8ba44b10d4822d2d9e076         |
| internalurl | http://controller:8776/v2/(tenant_id)s |
| publicurl   | http://controller:8776/v2/(tenant_id)s |
| region      | regionOne                              |
| service_id  | 16e038e449c94b40868277f1d801edb5       |

## 安装并配置块设备存储服务在控制节点服务器上的组件

1. 安装软件包：

```
yum install openstack-cinder python-cinderclient python-oslo-db
```

2. 编辑 `/etc/cinder/cinder.conf` 并完成下列操作：

a. 在[database]段，配置数据库访问相关参数：

```
[database]
...
connection = mysql://cinder:CINDER_DBPASS@controller/cinder
```

将 CINDER\_DBPASS 替换为之前为块设备服务数据库设定的密码。

b. 在[DEFAULT]段，对 RabbitMQ 消息代理相关参数进行配置：

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将 RABBIT\_PASS 替换为 RabbitMQ 服务中 guest 用户的密码。

c. 在 [DEFAULT] 和 [keystone\_authtoken] 部分，配置身份认证服务的访问：

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

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

将 CINDER\_PASS 替换为在身份认证服务中为 cinder 用户配置的密码。



## 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。

- d. 在 [DEFAULT] 部分，配置 `my_ip` 选项以使用控制节点上的管理网络接口的 IP 地址：

```
[DEFAULT]
...
my_ip = 10.0.0.11
```

- e. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

- ### 3. 初始化块设备服务的数据库：

```
su -s /bin/sh -c "cinder-manage db sync" cinder
```

## 完成安装

- 启动块设备存储服务，并将其配置为开机自启：

```
systemctl enable openstack-cinder-api.service openstack-cinder-scheduler.service
systemctl start openstack-cinder-api.service openstack-cinder-scheduler.service
```

## 安装并配置一个存储节点

这个部分将描述如何安装和配置存储节点，以使用块设备存储服务。简单来说，这个配置将一个存储节点关联到一个空的本地块存储设备 `/dev/sdb` 上，这个设备包含了一个合适的分区表，其中一个分区 `/dev/sdb1` 占用了整个设备。该服务使用 LVM 在这个设备上提供了逻辑卷，并通过 iSCSI 传输将这些逻辑卷提供给实例使用。您可以根据这些小修改的指导，通过额外的存储节点来增加您环境的规模。

## 配置前的准备

在您安装和配置卷服务之前，您必须先配置存储节点。类似于控制节点，存储节点包含一个管理网络接口上的网络。存储节点也需要一个适合您环境大小的空的块存储设备。要了解更多信息，请阅读[第 2 章 基本环境 \[11\]](#)。

- ### 1. 配置管理网络接口：

IP地址: 10.0.0.41

子网掩码: 255.255.255.0 (or /24)

默认网关: 10.0.0.1

2. 设置节点的主机名为block1.

3. 从控制节点上复制 `/etc/hosts` 文件的内容到存储节点上，并将下列内容添加到其中：

```
block1
10.0.0.41 block1
```

也要将这些内容添加到您环境中所有其他主机的 `/etc/hosts` 文件中。

4. 使用“其它节点服务器”一节[24]中的指示安装和配置NTP。

- ## 5. 安装 LVM 包：

```
yum install lvm2
```



## 注意

一些发行版默认包含了LVM。

6. 启动LVM的metadata服务并且设置该服务随系统启动：

```
systemctl enable lvm2-lvmetad.service
systemctl start lvm2-lvmetad.service
```

7. 创建LVM物理卷/dev/sdb1：

```
pvcreate /dev/sdb1
Physical volume "/dev/sdb1" successfully created
```



## 注意

如果您的系统使用了一个不同的设备名称，请对这些步骤进行相应的调整。

- ## 8. 创建LVM卷组cinder-volumes：

```
vgcreate cinder-volumes /dev/sdb1
Volume group "cinder-volumes" successfully created
```

块存储服务会在这个卷组中创建逻辑卷。

- 只有实例能够访问块存储卷。但是，底层的操作系统管理这些设备，关联到卷上。默认情况下，LVM 卷的扫描工具会扫描包含卷的块存储设备的 `/dev` 目录。如果租户在卷上使用了 LVM，扫描工具会检查这些卷并尝试缓存它们，这会在底层系统和租户卷上产生各种各样的问题。您必需重新配置 LVM，仅仅扫描包含了 `cinder-volume` 卷组的设备。修改配置文件 `/etc/lvm/lvm.conf` 并完成以下操作：

- 在 `devices` 部分，添加一个过滤器，接受 `/dev/sdb` 设备，拒绝其他所有设备：

```
devices {
...
filter = ["a/sdb/", "r/.*/"]
```

每个过滤器序列中的元素都以 **a** 开头，即为 **accept**，或以 **r** 开头，即为 **reject**，并好括一些设备名称的表示规则。您可以使用 **vgs -vvvv** 命令来测试过滤器。



警告

如果您的存储节点在操作系统磁盘上使用了 LVM，您还必需添加相关的设备到过滤器中。例如，如果 `/dev/sda` 设备包含在操作系统中：

```
filter = ["a/sda/", "a/sdb/", "r/.*/"]
```

类似地，如果您的计算节点在操作系统磁盘上使用了 LVM，您也必需修改这些节点上 `/etc/lvm/lvm.conf` 文件中的过滤器，将操作系统磁盘包含到过滤器中。例如，如果 `/dev/sda` 设备包含了操作系统：

```
filter = ["a/sda/", "r/*.*/"]
```

## 安装并配置块存储卷组件

1. 安装软件包：

```
yum install openstack-cinder targetcli python-oslo-db MySQL-python
```

2. 编辑 `/etc/cinder/cinder.conf` 并完成下列操作：

- a. 在[database]段，配置数据库访问相关参数：

```
[database]
...
connection = mysql://cinder:CINDER_DBPASS@控制器/cinder
```

将 CINDER\_DBPASS 替换为之前为块设备服务数据库设定的密码。

- b. 在 [DEFAULT] 段，对 RabbitMQ 消息代理相关参数进行配置：

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将RABBIT\_PASS替换为RabbitMQ中guest用户的密码。

- c. 在 [DEFAULT] 和 [keystone\_authtoken] 部分，配置身份认证服务的访问：



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

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

将 CINDER\_PASS 替换为在身份认证服务中为 cinder 用户配置的密码。



## 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。

- d. 在[DEFAULT]部分，设置my\_ip选项：

```
[DEFAULT]
...
my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
```

将其中的 `MANAGEMENT_INTERFACE_IP_ADDRESS` 替换为您存储节点的管理网络接口的 IP 地址，正如[样例架构](#)中的第一个节点的 `10.0.0.41` 地址。

- e. 在[DEFAULT]部分，配置镜像服务的位置：

```
[DEFAULT]
...
glance host = 控制器
```

- f. 在[DEFAULT]部分，配置块存储使用loadmiSCSI服务：

```
[DEFAULT]
...
iscsi_helper = loadm
```

- g. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

## 完成安装

- 启动块存储卷服务及其依赖的服务，并将其配置为随系统启动：

```
systemctl enable openstack-cinder-volume.service target.service
systemctl start openstack-cinder-volume.service target.service
```

## 验证操作

这个部分描述如何通过创建一个卷来验证块设备存储服务操作。

For more information about how to manage volumes, see the [OpenStack User Guide](#).



## 注意

在控制节点上执行这些命令。

1. 导入 admin 身份凭证以执行管理员用户专有的命令：

```
$ source admin-openrc.sh
```

2. 列出服务组件以验证是否每个进程都成功启动：

```
$ cinder service-list
```

| Binary           | Host       | Zone | Status  | State | Updated_at                 | Disabled Reason |
|------------------|------------|------|---------|-------|----------------------------|-----------------|
| cinder-scheduler | controller | nova | enabled | up    | 2014-10-18T01:30:54.000000 | None            |
| cinder-volume    | block1     | nova | enabled | up    | 2014-10-18T01:30:57.000000 | None            |

3. Source demo 租户凭证，以非管理员租户的身份来执行下列步骤：

```
$ source demo-openrc.sh
```

4. 创建一个 1 GB 的卷：

```
$ cinder create --display-name demo-volume1 1
```

| Property            | Value                                |
|---------------------|--------------------------------------|
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| created_at          | 2014-10-14T23:11:50.870239           |
| display_description | None                                 |
| display_name        | demo-volume1                         |
| encrypted           | False                                |
| id                  | 158bea89-07db-4ac2-8115-66c0d6a4bb48 |
| metadata            | {}                                   |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| volume_type         | None                                 |

- ### 5. 验证卷的创建和可用性：

```
$ cinder list
```

| ID                                   | Status    | Display Name | Size | Volume Type | Bootable | Attached to |
|--------------------------------------|-----------|--------------|------|-------------|----------|-------------|
| 158bea89-07db-4ac2-8115-66c0d6a4bb48 | available | demo-volume1 | 1    | None        | false    |             |

如果状态没有显示为 `available`，请检查控制节点和卷节点中 `/var/log/cinder` 目录下的日志，获取更多的信息。



T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

## 第 9 章 添加对象存储

# 目录

|                     |     |
|---------------------|-----|
| OpenStack对象存储 ..... | 92  |
| 安装并配置控制器节点 .....    | 92  |
| 安装和配置存储节点 .....     | 95  |
| 创建初始化的 rings .....  | 99  |
| 完成安装 .....          | 103 |
| 验证操作 .....          | 104 |
| 下一步 .....           | 104 |

OpenStack 对象存储服务 (swift) 通过 REST API 来提供对象存储和检索。在部署对象存储前，您的环境必须至少包含认证服务 (keystone)。

## 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)    | 在存储节点管理实际的对象，诸如文件。                                                                                         |
| 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。                                                                                      |

## 安装并配置控制器节点

This section describes how to install and configure the proxy service that handles requests for the account, container, and object services operating on the storage nodes. For simplicity, this guide installs and configures the proxy service on the controller node. However, you can run the proxy service on any node with network connectivity to the storage nodes. Additionally, you can install and configure the proxy service on multiple

RAFT

## ilo

AFT -



## ET - F

O - DEF

- ET - K

- O - DR

T-Ki

- DRA

- Kil

```
--internalurl 'http://控制器:8080/v1/AUTH_%(tenant_id)s'
--adminurl http://控制器:8080
--region regionOne
```

| Property    | Value                                        |
|-------------|----------------------------------------------|
| adminurl    | http://controller:8080/                      |
| id          | af534fb8b7ff40a6acf725437c586ebe             |
| internalurl | http://controller:8080/v1/AUTH_%(tenant_id)s |
| publicurl   | http://controller:8080/v1/AUTH_%(tenant_id)s |
| region      | regionOne                                    |
| service_id  | 75ef509da2c340499d454ae96a2c5c34             |

## 安装并配置controller节点组件

1. 安装软件包：



## 注意

完整的 OpenStack 环境已经包含了这些包的其中一部分。

```
yum install openstack-swift-proxy python-swiftclient python-keystone-auth-token
python-keystonemiddleware memcached
```

2. 从对象存储的仓库源中获取代理服务的配置文件：

```
curl -o /etc/swift/proxy-server.conf
https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/proxy-server.conf-sample
```

3. 修改配置文件 `/etc/swift/proxy-server.conf` 并完成以下步骤：

- a. 在[DEFAULT]部分，配置绑定端口，用户，和配置目录：

```
[DEFAULT]
...
bind_port = 8080
user = swift
swift_dir = /etc/swift
```

- b. 在 `[pipeline:main]` 部分，启用合适的模块：

```
[pipeline:main]
pipeline = authtoken cache healthcheck keystoneauth proxy-logging proxy-server
```



## 注意

For more information on other modules that enable additional features, see the [Deployment Guide](#).

- c. 在[app:proxy-server]部分中，启动账户管理：

```
[app:proxy-server]
...
allow_account_management = true
account autocreate = true
```

- d. 在[filter:keystoneauth]部分，配置操作者角色：

```
[filter:keystoneauth]
use = egg:swift#keystoneauth
...
operator_roles = admin, member
```



## 注意

你可能需要反注释这部分.

- e. 在[filter:authtoken]部分，配置身份认证服务的访问：

```
[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
admin_tenant_name = service
admin_user = swift
admin_password = SWIFT_PASS
delay_auth_decision = true
```

将 SWIFT\_PASS 替换为在身份认证服务中为 swift 用户配置的密码。



注意

你可能需要反注释这部分.



## 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。

- f. 在[filter:cache]部分，配置memcached的位置：

```
[filter:cache]
...
memcache servers = 127.0.0.1:11211
```

## 安装和配置存储节点

This section describes how to install and configure storage nodes that operate the account, container, and object services. For simplicity, this configuration references two storage nodes, each containing two empty local block storage devices. Each of the devices, `/dev/sdb` and `/dev/sdc`, must contain a suitable partition table with one partition occupying the entire device. Although the Object Storage service supports any file system with extended attributes (xattr), testing and benchmarking indicate the best performance and reliability on XFS. For more information on horizontally scaling your environment, see the [Deployment Guide](#).

## 配置前的准备

在您安装和配置卷服务之前，您必须先配置存储节点。类似于控制节点，存储节点包含一个管理网络接口上的网络。存储节点可以选择性地包含第二个网络接口到一个独立地网络上，用以复制。要了解更多信息，请阅读[第 2 章 基本环境 \[11\]](#)。

|           |         |
|-----------|---------|
| # object1 |         |
| 10.0.0.51 | object1 |
| # object2 |         |
| 10.0.0.52 | object2 |

- b. 使用“其它节点服务器”一节[24]中的指示安装和配置 NTP。
- c. 安装支持的工具包：

```
yum install xfsprogs rsync
```

- d. 将 /dev/sdb1 和 /dev/sdc1 分区格式化为 XFS：

```
mkfs.xfs /dev/sdb1
mkfs.xfs /dev/sdc1
```

- e. 创建挂载点目录结构：

```
mkdir -p /srv/node/sdb1
mkdir -p /srv/node/sdc1
```

- f. 编辑 `/etc/fstab` 文件并添加下列内容：

```
/dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 2
/dev/sdc1 /srv/node/sdc1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 2
```

- g. 挂载设备：



```
mount /srv/node/sdb1
mount /srv/node/sdc1
```

4. 添加以下内容到/etc/rsyncd.conf文件中：

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = MANAGEMENT_INTERFACE_IP_ADDRESS

[account]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/account.lock

[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```

将其中的 MANAGEMENT\_INTERFACE\_IP\_ADDRESS 替换为存储节点上的管理网络接口的 IP 地址。



## 注意

rsync 服务不需要认证，因此可以考虑将其运行在私有网络中。

5. 启动 rsync 服务并将其设置为随系统启动：

```
systemctl enable rsyncd.service
systemctl start rsyncd.service
```

## 安装并配置存储节点组件



## 注意

在每个存储节点上执行这些步骤。

1. 安装软件包：

```
yum install openstack-swift-account openstack-swift-container
openstack-swift-object
```

2. 从对象存储资源仓库中获取帐户、容器和对象服务的配置文件：

```
curl -o /etc/swift/account-server.conf
https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/account-server.conf-sample
curl -o /etc/swift/container-server.conf
```

`https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/container-server.conf-sample`

```
curl -o /etc/swift/object-server.conf
```

<https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/object-server.conf-sample>

3. 修改配置文件 `/etc/swift/account-server.conf` 并完成以下操作：

- a. 在 [DEFAULT] 部分，配置绑定 IP 地址、绑定端口、用户、配置文件目录和挂载点目录：

```
[DEFAULT]
...
bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
bind_port = 6002
user = swift
swift_dir = /etc/swift
devices = /srv/node
```

将其中的 MANAGEMENT\_INTERFACE\_IP\_ADDRESS 替换为存储节点上的管理网络接口的 IP 地址。

- b. 在[`pipeline:main`]部分，启用合适的模块：

```
[pipeline:main]
pipeline = healthcheck recon account-server
```



## 注意

For more information on other modules that enable additional features, see the [Deployment Guide](#).

- c. 在[filter:recon]部分，配置recon(metrics)缓存目录：

```
[filter:recon]
...
recon cache path = /var/cache/swift
```

4. 修改配置文件 `/etc/swift/container-server.conf` 并完成以下操作：

- a. 在 [DEFAULT] 部分，配置绑定 IP 地址、绑定端口、用户、配置文件目录和挂载点目录：

```
[DEFAULT]
...
bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
bind_port = 6001
user = swift
swift_dir = /etc/swift
devices = /srv/node
```

将其中的 MANAGEMENT\_INTERFACE\_IP\_ADDRESS 替换为存储节点上的管理网络接口的 IP 地址。

- b. 在 `[pipeline:main]` 部分，启用合适的模块：

```
[pipeline:main]
pipeline = healthcheck recon container-server
```



## 注意

For more information on other modules that enable additional features, see the [Deployment Guide](#).

- c. 在[filter:recon]部分，配置recon(metrics)缓存目录：

```
[filter:recon]
...
recon cache path = /var/cache/swift
```

5. 修改配置文件 `/etc/swift/object-server.conf` 并完成以下操作：

- a. 在 [DEFAULT] 部分，配置绑定 IP 地址、绑定端口、用户、配置文件目录和挂载点目录：

```
[DEFAULT]
...
bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
bind_port = 6000
user = swift
swift_dir = /etc/swift
devices = /srv/node
```

将其中的 MANAGEMENT\_INTERFACE\_IP\_ADDRESS 替换为存储节点上的管理网络接口的 IP 地址。

- b. 在 [pipeline:main] 部分，启用合适的模块：

```
[pipeline:main]
pipeline = healthcheck recon object-server
```



## 注意

For more information on other modules that enable additional features, see the [Deployment Guide](#).

- c. 在 [filter:recon] 部分，配置 recon (metrics) 缓存目录：

```
[filter:recon]
...
recon cache path = /var/cache/swift
```

6. 确认挂载点目录结构是否有合适的所有权：

```
chown -R swift:swift /srv/node
```

7. 创建 recon 目录并确认它有合适的所有权：

```
mkdir -p /var/cache/swift
chown -R swift:swift /var/cache/swift
```

## 创建初始化的 rings

Before starting the Object Storage services, you must create the initial account, container, and object rings. The ring builder creates configuration files that each node

uses to determine and deploy the storage architecture. For simplicity, this guide uses one region and zone with  $2^{10}$  (1024) maximum partitions, 3 replicas of each object, and 1 hour minimum time between moving a partition more than once. For Object Storage, a partition indicates a directory on a storage device rather than a conventional partition table. For more information, see the [Deployment Guide](#).

帐户 ring

帐户服务器使用帐户 ring 来维护一个容器的列表。

## 创建 ring



## 注意

在控制节点上执行这些步骤。

1. 切换到/etc/swift目录。
2. 创建基本的account.builder文件：

```
swift-ring-builder account.builder create 10 3 1
```

3. 添加每个节点到 ring 中：

```
swift-ring-builder account.builder
add r1z1-STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS:6002/DEVICE_NAME DEVICE WEIGHT
```

将其中的 `STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS` 替换为存储节点上的管理网络的 IP 地址。将其中的 `DEVICE_NAME` 替换为同一个存储节点上的一个存储设备的名称。例如，使用 [“安装和配置存储节点”一节 \[95\]](#) 中的第一个存储节点的 `/dev/sdb1` 存储设备，大小为 100：

```
swift-ring-builder account.builder add r1z1-10.0.0.51:6002/sdb1 100
```

在每个存储节点上为每个存储设备重复这个命令。样例架构需要该命令的四个变量。

- #### 4. 验证 ring 的内容：

```
swift-ring-builder account.builder
account.builder, build version 4
1024 partitions, 3.000000 replicas, 1 regions, 1 zones, 4 devices, 0.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices: id region zone ip address port replication ip replication port name weight
partitions balance meta
0 1 1 10.0.0.51 6002 10.0.0.51 6002 sdb1 100.00 768 0.00
1 1 1 10.0.0.51 6002 10.0.0.51 6002 sdc1 100.00 768 0.00
2 1 1 10.0.0.52 6002 10.0.0.52 6002 sdb1 100.00 768 0.00
3 1 1 10.0.0.52 6002 10.0.0.52 6002 sdc1 100.00 768 0.00
```

5. 平衡 ring :

```
swift-ring-builder account.builder rebalance
```



注意

这个过程会花费一些时间。

容器 ring

容器服务器使用容器环来维护对象的列表。但是，它不跟踪对象的位置。

## 创建 ring



## 注意

在控制节点上执行这些步骤。

1. 切换到/etc/swift目录。
2. 生成基本的container.builder文件：

```
swift-ring-builder container.builder create 10 3 1
```

3. 添加每个节点到 ring 中：

```
swift-ring-builder container.builder
add r1z1-STORAGE NODE MANAGEMENT INTERFACE IP ADDRESS:6001/DEVICE NAME DEVICE WEIGHT
```

将其中的 `STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS` 替换为存储节点上的管理网络的 IP 地址。将其中的 `DEVICE_NAME` 替换为同一个存储节点上的一个存储设备的名称。例如，使用 “[安装和配置存储节点](#)” 一节 [95] 中的第一个存储节点的 `/dev/sdb1` 存储设备，大小为 100：

```
swift-ring-builder container.builder add r1z1-10.0.0.51:6001/sdb1 100
```

在每个存储节点上为每个存储设备重复这个命令。样例架构需要该命令的四个变量。

- #### 4. 验证 ring 的内容：

```
swift-ring-builder container.builder
container.builder, build version 4
1024 partitions, 3,000,000 replicas, 1 regions, 1 zones, 4 devices, 0.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices: id region zone ip address port replication ip replication port name weight
partitions balance meta
0 1 1 10.0.0.51 6001 10.0.0.51 6001 sdb1 100.00 768 0.00
1 1 1 10.0.0.51 6001 10.0.0.51 6001 sdc1 100.00 768 0.00
2 1 1 10.0.0.52 6001 10.0.0.52 6001 sdb1 100.00 768 0.00
3 1 1 10.0.0.52 6001 10.0.0.52 6001 sdc1 100.00 768 0.00
```

5. 平衡 ring :

```
swift-ring-builder container.builder rebalance
```



## 注意

这个过程会花费一些时间。

## 对象环

对象服务器使用对象环来维护对象在本地设备上的位置列表。

## 创建 ring



注意

在控制节点上执行这些步骤。

1. 切换到/etc/swift目录。
2. 生成基本的 object.builder文件：

```
swift-ring-builder object.builder create 10 3 1
```

3. 添加每个节点到 ring 中：

```
swift-ring-builder object.builder
add r1z1-STORAGE NODE MANAGEMENT INTERFACE IP ADDRESS:6000/DEVICE NAME DEVICE WEIGHT
```

将其中的 `STORAGE_NODE_MANAGEMENT_INTERFACE_IP_ADDRESS` 替换为存储节点上的管理网络的 IP 地址。将其中的 `DEVICE_NAME` 替换为同一个存储节点上的一个存储设备的名称。例如，使用 [“安装和配置存储节点”一节 \[95\]](#) 中的第一个存储节点的 `/dev/sdb1` 存储设备，大小为 100：

```
swift-ring-builder object.builder add r1z1-10.0.0.51:6000/sdb1 100
```

在每个存储节点上为每个存储设备重复这个命令。样例架构需要该命令的四个变量。

- #### 4. 验证 ring 的内容：

```
swift-ring-builder object.builder
object.builder, build version 4
1024 partitions, 3,000000 replicas, 1 regions, 1 zones, 4 devices, 0.00 balance
The minimum number of hours before a partition can be reassigned is 1
Devices: id region zone ip address port replication ip replication port name weight
partitions balance meta
0 1 1 10.0.0.51 6000 10.0.0.51 6000 sdb1 100.00 768 0.00
1 1 1 10.0.0.51 6000 10.0.0.51 6000 sdc1 100.00 768 0.00
2 1 1 10.0.0.52 6000 10.0.0.52 6000 sdb1 100.00 768 0.00
3 1 1 10.0.0.52 6000 10.0.0.52 6000 sdc1 100.00 768 0.00
```

5. 平衡 ring :

```
swift-ring-builder object.builder rebalance
```



## 注意

这个过程会花费一些时间。

## 分发环配置文件

复制 `account.ring.gz`、`container.ring.gz` 和 `object.ring.gz` 文件到每个存储节点和其他运行了代理服务的额外节点的 `/etc/swift` 目录下：

## 完成安装

## 配置哈希和默认的存储策略

1. 从对象存储的仓库源中获取 `/etc/swift/swift.conf` 文件：

```
curl -o /etc/swift/swift.conf
https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/swift.conf-sample
```

2. 修改配置文件 `/etc/swift/swift.conf` 并完成以下操作：

- a. 在[swift-hash]部分，为您的环境配置哈希路径的前缀和或缀。

```
[swift-hash]
...
swift_hash_path_suffix = HASH_PATH_PREFIX
swift_hash_path_prefix = HASH_PATH_SUFFIX
```

将其中的 HASH PATH PREFIX 和 HASH PATH SUFFIX 替换为唯一的值。



警告

这些值要保密，并且不要修改或丢失。

- b. 在[storage-policy:0]部分，配置默认存储策略：

```
[storage-policy:0]
...
name = Policy-0
default = yes
```

3. 复制 `swift.conf` 文件到每个存储节点和其他运行了代理服务的额外节点的 `/etc/swift` 目录下：

4. 在所有节点上，确认配置文件目录是否有合适的所有权：

```
chown -R swift:swift /etc/swift
```

- 在控制节点和其他运行了代理服务的节点上，启动对象存储代理服务及其依赖服务，并将它们配置为随系统启动：

```
systemctl enable openstack-swift-proxy.service memcached.service
systemctl start openstack-swift-proxy.service memcached.service
```

6. 在存储节点上，启动对象存储服务，并将其设置为随系统启动：

```
systemctl enable openstack-swift-account.service openstack-swift-account-auditor.service
openstack-swift-account-reaper.service openstack-swift-account-replicator.service
systemctl start openstack-swift-account.service openstack-swift-account-auditor.service
openstack-swift-account-reaper.service openstack-swift-account-replicator.service
systemctl enable openstack-swift-container.service openstack-swift-container-auditor.service
openstack-swift-container-replicator.service openstack-swift-container-updater.service
systemctl start openstack-swift-container.service openstack-swift-container-auditor.service
openstack-swift-container-replicator.service openstack-swift-container-updater.service
systemctl enable openstack-swift-object.service openstack-swift-object-auditor.service
openstack-swift-object-replicator.service openstack-swift-object-updater.service
systemctl start openstack-swift-object.service openstack-swift-object-auditor.service
```

```
openstack-swift-object-replicator.service openstack-swift-object-updater.service
```

## 验证操作

这个部分将描述如何验证对象存储服务的操作。



## 注意

在控制节点上执行这些步骤。

1. Source demo 租户凭证：

```
$ source demo-openrc.sh
```

- ## 2. 显示服务状态：

```
$ swift stat
Account: AUTH_11b9758b7049476d9b48f7a91ea11493
Containers: 0
 Objects: 0
 Bytes: 0
Content-Type: text/plain; charset=utf-8
X-Timestamp: 1381434243.83760
X-Trans-Id: txdcdd594565214fb4a2d33-0052570383
X-Put-Timestamp: 1381434243.83760
```

3. 上传一个测试文件：

```
$ swift upload demo-container1 文件
```

将其中的 FILE 替换为要上传到 demo-container1 容器上的本地文件的名称。

4. 列出容器：

```
$ swift list
demo-container1
```

5. 下载一个测试文件：

```
$ swift download demo-container1 文件
```

将其中的 FILE 替换为上传到 demo-container1 容器的文件的名称。

## 下一步

您的OpenStack环境现在已经包含对象存储。您可以 [新建实例](#)或参考接下来的章节来添加更多的服务到您的环境中去。





- 

1

- xi

K

1

- E

E

- E

- E

E

E

- E

E



E

E

- E

E

- E

E



E

E

- E

```
$ keystone role-create --name heat_stack_user
```



## 注意

Orchestration 自动地分配 `heat_stack_user` 给在 `stack` 部署过程中创建的用户。默认情况下，这个角色会限制 API 的操作。为了避免冲突，请不要为用户添加 `heat_stack_owner` 角色。

- f. 创建 heat 和 heat-cfn 服务实体：

```
$ keystone service-create --name heat --type orchestration
--description "Orchestration"
+-----+
| Property | Value |
+-----+
description	Orchestration
enabled	True
id	031112165cad4c2bb23e84603957de29
name	heat
type	orchestration
+-----+

$ keystone service-create --name heat-cfn --type cloudformation
--description "Orchestration"
+-----+
| Property | Value |
+-----+
description	Orchestration
enabled	True
id	297740d74c0a446bbff867acdccb33fa
name	heat-cfn
type	cloudformation
+-----+
```

- #### 4. 创建 Orchestration 服务的 API 端点：

```
$ keystone endpoint-create
--service-id $(keystone service-list | awk '/ orchestration / {print $2}')
--publicurl http://控制器:8004/v1/%(tenant_id)s
--internalurl http://控制器:8004/v1/%(tenant_id)s
--adminurl http://控制器:8004/v1/%(tenant_id)s
--region regionOne
```

| Property    | Value                                   |
|-------------|-----------------------------------------|
| adminurl    | http://controller:8004/v1/%(tenant_id)s |
| id          | f41225f665694b95a46448e8676b0dc2        |
| internalurl | http://controller:8004/v1/%(tenant_id)s |
| publicurl   | http://controller:8004/v1/%(tenant_id)s |
| region      | regionOne                               |
| service_id  | 031112165cad4c2bb23e84603957de29        |

```
$ keystone endpoint-create
--service-id $(keystone service-list | awk '/ cloudformation / {print $2}')
--publicurl http://控制器:8000/v1
--internalurl http://控制器:8000/v1
--adminurl http://控制器:8000/v1
--region regionOne
```

| Property    | Value                            |
|-------------|----------------------------------|
| adminurl    | http://controller:8000/v1        |
| id          | f41225f665694b95a46448e8676b0dc2 |
| internalurl | http://controller:8000/v1        |
| publicurl   | http://controller:8000/v1        |
| region      | regionOne                        |
| service_id  | 297740d74c0a446bbff867acdccb33fa |

## 安装和配置Orchestration组件

1. 运行以下命令安装软件包：

```
yum install openstack-heat-api openstack-heat-api-cfn openstack-heat-engine
python-heatclient
```

2. 修改配置文件 `/etc/heat/heat.conf` 并完成以下操作：

- a. 在[database]段，配置数据库访问相关参数：

```
[database]
...
connection = mysql://heat:HEAT_DBPASS@控制器/heat
```

将其中的 HEAT\_DBPASS 替换为您为 Orchestration 数据库所设置的密码。

- b. 在[DEFAULT]段，对 RabbitMQ 消息代理相关参数进行配置：

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将 RABBIT\_PASS 替换为 RabbitMQ 服务中 guest 用户的密码。

- c. 在 [keystone authtoken] 和 [ec2authtoken] 部分中，配置身份认证服务的访问：

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

```
[ec2authtoken]
...
auth_uri = http://控制器:5000/v2.0
```

将其中的 HEAT\_PASS 替换为您在身份认证服务中为 heat 用户设置的密码。



## 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。

- d. 在 [DEFAULT] 部分，配置 metadata 和等待环境 URLs：

```
[DEFAULT]
...
heat_metadata_server_url = http://控制器:8000
heat_waitcondition_server_url = http://控制器:8000/v1/waitcondition
```

- e. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

- ### 3. 同步Orchestration数据库：

```
su -s /bin/sh -c "heat-manage db sync" heat
```

## 完成安装

- 启动 Orchestration 服务并将其设置为随系统启动：

```
systemctl enable openstack-heat-api.service openstack-heat-api-cfn.service
openstack-heat-engine.service
systemctl start openstack-heat-api.service openstack-heat-api-cfn.service
openstack-heat-engine.service
```

## 验证操作

这个不分将描述如何验证 Orchestration 模块 (heat) 的操作。

1. Source demo 租户凭证：

```
$ source demo-openrc.sh
```

2. The Orchestration module uses templates to describe stacks. To learn about the template language, see [the Template Guide](#) in the [Heat developer documentation](#).

在 `test-stack.yml` 文件中使用以下内容创建一个测试模板：

```
heat_template_version: 2014-10-16
description: A simple server.

parameters:
 ImageID:
 type: string
 description: Image use to boot a server
 NetID:
 type: string
 description: Network ID for the server

resources:
 server:
 type: OS::Nova::Server
 properties:
 image: { get_param: ImageID }
 flavor: ml.tiny
 networks:
 - network: { get_param: NetID }
```

```
outputs:
 private_ip:
 description: IP address of the server in the private network
 value: { get attr: [server, first address] }
```

3. 使用heat stack-create命令行以模板创建一个栈：

```
$ NET_ID=$(nova net-list | awk '/ demo-net / { print $2 }')
```

```
$ heat stack-create -f test-stack.yml
```

```
-P "ImageID=cirros-0.3.3-x86_64;NetID=$NET_ID" testStack
```

| id                                   | stack_name | stack_status       | creation_time        |
|--------------------------------------|------------|--------------------|----------------------|
| 477d96b4-d547-4069-938d-32ee990834af | testStack  | CREATE_IN_PROGRESS | 2014-04-06T15:11:01Z |

4. 使用heat stack-list命令行来验证栈的创建是否成功：

```
$ heat stack-list
+-----+-----+-----+-----+
| id | stack_name | stack_status | creation_time |
+-----+-----+-----+-----+
| 477d96b4-d547-4069-938d-32ee990834af | testStack | CREATE_COMPLETE | 2014-04-06T15:11:01Z |
+-----+-----+-----+-----+
```

## 下一步

您的 OpenStack 环境现在已经包含了 Orchestration。您可以[启动实例](#)或根据以下章节添加更多的服务到您的环境中。



|                                   |                                                                                                                                                                                                                                           |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                   | data coming from the agent). Notification messages are processed and turned into metering messages, which are sent to the message bus using the appropriate topic. Telemetry messages are written to the data store without modification. |
| 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。                                                                                                                                                                                                    |
| A data store                      | A database capable of handling concurrent writes (from one or more collector instances) and reads (from the API server).                                                                                                                  |
| 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。

## 安装配置控制节点服务器

这个部分将描述如何在控制节点上安装和配置 Telemetry 模块，即 ceilometer。Telemetry 模块使用分离的代理来从您环境的 OpenStack 服务中收集评估。

## 配置前的准备

安装和配置 Telemetry 之前，您必须先安装 MongoDB，创建一个 MongoDB 数据库、服务证书和 API 端点。

- ## 1. 安装MongoDB包：

```
yum install mongodb-server mongodb
```

2. 修改配置文件 `/etc/mongodb.conf` 并完成以下操作：

- a. 配置 `bind_ip` 关键字以使用控制节点的管理网络接口的 IP 地址。

```
bind_ip = 10.0.0.11
```

- b. By default, MongoDB creates several 1 GB journal files in the `/var/lib/mongodb/journal` directory. If you want to reduce the size of each journal file to 128 MB and limit total journal space consumption to 512 MB, assert the `smallfiles` key:

```
smallfiles = true
```

You can also disable journaling. For more information, see the [MongoDB manual](#).

- c. 启动 MongoDB 服务并将其配置为随系统启动：

```
systemctl enable mongod.service
systemctl start mongod.service
```



### 3. 创建ceilometer数据库：

```
mongo --host 控制器 --eval '
db = db.getSiblingDB("ceilometer");
db.addUser({user: "ceilometer",
pwd: "CEILOMETER_DBPASS",
roles: ["readWrite", "dbAdmin"]})'

MongoDB shell version: X.Y.Z
connecting to: controller:27017/test
{
 "user": "ceilometer",
 "pwd": "72f25aeee7ad4be52437d7cd3fc60f6f",
 "roles": [
 "readWrite",
 "dbAdmin"
],
 "_id": ObjectId("5489c22270d7fad1ba631dc3")
}
```

将其中的 CEILOMETER DBPASS 替换为一个合适的密码。

4. 导入 admin 身份凭证以执行管理员用户专有的命令：

```
$ source admin-openrc.sh
```

5. 创建服务证书，完成如下步骤：

a. 创建ceilometer用户：

```
$ keystone user-create --name ceilometer --pass CEILOMETER PASS
```

将其中的 CEILOMETER PASS 替换为一个合适的密码。

b. 为 ceilometer 用户添加 admin 角色。

```
$ keystone user-role-add --user ceilometer --tenant service --role admin
```

c. 创建ceilometer服务条目：

```
$ keystone service-create --name ceilometer --type metering
--description "Telemetry"
```

## 6. 创建Telemetry模块API端点：

```
$ keystone endpoint-create
--service-id $(keystone service-list | awk '/ metering / {print $2}')
--publicurl http://控制器:8777
--internalurl http://控制器:8777
--adminurl http://控制器:8777
--region regionOne
```

## 安装和配置Telemetry模块组件

1. 安装软件包：

```
yum install openstack-ceilometer-api openstack-ceilometer-collector
openstack-ceilometer-notification openstack-ceilometer-central openstack-ceilometer-alarm
python-ceilometerclient
```

2. Generate a random value to use as the metering secret:

```
$ openssl rand -hex 10
```

3. 修改配置文件 `/etc/ceilometer/ceilometer.conf` 并完成以下操作：

- a. 在[database]段，配置数据库访问相关参数：

```
[database]
...
connection = mongodb://ceilometer:CEILOMETER_DBPASS@控制器:27017/ceilometer
```

将其中的 CEILOMETER\_DBPASS 替换为您为 Telemetry 模块数据库所设置的密码。在给定的 RFC2396 字符串连接中，要避开一些特殊字符，例如 ':'、'/'、'+' 和 '@'。

- b. 在[DEFAULT]段，对 RabbitMQ 消息代理相关参数进行配置：

```
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将 RABBIT\_PASS 替换为 RabbitMQ 服务中 guest 用户的密码。

- c. 在 [DEFAULT] 和 [keystone authtoken] 部分，配置身份认证服务的访问：

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

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

将其中的 CEILOMETER\_PASS 替换为您在身份认证服务中为 celimeter 用户设置的密码。



## 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。

- d. 在[service\_credentials]部分，配置服务的证书：

```
[service_credentials]
...
os_auth_url = http://控制器:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = CEILOMETER_PASS
```

将其中的 CEILOMETER\_PASS 替换为您在身份认证服务中为 ceilometer 用户所设置的密码。

- e. In the [publisher] section, configure the metering secret:

```
[publisher]
...
metering_secret = METERING_SECRET
```

Replace `METERING_SECRET` with the metering secret that you generated in a previous step.

- f. (可选配置)可以在 [DEFAULT] 段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

## 完成安装

- 启动Telemetry服务并将其配置为随系统启动：

```
systemctl enable openstack-ceilometer-api.service openstack-ceilometer-notification.service
openstack-ceilometer-central.service openstack-ceilometer-collector.service
openstack-ceilometer-alarm-evaluator.service openstack-ceilometer-alarm-notifier.service
systemctl start openstack-ceilometer-api.service openstack-ceilometer-notification.service
openstack-ceilometer-central.service openstack-ceilometer-collector.service
openstack-ceilometer-alarm-evaluator.service openstack-ceilometer-alarm-notifier.service
```

## Install the Compute agent for Telemetry

Telemetry is composed of an API service, a collector and a range of disparate agents. This section explains how to install and configure the agent that runs on the compute node.

## 配置前的准备

1. 安装包：

```
yum install openstack-ceilometer-compute python-ceilometerclient python-pecan
```

2. 修改配置文件 `/etc/nova/nova.conf` 并在 `[DEFAULT]` 部分配置消息机制：

```
[DEFAULT]
...
instance_usage_audit = True
instance_usage_audit_period = hour
notify_on_state_change = vm_and_task_state
notification_driver = nova.openstack.common.notifier.rpc_notifier
notification_driver = ceilometer.compute.nova_notifier
```

- ### 3. 重启计算服务：

```
systemctl restart openstack-nova-compute.service
```

## To configure the Compute agent for Telemetry

修改配置文件 `/etc/ceilometer/ceilometer.conf` 并完成以下操作：

1. In the [publisher] section, configure the metering secret:

```
[publisher]
...
metering_secret = METERING_SECRET
```

Replace `METERING_SECRET` with the metering secret you chose for the Telemetry module.

2. 在[DEFAULT]段，对 RabbitMQ 消息代理相关参数进行配置：

```
[DEFAULT]
...
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将其中的 RABBIT\_PASS 替换为您所设置的 RabbitMQ 的 guest 帐户的密码。

3. 在[keystone authtoken]段，配置身份认证服务相关信息：

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

将其中的 CEILOMETER PASS 替换为您为 Telemetry 模块数据库所设置的密码。



## 注意

Comment out the `auth_host`, `auth_port`, and `auth_protocol` keys, since they are replaced by the `identity_uri` and `auth_uri` keys.

4. 在[service credentials]部分，配置服务的证书：

```
[service_credentials]
...
os_auth_url = http://控制器:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = CEILOMETER_PASS
os_endpoint_type = internalURL
os_region_name = regionOne
```

将其中的 CEILOMETER\_PASS 替换为您在身份认证服务中为 ceilometer 用户所设置的密码。

5. (可选配置)可以在[DEFAULT]段中开启详细日志配置，为后期的故障排除提供帮助：

```
[DEFAULT]
...
verbose = True
```

## 完成安装

- Start the Telemetry service and configure it to start when the system boots:

```
systemctl enable openstack-ceilometer-compute.service
```

```
systemctl start openstack-ceilometer-compute.service
```

## Configure the Image Service for Telemetry

## 配置前的准备

To retrieve image samples, you must configure the Image Service to send notifications to the message broker. Edit the `/etc/glance/glance-api.conf` and `/etc/glance/glance-registry.conf` files and complete the following actions:

1. 在 [DEFAULT] 部分，配置消息和 RabbitMQ 代理的访问：

```
[DEFAULT]
...
notification_driver = messaging
rpc_backend = rabbit
rabbit_host = 控制器
rabbit_password = RABBIT_PASS
```

将 RABBIT\_PASS 替换为 RabbitMQ 服务中 guest 用户的密码。

- ## 2. 重启镜像服务：

```
systemctl restart openstack-glance-api.service openstack-glance-registry.service
```

## Add the Block Storage service agent for Telemetry

1. To retrieve volume samples, you must configure the Block Storage service to send notifications to the bus.

Edit `/etc/cinder/cinder.conf` and add in the `[DEFAULT]` section on the controller and volume nodes:

```
control_exchange = cinder
notification_driver = cinder.openstack.common.notifier.rpc_notifier
```

- Restart the Block Storage services with their new settings.

On the controller node:

```
systemctl restart openstack-cinder-api.service openstack-cinder-scheduler.service
```

On the storage node:

```
systemctl restart openstack-cinder-volume.service
```

3. If you want to collect OpenStack Block Storage notification on demand, you can use `cinder-volume-usage-audit` from OpenStack Block Storage. For more information, [Block Storage audit script setup to get notifications](#).

## Configure the Object Storage service for Telemetry

1. Install the `python-ceilometerclient` package on your Object Storage proxy server:

```
yum install python-ceilometerclient
```

2. To retrieve object store statistics, the Telemetry service needs access to Object Storage with the ResellerAdmin role. Give this role to your `os_username` user for the `os_tenant_name` tenant:

```
$ keystone role-create --name ResellerAdmin
```

| Property | Value                            |
|----------|----------------------------------|
| id       | 462fa46c13fd4798a95a3bfbe27b5e54 |
| name     | ResellerAdmin                    |

```
$ keystone user-role-add --tenant service --user ceilometer
--role 462fa46c13fd4798a95a3bfbe27b5e54
```

3. You must also add the Telemetry middleware to Object Storage to handle incoming and outgoing traffic. Add these lines to the `/etc/swift/proxy-server.conf` file:

```
[filter:ceilometer]
use = egg:ceilometer#swift
```

4. Add ceilometer to the pipeline parameter of that same file:

```
[pipeline:main]
pipeline = healthcheck cache authtoken keystoneauth ceilometer proxy-server
```

5. Add ResellerAdmin to the operator roles parameter of that same file:

```
operator roles = Member,admin,swiftoperator, member ,ResellerAdmin
```

6. Add the system user swift to the system group ceilometer to give Object Storage access to the ceilometer.conf file.

```
usermod -a -G ceilometer swift
```

- Restart the service with its new settings:

```
systemctl restart openstack-swift-proxy.service
```

## 验证 Telemetry 的安装

这个部分将描述如何验证 Telemetry 模块的操作。



## 注意

在控制节点上执行这些步骤。

1. 导入 admin 身份凭证以执行管理员用户专有的命令：

```
$ source admin-openrc.sh
```

2. 列出可用的 meters :

\$ ceilometer meter-list

| Name                             | Type  | Unit  | Resource ID                          | User ID | Project ID |
|----------------------------------|-------|-------|--------------------------------------|---------|------------|
| image                            | gauge | image | acafc7c0-40aa-4026-9673-b879898e1fc2 | None    |            |
| efa984b0a914450e9a47788ad330699d |       |       |                                      |         |            |
| image.size                       | gauge | B     | acafc7c0-40aa-4026-9673-b879898e1fc2 | None    |            |
| efa984b0a914450e9a47788ad330699d |       |       |                                      |         |            |



## 注意

一些 meters 可能不会出现，直到您至少成功启动了一台实例之后。

- ### 3. 从映像服务器下载一个映像：

```
$ glance image-download "cirros-0.3.3-x86 64" > cirros.img
```

4. 再次列出可用的 meters 以验证镜像下载的检查：

```
$ ceilometer meter-list
```

| Name                             | Type  | Unit  | Resource ID                          | User ID | Project ID |
|----------------------------------|-------|-------|--------------------------------------|---------|------------|
| image                            | gauge | image | acafc7c0-40aa-4026-9673-b879898e1fc2 | None    |            |
| efa984b0a914450e9a47788ad330699d |       |       |                                      |         |            |
| image.download                   | delta | B     | acafc7c0-40aa-4026-9673-b879898e1fc2 | None    |            |
| efa984b0a914450e9a47788ad330699d |       |       |                                      |         |            |
| image.serve                      | delta | B     | acafc7c0-40aa-4026-9673-b879898e1fc2 | None    |            |
| efa984b0a914450e9a47788ad330699d |       |       |                                      |         |            |
| image.size                       | gauge | B     | acafc7c0-40aa-4026-9673-b879898e1fc2 | None    |            |
| efa984b0a914450e9a47788ad330699d |       |       |                                      |         |            |

5. 从 `image.download meter` 中获取使用统计数据：

```
$ ceilometer statistics -m image.download -p 60
```

| Period         | Period Start               | Period End                 | Count | Min        | Max        | Sum        | Avg        | Duration |
|----------------|----------------------------|----------------------------|-------|------------|------------|------------|------------|----------|
| Duration Start | Duration End               |                            |       |            |            |            |            |          |
| 60             | 2013-11-18T18:08:50        | 2013-11-18T18:09:50        | 1     | 13167616.0 | 13167616.0 | 13167616.0 | 13167616.0 | 0.0      |
|                | 2013-11-18T18:09:05.334000 | 2013-11-18T18:09:05.334000 |       |            |            |            |            |          |

## 下一步

您的 OpenStack 环境现在已经包含了 Telemetry。您可以[创建实例](#)或如前面章节所示添加更多的服务到您的环境中。

## 第 12 章 添加数据库服务

# 目录

|                 |     |
|-----------------|-----|
| 数据库服务概览 .....   | 120 |
| 安装数据库服务 .....   | 121 |
| 验证数据服务的安装 ..... | 124 |

使用数据库模块来创建云数据库资源。该整合的项目名称为 trove。



敬告

这个章节正在书写中。它可能包含一些错误的信息，并且会频繁地更新。

## 数据库服务概览

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 trove-api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。 |
|--------------|--------------------------------------------------|



T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT -

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

- T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT -

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT

- [illegible]

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

- T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

- T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT

- T - KiLo - DRAFT - KiLo - DRAFT - KiLo

T - Kilo - DRAFT - Kilo - DRAFT - M

T - Kilo - DRAFT - Kilo - DRAFT

- T - Kilo - DRAFT - Kilo - D

- T - Kilo - DRAFT - Kilo

- T - Kilo -

```
[composite:trove]
auth_uri = http://控制器:5000/v2.0
identity_uri = http://控制器:35357
auth_host = 控制器
admin_tenant_name = service
admin_user = trove
admin_password = TROVE PASS
```

- b. 修改每个文件 (api-paste.ini 除外) 中的 [DEFAULT] 部分，并为 OpenStack 的服务 URLs (可由身份认证服务目录处理)、logging 和 messaging 配置和 SQL 连接设置合适的值：

```
[DEFAULT]
log_dir = /var/log/trove
trove_auth_url = http://控制器:5000/v2.0
nova_compute_url = http://控制器:8774/v2
cinder_url = http://控制器:8776/v2
swift_url = http://控制器:8080/v1/AUTH_
sql_connection = mysql://trove:TROVE_DBPASS@控制器/trove
notifier_queue_hostname = 控制器
```

- c. 配置数据库模块，通过设置下列每个文件的 [DEFAULT] 配置组中的选项来使用 RabbitMQ 的消息 broker：

```
[DEFAULT]
control_exchange = trove
rabbit_host = 控制器
rabbit_userid = 客户机
rabbit_password = RABBIT_PASS
rabbit_virtual_host = /
rpc_backend = trove.openstack.common.rpc.impl kombu
```

4. 修改 `trove.conf` 文件，使其包含默认数据存储和网络标签正则表达式的合适的值，如下文所示：

```
[DEFAULT]
Config option for showing the IP address that nova does out.
add_addresses = True
network_label_regex = ^NETWORK_LABEL$
control_exchange = trove
```

5. 修改 `trove-taskmanager.conf` 文件，使其包含需要的设置，以连接到 OpenStack Compute 服务，如下文所示：

```
[DEFAULT]
Configuration options for talking to nova via the novaclient.
These options are for an admin user in your keystone config.
It proxy's the token received from the user to send to nova via this admin users creds,
basically acting like the client via that proxy token.
nova_proxy_admin_user = admin
nova_proxy_admin_pass = ADMIN_PASS
nova_proxy_admin_tenant_name = service
taskmanager_manager = trove.taskmanager.manager.Manager
log_file=trove-taskmanager.log
```

- ## 6. 准备trove管理员数据库：

```
$ mysql -u root -p
mysql> CREATE DATABASE trove;
mysql> GRANT ALL PRIVILEGES ON trove.* TO trove@'localhost'
IDENTIFIED BY 'TROVE_DBPASS';
mysql> GRANT ALL PRIVILEGES ON trove.* TO trove@'%'
IDENTIFIED BY 'TROVE_DBPASS';
```

## 7. 准备数据库服务：

a. 初始化数据库：

```
trove-manage db sync
```

- b. 创建一个数据库。您需要为每个您想使用的类型的数据库创建一个分离的数据存储，例如，MySQL、MongoDB、Cassandra。这个例子将向您展示如何为一个 MySQL 数据库创建数据存储：

```
su-s /bin/sh -c "trove-manage datastore update mysql "" trove
```

## 8. 创建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。

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，根据下列步骤进行：

- 添加下列几行到 `trove-guestagent.conf` 文件中：

```
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
log_file = trove-guestagent.log
```

- 更新数据仓储和版本，以通过 `trove-manage` 命令使用特定的镜像。

```
#trove-manage datastore_update datastore_name datastore_version
#trove-manage datastore_version_update datastore_name version_name
datastore manager glance image id 软件包 激活
```

这个实例将向您展示如何创建一个 5.5 版本的 MySQL 数据存储：

```
#trove-manage datastore_update mysql ""
#trove-manage datastore_version_update mysql 5.5 mysql glance_image_ID mysql-server-5.5 1
#trove-manage datastore_update mysql 5.5
```

上传 post-provisioning 配置验证规则：

```
#trove-manage db_load_datastore_config_parameters datastore_name version_name
/etc/datastore_name/validation-rules.json
```

MySQL 数据存储上传规则的例子：

```
trove-manage db load datastore config parameters
```

```
mysql 5.5 "$PYBASEDIR"/trove/templates/mysql/validation-rules.json
```

10. 您必需用身份认证服务注册数据库模块，这样，其他 OpenStack 服务才能找到它。注册服务并指定端点：

```
$ keystone service-create --name trove --type database
--description "OpenStack Database Service"
$ keystone endpoint-create
--service-id $(keystone service-list | awk '/ trove / {print $2}')
--publicurl http://控制器:8779/v1.0/(tenant_id)s
--internalurl http://控制器:8779/v1.0/(tenant_id)s
--adminurl http://控制器:8779/v1.0/(tenant_id)s
--region regionOne
```

11. 启动数据库服务并将其设置为随系统启动：

```
systemctl enable openstack-trove-api.service openstack-trove-taskmanager.service
openstack-trove-conductor.service
systemctl start openstack-trove-api.service openstack-trove-taskmanager.service
openstack-trove-conductor.service
```

## 验证数据服务的安装

验证数据库服务是否安装并正确配置，可以尝试执行 Trove 命令：

1. Source demo-openrc.sh 文件。

```
$ source ~/demo-openrc.sh
```

- ## 2. 获取 Trove 实例列表：

```
$ trove list
```

您应该看到类似如下的输出：

| id | name | datastore | datastore_version | status | flavor_id | size |
|----|------|-----------|-------------------|--------|-----------|------|
|----|------|-----------|-------------------|--------|-----------|------|

3. 假设您已经创建了一个您要使用的数据库类型的镜像，并已经更新数据存储以使用该镜像，您现在可以创建一个 Trove 实例（数据库）了。要实现创建，请使用 trove 的 create 命令。

这个示例会向您展示如何创建 MySQL 5.5 数据库：

```
$ trove create 名称2 --size=2 --databases DBNAME
--users USER:密码 --datastore_version mysql-5.5
--datastore mysql
```



在控制节点上安装数据处理服务：

- ### 1. 安装需要的包：

```
yum install openstack-sahara python-saharaclient
```

2. 修改配置文件 `/etc/sahara/sahara.conf`

- a. 首先，修改 `connection[database]` 部分的参数。这里所要提供的 URL 需要指向一个空的数据库。例如，连接到 MySQL 数据库的字符串为：

```
connection = mysql://sahara:SAHARA_DBPASS@控制器/sahara
```

- b. 切换到 [keystone\_authtoken] 部分。auth\_uri 参数需要指向 public 身份认证的 API 入口点。identity\_uri 需要指向 admin 身份认证的 API 入口点。例如：

```
auth_uri = http://控制器:5000/v2.0
identity_uri = http://控制器:35357
```

- c. 下一步，指定 `admin_user`、`admin_password` 和 `admin_tenant_name`。这些参数必须指定一个 `keystone` 用户，且这个用户在给定的租户中有 `admin` 角色。这些证书允许 `sahara` 认证和授权它的用户。
- d. 切换到 `[DEFAULT]` 部分。继续配置网络参数。如果您使用的是 `Neutron` 网络，那么请设置 `use_neutron=true`。否则，如果您使用的是 `nova-network`，请将给定的参数设置为 `false`。
- e. 这些设置对于第一次运行来说已经足够了。如果您为故障排除而想要增加日志，这里有两个配置文件中的参数可配置：`verbose` 和 `debug`。如果前一个参数设置为 `true`，`sahara` 会开始将 `INFO` 级别及以上的内容写入日志。如果 `debug` 参数设置为 `true`，`sahara` 会写入所有的日志，包括 `DEBUG` 的内容。

3. 如果您使用 MySQL 或 MariaDB 数据库的数据处理服务，您必须配置允许将大的作业文件保存到服务的内部数据库的最大的包的数目。

- a. 修改配置文件 `/etc/my.cnf` 并修改 `max allowed packet` 选项：

```
[mysqld]
max_allowed_packet = 256M
```

- b. 重启数据库服务：

```
systemctl restart mariadb.service
```

4. 完成下面的步骤以创建数据库：

- a. 以 root 用户身份通过数据库客户端连接到数据库服务：

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

- b. 创建 sahara 数据库：

```
CREATE DATABASE sahara;
```

- c. 为 sahara 数据库赋予适当的权限：

```
GRANT ALL PRIVILEGES ON sahara.* TO 'sahara'@'localhost' IDENTIFIED BY 'SAHARA_DBPASS';
GRANT ALL PRIVILEGES ON sahara.* TO 'sahara'@'%' IDENTIFIED BY 'SAHARA_DBPASS';
```

将其中的 SAHARA\_DBPASS 替换为您设置的一个合适的密码。

- d. 退出数据库客户端。
5. 创建数据库模式：

```
sahara-db-manage --config-file /etc/sahara/sahara.conf upgrade head
```

6. 您必须通过身份认证服务注册到数据处理服务中，这样其他的 OpenStack 服务才能找到它。注册服务并指定入口点：

```
$ keystone service-create --name sahara --type data_processing
--description "Data processing service"
$ keystone endpoint-create
--service-id $(keystone service-list | awk '/ sahara / {print $2}')
--publicurl http://控制器:8386/v1.1/% (tenant_id) s
--internalurl http://控制器:8386/v1.1/% (tenant_id) s
--adminurl http://控制器:8386/v1.1/% (tenant_id) s
--region regionOne
```

- ## 7. 启动 sahara 服务：

```
systemctl start openstack-sahara-all
```

8. (可选的)将数据处理服务设置为随系统启动

```
systemctl enable openstack-sahara-all
```

## 验证数据处理服务的安装

验证数据处理服务(sahara)已经安装并配置正确，可以尝试使用 sahara 客户端请求集群列表。

1. Source demo 租户凭证：

```
$ source demo-openrc.sh
```

- ## 2. 获取 sahara 集群列表：

```
$ sahara cluster-list
```

您应该看到类似如下的输出：

| name | id | status | node_count |
|------|----|--------|------------|
|      |    |        |            |
|      |    |        |            |
|      |    |        |            |

## 第 14 章 启动一个实例

# 目录

|                                        |     |
|----------------------------------------|-----|
| 使用 OpenStack 网络 (neutron) 启动一台实例 ..... | 128 |
| 使用传统网络 (nova-network) 启动一个实例 .....     | 134 |

An instance is a VM that OpenStack provisions on a compute node. This guide shows you how to launch a minimal instance using the CirrOS image that you added to your environment in the [第 4 章 添加镜像服务 \[39\]](#) chapter. In these steps, you use the command-line interface (CLI) on your controller node or any system with the appropriate OpenStack client libraries. To use the dashboard, see the [OpenStack User Guide](#).

Launch an instance using [OpenStack Networking \(neutron\)](#) or [legacy networking \(nova-network\)](#). For more information, see the [OpenStack User Guide](#).



## 注意

这些步骤关系到一些前面章节中所创建的组件，您必须调整某些值，如 IP 地址，以匹配您的环境配置。

## 使用 OpenStack 网络 (neutron) 启动一台实例

## 生成一个密钥对

大部分的云镜像支持 公钥认证 而不是传统的用户名/密码认证的方式。在启动实例之前，您必须生成一个公钥/私钥对使用 `ssh-keygen` 并将公钥添加到您的 OpenStack 环境中。

1. Source demo 租户凭证：

```
$ source demo-openrc.sh
```

- ## 2. 生成密钥对：

```
$ ssh-keygen
```

- ### 3. 添加公钥到您的 OpenStack 环境中：

```
$ nova keypair-add --pub-key ~/.ssh/id_rsa.pub demo-key
```



## 注意

这个命令执行后没有输出。

- #### 4. 验证公钥的添加：

```
$ nova keypair-list
```

| Name     | Fingerprint                                     |
|----------|-------------------------------------------------|
| demo-key | 6c:74:ec:3a:08:05:4e:9e:21:22:a6:dd:b2:62:b8:28 |



## 启动一台实例

启动一台实例，您必须至少指定一个类型、镜像名称、网络、安全组、密钥和实例名称。

1. 一个实例指定了虚拟机资源的大致分配，包括处理器、内存和存储。

列出可用类型：

| \$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        |           |  |  |

您的第一台实例使用的是 `m1.tiny` 类型。



## 注意

您也可以以 ID 引用类型。

2. 列出可用镜像：

```
$ nova image-list
```

| ID                                   | Name                | Status | Server |
|--------------------------------------|---------------------|--------|--------|
| acafc7c0-40aa-4026-9673-b879898e1fc2 | cirros-0.3.3-x86_64 | ACTIVE |        |

您的第一台实例使用的是 cirros-0.3.3-x86 64 镜像。

3. 列出可用网络：

```
$ neutron net-list
+-----+-----+-----+
| id | name | subnets |
+-----+-----+-----+
| 3c612b5a-d1db-498a-babb-a4c50e344cb1 | demo-net | 20bcd3fd-5785-41fe-ac42-55ff884e3180 192.168.1.0/24 |
| 9bce64a3-a963-4c05-bfcd-161f708042d1 | ext-net | b54a8d85-b434-4e85-a8aa-74873841a90d 203.0.113.0/24 |
+-----+-----+-----+
```

您的第一台实例使用的是 `demo-net` 租户网络。但是，您必须引用网络的 ID 而不是网络名称。

4. 列出可用的安全组：

```
$ nova secgroup-list
```

| Id | Name | Description |
|----|------|-------------|
|----|------|-------------|

|                                      |         |         |
|--------------------------------------|---------|---------|
| ad8d4ea5-3cad-4f7d-b164-ada67ec59473 | default | default |
|--------------------------------------|---------|---------|

您的第一台实例将使用 `default` 安全组。默认情况下，这个安全组生成了阻塞远程访问实例的防火墙规则。如果您想允许远程访问实例，启动它然后[配置远程访问](#)。

## 5. 启动实例：

将其中的 DEMO\_NET\_ID 替换为 demo-net 租户网络的 ID。

```
$ nova boot --flavor ml.tiny --image cirros-0.3.3-x86_64 --nic net-id=DEMO_NET_ID
--security-group default --key-name demo-key demo-instance1
```

| Property                             | Value                                                      |
|--------------------------------------|------------------------------------------------------------|
| OS-DCF:diskConfig                    | MANUAL                                                     |
| OS-EXT-AZ:availability_zone          | nova                                                       |
| OS-EXT-STS:power_state               | 0                                                          |
| OS-EXT-STS:task_state                | scheduling                                                 |
| OS-EXT-STS:vm_state                  | building                                                   |
| OS-SRV-USG:launched_at               | -                                                          |
| OS-SRV-USG:terminated_at             | -                                                          |
| accessIPv4                           |                                                            |
| accessIPv6                           |                                                            |
| adminPass                            | vFW7Bp8PQGNo                                               |
| config_drive                         |                                                            |
| created                              | 2014-04-09T19:24:27Z                                       |
| flavor                               | ml.tiny (1)                                                |
| hostId                               |                                                            |
| id                                   | 05682b91-81a1-464c-8f40-8b3da7ee92c5                       |
| image                                | cirros-0.3.3-x86_64 (acafc7c0-40aa-4026-9673-b879898e1fc2) |
| key_name                             | demo-key                                                   |
| metadata                             | { }                                                        |
| name                                 | demo-instance1                                             |
| os-extended-volumes:volumes_attached | [ ]                                                        |
| progress                             | 0                                                          |
| security_groups                      | default                                                    |
| status                               | BUILD                                                      |
| tenant_id                            | 7cf50047f8df4824bc76c2fdf66d11ec                           |
| updated                              | 2014-04-09T19:24:27Z                                       |
| user_id                              | 0e47686e72114d7182f7569d70c519c9                           |

## 6. 检查实例的状态：

```
$ nova list
+-----+-----+-----+-----+-----+-----+
| ID | Name | Status | Task State | Power State | Networks |
+-----+-----+-----+-----+-----+-----+
| 05682b91-81a1-464c-8f40-8b3da7ee92c5 | demo-instance1 | ACTIVE | - | Running | demo-net=192.168.1.3 |
+-----+-----+-----+-----+-----+-----+
```

当您的实例完成创建过程时，状态会从 BUILD 变为 ACTIVE。

## 使用虚拟机控制台访问您的实例

- 获取一个实例的 Virtual Network Computing (VNC) 会话的 URL 并从 web 浏览器访问它：

```
$ nova get-vnc-console demo-instance1 novnc
```

| Type  | Url                                                                             |
|-------|---------------------------------------------------------------------------------|
| novnc | http://controller:6080/vnc_auto.html?token=2f6dd985-f906-4bfc-b566-e87ce656375b |



## 注意

如果您的 web 浏览器运行在一台没有解析 controller 主机名的主机上，您可以将 controller 替换为控制节点管理网络的 IP 地址。

CirrOS 镜像包含传统的用户名/密码认证方式并需在登录提示中提供这些认证。登录到 CirrOS 后，我们建议您验证使用 ping 验证网络的连通性。

验证 demo-net 租户的网络网关：

```
$ ping -c 4 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=0.357 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=0.473 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=0.504 ms
64 bytes from 192.168.1.1: icmp_req=4 ttl=64 time=0.470 ms

--- 192.168.1.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2998ms
rtt min/avg/max/mdev = 0.357/0.451/0.504/0.055 ms
```

验证 ext-net 的外部网络：

```
$ ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data:
64 bytes from 174.143.194.225: icmp_req=1 ttl=53 time=17.4 ms
64 bytes from 174.143.194.225: icmp_req=2 ttl=53 time=17.5 ms
64 bytes from 174.143.194.225: icmp_req=3 ttl=53 time=17.7 ms
64 bytes from 174.143.194.225: icmp_req=4 ttl=53 time=17.5 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 17.431/17.575/17.734/0.143 ms
```

## 远程访问实例

1. 添加规则到 default 安全组中：

- a. 允许 ICMP (ping) :

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

| IP Protocol | From Port | To Port | IP Range  | Source Group |
|-------------|-----------|---------|-----------|--------------|
| icmp        | -1        | -1      | 0.0.0.0/0 |              |

- b. 允许安全 shell (SSH) 的访问：

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

---

| IP Protocol | From Port | To Port | IP Range  | Source Group |
|-------------|-----------|---------|-----------|--------------|
| tcp         | 22        | 22      | 0.0.0.0/0 |              |

2. 在 ext-net 外部网络上创建一个 浮动 IP 地址：

```
$ neutron floatingip-create ext-net
Created a new floatingip:
+-----+-----+
|Field |Value |
+-----+-----+
fixed_ip_address	
floating_ip_address	203.0.113.102
floating_network_id	9bce64a3-a963-4c05-bfcd-161f708042d1
id	05e36754-e7f3-46bb-9eaa-3521623b3722
port_id	
router_id	
status	DOWN
tenant_id	7cf50047f8df4824bc76c2fdf66d11ec
+-----+-----+
```

- ### 3. 将浮动 IP 地址与您的实例关联：

```
$ nova floating-ip-associate demo-instance1 203.0.113.102
```



## 注意

这个命令执行后没有输出。

4. 检查这个浮动 IP 地址的状态：

```
$ nova list
+-----+-----+-----+-----+-----+-----+
| ID | Name | Status | Task State | Power State | Networks |
+-----+-----+-----+-----+-----+-----+
| 05682b91-81a1-464c-8f40-8b3da7ee92c5 | demo-instance1 | ACTIVE | - | Running | demo-net=192.168.1.3, 203.0.113.102 |
+-----+-----+-----+-----+-----+-----+
```

5. 在控制节点或其他主机的外部网络上使用 ping 验证网络的连通性：

```
$ ping -c 4 203.0.113.102
PING 203.0.113.102 (203.0.113.112) 56(84) bytes of data.
64 bytes from 203.0.113.102: icmp_req=1 ttl=63 time=3.18 ms
64 bytes from 203.0.113.102: icmp_req=2 ttl=63 time=0.981 ms
64 bytes from 203.0.113.102: icmp_req=3 ttl=63 time=1.06 ms
64 bytes from 203.0.113.102: icmp_req=4 ttl=63 time=0.929 ms

--- 203.0.113.102 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.929/1.539/3.183/0.951 ms
```

6. 通过控制节点或任意外部网络上的主机使用 SSH 访问您的实例：

```
$ ssh cirros@203.0.113.102
The authenticity of host '203.0.113.102 (203.0.113.102)' can't be established.
RSA key fingerprint is ed:05:e9:c7:52:a0:ff:83:68:94:c7:d1:f2:f8:e2:e9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '203.0.113.102' (RSA) to the list of known hosts.
```

\$



## 注意

如果您的主机在之前的步骤中没有包含所创建的公钥或私钥对，SSH 会提示使用 `cirros` 用户相关的密码。

## 附加一个块设备存储的卷到您的实例上

如果您的环境包含块设备存储服务，您可以附加一个卷给实例。

1. Source demo 租户凭证：

```
$ source demo-openrc.sh
```

2. 列出卷：

\$ nova volume-list

| ID                                   | Status    | Display Name | Size | Volume Type | Attached to |
|--------------------------------------|-----------|--------------|------|-------------|-------------|
| 158bea89-07db-4ac2-8115-66c0d6a4bb48 | available | demo-volume1 | 1    | None        |             |

3. 附件卷 demo-volume1 给 demo-instance1 实例：

```
$ nova volume-attach demo-instance1 158bea89-07db-4ac2-8115-66c0d6a4bb48
```

|          |                                      |
|----------|--------------------------------------|
| Property | Value                                |
| device   | /dev/vdb                             |
| id       | 158bea89-07db-4ac2-8115-66c0d6a4bb48 |
| serverId | 05682b91-81a1-464c-8f40-8b3da7ee92c5 |
| volumeId | 158bea89-07db-4ac2-8115-66c0d6a4bb48 |



## 注意

您必须使用实例的 ID 来关联卷，不能使用实例名称。

4. 列出卷：

\$ nova volume-list

| ID                                   | Status | Display Name | Size | Volume Type | Attached to |
|--------------------------------------|--------|--------------|------|-------------|-------------|
| 158bea89-07db-4ac2-8115-66c0d6a4bb48 | in-use | demo-volume1 | 1    | None        |             |
| 05682b91-81a1-464c-8f40-8b3da7ee92c5 |        |              |      |             |             |

卷 demo-volume1 的状态应该显示被 ID 为 demo-instance1 的实例所 in-use 使用。

5. 使用 SSH 通过控制节点或任意主机的外部网络访问实例，并使用 `fdisk` 命令来验证卷是否以 `/dev/vdb` 的块设备存储存在：

```
$ ssh cirros@203.0.113.102
$ sudo fdisk -l
```



## 启动一台实例

启动一台实例，您必须至少指定一个类型、镜像名称、网络、安全组、密钥和实例名称。

1. 一个实例指定了虚拟机资源的大致分配，包括处理器、内存和存储。

列出可用类型：

| 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        |           |

您的第一台实例使用的是 `m1.tiny` 类型。



## 注意

您也可以以 ID 引用类型。

2. 列出可用镜像：

```
$ nova image-list
```

| ID                                   | Name                | Status | Server |
|--------------------------------------|---------------------|--------|--------|
| acafc7c0-40aa-4026-9673-b879898e1fc2 | cirros-0.3.3-x86_64 | ACTIVE |        |

您的第一台实例使用的是 `cirros-0.3.3-x86_64` 镜像。

3. 列出可用网络：



## 注意

您必需 `source admin` 租户的凭证来执行该步骤，然后再 `source demo` 租户的凭证来执行剩下的步骤。

```
$ source admin-openrc.sh
```

```
$ nova net-list
```

| ID                                   | Label    | CIDR            |  |
|--------------------------------------|----------|-----------------|--|
| 7f849be3-4494-495a-95a1-0f99ccb884c4 | demo-net | 203.0.113.24/29 |  |

您的第一台实例使用的是 demo-net 租户网络。但是，您必须引用网络的 ID 而不是网络名称。

4. 列出可用的安全组：

```
$ nova secgroup-list
```

| Id                                   | Name    | Description |
|--------------------------------------|---------|-------------|
| ad8d4ea5-3cad-4f7d-b164-ada67ec59473 | default | default     |

您的第一台实例使用的是 `default` 安全组。默认情况下，这个安全组实现了一个防火墙，会阻塞远程对实例的访问。如果您想要授权远程访问实例，启动它然后 [配置远程访问](#)。

## 5. 启动实例：

将其中的 DEMO NET ID 替换为 demo-net 租户网络的 ID。

```
$ nova boot --flavor m1.tiny --image cirros-0.3.3-x86_64 --nic net-id=DEMO_NET_ID
--security-group default --key-name demo-key demo-instance1
```

| Property                             | Value                                                      |
|--------------------------------------|------------------------------------------------------------|
| OS-DCF:diskConfig                    | MANUAL                                                     |
| OS-EXT-AZ:availability_zone          | nova                                                       |
| OS-EXT-STS:power_state               | 0                                                          |
| OS-EXT-STS:task_state                | scheduling                                                 |
| OS-EXT-STS:vm_state                  | building                                                   |
| OS-SRV-USG:launched_at               | -                                                          |
| OS-SRV-USG:terminated_at             | -                                                          |
| accessIPv4                           |                                                            |
| accessIPv6                           |                                                            |
| adminPass                            | ThZqrg7ach78                                               |
| config_drive                         |                                                            |
| created                              | 2014-04-10T00:09:16Z                                       |
| flavor                               | m1.tiny (1)                                                |
| hostId                               |                                                            |
| id                                   | 45ea195c-c469-43eb-83db-1a663bbad2fc                       |
| image                                | cirros-0.3.3-x86_64 (acafc7c0-40aa-4026-9673-b879898e1fc2) |
| key_name                             | demo-key                                                   |
| metadata                             | {}                                                         |
| name                                 | demo-instance1                                             |
| os-extended-volumes:volumes_attached | []                                                         |
| progress                             | 0                                                          |
| security_groups                      | default                                                    |
| status                               | BUILD                                                      |
| tenant_id                            | 93849608fe3d462ca9fa0e5dbfd4d040                           |
| updated                              | 2014-04-10T00:09:16Z                                       |
| user_id                              | 8397567baf4746cca7a1e608677c3b23                           |

### 6. 检查实例的状态：

\$ nova list

| ID                                   | Name | Status         | Task State | Power State | Networks              |
|--------------------------------------|------|----------------|------------|-------------|-----------------------|
| 45ea195c-c469-43eb-83db-1a663bbad2fc |      | demo-instance1 | ACTIVE     | Running     | demo-net=203.0.113.26 |

当您的实例完成创建过程时，状态会从 BUILD 变为 ACTIVE。



## 使用虚拟机控制台访问您的实例

- 获取一个实例的 Virtual Network Computing (VNC) 会话的 URL 并从 web 浏览器访问它：

```
$ nova get-vnc-console demo-instance1 novnc
```

| Type  | Url                                                                             |
|-------|---------------------------------------------------------------------------------|
| novnc | http://controller:6080/vnc_auto.html?token=2f6dd985-f906-4bfc-b566-e87ce656375b |



## 注意

如果您的 web 浏览器运行在一台没有解析 controller 主机名的主机上，您可以将 controller 替换为控制节点管理网络的 IP 地址。

CirrOS 镜像包含传统的用户名/密码认证方式并需在登录提示中提供这些这些认证。登录到 CirrOS 后，我们建议您验证使用ping验证网络的连通性。

验证 demo-net 网络：

```
$ ping -c 4 openstack.org
PING openstack.org (174.143.194.225) 56(84) bytes of data:
64 bytes from 174.143.194.225: icmp_req=1 ttl=53 time=17.4 ms
64 bytes from 174.143.194.225: icmp_req=2 ttl=53 time=17.5 ms
64 bytes from 174.143.194.225: icmp_req=3 ttl=53 time=17.7 ms
64 bytes from 174.143.194.225: icmp_req=4 ttl=53 time=17.5 ms

--- openstack.org ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 17.431/17.575/17.734/0.143 ms
```

## 远程访问实例

1. 添加规则到 default 安全组中：

- a. 允许 ICMP (ping) :

| IP Protocol | From Port | To Port | IP Range  | Source Group |
|-------------|-----------|---------|-----------|--------------|
| icmp        | -1        | -1      | 0.0.0.0/0 |              |

- b. 允许安全 shell (SSH) 的访问：

| IP Protocol | From Port | To Port | IP Range  | Source Group |
|-------------|-----------|---------|-----------|--------------|
| tcp         | 22        | 22      | 0.0.0.0/0 |              |

2. 在控制节点或其他主机的外部网络上使用 ping 验证网络的连通性：

```
$ ping -c 4 203.0.113.26
PING 203.0.113.26 (203.0.113.26) 56(84) bytes of data.
```

```
64 bytes from 203.0.113.26: icmp_req=1 ttl=63 time=3.18 ms
64 bytes from 203.0.113.26: icmp_req=2 ttl=63 time=0.981 ms
64 bytes from 203.0.113.26: icmp_req=3 ttl=63 time=1.06 ms
64 bytes from 203.0.113.26: icmp_req=4 ttl=63 time=0.929 ms

--- 203.0.113.26 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3002ms
rtt min/avg/max/mdev = 0.929/1.539/3.183/0.951 ms
```

3. 通过控制节点或任意外部网络上的主机使用 SSH 访问您的实例：

```
$ ssh cirros@203.0.113.26
The authenticity of host '203.0.113.26 (203.0.113.26)' can't be established.
RSA key fingerprint is ed:05:e9:e7:52:a0:ff:83:68:94:c7:d1:f2:f8:e2:e9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '203.0.113.26' (RSA) to the list of known hosts.
$
```



## 注意

如果您的主机在之前的步骤中没有包含所创建的公钥或私钥对，SSH 会提示使用 cirros 用户相关的密码。

## 附加一个块设备存储的卷到您的实例上

如果您的环境包含块设备存储服务，您可以附加一个卷给实例。

1. Source demo 租户凭证：

```
$ source demo-openrc.sh
```

2. 列出卷：

```
$ nova volume-list
```

| ID                                   | Status    | Display Name | Size | Volume Type | Attached to |
|--------------------------------------|-----------|--------------|------|-------------|-------------|
| 158bea89-07db-4ac2-8115-66c0d6a4bb48 | available | demo-volume1 | 1    | None        |             |

3. 附件卷 demo-volume1 给 demo-instance1 实例：

```
$ nova volume-attach demo-instance1 158bea89-07db-4ac2-8115-66c0d6a4bb48
```

| Property | Value                                |
|----------|--------------------------------------|
| device   | /dev/vdb                             |
| id       | 158bea89-07db-4ac2-8115-66c0d6a4bb48 |
| serverId | 45ea195c-c469-43eb-83db-1a663bbad2fc |
| volumeId | 158bea89-07db-4ac2-8115-66c0d6a4bb48 |



## 注意

您必须使用实例的 ID 来关联卷，不能使用实例名称。

4. 列出卷：

\$ nova volume-list

| ID                                   | Status | Display Name | Size | Volume Type | Attached to                          |
|--------------------------------------|--------|--------------|------|-------------|--------------------------------------|
| 158bea89-07db-4ac2-8115-66c0d6a4bb48 | in-use | demo-volume1 | 1    | None        | 45ea195c-c469-43eb-83db-1a663bbad2fc |

卷 demo-volume1 的状态应该显示被 ID 为 demo-instance1 的实例所 in-use 使用。

5. 使用 SSH 通过控制节点或任意主机的外部网络访问实例，并使用 `fdisk` 命令来验证卷是否以 `/dev/vdb` 的块设备存储存在：

```
$ ssh cirros@203.0.113.102
$ sudo fdisk -l

Disk /dev/vda: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

 Device Boot Start End Blocks Id System
/dev/vda1 * 16065 2088449 1036192+ 83 Linux

Disk /dev/vdb: 1073 MB, 1073741824 bytes
16 heads, 63 sectors/track, 2080 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

Disk /dev/vdb doesn't contain a valid partition table
```



## 注意

您必须创建一个分开的表和文件系统以使用卷。

If your instance does not launch or seem to work as you expect, see the [OpenStack Operations Guide](#) for more information or use one of the [many other options](#) to seek assistance. We want your environment to work!

## 附录 A. 保留的用户ID

OpenStack 保留特定的用户 ID 来运行特定的服务和自己的特定文件。这些用户 ID 是根据发行版的包设置的。下面的表格是一个预览。

表 A.1. 保留的用户ID

| 名称         | 描述                        | ID        |
|------------|---------------------------|-----------|
| ceilometer | OpenStack Ceilometer 守护进程 | 166       |
| cinder     | OpenStack Cinder 守护进程     | 165       |
| glance     | OpenStack Glance 守护进程     | 161       |
| heat       | OpenStack Heat 守护进程       | 187       |
| keystone   | OpenStack Keystone 守护进程   | 163       |
| neutron    | OpenStack Neutron 守护进程    | 164       |
| nova       | OpenStack Nova 守护进程       | 162       |
| swift      | OpenStack Swift 守护进程      | 160       |
| trove      | OpenStack Trove 守护进程      | 在包安装过程中分配 |

每个用户都属于一个和用户同名的用户组。

## 附录 B. 社区支持

# 目录

|                             |     |
|-----------------------------|-----|
| 文档 .....                    | 141 |
| 问答论坛 .....                  | 142 |
| OpenStack 邮件列表 .....        | 142 |
| OpenStack 维基百科 .....        | 142 |
| Launchpad的Bug区 .....        | 143 |
| The OpenStack 在线聊天室频道 ..... | 144 |
| 文档反馈 .....                  | 144 |
| OpenStack分发包 .....          | 144 |

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](https://docs.openstack.org).

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

以下书籍解释了如何安装一个基于OpenStack云及其相关的组件

- 基于openSUSE 13.1 或 SUSE Linux Enterprise Server 11 SP3的安装向导
- 基于Red Hat Enterprise Linux 7, CentOS 7, 以及 Fedora 20的安装向导
- 基于Ubuntu 14.04的安装向导

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
- 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：



T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

- T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -

- T - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo - DRAFT - Kilo -





# 术语表

## API

## 应用程序接口

## 块存储

OpenStack核心项目，它管理卷、卷快照，以及卷类型。块存储的项目名称叫做cinder。

## CirrOS

在云环境（例如OpenStack）中用于测试镜像,按照最小的Linux发行版来设计。

## cloud controller node

A node that runs network, volume, API, scheduler, and image services. Each service may be broken out into separate nodes for scalability or availability.

# 数据库

An integrated project that provide scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines. The project name of Database Service is trove.

## DHCP

Dynamic Host Configuration Protocol, A network protocol that configures devices that are connected to a network so that they can communicate on that network by using the Internet Protocol (IP). The protocol is implemented in a client-server model where DHCP clients request configuration data, such as an IP address, a default route, and one or more DNS server addresses from a DHCP server.

## DNS

Domain Name Server. A hierarchical and distributed naming system for computers, services, and resources connected to the Internet or a private network, Associates a human-friendly names to IP addresses.

## dnsmasq

Daemon that provides DNS, DHCP, BOOTP, and TFTP services for virtual networks.

extended attributes (xattr)

File system option that enables storage of additional information beyond owner, group, permissions, modification time, and so on. The underlying Object Storage file system must support extended attributes.

firewall

Used to restrict communications between hosts and/or nodes, implemented in Compute using iptables, arptables, ip6tables, and etables.

## flat network

Virtual network type that uses neither VLANs nor tunnels to segregate tenant traffic. Each flat network typically requires a separate underlying physical interface defined by bridge mappings. However, a flat network can contain multiple subnets.

## 网关

An IP address, typically assigned to a router, that passes network traffic between different networks.

generic receive offload (GRO)

Feature of certain network interface drivers that combines many smaller received packets into a large packet before delivery to the kernel IP stack.

generic routing encapsulation (GRE)

Protocol that encapsulates a wide variety of network layer protocols inside virtual point-to-point links.

hypervisor

Software that arbitrates and controls VM access to the actual underlying hardware.

IaaS

基础设施即服务。IaaS是一种配置模式，将数据中心的物理组件，如存储、硬件、服务器以及网络等以组织外包的方式提供。服务运营商提供设备，负责机房以及操作和维护。用户只需要按需使用并付费即可。IaaS是云服务模式的一种。

ICMP

Internet Control Message Protocol, used by network devices for control messages. For example, ping uses ICMP to test connectivity.

## Identity Service

The OpenStack core project that provides a central directory of users mapped to the OpenStack services they can access. It also registers endpoints for OpenStack services. It acts as a common authentication system. The project name of the Identity Service is keystone.

## 镜像服务

OpenStack核心项目之一，提供发现、注册和交付磁盘和服务器的镜像的服务。项目的名称叫 glance。

interface

A physical or virtual device that provides connectivity to another device or medium.

Internet protocol (IP)

Principal communications protocol in the internet protocol suite for relaying datagrams across network boundaries.

ipset

Extension to iptables that allows creation of firewall rules that match entire "sets" of IP addresses simultaneously. These sets reside in indexed data structures to increase efficiency, particularly on systems with a large quantity of rules.

iptables

Used along with arptables and ebtables, iptables create firewalls in Compute. iptables are the tables provided by the Linux kernel firewall (implemented as different Netfilter modules) and the chains and rules it stores. Different kernel modules and programs are currently used for different protocols: iptables applies to IPv4, ip6tables to IPv6, arptables to ARP, and ebtables to Ethernet frames. Requires root privilege to manipulate.

iSCSI

The SCSI disk protocol tunneled within Ethernet, supported by Compute, Object Storage, and Image Service.

jumbo frame

Feature in modern Ethernet networks that supports frames up to approximately 9000 bytes.

load balancer

A load balancer is a logical device that belongs to a cloud account. It is used to distribute workloads between multiple back-end systems or services, based on the criteria defined as part of its configuration.

## Logical Volume Manager (LVM)

Provides a method of allocating space on mass-storage devices that is more flexible than conventional partitioning schemes.

maximum transmission unit (MTU)

Maximum frame or packet size for a particular network medium. Typically 1500 bytes for Ethernet networks.

message broker

The software package used to provide AMQP messaging capabilities within Compute. Default package is RabbitMQ.

Metadata agent

OpenStack Networking agent that provides metadata services for instances.

multi-host

High-availability mode for legacy (nova) networking. Each compute node handles NAT and DHCP and acts as a gateway for all of the VMs on it. A networking failure on one compute node doesn't affect VMs on other compute nodes.

network namespace

Linux kernel feature that provides independent virtual networking instances on a single host with separate routing tables and interfaces. Similar to virtual routing and forwarding (VRF) services on physical network equipment.

网络

A virtual network that provides connectivity between entities. For example, a collection of virtual ports that share network connectivity. In Networking terminology, a network is always a layer-2 network.

## Network Address Translation (NAT)

The process of modifying IP address information while in transit. Supported by Compute and Networking.

网络

A core OpenStack project that provides a network connectivity abstraction layer to OpenStack Compute. The project name of Networking is neutron.

## Open vSwitch

Open vSwitch是一款产品级的，多层的虚拟交换机，基于开源Apache2.0许可证分发。被设计用于基于可编程扩展的大规模网络自动化，支持标准的管理接口和协议(例如NetFlow, sFlow, SPAN, RSPAN, CLI, LACP, 802.1ag)。

OpenStack

OpenStack是一个云操作系统，通过数据中心可控制大型的计算、存储、网络等资源池。所有的管理通过前端界面管理员就可以完成，同样也可以通过web接口让最终用户部署资源。OpenStack是一个开放源代码的项目，基于Apeche许可证2.0发布。

path MTU discovery (PMTUD)

Mechanism in IP networks to detect end-to-end MTU and adjust packet size accordingly.

plug-in

Software component providing the actual implementation for Networking APIs, or for Compute APIs, depending on the context.

## Quick EMUlator (QEMU)

QEMU is a generic and open source machine emulator and virtualizer.

One of the hypervisors supported by OpenStack, generally used for development purposes.

RESTful

A kind of web service API that uses REST, or Representational State Transfer. REST is the style of architecture for hypermedia systems that is used for the World Wide Web.

security group

A set of network traffic filtering rules that are applied to a Compute instance.

## Telemetry

An integrated project that provides metering and measuring facilities for OpenStack. The project name of Telemetry is ceilometer.

trove

OpenStack为提供数据库服务的应用程序项目。

virtual machine (VM)

An operating system instance that runs on top of a hypervisor. Multiple VMs can run at the same time on the same physical host.

## Virtual Network Computing (VNC)

Open source GUI and CLI tools used for remote console access to VMs. Supported by Compute.

virtual private network (VPN)

Provided by Compute in the form of cloudpipes, specialized instances that are used to create VPNs on a per-project basis.

XFS

High-performance 64-bit file system created by Silicon Graphics. Excels in parallel I/O operations and data consistency.