Red Hat OpenStack Platform 9
网络指南

OpenStack Networking 高级指南

OpenStack Team

Red Hat OpenStack Platform 9 网络指南

OpenStack Networking 高级指南

OpenStack Team
rhos-docs@redhat.com

法律通告

Copyright © 2016 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

摘要

执行常见 OpenStack Networking 任务的实用手册。

目录

序言

OpenStack Networking（*neutron*）是 Red Hat OpenStack Platform 9 中的一个软件定义的网络组件。

1. OPENSTACK NETWORKING 和 SDN

软件定义的网络（Software-defined Networking，简称 SDN）是一个用来描述虚拟网络功能的术语。虚拟环境中的服务器和物理服务器一样，仍然需要网络连接来接收和发送数据。SDN 通过把网络设备（如路由器和交换机）移到相同的虚拟空间来满足服务器对网络连接功能的要求。如果您熟悉基本的网络概念，则对了解网络功能的虚拟化不会有太多困难。

本文档中的第 1 部分向管理员介绍了基本的管理和故障排除任务，第 2 部分以实用手册的形式展示了 OpenStack Networking 的高级功能。如果您对网络的基本概念已有了解，则可以从第 1 部分开始阅读；如果您需要了解网络的基本概念，序言部分提供了相关的内容。

1.1. 本文档中包括的主题

- **序言** - 介绍了在一个大型机构中实施 SDN 可能要涉及到的“政治因素”，并包括了对常规网络概念的简单介绍。
- **第 1 部分** - 包括了常见管理任务和基本故障排除的步骤：
  - 添加和删除网络资源
  - 基本的网络故障排除
  - 租户网络的故障排除
- **第 2 部分** - 以实用手册的形式展示了 OpenStack Networking 的高级功能。包括：
  - 为虚拟路由配置 3 层高可用性功能
  - 配置 SR-IOV、DVR 以及其它 Neutron 功能

2. 虚拟网络的“政治因素”

通过 SDN，工程师可以基于 OpenStack 或 RHEV 在虚拟环境中实施虚拟路由器和交换机。另外，SDN 还会改变计算机间的数据包传输的处理方式。在没有使用 SDN 以前，处理网络数据的路由器和交换机都是物理设备，并需要使用各种网线把在这些物理设备进行连接。在使用 SDN 时，部署和操作这些网络功能只需点几个按钮。

在任何大型虚拟环境中使用 SDN 都可能会在不同部门间产生一些“矛盾”。虚拟系统工程师可能并不了解一些高级的网络概念，而他们却需要在云环境中管理虚拟路由器和交换机，并需要考虑 IP 地址分配、VLAN 隔离以及子网等与网络相关的问题。同时，网络工程师会发现，原来由他们所负责的技术成为了其它部门讨论的话题，并由此产生种种不适，甚至感到自己的工作稳定性受到了威胁。另外，这还会把故障排除的过程复杂化：当系统出现问题，或系统间无法进行网络连接时，虚拟系统工程师可能会需要网络工程师的介入来查看数据包是否在物理交换设备间出现了问题。

当把您的虚拟网络看做为物理网络的一个扩展部分时，以上问题可能就会变得简单。默认网关、路由、子网这些概念的含义和作用都是相同的，虚拟网络仍然使用 TCP/IP、VLAN 和 MAC 地址。通常情况下，虚拟交换机是基于物理交换机上的 VLAN 配置的，它们只是物理网络的一个扩展。

除了做好各部门间的协调外，还可以使用一些已有技术来帮助解决以上问题。例如，Cisco’s Nexus 系列产品允许 OpenStack 操作者部署运行已被大众熟悉的 Cisco NX-OS 的虚拟路由器。这样，网络

工程师就可以使用和登录到 Cisco 物理网络设备同样的方式登录并管理虚拟网络端口。如果网络工程师不需要管理虚拟网络，在开始设计虚拟网络时邀请他们介入也会是一个好的做法。OpenStack 节点需要使用物理网络架构、IP 地址需要被分配、物理交换机的接口需要进行配置来支持 VLAN。另外，除了故障排除外，在其它一些情况下也需要虚拟系统团队和网络团队的合作。例如，在调整虚拟机的 MTU 大小时，包括虚拟和物理交换机和路由器在内的所有端点都需要进行设置，这将需要两个团队间的密切合作。

在您的虚拟环境中，网络工程师仍然非常重要，特别是在您使用 SDN 技术时。因为 SDN 的复杂性，网络工程师在网络方面的专业知识对您系统的正常运行会大有益处。

第 1 章 网络概述

1.1. 网络如何工作

*Networking（网络）*这个术语被用来代表在一个计算机和另外一个计算机间进行的信息传递。在一个最基本的层面上，它是通过在分别安装了网卡（NIC）的两个机器间的网线上实现的。

注意

在 OSI 网络模型中，它是第 1 层。

如果您需要把多于两个的计算机进行连接，则需要添加一个交换机。企业级的交换机都会带有多个以太网端口用来连接额外的机器。现在，您就有了一个“局域网”（Local Area Network，简称 LAN）。

交换机位于 OSI 模型的第 2 层，它会比第 1 层更智能一些：每个 NIC 都有一个与硬件关联的唯一 MAC 地址，连接到同一个交换机的设备可以通过 MAC 地址相互进行通讯。交换机会管理一个记录了哪些 MAC 地址连接到哪个端口的信息列表。当一个设备需要和另外一个设备进行通讯时，交换机就可以根据这个列表中的信息正确处理网络数据，并调整用来记录 MAC 地址到端口的映射信息的 FIB（Forwarding Information Base）中的数据。

1.2. 连接两个 LAN

当您有两个分别运行于两个相互独立的交换机上的 LAN，并需要在它们之间共享信息时，您有两个选择可以实现这个功能：

- 选择一：使用“主干网线（trunk cable）”把两个交换机连接在一起。您需要把网线的两端分别插入到两个交换机的各一个端口中，然后把这些端口配置为主干端口（trunk port）。通过这些配置，您实际上把这两个交换机配置为一个大的逻辑交换机，连接到这两个交换机上的所有设备就可以相互进行通讯。这个方案会带来扩展性问题，当所连接的交换机数量太多时，相应的工作负载会成为一个问题。
- 选择二：使用网线把所有交换机连接到一个路由器设备，从而可以使路由器知道每个交换机的配置。连接到交换机的一端会被分配一个 IP 地址，它被称为默认网关。默认网关定义了，当所发送数据的目的地没有位于同一个 LAN 时，它需要先被发送到的地址。当数据被发送到这个默认网关所定义的地址上后，路由器会处理以后需要发送的地址。因为路由器知道哪些网络位于哪些端口的信息，它可以把数据发送到正确的目的地。路由功能位于 OSI 模型的第 3 层，IP 地址和子网也都在这一层上实现。

注意

互联网本身就是通过这个概念进行工作的。不同机构的独立网络通过交换机和路由器连接到一起，通过相关的默认网关，网络数据就可以被发送到正确的地址。

1.2.1. VLAN

使用 VLAN 可以对连接到同一个网络交换机上的计算机进行分离。换句话说，您可以通过配置端口来把交换机上的网络进行逻辑分割，从而形成多个“迷你” LAN，达到通过分隔网络数据实现数据安全的目的。例如，您的交换机共有 24 个端口，您可以把端口 1-6 配置为属于 **`VLAN200`**，把端口 7-18 配置为属于 **`VLAN201`**。连接到 **`VLAN200`** 中的计算机将和连接到 **`VLAN201`** 中的计算机完全分离，它们

将无法直接进行通讯。如果您需要这两个网络中的计算机进行相互通讯，则需要通过一个路由器（这和使用两个独立的物理交换机的情况一样）。在这种情况下，防火墙就可以被用来监控 VLAN 间的网络通讯。

1.2.2. 防火墙

防火墙和 IP 路由操作位于同一个 OSI 层中，但在管理基于 TCP/UDP 端口号的网络数据时它也可以在第 4 层工作。防火墙通常会和路由器处于相同的网段中来处理所有网络间的通讯。防火墙代表了一组预定义的规则，这些规则指定了哪些网络流量可以进入网络，以及哪些网络流量不能进入网络。这些规则可以高度“颗粒化”，例如：

"**VLAN200** 中的服务器只可以和 **VLAN201** 中的计算机进行通讯，并且只能在周四下午进行，并且只允许单向传输加密的网络数据（HTTPS）"。

为了保证这些规则的有效性，一些防火墙还会在第 5 层到第 7 层中执行深度包检测（Deep Packet Inspection，简称 DPI）。网络黑客通常会使用网络数据伪装的手法来对系统的安全性进行破坏，而 DPI 会检查数据包中的实际内容来确定它们的真实性，从而降低网络安全被破坏的可能性。

1.3. OpenStack 中的网络

以上概念在 OpenStack 中完全适用，只是在 OpenStack 中被称为软件定义的网络（SDN）。虚拟交换机（使用 Open vSwitch）和虚拟路由器（**l3-agent**）提供了在实例间进行通讯的功能，并可以为实例提供与外部网络进行通讯的能力。Open vSwitch 网桥会为实例分配虚拟端口，并可以使入站和出站的网络数据分散到相关的物理网络中。

1.4. 高级 OpenStack 网络概念

1.4.1. 第 3 层高可用性

OpenStack Networking 在一个中央化的网络节点（一个专门用于运行虚拟网络组件的物理服务器）上运行虚拟路由器。这些虚拟路由器被用来处理发送到虚拟机或从虚拟机发出的网络数据，因此它们是保证整个环境的网络正常运行的重要元素。因为物理机可能会出现故障（这种情况是*不可避免的*），所以当网络节点出现故障时，您的虚拟机将无法正常工作。

OpenStack Networking 使用*第 3 层高可用性*功能来帮助减少以上问题的出现。它通过实现业界标准的 *VRRP* 协议来保护虚拟路由器和浮动 IP 地址。在启用了*第 3 层高可用性*功能后，租户的虚拟路由器会被随机地在多个网络节点上进行分配，其中的一个路由器被指定为“活跃的”路由器，其它路由器作为“备份”路由器。当运行活跃路由器的网络节点出现故障时，备份路由器会作为活跃路由器使用。

注意

这里的"第 3 层"代表了 OSI 模型中的第 3 层，因此它会保护 OSI 模型第 3 层中提供的网络功能，包括路由和 IP 地址。

如需了解更多相关信息，请参阅"*配置第 3 层高可用性*"一章。

1.4.2. HAproxy

作为高可用性（HA）解决方案的一部分，HAProxy 是一个用来在所有可用控制器间分配连接的内建负载均衡程序（与 LBaaS 不同）。因为 HAProxy 是基于分配给每个服务的 UDP/TCP 端口号的，所以它需要在第 4 层中工作。VRRP 会决定连接要被初始发送到哪个 HAProxy 实例，然后 HAProxy 会把连接分配到后端的服务器。这种配置可以被认为是高可用性的负载均衡服务，这与以下介绍的

LBaaS 有所不同。

1.4.3. 负载均衡既服务（LBaaS）

负载均衡即服务（Load Balancing-as-a-Service，简称 LBaaS）使 OpenStack Networking 可以在相关的实例中平均分配入站的网络流量。这保证了入站的网络流量所带来的工作负载以可预测的方式在多个实例中共享，从而达到有效使用系统资源的目的。入站的网络流量以下面的方法之一进行分配：

- *轮转（Round robin）* - 在多个实例间均匀轮转分配请求。
- *源 IP（Source IP）* - 从同一个源 IP 地址发送的请求会被分配到同一个实例。
- *最少连接数量（Least connections）* - 把请求分配到有最少活跃连接数量的实例。

如需了解更多相关信息，请参阅*"配置负载均衡既服务（LBaaS）"*。

1.4.4. IPv6

OpenStack Networking 支持在租户网络中使用 IPv6，这意味着您可以为虚拟机动态地分配 IPv6 地址。OpenStack Networking 也可以集成物理路由器上的 SLAAC，从而使虚拟机可以从您已有的 DHCP 系统中获得 IPv6 地址。

如需了解更多相关信息，请参阅*"在租户网络中使用 IPv6"* 一章。

1.4.5. CIDR 格式

IP 地址通常首先会在子网的地址段中进行分配。例如，子网掩码为 **255.555.255.0** 的 IP 地址段 **192.168.100.0 - 192.168.100.255** 可以提供 **254** 个 IP 地址（第一个和最后一个地址会被保留）。

这些子网可以使用多种形式代表：

- 常规形式：子网地址的传统表示方法是使用网络地址和子网掩码。例如：
  - 网络地址：*192.168.100.0*
  - 子网掩码：*255.255.255.0*
- CIDR 格式：这种格式把子网掩码缩短为使用实际的位数进行代表。例如，**192.168.100.0/24**。其中的 **/24** 代表了 **255.255.255.0**（这个子网掩码的二进制形式共有 24 位 1）。在 **ifcfg-xxx** 脚本中可以使用 CIDR 格式来替代 **NETMASK** 值：

```
#NETMASK=255.255.255.0
PREFIX=24
```

第 2 章 OPENSTACK NETWORKING 概念

OpenStack Networking 具有一系列系统服务用来管理关键服务（如路由、DHCP 和元数据），这些服务构成了*网络节点*这一概念。网络节点是分配给一个物理服务器的概念性角色。通常，*网络节点*角色会分配给一个物理服务器，这个服务器专门用于处理与实例相关的第 3 层网络流量。在 OpenStack Networking 中，您可以为多个物理服务器分配这个角色，从而可以在出现硬件故障时实现容错的功能。如需了解更多相关信息，请参阅*第 3 层高可用性*一章。

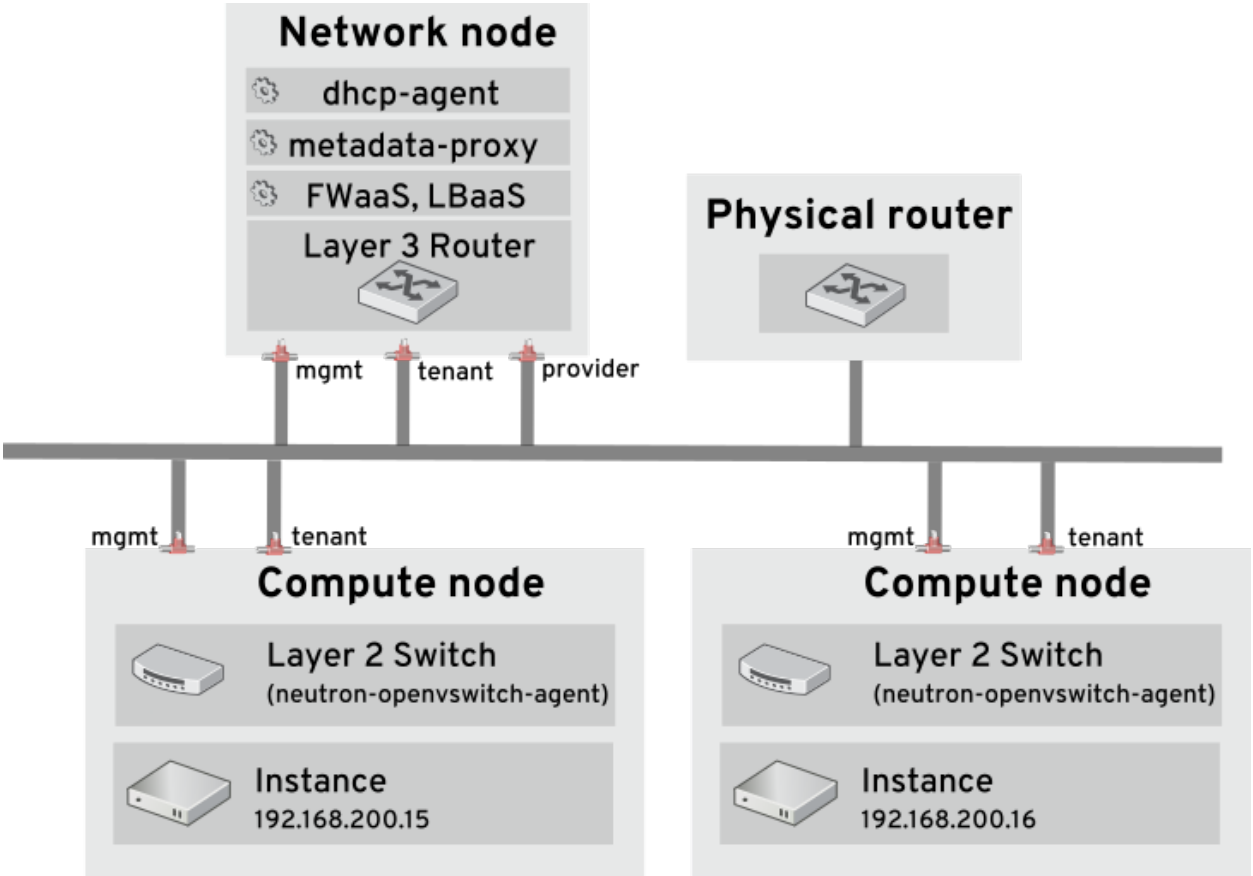
2.1. 安装 OpenStack Networking（neutron）

2.1.1. 安装方式

在 Red Hat OpenStack Platform 9（Mitaka）中，OpenStack Networking 组件作为 *RHEL OpenStack director* 实施的一部分被安装。请参阅 *RHEL OpenStack director* 安装指南来获取更多相关信息。

2.2. OpenStack Networking 示意图

这个示意图显示了一个 OpenStack Networking 实施示例，它带有一个专用的 OpenStack Networking 节点来提供 L3 路由和 DHCP 功能，并运行 *FWaaS* 和 *LBaaS* 服务。另外，还有两个 Compute 节点运行 Open vSwitch（*ovs-agent*），每个上面有两个物理网卡，一个用来处理租户网络数据，一个用来管理网络连接。OpenStack Networking 节点还带有第 3 个网卡用来处理供应商网络流量：



2.3. 安全组

安全组和相关规则根据网络数据的类型和传送方向过滤发送到特定 neutron 端口的网络流量，它在 Compute 实例的防火墙规则之外提供了额外的安全层。安全组是带有一个或多个安全规则的容器

项，它可以管理到多个 Compute 实例的网络流量。那些为浮动 IP 地址、OpenStack Networking LBaaS VIP、路由器接口以及实例所创建的端口都需要与一个安全组相关联。如果没有指定，端口会使用默认的安全组，它会拒绝所有入站的网络流量，而只允许出站的网络流量。如果需要，可以为默认的安全组添加额外的规则，或创建新的安全组。Open vSwitch、Linux Bridge、VMware NSX、NEC 和 Ryu 网络插件当前都支持安全组功能。

注意

和 Compute 的安全组不同，OpenStack Networking 的安全组是基于一个端口的，而不是基于一个实例的。

2.4. Open vSwitch

Open vSwitch（OVS）是一个用来替代 Linux 软件网桥的 SDN 虚拟交换机。OVS 为虚拟网络提供了网络数据交换服务，并支持业界标准的 *NetFlow*、*OpenFlow* 和 *sFlow*。Open vSwitch 同时也可以和使用第 2 层功能（*STP*、*LACP* 和 *802.1Q VLAN tagging*）的交换机进行集成。Open vSwitch 版本 *1.11.0-1.el6* 或更新版本支持 Open vSwitch 隧道（tunneling）功能。

注意

不要在 LACP 中使用基于 OVS 的绑定，因为这个配置会出现问题，而且不被支持。可以使用 **bond_mode=balance-slb** 作为替代来实现相关的功能。另外，您仍然可以使用带有 Linux 绑定的 LACP。

2.5. Modular Layer 2（ML2）

ML2 是从 OpenStack 的 Havana 发行版本开始提供的一个新的 OpenStack Networking 核心插件。ML2 取代了以前的单插件模式，而支持同时使用混合的网络技术。Open vSwitch 和 linuxbridge 插件已过时，并将不会包括在以后的发行版本中。它们的功能已在 ML2 中实现。

注意

ML2 是默认的 OpenStack Networking 插件，它使用 Open vSwitch 作为默认的机制驱动。

2.5.1. 使用 ML2 的原因

在以前，OpenStack Networking 只能使用在部署时所选择的插件。例如，运行 Open vSwitch 插件的系统只能使用 Open vSwitch，而不能使用其它插件（如 linuxbridge）。这会在一个有多种要求的环境中造成相关的限制。

2.5.2. ML2 网络类型

多种网络段类型可以同时运行。另外，这些网络段还可以使用 ML2 对多段网络的支持功能进行相互的连接。端口会自动与相关的网络段相关联，而不需要把它们与特定网络端绑定。根据所使用的驱动，ML2 支持以下网络端类型：* flat * GRE * local * VLAN * VXLAN

不同的*类型*驱动在 **ml2_conf.ini** 文件中的 ML2 项中指定：

```
[ml2]
type_drivers = local,flat,vlan,gre,vxlan
```

2.5.3. ML2 机制

现在，插件以带有通用代码库的机制被重新实现。使用这个机制，可以重复使用通用的代码，从而简化了维护和测试代码的复杂性。

注意

如需获得支持的机制驱动信息，请参阅 Release Notes。

不同的机制驱动在 **ml2_conf.ini** 文件中的 ML2 项中指定。例如：

```
[ml2]
mechanism_drivers = openvswitch,linuxbridge,l2population
```

注意

如果使用 Red Hat OpenStack Platform director 进行部署，则这些设置会由 puppet 进行管理，而不应该被手工修改。

2.6. OpenStack 的网络后端

Red Hat OpenStack 提供了两个不同的网络后端：Nova networking 和 OpenStack Networking（neutron）。Nova networking 在 OpenStack 技术的发展规划中已过时，但当前仍然被提供；而 OpenStack Networking 被看作为一个 OpenStack 发展规划中的核心 SDN 组件，并在不断被开发完善。请注意，当前还没有在 Nova networking 和 OpenStack Networking 间进行迁移的方案，如果您计划先使用 Nova networking，然后再升级到 OpenStack Networking，这一点将会是一个非常重要的考虑因素。当前，如需在这两种技术间进行迁移，则需手工进行，并需要一定的停机时间。

注意

Nova networking 不能使用 Red Hat OpenStack Platform Director 部署。

2.6.1. 使用 OpenStack Networking（neutron）

- 如果您需要一个覆盖网络（overlay network）解决方案：OpenStack Networking 支持使用 GRE 或 VXLAN 隧道（tunneling）来对虚拟机的网络流量进行隔离。当使用 GRE 或 VXLAN 时，在网络 fabric 上不需要 VLAN 的配置，对物理网络的唯一要求就是在节点间提供 IP 连接。另外，在理论上，VXLAN 或 GRE 对 ID 数量的限制是一千六百万（16,000,000），这远远超过了 802.1q VLAN ID 的最大数量 - 4094。Nova networking 是基于 802.1q VLAN 实现网络隔离的，它不支持 GRE 或 VXLAN 隧道功能。
- 如果不同的租户需要使用相互重叠的 IP 地址：OpenStack Networking 使用 Linux 内核的网络命名空间，它允许不同的租户在同一个 Compute 节点上使用相同的子网范围（例如 192.168.1/24），而不会造成相互间的影响。相反，Nova networking 只提供扁平化的网络拓扑结构，在为所有租户分配子网时必须十分小心。
- 如果需要一个经过红帽认证的第三方 OpenStack Networking 插件：在默认情况下，Red Hat OpenStack Platform 8 使用带有Open vSwitch（OVS）机制驱动的开源 ML2 内核插件。因为

OpenStack Networking 具有可插入式的构架，所以用户可以基于物理网络 fabric 和其它的网络要求，使用第三方的 OpenStack Networking 插件替代默认的 ML2/Open vSwitch 驱动配置。红帽一直不断致力于通过我们的合作伙伴计划为 Red Hat OpenStack Platform 认证更多的 OpenStack Networking 插件。如需了解更多与认证计划相关的信息，以及已认证的 OpenStack Networking 插件，请访问 http://marketplace.redhat.com。

- 如果需要 Firewall-as-a-service（FWaaS）或 Load-Balancing-as-a-service（LBaaS）：只有 OpenStack Networking 提供这些服务，Nova networking 不提供这些服务。租户可以在不通过管理员的情况下，使用 dashboard 来管理这些服务。

2.6.2. 使用 Nova Networking

- 如果您需要扁平化（untagged）网络或 VLAN（802.1q tagged）网络：这涉及到对可扩展性的要求（在理论上，最大的 VLAN ID 数量是 4094，而物理交换机实际支持的数量可能远远少于这个数量），以及对管理和部署的要求。在物理网络上需要进行特定的配置来在节点间分配 VLAN。
- 如果不同的租户不需要相互重叠的 IP 地址：这通常只适用于小的、私用的环境。
- 如果您不需要 SDN 解决方案，或不需要和物理网络 fabric 进行交互。
- 如果您不需要自服务的 VPN、防火墙或负载均衡服务。

2.7. L2 Population

L2 Population 驱动支持使用广播、多播和单播方式在覆盖网络中进行扩展。在默认情况下，Open vSwitch GRE 和 VXLAN 会复制广播到包括那些没有运行目标网络的每个代理。这种设计导致了大量的网络和处理负载。而 L2 Population 驱动使用了另外一种设计观念，它为 ARP 解析以及 MAC 地址学习功能提供了一个部分的网状网络（mesh），网络数据会被打包作为目标单波只发送到相关的代理上。为了启用 L2 population，您需要把它添加到机制驱动列表中。另外，您还需要最少启用一个隧道（tunneling）驱动（GRE、VXLAN 或两个都启用）。在 ml2_conf.ini 文件中添加适当的配置：

```
[ml2]
type_drivers = local,flat,vlan,gre,vxlan
mechanism_drivers = openvswitch,linuxbridge,l2population
```

在 ovs_neutron_plugin.ini 文件中启用 L2 population。这需要在所有运行 L2 agent 的节点上都启用：

```
[agent]
l2_population = True
```

2.8. OpenStack Networking 服务

OpenStack Networking 集成了多个组件来为您的环境提供网络功能：

2.8.1. L3 Agent

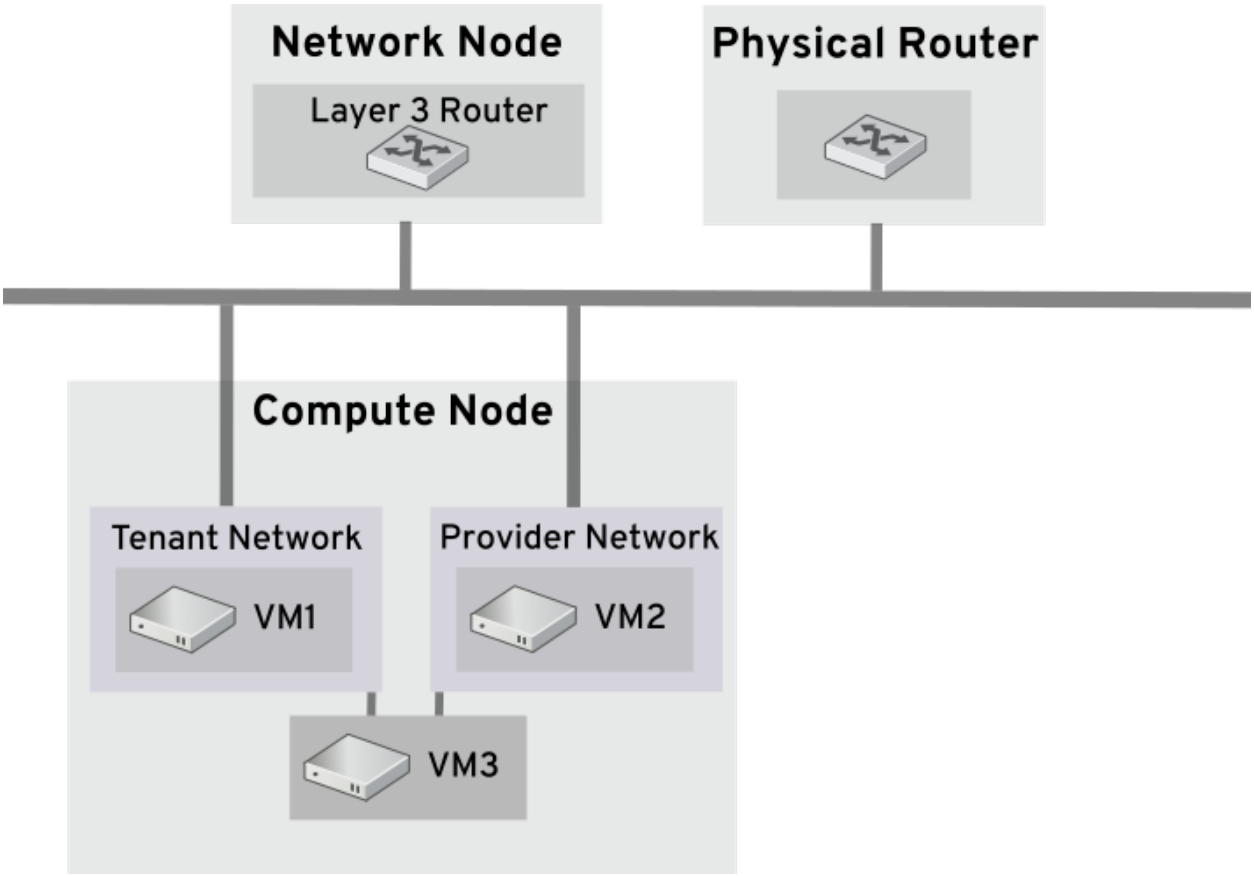
L3 agent 是 *openstack-neutron* 软件包的一部分，它的功能是作为一个虚拟的第 3 层路由器来处理网络流量，并为第 2 层网络提供网关服务。L3 agent 所在节点的网络接口不能带有手工配置的、连接到一个外部网络的 IP 地址。它需要具有一组可以被 OpenStack Networking 使用的、来自于外部网络的 IP 地址，这些 IP 地址会分配给用来连接内部网络和外部网络的路由器。这组地址的规模需要足够大来为每个路由器分配 IP 地址，并可以分配相关的浮动 IP。

- **DHCP Agent** - OpenStack Networking DHCP agent 可以为在网络中运行的虚拟机分配 IP 地址。如果这个 agent 被启用，并在创建子网时运行，则 DHCP 功能会在子网中默认启用。

- **Plug-in Agent** - 许多网络插件都可以利用它们自己的 agent（包括 Open vSwitch 和 Linux Bridge）。针对于特定插件的 agent 会在每个管理网络流量的节点上运行，包括所有计算节点，以及运行专用 agent（*neutron-dhcp-agent* 和 *neutron-l3-agent*）的节点。

2.9. 租户和供应商网络

下图展示了租户网络和供应商网络的类型，以及它们如何在整个 OpenStack Networking 拓扑中工作：



2.9.1. 租户网络

租户网络（Tenant network）是用户创建的、用来在项目中提供连接功能的网络，它们在默认情况下被完全分离，并且不会和其它项目进行共享。OpenStack Networking 支持以下网络类型：

- **Flat（扁平化）** - 所有实例都位于同一个网络中。这个网络可以被主机共享，并且没有 VLAN tagging 或其它网络分离功能。
- **VLAN（虚拟局域网）** - OpenStack Networking 允许用户使用 VLAN ID （802.1Q tagged）在物理网络中创建多个供应商（provider）或租户（tenant）网络。这将使实例可以和环境中的其它实例进行通讯，并可以和位于同一个第 2 层 VLAN 中的专用服务器、防火墙、负载均衡设备以及其它网络基础设施进行通讯。

注意

您也可以为租户网络配置 QoS 策略。如需了解更多相关信息，请参阅 第 10 章 *配置 QoS*。

2.9.2. VXLAN 和 GRE 隧道

VXLAN 和 GRE 使用网络覆盖（network overlay）来支持实例间的私人通讯。如果要进行到 GRE 或 VXLAN 租户网络外的网络通讯，则需要一个 OpenStack Networking 路由器。另外，如果需要连接到和外部网络（包括 Internet）直接连接的租户网络时，也需要一个路由器，路由器提供了使用浮动 IP 地址从一个外部网直接连接到实例的功能。

2.9.3. 供应商网络

供应商网络（Provider network）由 OpenStack 管理员创建，并直接映射到数据中心中存在的一个物理网络上。这个网络的网络类型包括 flat（untagged）和 VLAN（802.1Q tagged）。作为网络创建的一部分，供应商网络可以被租户共享。

2.9.3.1. 扁平化供应商网络

您可以使用扁平化（flat） 供应商网络直接把实例连接到外部网络。当有多个物理网络（例如，*physnet1* 和 *physnet2*），而且有独立的物理接口（*eth0 - > physnet1* 和 *eth1 → physnet2*），并需要把每个 Compute 节点和 Network 节点连接到它们的外部网络时使用这个网络类型。如果您需要使用在一个接口上的多个 vlan-tagged 接口连接到多个供应商网络，请参阅 第 7.2 节 “使用 VLAN 供应商网络”。

2.9.3.2. 配置控制器节点

1. 编辑 */etc/neutron/plugin.ini*（它是到 */etc/neutron/plugins/ml2/ml2_conf.ini* 文件的 symbolic link），把 **`flat`** 添加到已存在的值列表中，把 **`flat_networks`** 设置为 **`*`**：

```
type_drivers = vxlan,flat
flat_networks =*
```

2. 创建一个外部网络作为 flat 网络，并把它和配置的 *physical_network* 进行关联。如果把它作为一个共享网络创建（使用 **`--shared`**），将允许其它用户在其中直接创建实例。

```
neutron net-create public01 --provider:network_type flat --
provider:physical_network physnet1 --router:external=True --shared
```

3. 使用 **`neutron subnet-create`** 或 Dashboard 创建一个子网。

```
# neutron subnet-create --name public_subnet --enable_dhcp=False --
allocation_pool start=192.168.100.20,end=192.168.100.100 --
gateway=192.168.100.1 public 192.168.100.0/24
```

4. 重启 **`neutron-server`** 服务以使所做的改变生效：

```
systemctl restart neutron-server
```

2.9.3.3. 配置 Network 节点和 Compute 节点：

在 network 节点和 compute 节点上进行这些操作。这将把节点连接到外部网络，并使实例可以直接和外部网络进行通讯。

1. 创建一个外部网桥（br-ex），并为它添加一个相关联的端口（eth1）：

在 */etc/sysconfig/network-scripts/ifcfg-br-ex* 中创建外部网桥：

```
DEVICE=br-ex
TYPE=OVSBridge
DEVICETYPE=ovs
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

在 */etc/sysconfig/network-scripts/ifcfg-eth1* 中，把 **eth1** 配置为与 **br-ex** 相连：

```
DEVICE=eth1
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

重启节点或网络服务以使所做的修改生效。

2. 在 */etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini* 中配置物理网络，把网桥映射到相关的物理网络：

```
bridge_mappings = physnet1:br-ex
```

注意

如需了解更多与网桥相关的信息，请参阅 第 11 章 *配置网桥映射*。

3. 在 Network 节点和 Compute 节点上重启 *neutron-openvswitch-agent* 服务以使所做改变生效：

```
systemctl restart neutron-openvswitch-agent
```

2.9.3.4. 配置网络节点

1. 把 */etc/neutron/l3_agent.ini* 文件中的 **external_network_bridge =** 设为空值。

以前，当只使用一个网桥连接到外部网络时，OpenStack Networking 会使用 **external_network_bridge**。现在，这个值可以是一个空字符串，这将允许多个外部网桥，OpenStack Networking 会为每个到 **br-int** 的网桥创建一个 patch。

```
# Name of bridge used for external network traffic. This should be set
to
# empty value for the linux bridge
external_network_bridge =
```

2. 重启 **neutron-l3-agent** 以使所做的修改生效。

```
systemctl restart neutron-l3-agent
```

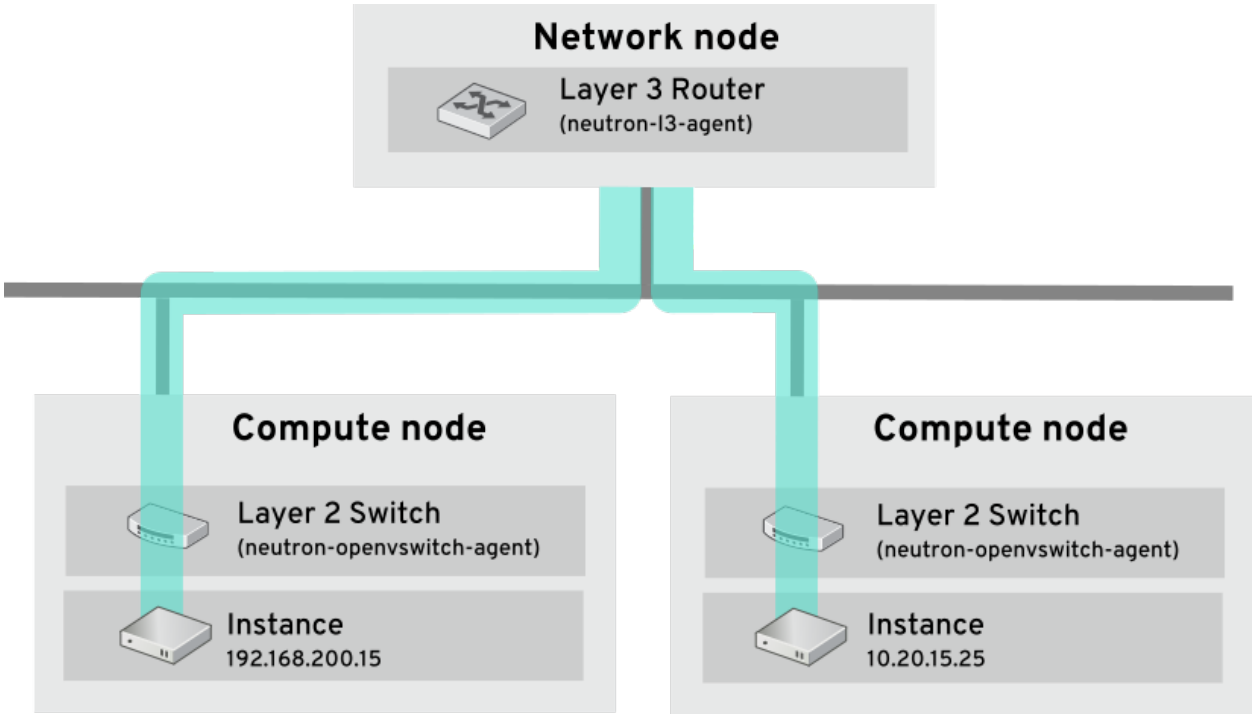
注意

如果有多个 flat 供应商网络，每个网络都需要有独立的物理网络接口和网桥来与外部网络连接。请正确配置 *ifcfg-** 脚本，并在 **`bridge_mappings`** 中设置网络映射（使用一个由逗号分隔的网络列表）。如需了解更多与网络映射相关的信息，请参阅第 11 章 *配置网桥映射*。

2.10. 第 2 层和第 3 层网络

在设计网络时，您需要考虑多数网络流量在哪里进行。因为在不同逻辑网络间传递数据需要经过路由器，所以在同一个网络中传递网络数据比在不同网络间传递数据的速度要快。

下图展示了在独立 VLAN 中的实例间进行网络数据传输的过程。

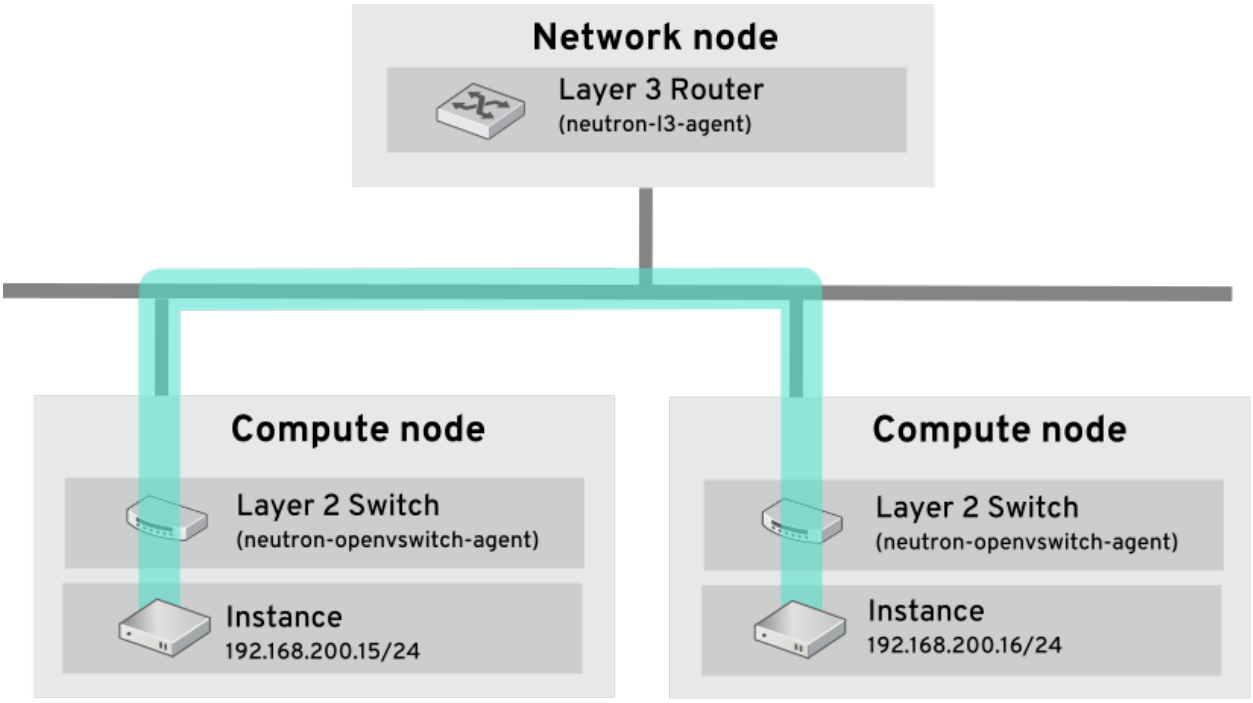


注意

即使在这种配置中使用高性能的硬件路由器，仍然会造成额外的速度延迟。

2.10.1. 尽量使用网络交换

网络数据交换的操作发生在更低的网络层上（第 2 层），因此它的速度比在第 3 层上的数据路由要快很多。在设计网络时，您需要考虑把经常要进行通讯的系统放置到临近的位置。例如，下图展示了一个包括了两个物理节点的交换网络，这就使其中两个实例可以在不需要进行网络路由的情况下就可以直接进行通讯。您可以注意到，实例共享相同的子网，这代表它们在同一个逻辑网络中：



为了使独立节点上的实例可以象在同一个逻辑网络中一样进行通讯，需要使用“隧道（tunnel）”技术，如 VXLAN 或 GRE。我们推荐您考虑调整端点上的 MTU 的设置，从而可以处理隧道头所包括的额外数据，否则网络性能可能会因为网络被分隔受到负面影响。另外，您还可以使用支持的、带有 VXLAN offload 功能的硬件来进一步提高 VXLAN 隧道的性能。如需了解这些硬件的完整列表，请参阅 https://access.redhat.com/articles/1390483

部分 I. 常见任务

常见的管理任务以及基本的故障排除步骤。

第 3 章 常见的管理任务

OpenStack Networking（neutron）是一个软件定义的 Red Hat OpenStack Platform 网络组件。通过虚拟网络的基本架构，实例和外部物理网络就可以进行网络连接。

本节介绍了常见的管理任务，如根据您的 Red Hat OpenStack Platform 部署添加、删除子网和路由器。

3.1. 创建网络

通过创建网络，您的实例间将可以进行通讯，并可以使用 DHCP 获得 IP 地址。网络也可以和您的 Red Hat OpenStack Platform 部署或其它部署中的外部网络（如物理网络）进行集成。通过集成，您的实例将可以和外部系统进行交流。如需了解把您的网络和外部物理网络进行集成的信息，请参阅“桥接物理网络”。

在创建网络时，一个重要因素您需要进行考虑：网络可以包括多个子网。这一点对于需要在同一个网络中提供相互独立的不同系统的环境非常重要。例如，您可以使 webserver 的网络流量只在一个子网中有效，而数据库网络流量在另外的子网中有效。每个子网都独立于其它子网，如果实例需要和其它子网进行通讯，必须经过一个网络路由器。您可以考虑把相互间需要进行大量网络通讯的系统放置在同一个子网中，这将会尽量避免网络路由，从而提高网络的性能。

1. 在 dashboard 中选择**项目 > 网络 > 网络**。

2. 点 **+创建网络**并设置以下项：

项	描述
网络名称	根据网络要实现的任务，为它取一个有意义的名称。如果您需要把它和一个外部 VLAN 进行集成，可以考虑把 VLAN ID 值加在名称中。例如：如果您在这个子网中提供 HTTP web 服务器，而它的 VLAN tag 是 122，则可以为它取名为 * webservers_122；如果您计划把这个网络作为私人网络使用，并不准备和外部网络进行连接，则可以为它取名为 * internal-only。
管理状态	指定这个网络是否马上有效。您可以创建一个网络，但使它处于 Down 状态（在逻辑上，这个网络已经存在，但还没有被激活）。如果您不想把新创建的网络马上加入到生产环境中，可以使用这个功能。

3. 点**下一步**按钮，在**子网**标签页中指定以下项：

项	描述

项	描述
创建子网	指定是否需要创建一个子网。例如，当您只希望创建一个网络结构，而并不需要进行网络连接时，可以不创建子网。
子网名	为子网输入一个适当的名称。
网络地址	输入 CIDR 格式的地址（一个包括 IP 地址范围和子网掩码的值）。其中的子网掩码值是通过计算用于进行子网掩码的位数决定的。例如，子网掩码 255.255.255.0 有 24 位，如需指定使用这个掩码的 IPv4 地址范围 192.168.122.0，使用 192.168.122.0/24。
IP 版本	指定 IP 版本，有效值是 IPv4 和 IPv6。*网络地址*项中设置的 IP 地址范围需要和这个值相匹配。
网关 IP	作为默认网关的路由器的 IP 地址。所有和外部网络进行通讯的数据都需要经过这个地址进行路由，它需要在网络地址项中指定的地址范围之内。例如，如果您的 CIDR 网络地址是 192.168.122.0/24，您的默认网关可以是 192.168.122.1。
禁用网关	禁用网络数据转发，使这个网络独立于其它环境。

4. 点**下一步**来设置 **DHCP** 选项：

- 启**用 DHCP** - 在这个子网中启用 DHCP 服务。DHCP 可以为您的实例自动分配 IP 设置。
- **IPv6 地址** - 如果创建一个 IPv6 网络，指定 IPv6 地址和其它额外信息是如何分配的。
  - **未指定选项** - 如果手工设置 IP 地址，或使用非 OpenStack 支持的方法分配地址，则使用这个选项。
  - **SLAAC (Stateless Address Autoconfiguration)** - 实例基于 OpenStack Networking 路由器发送的 RA（Router Advertisement）信息生成 IPv6 地址。使用这个配置的结果是，在创建 OpenStack Networking 子网时，ra_mode 和 address_mode 都被设置为 slaac。
  - **DHCPv6 stateful** - 实例从 OpenStack Networking DHCPv6 服务接收 IPv6 地址以及其它选项（如 DNS）。使用这个配置的结果是，在创建网络时，ra_mode 和 address_mode 都被设置为 dhcpv6-stateful。
  - **DHCPv6 stateless** - 实例基于 OpenStack Networking 路由器发送的 RA（Router Advertisement）信息生成 IPv6 地址。其它额外的选项（如 DNS）由 OpenStack Networking DHCPv6 服务分配。使用这个配置的结果是，在创建网络时，ra_mode set 和 address_mode 都被设置为 dhcpv6-stateless。

- **分配池** - DHCP 分配的 IP 地址范围。例如，192.168.22.100,192.168.22.100 代表范围内的所有 *up* 的地址都可以被分配。
- **DNS 域名解析服务器** - 网络中可用的 DNS 服务器的 IP 地址。DHCP 把这些地址分配给实例用来进行域名解析。
- **主机路由** - 静态主机路由。首先以 CIDR 的格式指定目标网络，然后指定下一个处理路由的地址。例如：192.168.23.0/24, 10.1.31.1。如果您需要为实例静态分配路由信息，请设置这个选项。

5. 点**创建**。

创建的网络可以在**网络**标签页中查看。您可以根据需要，通过点**编辑**来修改它的选项。现在，当您创建实例时，可以配置它来使用这个子网，并获得相应的 DHCP 选项。

3.2. 创建高级网络

当管理员通过**管理员**界面创建网络时，可以使用高级网络选项。这些选项定义了要使用的网络类型，并允许指定租户：

1. 在 dashboard 中，选择**管理员 > 网络 > 创建网络 > 项目**。在**项目**中选择一个项目来运行这个新网络。

2. 检查**供应商网络类型**中的选项：

- **Local** - 网络流量只在本地 Compute 主机上传输，它的实际效果是和所有外部网络隔离。
- **Flat** - 通讯数据被限制在一个单独的网络中，并可以和其它主机共享。没有 VLAN tagging 或其它网络分离的功能。
- **VLAN** - 使用代表了物理网络中的一个 VLAN 的 VLAN ID 来创建网络。这可以使实例和处于相同 2 层 VLAN 中的系统进行交流。
- **GRE** - 使用一个跨多个节点的网络覆盖（network overlay）来为实例间提供私人网络通讯。从网络覆盖出站的网络流量需要被路由。
- **VXLAN** - 和 GRE 类似，使用一个跨多个节点的网络覆盖（network overlay）来为实例间提供私人网络通讯。从网络覆盖出站的网络流量需要被路由。

点**创建网络**，检查项目的网络拓扑来确认网络已被成功创建。

3.3. 添加网络路由

为了使网络数据可以路由到您的新网络，或从新网络中路由出来，您需要把它的子网添加为一个已存在虚拟路由器的接口：

1. 在 dashboard 中选择**项目 > 网络 > 路由**。

2. 在**路由**列中选您的虚拟路由器名称，点 **+添加接口**。在子网列表中选择您的新子网的名称。您也可以为这个接口指定一个 IP 地址。

3. 点**增加接口**。

现在，您网络中的实例就可以和子网以外的系统进行交流。

3.4. 删除网络

在一些情况下，您可能需要删除以前创建的网络。为了成功删除一个网络，您需要首先删除或断开所有使用它的接口。以下介绍了在项目中删除网络的方法。

1. 在 dashboard 中，选择**项目 > 网络 > 网络**。删除和目标网络的子网相关联的所有路由接口。删除一个接口的步骤是：在**网**络列中点目标网络来找到需要删除的网络的 ID（在 ID 项中显示）。所有和这个网络相关联的子网都在它们的**网络 ID** 项中共享这个值。

2. 选择**项目 > 网络 > 路由**，在**路由**列表中选您的虚拟路由器的名称，并找到附加到需要删除的子网上的接口。您可以通过它的 IP 地址（网关的 IP 地址）来和其它接口区分开。另外，它的网络接口 ID 应该和上一步获得的 ID 相匹配。

3. 点接口的**删除接口**按钮。

选择项目 > 网络 > 网络，点您的网络名。点目标子网的删除子网按钮。

注意

如果您还不能删除子网，请检查它是否还被其它实例使用。

选**项目 > 网络 > 网络**，然后选项您需要删除的网络。提示时点**删除网络**，在下一个界面中再次确认。

3.5. 清除（**purge**）租户的网络

在 Red Hat OpenStack Platform 9 以前的版本中，在删除一个项目后，还存在分配给这个项目的资源，这包括网络、路由和端口。这些资源需要被手工删除，而且需要以正确的顺序删除。在 Red Hat OpenStack Platform 9 中，可以使用 **`neutron purge`** 命令来删除以前属于一个特定项目的所有 neutron 资源。

例如，在删除 **`test-project`** 前，需要 purge 它的 neutron 资源：

```
# openstack project list
+----------------------------------+--------------+
| ID                               | Name         |
+----------------------------------+--------------+
| 02e501908c5b438dbc73536c10c9aac0 | test-project |
| 519e6344f82e4c079c8e2eabb690023b | services     |
| 80bf5732752a41128e612fe615c886c6 | demo         |
| 98a2f53c20ce4d50a40dac4a38016c69 | admin        |
+----------------------------------+--------------+

# neutron purge 02e501908c5b438dbc73536c10c9aac0
Purging resources: 100% complete.
Deleted 1 security_group, 1 router, 1 port, 1 network.

# openstack project delete 02e501908c5b438dbc73536c10c9aac0
```

3.6. 创建子网

子网为相关的实例提供了网络连接功能。在每个实例的创建过程中，都需要被分配一个子网，您需要仔细考虑实例和子网的设置，以最好地满足您的网络连接需求。子网是在已存在的网络中创建的。一个需要考虑的重要因素是：OpenStack Networking 中的租户网络可以包括多个子网。这一点对于需要在同一个网络中提供相互独立的不同系统环境的情况非常重要。例如，您可以使 webserver 的网络

流量只在一个子网中有效，而数据库网络流量在另外的子网中有效。每个子网都独立于其它子网，如果实例需要和其它子网进行通讯，必须经过一个网络路由器。您可以考虑把相互间需要进行大量网络通讯的系统放置在同一个子网中，这将会尽量避免网络路由，从而提高网络的性能。

3.6.1. 创建一个新子网

在 dashboard 中，选择**项目 > 网络 > 网络**，然后在**网络**项中点您的网络名称。

1. 点**创建子网**，然后指定以下项。

项	描述
子网名	子网的名称
网络地址	CIDR 格式的地址（一个包括 IP 地址范围和子网掩码的值）。其中的子网掩码值是通过计算用于进行子网掩码的位数决定的。例如，子网掩码 255.255.255.0 有 24 位，如需指定使用这个掩码的 IPv4 地址范围 192.168.122.0，使用 192.168.122.0/24。
IP 版本	指定 IP 版本，有效值是 IPv4 和 IPv6。网络地址项中设置的 IP 地址范围需要和这个值相匹配。
网关 IP	作为默认网关的路由器的 IP 地址。所有和外部网络进行通讯的数据都需要经过这个地址进行路由，它需要在网络地址项中指定的地址范围之内。例如，如果您的 CIDR 网络地址是 192.168.122.0/24，您的默认网关可以是 192.168.122.1。
禁用网关	禁用网络数据转发，使这个网络独立于其它环境。

2. 点**下一步**来设置 DHCP 选项：

- 启**用 DHCP** - 在这个子网中启用 DHCP 服务。DHCP 可以为您的实例自动分配 IP 设置。
- **IPv6 地址** - 如果创建一个 IPv6 网络，指定 IPv6 地址和其它额外信息是如何分配的。
  - **未指定选项** - 如果手工设置 IP 地址，或使用非 OpenStack 支持的方法分配地址，则使用这个选项。
  - **SLAAC (Stateless Address Autoconfiguration)** - 实例基于 OpenStack Networking 路由器发送的 RA（Router Advertisement）信息生成 IPv6 地址。使用这个配置的结果是，在创建 OpenStack Networking 子网时，ra_mode 和 address_mode 都被设置为 slaac。
  - **DHCPv6 stateful** - 实例从 OpenStack Networking DHCPv6 服务接收 IPv6 地址以及其它选项（如 DNS）。使用这个配置的结果是，在创建网络时，ra_mode 和 address_mode 都被设置为 dhcpv6-stateful。

- **DHCPv6 stateless** - 实例基于 OpenStack Networking 路由器发送的 RA（Router Advertisement）信息生成 IPv6 地址。其它额外的选项（如 DNS）由 OpenStack Networking DHCPv6 服务分配。使用这个配置的结果是，在创建网络时，ra_mode set 和 address_mode 都被设置为 dhcpv6-stateless。

- **分配池** - DHCP 分配的 IP 地址范围。例如，192.168.22.100,192.168.22.100 代表范围内的所有 *up* 的地址都可以被分配。
- **DNS 域名解析服务器** - 网络中可用的 DNS 服务器的 IP 地址。DHCP 把这些地址分配给实例用来进行域名解析。
- **主机路由** - 静态主机路由。首先以 CIDR 的格式指定目标网络，然后指定下一个处理路由的地址。例如：192.168.23.0/24, 10.1.31.1。如果您需要为实例静态分配路由信息，请设置这个选项。

3. 点**创建**。

新建的子网可以在网络的子网标签页中看到。您可以根据需要，通过点编辑来修改它的选项。现在，当您创建实例时，可以配置它来使用这个新创建的子网，并获得相应的 DHCP 选项。

3.7. 删除子网

当一个子网不再被使用时，您可以删除它。当您删除子网时，如果有实例还在使用它，删除操作会失败，dashboard 会显示一个错误信息。以下介绍了在一个网络中删除一个特定子网的步骤。

在 dashboard 中，选择**项目 > 网络 > 网络**，然后点您的网络名。选择目标子网后点**删除子网**。

3.8. 添加一个路由器

OpenStack Networking 使用一个基于 SDN 的虚拟路由器提供网络路由服务。当您的实例需要和外部子网（包括物理网络之外的网络）进行通讯时，需要使用路由器。路由器和子网使用接口进行连接，每个子网都需要有自己的到路由器的接口。一个路由器的默认网关定义了它所接收的数据要发送到的下一个地址。它的网络通常会被设置为使用一个虚拟网桥把数据路由到外部物理网络。

1.在 dashboard 中，选择**项目 > 网络 > 路由**，然后点 **+创建路由**。

2. 为新的路由设置一个名称，然后点**创建路由**。

3. 点**路由**列表中的新路由器旁的**设置网关**。

4. 在**外部网络**列表中，指定接收发送到外部地址的网络数据的网络。

5. 点**设置网关**。在添加了一个路由后，您需要配置已经创建的子网来使用这个路由器发送网络数据，您需要通过在子网和路由器间创建一个接口来实现它。

3.9. 删除一个路由

如果一个路由没有和其它接口相连，您可以删除这个路由。以下介绍了删除路由接口，然后再删除路由本身的方法。

1. 在 dashboard 中，选择**项目 > 网络 > 路由**，然后点您需要删除的路由名称。

2. 选择类型为 **Internal Interface** 的接口，点**删除接口**。

3. 在**路由**列表中选择目标路由，点**删除路由**。

3.10. 添加一个接口

使用接口可以把路由器和子网进行连接。路由器可以把实例所发出的数据发送到所在子网以外。以下过程会添加一个路由器接口并把它连接到一个子网。它使用"网络拓扑"功能来以图形化的形式显示所有的虚拟路由器，并可以通过这个功能执行网络管理的任务。

1. 在 dashboard 中，选项**目 > 网**络 **> 网**络**拓扑**。

2. 找到您需要管理的路由器，把鼠标光标移到它上面，然后点**添加接口**。

3. 指定要连接到这个路由器的子网。您可以指定一个 IP 地址，它可以用于测试和故障排除，如果可以 ping 到这个地址，则说明数据可以被正确路由。

4. 点**添加接口**。

网络**拓扑**图会自动更新来显示在路由器和子网间添加的新接口。

3.11. 删除一个接口

当您不再需要路由器来处理到一个子网的接口的数据时，可以把这个接口从子网中删除。以下介绍了删除接口的步骤：

1. 在 dashboard 中选择项**目 > 网**络 **> 路由**。

2. 点您需要删除的接口所在的路由。

3. 选择接口（类型应该是 **Internal Interface**），点**删除接口**。

3.12. 配置 IP 地址

您可以使用以下介绍的方法来在 OpenStack Networking 中管理 IP 地址的分配。

3.12.1. 创建浮动 IP 地址池

通过使用浮动 IP 地址，您可以把外部的网络流量发送到您的 OpenStack 实例中。首先，您需要定义一组可以被路由的外部 IP 地址，然后可以把它们动态地分配给实例。OpenStack Networking 会把流入的网络数据发送到相关实例的浮动 IP 地址上。

注意

OpenStack Networking 会把浮动 IP 地址分配给在同一个 IP 范围/CIDR 中的所有项目（租户）。这意味着，浮动 IP 地址子网可以分配给任何项目或所有项目。您可以通过为特定项目设置配额来对这个进行管理。例如，您可以把 *ProjectA* 和 *ProjectB* 的默认配额设为 *10*，把 *ProjectC* 的配额设为 *0*。

浮动 IP 地址池在创建一个外部子网时被定义。如果这个子网只包括主机的浮动 IP 地址，您可以使用 *enable_dhcp=False* 选项禁用 DCHP 服务。

```
# neutron subnet-create --name SUBNET_NAME --enable_dhcp=False --
allocation_pool start=IP_ADDRESS,end=IP_ADDRESS --gateway=IP_ADDRESS
NETWORK_NAME CIDR
```

例如：

```
# neutron subnet-create --name public_subnet --enable_dhcp=False --
allocation_pool start=192.168.100.20,end=192.168.100.100 --
gateway=192.168.100.1 public 192.168.100.0/24
```

3.12.2. 分配一个特定的浮动 IP

您可以使用 nova 命令来为实例分配一个特定的浮动 IP 地址。或者通过 dashboard 进行（请参阅 3.1.2 "更新一个实例（操作菜单）"）

```
# nova floating-ip-associate INSTANCE_NAME IP_ADDRESS
```

在这个例子中，为名为 corp-vm-01 的实例分配了一个浮动 IP 地址：

```
# nova floating-ip-associate corp-vm-01 192.168.100.20
```

3.12.3. 分配一个随机的浮动 IP

浮动 IP 地址可以被动态地分配给实例。您不需要选择一个特定的 IP 地址，而只是要求 OpenStack Networking 从地址池中分配一个。从前面创建的池中分配一个浮动 IP 地址：

```
# neutron floatingip-create public
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| fixed_ip_address    |                                      |
| floating_ip_address | 192.168.100.20                       |
| floating_network_id | 7a03e6bc-234d-402b-9fb2-0af06c85a8a3 |
| id                  | 9d7e2603482d                         |
| port_id             |                                      |
| router_id           |                                      |
| status              | ACTIVE                               |
| tenant_id           | 9e67d44eab334f07bf82fa1b17d824b6     |
+---------------------+--------------------------------------+
```

当获得一个 IP 地址后，您可以把它分配给一个特定的实例。找到和您的实例相关的端口 ID（它和与实例相关联的固定 IP 地址相匹配）。这个端口 ID 会在以下步骤中使用来把实例的端口 ID 和浮动 IP 地址 ID 相关联。您还可以通过检查实例的第 3 栏中的 MAC 地址来确定正确的端口 ID。

```
# neutron port-list
+--------+------+-------------+-----------------------------------
--------------------+
| id     | name | mac_address | fixed_ips
|
+--------+------+-------------+-----------------------------------
--------------------+
| ce8320 |      | 3e:37:09:4b | {"subnet_id": "361f27", "ip_address":
"192.168.100.2"} |
| d88926 |      | 3e:1d:ea:31 | {"subnet_id": "361f27", "ip_address":
"192.168.100.5"} |
| 8190ab |      | 3e:a3:3d:2f | {"subnet_id": "b74dbb", "ip_address":
"10.10.1.25"}|
+--------+------+-------------+-----------------------------------
--------------------+
```

使用 **neutron** 命令把浮动 IP 地址和一个实例的相关端口进行关联：

```
# neutron floatingip-associate 9d7e2603482d 8190ab
```

3.13. 创建多个浮动 IP 地址池

OpenStack Networking 支持每个 L3 agent 有一个浮动 IP 地址池。因此，如果您需要增加 L3 agent 的数量，您将可以创建额外的浮动 IP 地址池。

注意

请确保在您的环境中，只有一个 L3 agent 的 **handle_internal_only_routers** 被配置为 **True**（在 */etc/neutron/neutron.conf* 文件中）。这个选项会把 L3 agent 配置为只管理非外部的路由器。

3.14. 桥接物理网络

以下介绍了把您的虚拟网络桥接到物理网络，从而使虚拟实例可以进行网络连接的方法。其中，我们使用物理 **eth0** 接口作为一个例子，把它映射到 **br-ex** 网桥；虚拟网桥会作为物理网络和虚拟网络之间的一个中间介质。其结果是，所有通过 **eth0** 的网络流量都会使用配置的 Open vSwitch 来访问实例。使用以下方法把一个物理 NIC 映射到虚拟 Open vSwitch 网桥：

注意

IPADDR、**NETMASK GATEWAY** 和 **DNS1**（名称解析服务器）必须被更新来与您的网络相匹配。

```
# vi /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
Configure the virtual bridge with the IP address details that were
previously allocated to eth0:
# vi /etc/sysconfig/network-scripts/ifcfg-br-ex
DEVICE=br-ex
DEVICETYPE=ovs
TYPE=OVSBridge
BOOTPROTO=static
IPADDR=192.168.120.10
NETMASK=255.255.255.0
GATEWAY=192.168.120.1
DNS1=192.168.120.1
ONBOOT=yes
```

现在，您可以为实例分配浮动 IP 地址，并使物理网络可以使用它们。

第 4 章 规划 IP 地址的使用

一个 OpenStack 环境可能会需要使用比想象中更多的 IP 地址。本节的内容可以帮助您更准确地预测所需的地址数量，并介绍了它们需要在哪里被使用。

注意

VIP（也被称为 Virtual IP Addresses - 虚拟 IP 地址） - VIP 地址用来支持 HA 服务，它就是一个被多个控制器节点共享的 IP 地址。

4.1. 使用多个 VLAN

在计划您的 OpenStack 部署时，首先应该从一组子网开始，然后才能计划不同的地址需要被如何使用。当有了多个子网后，您就可以把不同系统间的网络数据进行分隔来组成 VLAN。例如，在一般情况下，管理网络流量和 API 网络流量不应该和处理 web 网络流量的系统共享相同的网络。另外，不同 VLAN 间的网络流量都需要通过一个路由器，这样就可以在路由器上设置防火墙来进一步控制网络流量的安全。

4.2. 隔离 VLAN 的网络数据

一般情况下，您需要为不同的网络类型分配不同的 VLAN。例如，您可以为每一个网络类型都分配一个它们自己的 VLAN。这样，只有 External 网络需要可以路由到外部的物理网络中。在 Red Hat OpenStack Platform 9 中，DHCP 服务由 *director* 提供。

注意

每个 OpenStack 部署不一定都需要本节中提到的所有 VLAN。例如，您的云用户不需要动态创建特定的虚拟网络，则不需要租户网络；如果您只希望所有虚拟机都直接连接到其它物理系统连接到的交换机时，您可能只需要把您的 Compute 节点直接连接到一个供应商网络中，从而使您的实例直接使用供应商网络。

- **Provisioning 网络** - 这个 VLAN 被专门用于使用 director 通过 PXE 引导部署新节点。OpenStack Orchestration（heat）会在 overcloud 的裸机服务器上安装 OpenStack；它们被附加到物理网络上来从undercloud 系统中获得平台的安装镜像。
- **Internal API 网络** - Internal API 网络被用来处理 OpenStack 服务间的通讯，包括 API 通讯、RPC 消息数据以及数据库通讯。另外，这个网络还被用来处理控制器节点间的操作信息。在设计您的 IP 地址分配时，需要考虑到，每个 API 服务都需要自己的 IP 地址。以下的每个服务都需要一个 IP 地址：
  - vip-msg（ampq）
  - vip-keystone-int
  - vip-glance-int
  - vip-cinder-int
  - vip-nova-int
  - vip-neutron-int
  - vip-horizon-int

- vip-heat-int
- vip-ceilometer-int
- vip-swift-int
- vip-keystone-pub
- vip-glance-pub
- vip-cinder-pub
- vip-nova-pub
- vip-neutron-pub
- vip-horizon-pub
- vip-heat-pub
- vip-ceilometer-pub
- vip-swift-pub

注意

在使用高可用性功能时，Pacemaker 需要可以在不同的物理节点间移动 VIP 地址。

- **Storage（存储）网络** - 块存储、NFS、iSCSI 和其它存储。因为性能的原因，在理想情况下这个网络应该位于一个完全独立的物理以太网连接环境中。
- **Storage Management（存储管理）网络** - OpenStack Object Storage（swift）使用这个网络来在相关的副本节点中同步数据项。代理服务（proxy service）在用户请求和底层的存储层间起到一个中间接口的作用。这个代理会接收用户的请求，并找到所需的副本来获得所需的数据。使用 Ceph 作为后端的服务会通过 Storage Management 网络进行连接，因为它们不会和 Ceph 直接进行交流，而是使用前端的服务。请注意，RBD 驱动是个例外，它会直接连接到 Ceph。
- **Tenant（租户）网络** - Neutron 为每个租户提供自己的网络。这可以通过使用 VLAN 隔离（VLAN segregation，每个租户网络都是一个网络 VLAN）实现，也可以使用 VXLAN 或 GRE 隧道（tunneling）实现。每个租户网络的网络数据会被相互隔离，并都有一个相关联的 IP 子网。多个租户子网可以使用相同的地址。
- **External（外部）网络** - 外部网络包括了公共 API 的端点以及到 Dashboard（horizon）的连接。您也可以使用它作为 SNAT，但这不是必须的。在一个生产环境的部署中，通常会使用一个独立的网络用于浮动 IP 地址和 NAT。
- **Provider（供应商）网络** - 实例可以使用这些网络附加到已存在的网络基础设施中。通过扁平化网络结构（flat networking）或 VLAN tag，您可以使用供应商网络直接映射到数据中心中的一个物理网络。这可以使一个实例共享相同的 2 层网络作为 OpenStack Networking 基础架构的一个外部系统。

4.3. IP 地址的消耗

以下系统会消耗 IP 地址分配范围内的 IP 地址：

- **Physical（物理）节点** - 每个物理 NIC 都需要一个 IP 地址；一般情况下，最好是每个物理 NIC 都只提供特定的功能。例如，管理网络有它自己专用的物理 NIC，NFS 网络也有自己专用的 NIC（有时，多个 NIC 连接到不同的交换机来提供冗余容错功能）。
- **用于高可用性的虚拟 IP（VIP）** - 控制器节点间共享的每个网络都需要 1 到 3 个 IP。

4.4. 虚拟网络

这些虚拟资源会消耗 OpenStack Networking 中的 IP 地址。它们被认为是云基础架构的本地资源，并不需要被外部物理网络中的系统访问：

- **Tenant（租户）网络** - 每个租户网络都需要一个子网来为实例分配 IP 地址。
- **虚拟路由** - 每个插入到子网中的路由接口都需要一个 IP 地址。
- **实例** - 每个实例都需要从它所在的租户子网中获得一个 IP 地址。如果实例有入站网络流量，则还需要从相关的外部网络中分配一个浮动 IP 地址。
- **Management（管理）网络流量** - 包括 OpenStack 服务和 API 的网络流量。在 Red Hat OpenStack Platform 9 中，所需虚拟 IP 地址的数量已被减小，所有服务将共享一组少量的 VIP。API、RPC 和数据库服务还在一个内部的 API VIP 上进行通讯。

4.5. 网络计划示例

这个例子分配了一组网络来运行 7 个子网，每个子网需要一定数量的地址：

表 4.1. 子网计划示例

子网名	地址范围	地址的数量	子网掩码
Provisioning 网络	192.168.100.1 - 192.168.100.250	250	255.255.255.0
Internal API 网络	172.16.1.10 - 172.16.1.250	241	255.255.255.0
Storage	172.16.2.10 - 172.16.2.250	241	255.255.255.0
Storage Management	172.16.3.10 - 172.16.3.250	241	255.255.255.0
Tenant network (GRE/VXLAN)	172.19.4.10 - 172.16.4.250	241	255.255.255.0

子网名	地址范围	地址的数量	子网掩码
External network（包括浮动 IP）	10.1.2.10 - 10.1.3.222	469	255.255.254.0
Provider 网络（基础架构）	10.10.3.10 - 10.10.3.250	241	255.255.252.0

第 5 章 检查 OPENSTACK NETWORKING 路由端口

OpenStack Networking 中的虚拟路由器使用端口来进行子网间的连接。您可以通过检查这些端口的状态来确定它们是否被正常连接。

5.1. 查看当前端口状态

以下过程会列出附加到特定路由器上的端口列表，然后演示了如何获得端口状态（DOWN 或 ACTIVE）的方法。

1. 查看附加到名为 **r1** 的路由器上的所有端口：

```
#  neutron router-port-list r1
```

结果输出：

```
+--------------------------------------+------+-------------------
+-------------------------------------------------------------------
--------------------+
| id                                   | name | mac_address       |
fixed_ips
|
+--------------------------------------+------+-------------------
+-------------------------------------------------------------------
--------------------+
| b58d26f0-cc03-43c1-ab23-ccdb1018252a |      | fa:16:3e:94:a7:df |
{"subnet_id": "a592fdba-babd-48e0-96e8-2dd9117614d3", "ip_address":
"192.168.200.1"} |
| c45e998d-98a1-4b23-bb41-5d24797a12a4 |      | fa:16:3e:ee:6a:f7 |
{"subnet_id": "43f8f625-c773-4f18-a691-fd4ebfb3be54", "ip_address":
"172.24.4.225"}  |
+--------------------------------------+------+-------------------
+-------------------------------------------------------------------
--------------------+
```

2. 使用端口的 ID（左面列中的值）运行以下命令来查看端口详情。命令运行结果包括端口的 **status** 值（在下面的例子中是 **ACTIVE**）：

```
# neutron port-show b58d26f0-cc03-43c1-ab23-ccdb1018252a
```

结果输出：

```
+-----------------------+-----------------------------------------
------------------------------------------+
| Field                 | Value
|
+-----------------------+-----------------------------------------
------------------------------------------+
| admin_state_up        | True
|
| allowed_address_pairs |
|
| binding:host_id       | node.example.com
|
```

```
| binding:profile       | {}
|
| binding:vif_details   | {"port_filter": true, "ovs_hybrid_plug":
true}                                        |
| binding:vif_type      | ovs
|
| binding:vnic_type     | normal
|
| device_id             | 49c6ebdc-0e62-49ad-a9ca-58cea464472f
|
| device_owner          | network:router_interface
|
| extra_dhcp_opts       |
|
| fixed_ips             | {"subnet_id": "a592fdba-babd-48e0-96e8-
2dd9117614d3", "ip_address": "192.168.200.1"} |
| id                    | b58d26f0-cc03-43c1-ab23-ccdb1018252a
|
| mac_address           | fa:16:3e:94:a7:df
|
| name                  |
|
| network_id            | 63c24160-47ac-4140-903d-8f9a670b0ca4
|
| security_groups       |
|
| status                | ACTIVE
|
| tenant_id             | d588d1112e0f496fb6cac22f9be45d49
|
+-----------------------+-----------------------------------------
-------------------------------------------+
```

使用不同端口的 ID 执行以上命令来获得它们的状态。

第 6 章 对供应商网络进行故障排除

在开始看来，使用虚拟路由器和交换机（也被称为软件定义的网络，简称 SDN）会使网络系统变得复杂。实际上，对 OpenStack Networking 中的连接进行故障排除的过程和对物理网络进行故障排除的方法非常相似。虚拟基础架构可以看做为物理网络的一个主干扩展，而不是一个完全独立的环境。

6.1. 故障排除的方法

- 基本的 ping 测试
- 对 VLAN 网络进行故障排除
- 从租户网络内进行故障排除

6.2. 基本的 ping 测试

ping 命令是一个非常有用的分析网络连接问题的工具。使用它可以获得基本的网络连接状态，但无法排除所有的连接问题，如防火墙阻止的特定应用程序的连接。ping 命令的工作流程是，向指定的地址发送数据，然后输出这个操作是否成功作为结果。

注意

ping 命令需要 ICMP 网络数据可以通过所有相关的防火墙。

从有网络连接问题的系统上运行 ping 命令进行测试非常有必要。当一个系统看起来已完全没有网络连接时，需要可以通过 VNC 管理控制台连接到系统的命令行以便运行 ping 命令。

例如，下面的 ping 测试命令验证了网络可以正常工作所需的多个网络层的功能（域名解析功能、IP 路由功能和网络交换功能）：

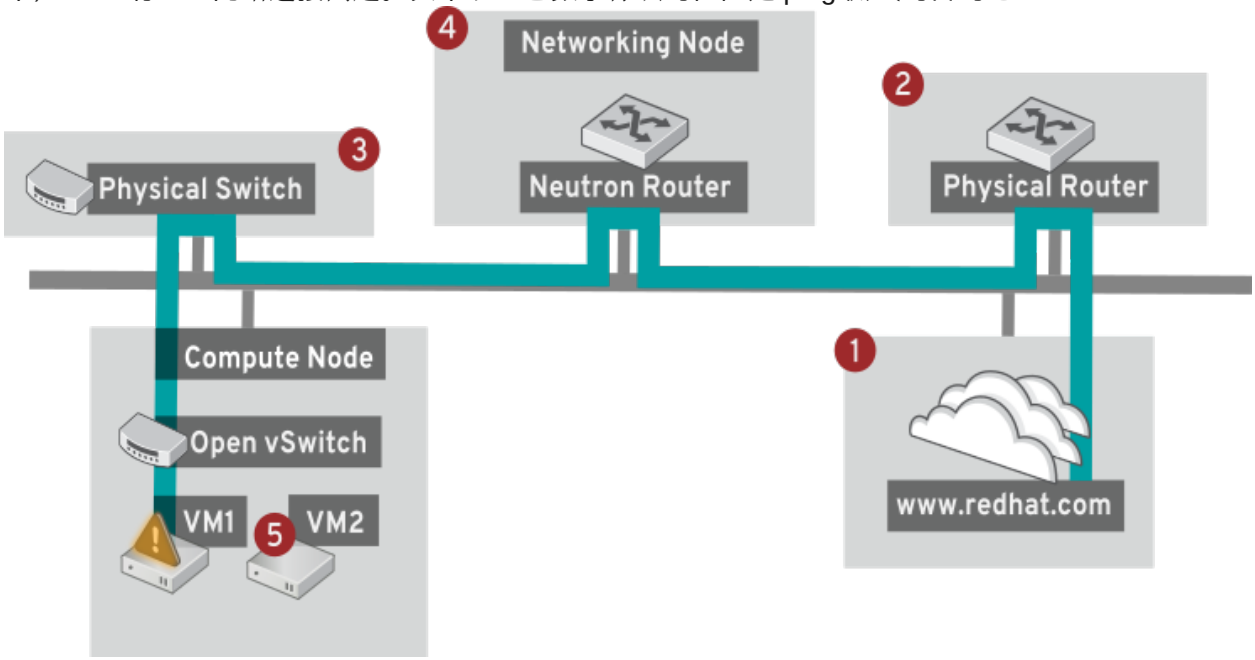
```
$ ping www.redhat.com

PING e1890.b.akamaiedge.net (125.56.247.214) 56(84) bytes of data.
64 bytes from a125-56.247-214.deploy.akamaitechnologies.com
(125.56.247.214): icmp_seq=1 ttl=54 time=13.4 ms
64 bytes from a125-56.247-214.deploy.akamaitechnologies.com
(125.56.247.214): icmp_seq=2 ttl=54 time=13.5 ms
64 bytes from a125-56.247-214.deploy.akamaitechnologies.com
(125.56.247.214): icmp_seq=3 ttl=54 time=13.4 ms
^C
```

在获得了所需的结果后，您可以使用 Ctrl-c 中断 ping 命令。如果命令输出中的数据包丢失（packet loss）值是 0，则说明网络连接处于正常状态：

```
--- e1890.b.akamaiedge.net ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 13.461/13.498/13.541/0.100 ms
```

另外，根据测试的目标，从 ping 测试的结果中可以发现造成网络问题的原因。例如，在以下的图例中，*VM1* 有一些网络连接问题。其中以红色数字标识的位置是 ping 测试的目的地址：



1. internet - 一个常见的做法是首先 ping 一个 internet 地址，如 www.redhat.com。

- *Success*：这意味着，所有中间的网络端点都可以正常工作，包括虚拟网络和物理网络系统。
- *Failure*：多种原因都可以造成 ping 一个 internet 地址失败。如果网络中的其它系统可以成功 ping 到 internet，则说明 internet 本身的连接是正常的，您需要进一步对本地网络系统进行检查。

2. Physical（物理）路由器 - 被网络管理员设置为处理向外部地址发送数据的路由器。

- *Success*：ping 默认网关可以显示本地网络以及所依赖的底层网络交换是否可以正常工作。因为它们中的数据包还没有到达路由器，所以无法说明默认网关是否有路由问题。
- *Failure*：这说明 VM1 和默认网关之间的连接有问题。路由器/交换机可能出现问题，或您使用了一个错误的网关。您可以把有问题的系统的设置和其它可以正常工作的系统上的配置进行比较。尝试 ping 本地网络中的其它服务器。

3. Neutron 路由 - 这是一个 RHEL OpenStack 用来处理虚拟机网络数据的虚拟 SDN（软件定义的网络）.

- *Success*：防火墙允许 ICMP 数据，Networking 节点在线。
- *Failure*：确认实例的安全组是否允许 ICMP 数据。检查 Networking 节点是否在线，所有需要的服务是否都在运行。

4. 物理交换机 - 物理交换机被用来管理相同物理网络中的节点间的通讯。

- *Success*：虚拟机发送到物理交换机的数据需要通过虚拟网络系统。如果 ping 命令成功，则意味着这一段的网络在正常工作。
- *Failure*：物理交换机端口是否被配置为处理相关的 VLAN？

5. VM2 - 尝试 ping 同一个子网中的同一个 Compute 节点上的虚拟机。

- *Success*：VM1 上的 NIC 驱动和基本的 IP 配置可以正常工作。
- *Failure*：检查 VM1 上的网络配置。也可能是 VM2 上的防火墙配置阻塞了 ping 的数据。

6.3. 对 VLAN 网络进行故障排除

OpenStack Networking 可以通过 SDN 交换机进行 VLAN 网络的端口汇聚（trunk）。对 VLAN-tagged 供应商网络的支持意味着虚拟实例可以在物理网络中和服务器子网进行集成。

为了对一个 VLAN 供应商网络的连接性进行故障排除，尝试 ping 在创建网络时指定的网关 IP。例如，您使用以下命令创建了网络：

```
# neutron net-create provider --provider:network_type=vlan --
provider:physical_network=phy-eno1 --provider:segmentation_id=120 --
router:external=True
# neutron subnet-create "provider" --allocation-pool
start=192.168.120.1,end=192.168.120.253 --disable-dhcp --gateway
192.168.120.254 192.168.120.0/24
```

您可以尝试 ping 其中的网关 IP - **192.168.120.254**

如果失败，请确认您是否正确配置了与 VLAN 相关的网络设置（在创建网络时定义）。在上面的例子中，OpenStack Networking 被定义为把 VLAN 120 端口汇聚（trunk）到供应商网络。这个选项通过 **--provider:segmentation_id=120** 参数设置。

检查网桥接口（在这里，它的名称是 **br-ex**） 上的 VLAN 网络流设置：

```
# ovs-ofctl dump-flows br-ex

 NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=987.521s, table=0, n_packets=67897,
n_bytes=14065247, idle_age=0, priority=1 actions=NORMAL
  cookie=0x0, duration=986.979s, table=0, n_packets=8, n_bytes=648,
idle_age=977, priority=2,in_port=12 actions=drop
```

6.3.1. 查看 VLAN 配置和日志文件

- **OpenStack Networking（neutron）agent** - 使用 **neutron** 命令验证所有 agent 都在运行，并以正确的名称注册：

```
# neutron agent-list
+--------------------------------------+--------------------+------
----------------+-------+----------------+
| id                                   | agent_type         | host
| alive | admin_state_up |
+--------------------------------------+--------------------+------
----------------+-------+----------------+
| a08397a8-6600-437d-9013-b2c5b3730c0c | Metadata agent     |
rhelosp.example.com   | :-)   | True           |
| a5153cd2-5881-4fc8-b0ad-be0c97734e6a | L3 agent           |
rhelosp.example.com   | :-)   | True           |
| b54f0be7-c555-43da-ad19-5593a075ddf0 | DHCP agent         |
rhelosp.example.com   | :-)   | True           |
| d2be3cb0-4010-4458-b459-c5eb0d4d354b | Open vSwitch agent |
rhelosp.example.com   | :-)   | True           |
+--------------------------------------+--------------------+------
----------------+-------+----------------+
```

- 检查 **/var/log/neutron/openvswitch-agent.log** - 这个日志应该显示，创建的过程使用 **ovs-ofctl** 命令来配置 VLAN 端口汇聚功能（VLAN trunking）。

- 检查 **/etc/neutron/l3_agent.ini** 文件中的 **external_network_bridge** 设置。如果它是一个“硬代码”值，则无法通过 L3-agent 使用供应商网络，也无法创建所需的网络流程。
- 检查 **/etc/neutron/plugin.ini** 文件中 **network_vlan_ranges** 的值。如果它是一个供应商网络，则不需要指定 VLAN ID。只有在您使用 VLAN 隔离租户网络时才需要指定 ID。

6.4. 从租户网络内进行故障排除

在 OpenStack Networking 中，所有网络流量都会被限制在网络命名空间内。这样，租户就可以在不相互影响的情况下进行网络配置。例如，网络命名空间允许不同租户都使用192.168.1.1/24 作为子网范围，而不会相互影响。

为了对租户网络进行故障排除，首先需要决定网络包括在哪个命名空间中：

1. 使用 **neutron** 命令列出所有租户网络：

```
# neutron net-list
+--------------------------------------+-------------+-------------
-----------------------------------------+
| id                                   | name        | subnets
|
+--------------------------------------+-------------+-------------
-----------------------------------------+
| 9cb32fe0-d7fb-432c-b116-f483c6497b08 | web-servers | 453d6769-fcde-
4796-a205-66ee01680bba 192.168.212.0/24 |
| a0cc8cdd-575f-4788-a3e3-5df8c6d0dd81 | private     | c1e58160-707f-
44a7-bf94-8694f29e74d3 10.0.0.0/24      |
| baadd774-87e9-4e97-a055-326bb422b29b | private     | 340c58e1-7fe7-
4cf2-96a7-96a0a4ff3231 192.168.200.0/24 |
| 24ba3a36-5645-4f46-be47-f6af2a7d8af2 | public      | 35f3d2cb-6e4b-
4527-a932-952a395c4bb3 172.24.4.224/28  |
+--------------------------------------+-------------+-------------
-----------------------------------------+
```

在这个例子中，我们将检查 **web-servers** 网络。记录下 *web-server* 行中的 **id** 值（这里是 **9cb32fe0-d7fb-432c-b116-f483c6497b08**）。在下一步的输出中，可以看到这个值被附加到一个网络命名空间名的后面。

2. 使用 **ip** 命令列出所有网络命名空间：

```
# ip netns list
qdhcp-9cb32fe0-d7fb-432c-b116-f483c6497b08
qrouter-31680a1c-9b3e-4906-bd69-cb39ed5faa01
qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b
qdhcp-a0cc8cdd-575f-4788-a3e3-5df8c6d0dd81
qrouter-e9281608-52a6-4576-86a6-92955df46f56
```

在结果中，一个命名空间与 **web-server** 网络 id 匹配。它由 **qdhcp-9cb32fe0-d7fb-432c-b116-f483c6497b08** 代表。

3. 在命名空间中运行命令来检查 **web-servers** 网络的配置。您需要在故障排除命令前加上 **ip netns exec (namespace)**。例如：

a) 查看 **web-servers** 网络的路由表：

```
# ip netns exec qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b route -n
```

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref
Use Iface
0.0.0.0         172.24.4.225    0.0.0.0         UG    0      0
0 qg-8d128f89-87
172.24.4.224    0.0.0.0         255.255.255.240 U     0      0
0 qg-8d128f89-87
192.168.200.0   0.0.0.0         255.255.255.0   U     0      0
0 qr-8efd6357-96
```

b) 查看 **web-servers** 网络的路由表：

```
# ip netns exec qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b route -n

Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref
Use Iface
0.0.0.0         172.24.4.225    0.0.0.0         UG    0      0
0 qg-8d128f89-87
172.24.4.224    0.0.0.0         255.255.255.240 U     0      0
0 qg-8d128f89-87
192.168.200.0   0.0.0.0         255.255.255.0   U     0      0
0 qr-8efd6357-96
```

6.4.1. 在命名空间内执行高级的 ICMP 测试

1. 使用 **tcpdump** 命令获取 ICMP 网络流量。

```
# ip netns exec qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b tcpdump -
qnntpi any icmp
```

在执行下一步前，可能没有输出：

2. 在一个独立的命令行窗口中运行一个到外部网络的 ping 测试命令：

```
# ip netns exec qrouter-62ed467e-abae-4ab4-87f4-13a9937fbd6b  ping
www.redhat.com
```

3. 在运行 **tcpdump** 的终端中检查 ping 测试的结果详情。

```
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked), capture
size 65535 bytes
IP (tos 0xc0, ttl 64, id 55447, offset 0, flags [none], proto ICMP (1),
length 88)
    172.24.4.228 > 172.24.4.228: ICMP host 192.168.200.20 unreachable,
length 68
 IP (tos 0x0, ttl 64, id 22976, offset 0, flags [DF], proto UDP (17),
length 60)
    172.24.4.228.40278 > 192.168.200.21: [bad udp cksum 0xfa7b ->
0xe235!] UDP, length 32
```

注意

在对网络数据进行 **tcpdump** 分析时，您可能会发现返回的数据包会发送到路由器接口，而不是实例。这是一个正常的行为，因为 qrouter 会在返回的数据包上执行 DNAT。

第 7 章 把一个实例连接到物理网络

本章介绍了如何使用供应商网络把实例直接连接到一个外部网络。

OpenStack Networking 拓扑概述：

OpenStack Networking（neutron）有两类服务分布在多个节点类型中。

- **Neutron API 服务器** - 这个服务运行 OpenStack Networking API 服务器，它为最终用户提供了 API，以及和 OpenStack Networking 进行通讯的服务。这个服务器还集成了底层的数据库来存储和获取租户网络、路由器、负载均衡设备以及其它信息。
- **Neutron agent（代理）** - 为 OpenStack Networking 执行网络功能的服务：
  - **`neutron-dhcp-agent`** - 为租户私人网络管理 DHCP IP 地址。
  - **`neutron-l3-agent`** - 在租户私人网络、外部网络和其它系统间执行第 3 层路由。
  - **`neutron-lbaas-agent`** - 配置由租户创建的 LBaaS 路由器。
- **Compute（计算）节点** - 这个节点包括了运行虚拟机的虚拟机监控程序（hypervisor）。Compute 节点需要通过网线直接连接到网络以便为实例提供外部的网络连接。

服务的位置：

OpenStack Networking 服务可以在同一个物理服务器上运行，也可以分别在专用的服务器上运行（服务器的名称与它们所运行的服务相匹配）：

- *控制器节点（Controller node）* - 运行 API 服务的服务器。
- *网络节点（Network node）* - 运行 OpenStack Networking agent 的服务器。
- *计算节点（Compute node）* - 运行实例的虚拟机监控程序服务器。

本章中介绍的步骤假设您的环境中已经部署了这 3 个节点类型。如果您的 Controller 节点和 Network 节点都运行在相同的物理节点上，针对于这两类节点的步骤都需要在这个服务器上执行。这同样适用于高可用性（HA）环境，在 HA 环境中，所有 3 个节点都可能运行 Controller 节点和 Network 节点以实现高可用性。因此，适用于 Contorller 节点和适用于 Network 节点的步骤都需要在这 3 个节点上运行。

7.1. 使用扁平化供应商网络

以下步骤创建可以把实例直接连接到外部网络的扁平化（flat）供应商网络。当您有多个物理网络（如 **`physnet1`**、**`physnet2`**）和独立的物理接口（**`eth0 -> physnet1`** 和 **`eth1 -> physnet2`**），并需要把每个 Compute 节点和 Network 节点连接到这些外部网络时，可以进行这个操作。

注意

如果您需要把同一个 NIC 中的多个 VLAN-tagged 接口连接到多个供应商网络，请参阅 *VLAN 供应商网络*。

配置控制器节点：

1. 编辑 **/etc/neutron/plugin.ini**（它是到 **/etc/neutron/plugins/ml2/ml2_conf.ini** 文件的 symbolic link），把 **flat** 添加到已存在的值列表中，把 **flat_networks** 设置为 *****：

```
type_drivers = vxlan,flat
flat_networks =*
```

2. 创建一个扁平化外部网络，并把它与配置的 **physical_network** 相关联。如果把它创建为一个共享网络，则允许其它用户把他们的实例直接连接到这个网络：

```
neutron net-create public01 --provider:network_type flat --
provider:physical_network physnet1 --router:external=True --shared
```

3. 使用 **neutron subnet-create** 或 OpenStack Dashboard 在这个外部网络中创建一个子网。

4. 重启 **neutron-server** 服务以使所做的改变生效：

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

配置网络节点和计算节点：

这些步骤需要在 Network 节点和 Compute 节点上进行。这些节点将会被连接到外部网络，并允许实例与外部网络直接进行通讯。

1. 创建 Open vSwitch 网桥和端口。这一步将创建一个外部网桥（**br-ex**）并添加相应的端口（**eth1**）：

i. 编辑 **/etc/sysconfig/network-scripts/ifcfg-eth1**：

```
DEVICE=eth1
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

ii. 编辑 **/etc/sysconfig/network-scripts/ifcfg-br-ex**：

```
DEVICE=br-ex
TYPE=OVSBridge
DEVICETYPE=ovs
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

2. 重启 **network** 服务以使所做改变生效：

```
# systemctl restart network.service
```

3. 在 **/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini** 中配置物理网络，把网桥映射到物理网络：

注意

如需了解更多与配置 **bridge_mappings** 相关的信息，请参阅本指南中的*配置网桥映射*一章。

```
bridge_mappings = physnet1:br-ex
```

4. 在 Network 节点和 Compute 节点上重启 **neutron-openvswitch-agent** 服务以使所做改变生效：

```
systemctl restart neutron-openvswitch-agent
```

配置 Network 节点：

1. 把 **/etc/neutron/l3-agent.ini** 文件中的 **external_network_bridge =** 设为空值。这将会使用外部供应商网络。

```
# Name of bridge used for external network traffic. This should be set
to
# empty value for the linux bridge
external_network_bridge =
```

2. 重启 **neutron-l3-agent** 以使所做改变生效：

```
systemctl restart neutron-l3-agent.service
```

注意

如果有多个 flat 供应商网络，每个网络都需要有独立的物理网络和网桥来与外部网络连接。请正确配置 *ifcfg-** 脚本，当在 **bridge_mappings** 中指定网络时，使用由逗号分隔的网络列表。如需了解更多与配置 **bridge_mappings** 相关的信息，请参阅本指南中的*配置网桥映射*一章。

把一个实例连接到外部网络：

在网络创建后，可以把它与一个实例进行相连来测试它是否可以正常工作：

1. 创建一个新实例。

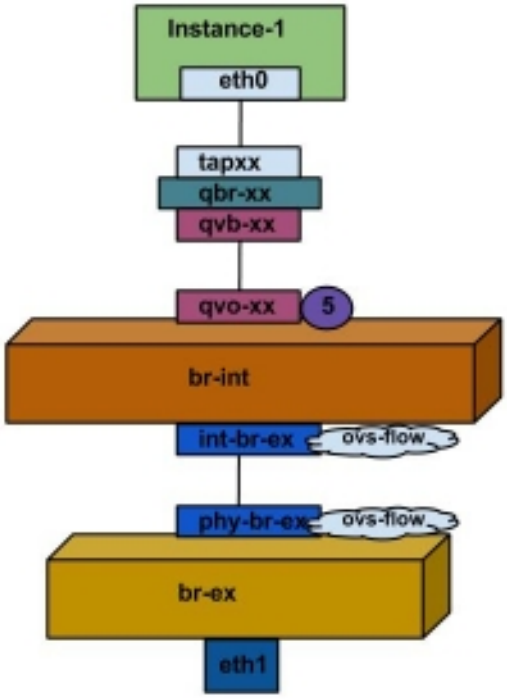
2. 从 dashboard 的**网络**标签页中把新实例直接添加到新创建的外部网络中。

数据包传输是如何工作的？

本节介绍了在扁平化供应商网络被配置后，数据是如何传输到一个实例，以及数据如何从一个实例发送。

7.1.1. 出站网络数据的传输

数据包会离开实例并直接到达一个外部网络：一旦配置了 **br-ex**，添加物理接口，并把实例分配到一个 Compute 节点上。通过这一步所获得的接口和网桥将会和下图所示类似：



1. 从实例的 **eth0** 接口发送的数据包会首先到达 linux 网桥 **qbr-xx**。

2. 网桥 **qbr-xx** 通过 **qvb-xx <-> qvo-xxx** 连接到 **br-int**。

3. 接口 **qvb-xx** 被连接到 **qbr-xx** linux 网桥，**qvo-xx** 被连接到 **br-int** Open vSwitch (OVS) 网桥。

在 Linux 网桥上配置 qbr-xx：

```
qbr269d4d73-e7  8000.061943266ebb no  qvb269d4d73-e7
       tap269d4d73-e7
```

在 br-int 上配置 qvo-xx：

```
  Bridge br-int
        fail_mode: secure
            Interface "qvof63599ba-8f"
        Port "qvo269d4d73-e7"
            tag: 5
            Interface "qvo269d4d73-e7"
```

注意

端口 **qvo-xx** 以与扁平化供应商网络相关联的内部 VLAN tag 进行标记（tag）。在这个例子中，VLAN tag 是 **5**。一旦数据包到达了 **qvo-xx**，VLAN tag 会被添加到数据包头中。

然后，数据包会根据 **int-br-ex <-> phy-br-ex** 被移到 **br-ex** OVS 网桥。

在 **br-int** 上配置 patch-peer 的一个示例：

```
    Bridge br-int
        fail_mode: secure
       Port int-br-ex
            Interface int-br-ex
                type: patch
                options: {peer=phy-br-ex}
```

在 **br-ex** 上配置 patch-peer 的一个示例：

```
    Bridge br-ex
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
        Port br-ex
            Interface br-ex
                type: internal
```

当数据包到达 **br-ex** 中的 **phy-br-ex** 后，**br-ex** 内的 OVS 数据流会删除 VLAN tag（5），并转发到物理接口上。

在以下例子中，输出显示了 **phy-br-ex** 的端口号是 **2**。

```
# ovs-ofctl show br-ex
OFPT_FEATURES_REPLY (xid=0x2): dpid:00003440b5c90dc6
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS
ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC
SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST
ENQUEUE

 2(phy-br-ex): addr:ba:b5:7b:ae:5c:a2
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

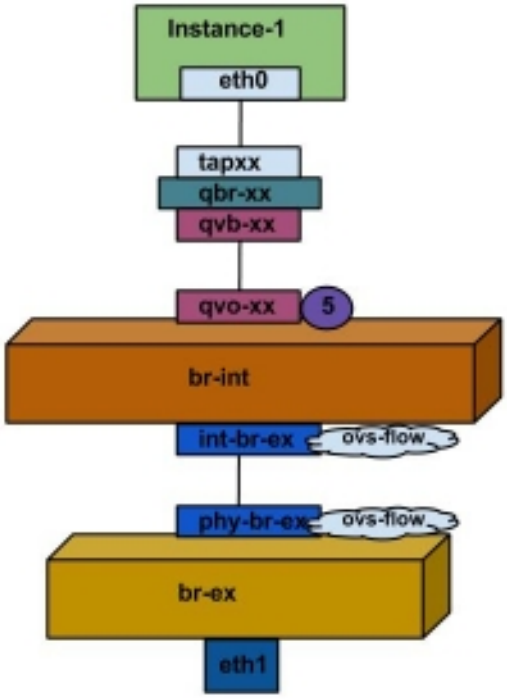
以下输出显示了，所有到达 **phy-br-ex**（**in_port=2**） 的数据包都带有一个 VLAN tag **5**（**dl_vlan=5**）。然后，VLAN tag 被删除，数据包被转发到（**actions=strip_vlan,NORMAL**）。

```
# ovs-ofctl dump-flows br-ex
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=4703.491s, table=0, n_packets=3620,
n_bytes=333744, idle_age=0, priority=1 actions=NORMAL
 cookie=0x0, duration=3890.038s, table=0, n_packets=13, n_bytes=1714,
idle_age=3764, priority=4,in_port=2,dl_vlan=5 actions=strip_vlan,NORMAL
 cookie=0x0, duration=4702.644s, table=0, n_packets=10650,
n_bytes=447632, idle_age=0, priority=2,in_port=2 actions=drop
```

然后，这个数据包被转发到物理接口。如果这个物理接口是一个不同的 vlan tagged 接口，接口会把它的 vlan tag 添加到数据包中。

7.1.2. 入站网络数据的传输

本节介绍了从外部网络发出的入站数据到达实例接口的流程。



1. 入站网络数据首先到达物理节点的 **eth1**。

2. 数据包被发送到 **br-ex** 网桥。

3. 数据包根据 **phy-br-ex <--> int-br-ex** 被移到 **br-int**。

在下面的例子中，**int-br-ex** 使用端口 **15**。请参阅包括 **15(int-br-ex)** 的项：

```
ovs-ofctl show br-int
OFPT_FEATURES_REPLY (xid=0x2): dpid:00004e67212f644d
n_tables:254, n_buffers:256
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS
ARP_MATCH_IP
actions: OUTPUT SET_VLAN_VID SET_VLAN_PCP STRIP_VLAN SET_DL_SRC
SET_DL_DST SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST
ENQUEUE
 15(int-br-ex): addr:12:4e:44:a9:50:f4
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

观察 br-int 上的网络流量

1. 当数据包到达 **int-br-ex** 后，**br-int** 网桥中的一个 OVS 流规则会把内部 VLAN tag **5** 添加到数据包中。请参阅 **actions=mod_vlan_vid:5** 的项：

```
# ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=5351.536s, table=0, n_packets=12118,
n_bytes=510456, idle_age=0, priority=1 actions=NORMAL
 cookie=0x0, duration=4537.553s, table=0, n_packets=3489,
n_bytes=321696, idle_age=0, priority=3,in_port=15,vlan_tci=0x0000
actions=mod_vlan_vid:5,NORMAL
 cookie=0x0, duration=5350.365s, table=0, n_packets=628, n_bytes=57892,
idle_age=4538, priority=2,in_port=15 actions=drop
 cookie=0x0, duration=5351.432s, table=23, n_packets=0, n_bytes=0,
idle_age=5351, priority=0 actions=drop
```

2. 第 2 个规则管理到达 int-br-ex (in_port=15) 上的不带有 VLAN tag (vlan_tci=0x0000) 的数据包：它把 VLAN tag 5 添加到数据包上（**actions=mod_vlan_vid:5,NORMAL**），然后把它转发到 **qvo-xxx**。

3. qvo-xxx 接受数据包并在删除 VLAN tag 后把它转发到 **qvb-xx**。

4. 数据包然后会到达实例。

注意

VLAN tag 5 是一个带有扁平化供应商网络的测试 Compute 节点的例子，**neutron-openvswitch-agent** 会自动分配这个值。它可能会和您自己的扁平化供应商网络的值不同，在两个独立 Compute 节点上的同一个网络的值也可能不同。

7.1.3. 故障排除

如果一个扁平化的供应商网络出现了问题，可以从上一节中所显示的输出中获取信息进行故障排除。以下介绍的步骤会进一步帮助您进行故障排除。

1. 检查 **bridge_mappings**：

确认使用的物理网络名（例如 **physnet1**）和 **bridge_mapping** 配置的内容相匹配。例如：

```
# grep bridge_mapping
/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini
bridge_mappings = physnet1:br-ex

# neutron net-show provider-flat
...
| provider:physical_network | physnet1
...
```

2. 检查网络配置：

确认网络被创建为 **external**，类型是 **flat**：

```
# neutron net-show provider-flat
...
| provider:network_type     | flat                                 |
```

```
| router:external           | True                                 |
...
```

3. 检查 patch-peer：

运行 **ovs-vsctl show**，确认 **br-int** 和 **br-ex** 通过使用 patch-peer **int-br-ex <--> phy-br-ex** 进行连接。

这个连接在 **neutron-openvswitch-agent** 服务重启时被创建，并需要 **bridge_mapping** 在 **/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini** 中被正确配置。如果没有被创建，请再次检查 **bridge_mapping** 的设置（即使已重启了服务）。

注意

如需了解更多与配置 **bridge_mappings** 相关的信息，请参阅本指南中的*配置网桥映射*一章。

4. 检查网络数据流：

运行 **ovs-ofctl dump-flows br-ex** 和 **ovs-ofctl dump-flows br-int**，检查内部 VLAND ID 是否已从向外的数据包中删除，以及 VLAN ID 是否已被添加到向内的数据包中。当在特定 Compute 节点的这个网络上分配实例时，这个数据流会被创建。

- 如果在分配实例后没有创建这个数据流，请检查网络是否被创建为 **flat**，是否是 **external**，以及 **physical_network** 名是否正确。另外，请检查 **bridge_mapping** 的设置。
- 最后，检查 **ifcfg-br-ex** 和 **ifcfg-ethx** 的配置。确认 **ethX** 已作为 **br-ex** 内的端口被添加，它们在 **ip a** 中的输出中都带有 **UP** 标识。

例如，以下输出显示 **eth1** 是 **br-ex** 中的一个端口：

```
    Bridge br-ex
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
        Port "eth1"
            Interface "eth1"
```

以下例子显示，**eth1** 被配置为一个 OVS 端口，内核知道从接口上传输所有数据包，并把它们发送到 OVS 网桥 **br-ex**。这些可以在 **master ovs-system** 项中找到。

```
# ip a
5: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master
ovs-system state UP qlen 1000
```

7.2. 使用 VLAN 供应商网络

这个过程会创建一个可以把实例直接连接到外部网络的 VLAN 供应商网络。如果您需要把一个独立 NIC 上的多个 VLAN-tagged 接口连接到多个供应商网络，则要执行这些操作。这个例子使用一个名为 **physnet1** 的网络，它具有一组 VLAN（**171-172**）。网络节点和计算节点使用名为 **eth1** 的物理接口连接到物理网络。连接这些接口的交换机网络被配置为对所需的 VLAN 进行端口汇聚

（trunk）。
以下过程使用上面提供的 VLAN ID 和名称对 VLAN 供应商网络进行配置。

配置控制器节点：

1. 编辑 **/etc/neutron/plugin.ini**（到 **/etc/neutron/plugins/ml2/ml2_conf.ini** 文件的符合链接）文件来启用 *vlan* 机制驱动，把 *vlan* 添加到存在的值列表中。如：

```
[ml2]
type_drivers = vxlan,flat,vlan
```

2. 配置 **network_vlan_ranges** 的设置来反映使用的物理网络和 VLAN。例如：

```
[ml2_type_vlan]
network_vlan_ranges=physnet1:171:172
```

3. 重启 *neutron-server* 服务以使所做的修改生效：

```
systemctl restart neutron-server
```

4. 创建外部网络作为 *vlan* 类型，并把它们与配置的 **physical_network** 相关联。把它创建为一个 *--shared* 网络来允许其它用户直接连接到实例。这个例子会创建两个网络：一个为 VLAN 171，另一个为 VLAN 172：

```
neutron net-create provider-vlan171 \
   --provider:network_type vlan \
   --router:external true \
   --provider:physical_network physnet1 \
   --provider:segmentation_id 171 --shared

neutron net-create provider-vlan172 \
   --provider:network_type vlan \
   --router:external true \
   --provider:physical_network physnet1 \
   --provider:segmentation_id 172 --shared
```

5. 使用 **neutron subnet-create** 或 dashboard 创建一组子网，并把它们配置为使用外部网络。您需要确保从网络管理员处获得的外部子网信息正确地与每个 VLAN 相关联。在这个例子中，VLAN 171 使用子网 *10.65.217.0/24*，VLAN 172 使用 *10.65.218.0/24*：

```
neutron subnet-create \
   --name subnet-provider-171 provider-171 10.65.217.0/24 \
   --enable-dhcp \
   --gateway 10.65.217.254 \

neutron subnet-create \
   --name subnet-provider-172 provider-172 10.65.218.0/24 \
   --enable-dhcp \
   --gateway 10.65.218.254 \
```

配置 Network 节点和 Compute 节点：

这些步骤必须在 network 节点和 compute 节点上都进行。这将把节点连接到外部网络，并使实例可以直接和外部网络进行通讯。

1. 创建一个外部网桥（*br-ex*），并把它与端口（*eth1*）进行关联：

- 在这个例子中，把 *eth1* 配置为使用 *bre-ex*：

```
/etc/sysconfig/network-scripts/ifcfg-eth1

DEVICE=eth1
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

- 以下例子被用来配置 *br-ex* 网桥：

```
/etc/sysconfig/network-scripts/ifcfg-br-ex:

DEVICE=br-ex
TYPE=OVSBridge
DEVICETYPE=ovs
ONBOOT=yes
NM_CONTROLLED=no
BOOTPROTO=none
```

2. 重启节点或重启 *network* 服务以使对网络的修改生效。例如：

```
# systemctl restart network
```

3. 在 **/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini** 中配置物理网络，把网桥映射到相关的物理网络：

```
bridge_mappings = physnet1:br-ex
```

注意

如需了解更多与配置 **bridge_mappings** 相关的信息，请参阅本指南中的*配置网桥映射*一章。

4. 在网络节点和计算节点上重启 **neutron-openvswitch-agent** 服务以使所做的修改生效：

```
systemctl restart neutron-openvswitch-agent
```

配置 Network 节点：

1. 把 **/etc/neutron/l3-agent.ini** 文件中的 **external_network_bridge =** 设置为空。在使用供应商外部网络时需要这个配置。如果是基于外部网络的网桥，则需要添加 **external_network_bridge = br-ex**：

```
# Name of bridge used for external network traffic. This should be set
to
# empty value for the linux bridge
external_network_bridge =
```

2. 重启 **neutron-l3-agent** 以使所做的修改生效。

```
systemctl restart neutron-l3-agent
```

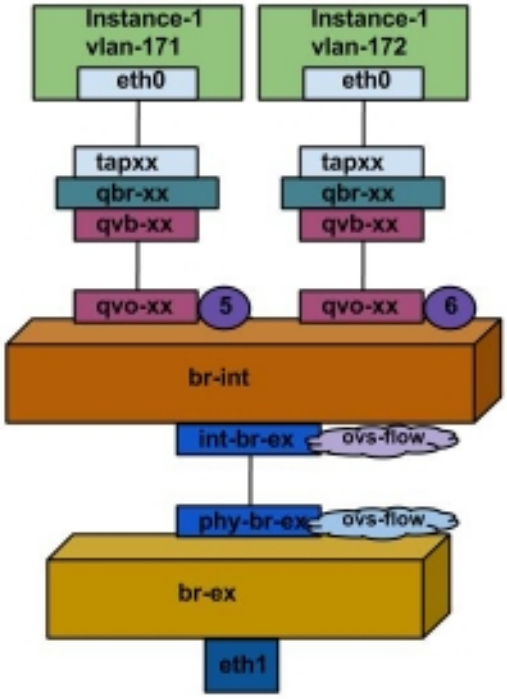
3. 创建一个新实例，并使用 dashboard 中的**网络**标签页把新实例直接添加到新创建的外部网络中。

数据包传输是如何工作的？

本节介绍了在供应商网络被配置后，数据是如何传输到一个实例，以及数据如何从一个实例发送。

7.2.1. 出站网络数据的传输

本节介绍了数据包离开一个实例并直接到达一个 VLAN 供应商外部网络的流程。这个例子使用了两个附加到两个 VLAN 网络的实例（171 和 172）。在配置完 *br-ex* 后，为它添加一个物理网络，并把一个实例分配到一个 Compute 节点。通过这一步所获得的接口和网桥将会和下图所示类似：



1. 如上所示，离开实例的 *eth0* 接口的数据会首先到达与实例相连接的 linux 网桥 *qbr-xx*。

2. *qbr-xx* 使用 *qvb-xx* <→ *qvo-xxx* 连接到 *br-int*。

3. *qvb-xx* 被连接到 linux 网桥 *qbr-xx*；*qvo-xx* 连接到 Open vSwitch 网桥 *br-int*。

以下是 Linux 网桥上的 *qbr-xx* 配置。

因为有两个实例，所以需要两个 linux 网桥：

```
# brctl show
bridge name bridge id  STP enabled interfaces
qbr84878b78-63  8000.e6b3df9451e0 no  qvb84878b78-63
       tap84878b78-63

qbr86257b61-5d  8000.3a3c888eeae6 no  qvb86257b61-5d
       tap86257b61-5d
```

***br-int* 上的 *qvo-xx* 配置：**

```
            options: {peer=phy-br-ex}
        Port "qvo86257b61-5d"
            tag: 3

            Interface "qvo86257b61-5d"
        Port "qvo84878b78-63"
            tag: 2
            Interface "qvo84878b78-63"
```

- **qvo-xx** 使用与 VLAN 供应商网络相关联的内部 VLAN tag 进行标记（tag）。在这个例子中，内部 VLAN tag 2 与 VLAN 供应商网络 **provider-171** 相关联；VLAN tag 3 与 VLAN 供应商网络 **provider-172** 相关联。一旦数据包到达 *qvo-xx*，相关的 VLAN tag 就会被添加到数据包的头数据中。
- 然后，数据包会使用 patch-peer **int-br-ex** <→ **phy-br-ex** 移到 *br-ex* OVS网桥。以下是 *br-int* 中的一个 patch-peer 示例：

```
    Bridge br-int
        fail_mode: secure
       Port int-br-ex
            Interface int-br-ex
                type: patch
                options: {peer=phy-br-ex}
```

在 *br-ex* 上配置 patch peer 的示例：

```
    Bridge br-ex
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
        Port br-ex
            Interface br-ex
                type: internal
```

- 当这个数据包到达 *br-ex* 上的 *phy-br-ex* 时，*br-ex* 内的一个 ovs flow 会使用与 VLAN 供应商网络相关联的实际的 VLAN tag 替换内部 VLAN tag。

以下命令的输出显示了，*phy-br-ex* 端口号是 **4**：

```
# ovs-ofctl show br-ex
 4(phy-br-ex): addr:32:e7:a1:6b:90:3e
```

```
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

以下命令显示了到达 phy-br-ex（**in_port=4**）的数据包带有 vlan tag 2（**dl_vlan=2**）。Open vSwitch 会把这个 VLAN tag 替换为 171（**actions=mod_vlan_vid:171,NORMAL**）后转发数据包。 它还显示了到达 phy-br-ex（**in_port=4**）的数据包带有 vlan tag 3（**dl_vlan=3**）。Open vSwitch 会把这个 VLAN tag 替换为 172（**actions=mod_vlan_vid:172,NORMAL**）后转发数据包。这些规则由 neutron-openvswitch-agent 自动添加。

```
# ovs-ofctl dump-flows br-ex
NXST_FLOW reply (xid=0x4):
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6527.527s, table=0, n_packets=29211,
n_bytes=2725576, idle_age=0, priority=1 actions=NORMAL
 cookie=0x0, duration=2939.172s, table=0, n_packets=117, n_bytes=8296,
idle_age=58, priority=4,in_port=4,dl_vlan=3
actions=mod_vlan_vid:172,NORMAL
 cookie=0x0, duration=6111.389s, table=0, n_packets=145, n_bytes=9368,
idle_age=98, priority=4,in_port=4,dl_vlan=2
actions=mod_vlan_vid:171,NORMAL
 cookie=0x0, duration=6526.675s, table=0, n_packets=82, n_bytes=6700,
idle_age=2462, priority=2,in_port=4 actions=drop
```

- 然后，这个数据包会被发送到物理接口 *eth1*。

7.2.2. 入站网络数据的传输

- 从外部网络发送到实例的数据包会首先到达 *eth1*，然后到达 *br-ex*。
- 从 br-ex 上，数据包通过 patch-peer **phy-br-ex <-> int-br-ex** 被移到 br-int。

以下命令显示了 *int-br-ex* 有一个端口号是 15：

```
# ovs-ofctl show br-int
 18(int-br-ex): addr:fe:b7:cb:03:c5:c1
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

- 当数据包到达 *int-br-ex* 时，*br-int* 内的一个 OVS flow 规则会在数级包内为 **provider-171** 添加 VLAN tag 2，为 **provider-172** 添加 VLAN tag 3：

```
# ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=6770.572s, table=0, n_packets=1239,
n_bytes=127795, idle_age=106, priority=1 actions=NORMAL
 cookie=0x0, duration=3181.679s, table=0, n_packets=2605,
n_bytes=246456, idle_age=0, priority=3,in_port=18,dl_vlan=172
actions=mod_vlan_vid:3,NORMAL
 cookie=0x0, duration=6353.898s, table=0, n_packets=5077,
n_bytes=482582, idle_age=0, priority=3,in_port=18,dl_vlan=171
actions=mod_vlan_vid:2,NORMAL
```

```
 cookie=0x0, duration=6769.391s, table=0, n_packets=22301,
n_bytes=2013101, idle_age=0, priority=2,in_port=18 actions=drop
 cookie=0x0, duration=6770.463s, table=23, n_packets=0, n_bytes=0,
idle_age=6770, priority=0 actions=drop
```

第 2 个规则表示，一个带有 VLAN tag 172（**dl_vlan=172**）的数据包到达 int-br-ex（**in_port=15**），把 VLAN tag 替换为 3（**actions=mod_vlan_vid:3,NORMAL**）并转发。第 3 个规则表示，一个带有 VLAN tag 171（**dl_vlan=171**）的数据包到达 int-br-ex（**in_port=15**），把 VLAN tag 替换为 2（**actions=mod_vlan_vid:2,NORMAL**）并转发。

- 当内部的 VLAN tag 被加入到数据包后，qvo-xxx 就可以接受它，并在删除 VLAN tag 后把数据包转发到 qvb-xx。最后，数据包就可以到达相关的实例。

请注意，在这个例子中，neutron-openvswitch-agent 会自动在 Compute 节点上使用 VLAND tag 2 和 3 来分别代表 VLAN 供应商网络 provider-171 和 provider-172。这可能和您实际的 VLAN 供应商网络不同，并且两个不同 Compute 节点上的值也可能不同。

7.2.3. 故障排除

在对一个 VLAN 供应商网络中的网络连接性进行故障排除时，可以参阅本节中介绍的与数据包流相关的内容。另外，请检查以下配置选项：

1. 确认使用的网络名是一致的。在这个例子中，**physnet1** 在创建网络和 **bridge_mapping** 配置中被一致使用：

```
# grep bridge_mapping
/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini
bridge_mappings = physnet1:br-ex

# neutron net-show provider-vlan171
...
| provider:physical_network | physnet1
...
```

2. 确认网络被创建为 **external**，类型是 **vlan**，并且使用了正确的 **segmentation_id** 值：

```
# neutron net-show provider-vlan171
...
| provider:network_type     | vlan                                 |
| provider:physical_network | physnet1                             |
| provider:segmentation_id  | 171                                  |
...
```

3. 运行 **ovs-vsctl show**，确认 *br-int* 和 *br-ex* 使用 patch-peer **int-br-ex** <→ **phy-br-ex** 进行了连接。
这个连接会在重启 *neutron-openvswitch-agent* 后被创建（需要在 **/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini** 中正确配置了 **bridge_mapping**）。
如果在重启服务后没有创建这个连接，请再次检查 *bridge_mapping* 的设置。

4. 运行 **ovs-ofctl dump-flows br-ex** 和 **ovs-ofctl dump-flows br-int** 来检查出站数据流，确认数据流把内部 VLAN ID 映射到外部 VLAN id（*segmentation_id*）。对于入站的数据包，把外部 VLAN ID 映射到内部 VLAN ID。
当第一次把实例连接到这个网络时，neutron OVS 代理会添加数据流。如果在连接实例后没有创建数

据流，请检查网络是否被创建为 **vlan**，并且是 **external**，检查 **physical_network** 名称是否正确。另外，再次检查 **bridge_mapping** 的设置。

5. 最后，重新检查 *ifcfg-br-ex* 和 *ifcfg-ethx* 的配置。确认 *ethX* 作为一个端口在 *br-ex* 内添加，并且它们在 **ip a** 命令输出中有 **UP** 标记。
例如，以下输出显示，*eth1* 是 *br-ex* 中的一个端口。

```
    Bridge br-ex
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
        Port "eth1"
            Interface "eth1"
```

以下命令显示，*eth1* 作为一个端口被添加，内核知道把所有数据包从节点移到 OVS 网桥 *br-ex*。这些信息包括在 **master ovs-system** 项中。

```
# ip a
5: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master
ovs-system state UP qlen 1000
```

7.3. 启用 Compute 元数据访问

使用这种方式连接的实例会直接附加到供应商外部网络，并带有作为默认网关的外部路由器，而不是使用 OpenStack Networking（neutron）路由器。这意味着，*neutron* 路由不能作为从实例到 *nova-metadata* 服务器的元数据请求的代理，这会在运行 *cloud-init* 时出现错误。但是，这个问题可以通过把 dhcp agent 配置为元数据请求代理来解决。您可以在 **/etc/neutron/dhcp_agent.ini** 中启用这个功能。例如：

```
enable_isolated_metadata = True
```

7.4. 浮动 IP 地址

请注意，相同的网络可以被用来为实例分配浮动 IP 地址，即使它们同时已被加入到私人网络中。从这个网络中分配的浮动 IP 会和 Network 节点上的 *qrouter-xxx* 命名空间相关联，并会执行 *DNAT-SNAT* 来关联私人 IP 地址。与之相反，被分配用来进行外部网络访问的 IP 地址会直接和实例内部相关联，从而使实例可以直接和外部网络进行交流。

第 8 章 为 OPENSTACK 网络配置物理交换机

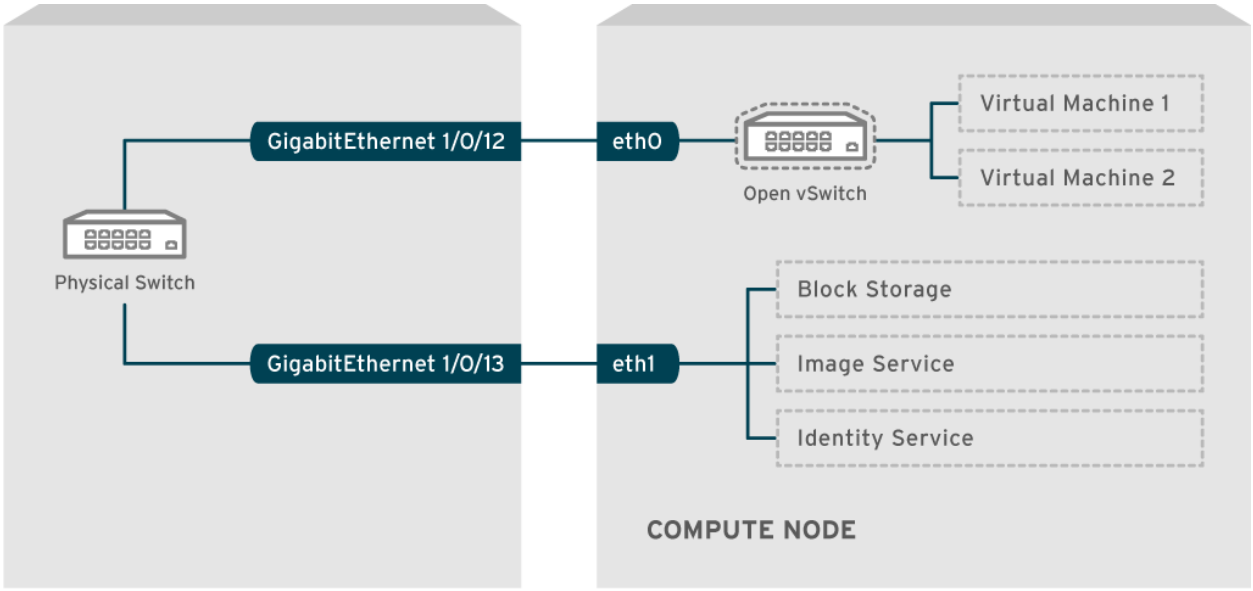
本章介绍了 OpenStack Networking 所需的常规物理交换机配置步骤。另外，也包括了对一些流行厂商（如 Cisco、Extreme Networks 和 Juniper）的交换机进行配置的介绍。

8.1. 规划物理网络环境

OpenStack 节点中的物理网络适配器可能会需要处理不同类型的网络流量，如与实例相关的网络流量、存储数据网络流量或身份验证请求的网络流量。网卡所要处理的不同网络流量类型会影响到如何在物理交换机上对它们的端口进行配置。

首先，您需要决定 Compute 节点上的不同物理网络需要处理哪种类型的网络流量。然后，当网卡被插入到物理交换机接口后，物理交换机接口需要进行相应的配置来允许主干网络流量（trunked traffic）或一般的网络流量。

例如，这个图示有一个带有两个网卡（eth0 和 eth1）的 Compute 节点，每个网络都被连接到一个物理交换机的 Gigabit 以太网网口中。eth0 被用来处理与实例相关的网络流量，eth1 用来处理与 OpenStack 服务相关的网络数据：



注意

这个图示没有包括提供容错功能的额外网卡。

8.2. 配置 Cisco Catalyst 交换机

8.2.1. 配置主干（trunk）端口

OpenStack Networking 允许实例连接到已存在于物理网络中的 VLAN。*主干（trunk）*这个术语被用来描述一个端口，这个端口允许多个 VLAN 通过它。通过主干，VLAN 就可以分布于包括虚拟交换机在内的多个交换机中。例如，物理网络中的带有 **VLAN110** tag 的网络数据可以到达 Compute 节点，它的 8021q 模块会把带有 tag 的网络数据发送到 vSwitch 中的相关 VLAN 中。

8.2.1.1. 为 Cisco Catalyst 交换机配置主干（trunk）端口

如果使用运行 Cisco IOS 的 Cisco Catalyst 交换机，则需要使用以下配置来允许到 VLAN 110 和 111 的数据可以被发送到您的实例。这个配置假设物理节点已有一个连接到物理交换机中的 **GigabitEthernet1/0/12** 接口的以太网网线。

注意

以下只是一个示例。如果在没有根据您的具体情况对相关值进行修改的情况下把它们复制到您的交换机配置中，则可能会导致一些意料外的问题。

```
interface GigabitEthernet1/0/12
  description Trunk to Compute Node
  spanning-tree portfast trunk
  switchport trunk encapsulation dot1q
  switchport mode trunk
  switchport trunk native vlan 1
  switchport trunk allowed vlan 1,110,111
```

这些设置描述如下：

项	描述
interface GigabitEthernet1/0/12	这是节点网卡输入的交换机端口。这只是一个示例，您需要确保根据具体情况使用了正确的端口。可以使用 show interface 命令查看端口列表。
description Trunk to Compute Node	使用 show interface 命令列出所有接口时显示的描述信息。它应该提供了足够信息以使用户可以知道哪个系统被插入到这个端口，以及这个连接所要实现的功能。
spanning-tree portfast trunk	假设您的环境使用 STP，告诉 Port Fast 这个端口被用来处理 trunk 网络流量。
switchport trunk encapsulation dot1q	启用 802.1q trunking 标准（而不是 ISL）。这会根据您所使用的交换机所支持的标准有所不同。
switchport mode trunk	把这个端口配置为一个主干（trunk）端口，而不是一个访问端口。这意味着，它将允许所有 VLAN 网络流量通过虚拟交换机。
switchport trunk native vlan 1	设置一个原生的 VLAN，交换机会发送没有经过 tag 处理的（non-VLAN）网络数据。

项	描述
switchport trunk allowed vlan 1,110,111	指定哪些 VLAN 允许通过这个 trunk 端口。

注意

因为这个端口与 SDN 交换机相集成，所以只配置 spanning-tree portfast 可能会导致 switching loop 问题使端口被阻塞。

8.2.2. 配置访问端口

因为不是 Compute 节点上的所有网卡都需要处理网络数据，所以不需要配置为允许多个 VLAN 通过。这些端口只需要配置一个 VLAN 就可能实现其它操作需求（如传递管理网络流量或 Block Storage 的数据）。这些端口通常被称为访问端口（access port），它们的配置会比 trunk 端口的配置简单。

8.2.2.1. 为 Cisco Catalyst 交换机配置访问端口

继续使用以前的示图，这个例子把 Cisco Catalyst 交换机上的 GigabitEthernet1/0/13 配置为 eth1 的一个访问端口。这个配置假设您的物理节点有一个连接到物理交换机中的 **GigabitEthernet1/0/12** 接口的以太网网线。

注意

以下只是一个示例。如果在没有根据您的具体情况对相关值进行修改的情况下把它们复制到您的交换机配置中，则可能会导致一些意料外的问题。

```
interface GigabitEthernet1/0/13
 description Access port for Compute Node
 switchport mode access
 switchport access vlan 200
 spanning-tree portfast
```

这些设置描述如下：

项	描述
interface GigabitEthernet1/0/13	这是节点网卡输入的交换机端口。这个接口值只是一个例子，您需要确保根据具体情况使用了正确的端口。可以使用 show interface 命令查看端口列表。

项	描述
description Access port for Compute Node	使用 show interface 命令列出所有接口时显示的描述信息。它应该提供了足够信息以使用户可以知道哪个系统被插入到这个端口，以及这个连接所要实现的功能。
switchport mode access	把这个端口配置为一个访问端口，而不是一个主干（trunk）端口。
switchport access vlan 200	把这个端口配置为允许 VLAN 200 中的网络数据。您的 Compute 节点也应该被配置为使用这个 VLAN 中的一个 IP 地址。
spanning-tree portfast	在使用 STP 时，这个设置会告知 STP 不要在初始连接时把它作为一个主干。这个设置可以加快初始连接时（如重启服务器）端口握手的速度。

8.2.3. 配置 LACP 端口聚合

LACP 允许您把多个物理网卡结合在一起成为一个单独的逻辑通道。LACP 在 Linux 中也被称为绑定模式 4，它会创建一个动态的绑定来实现负载均衡和容错功能。LACP 需要在物理的 NIC 端和物理的交换机端口端都进行配置。

8.2.3.1. 在网络网卡上配置 LACP

1. 编辑 */home/stack/network-environment.yaml* 文件：

```
- type: linux_bond
  name: bond1
  mtu: 9000
  bonding_options:{get_param: BondInterfaceOvsOptions};
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000
```

2. 配置 Open vSwitch 网桥来使用 **LACP**：

```
BondInterfaceOvsOptions:
    "mode=802.3ad"
```

如需了解配置网络绑定的信息，请参阅 Director 安装和使用指南。

8.2.3.2. 在 Cisco Catalyst 交换机上配置 LACP

在这个例子中，Compute 节点有两个使用 VLAN 100 的网卡：

1. 把 Compute 节点的两个网卡物理地连接到交换机（例如，连接到端口 12 和 13）。

2. 创建 LACP 端口频道：

```
interface port-channel1
  switchport access vlan 100
  switchport mode access
  spanning-tree portfast disable
  spanning-tree bpduguard disable
  spanning-tree guard root
```

3. 配置交换机端口 12（Gi1/0/12）和 13（Gi1/0/13）：

```
sw01# config t
Enter configuration commands, one per line.  End with CNTL/Z.

sw01(config) interface GigabitEthernet1/0/12
   switchport access vlan 100
   switchport mode access
   speed 1000
   duplex full
   channel-group 10 mode active
   channel-protocol lacp

interface GigabitEthernet1/0/13
  switchport access vlan 100
  switchport mode access
  speed 1000
  duplex full
  channel-group 10 mode active
  channel-protocol lacp
```

4. 复查新的端口频道。这个结果输出应该列出新创建的端口频道 **Po1**，它包括端口 **Gi1/0/12** 和 **Gi1/0/13**：

```
sw01# show etherchannel summary
<snip>

Number of channel-groups in use: 1
Number of aggregators:           1

Group  Port-channel  Protocol    Ports
------+-------------+-----------+-----------------------------------
-------------
1      Po1(SD)         LACP      Gi1/0/12(D)  Gi1/0/13(D)
```

注意

为了使所做的修改生效，把 running-config 复制到 startup-config：**`copy running-config startup-config`**。

8.2.4. 配置 MTU 设置

一些特定的网络流量可能需要对 MTU 的大小进行调整才可以正常工作。例如，特定的 NFS 或 iSCSI 网络数据可能需要设置巨型帧（9000 字节）。

注意

对 MTU 设置的修改需要在数据经过的所有节点上进行，这包括虚拟交换机。如需了解更多与在 OpenStack 环境中修改 MTU 的信息，请参阅 第 9 章 *配置 MTU 设置*。

8.2.4.1. 在 Cisco Catalyst 交换机上配置 MTU 的设置

这个例子会在 Cisco Catalyst 3750 交换机上启用巨型帧（jumbo frame）。

1. 查看当前的 MTU 设置：

```
sw01# show system mtu

System MTU size is 1600 bytes
System Jumbo MTU size is 1600 bytes
System Alternate MTU size is 1600 bytes
Routing MTU size is 1600 bytes
```

2. MTU 的设置会在 3750 交换机的范围内改变，而不是只针对于特定接口。这个命令把交换机配置为使用 9000 字节的巨型帧：

```
sw01# config t
Enter configuration commands, one per line.  End with CNTL/Z.

sw01(config)# system mtu jumbo 9000
Changes to the system jumbo MTU will not take effect until the next
reload is done
```

注意

为了使所做的修改生效，把 running-config 复制到 startup-config：**`copy running-config startup-config`**。

3. 在可能的情况下，重新加载交换机以使改变生效。在重新加载的过程中，所有依赖于这个交换机的设备上的网络功能都会暂时失效。

```
sw01# reload
Proceed with reload? [confirm]
```

4. 在交换机重新加载完成后，检查新的 MTU 设置：

```
sw01# show system mtu

System MTU size is 1600 bytes
System Jumbo MTU size is 9000 bytes
System Alternate MTU size is 1600 bytes
Routing MTU size is 1600 bytes
```

8.2.5. 配置 LLDP 发现（LLDP discovery）

ironic-python-agent 服务会监听来自于连接的交换机中的 LLDP 数据包。它所收集的信息包括交换机名、端口详情和有效的 VLAN。与 CDP（Cisco Discovery Protocol）相似，LLDP 会在 director 的 **introspection（内省）**过程中帮助发现物理硬件。

8.2.5.1. 在 Cisco Catalyst 交换机上配置 LLDP

1. 使用 **lldp run** 在 Cisco Catalyst 交换机的全局范围内启用 LLDP：

```
sw01# config t
Enter configuration commands, one per line.  End with CNTL/Z.

sw01(config)# lldp run
```

2. 查看附近 LLDP 兼容的设置：

```
sw01# show lldp neighbor
Capability codes:
    (R) Router, (B) Bridge, (T) Telephone, (C) DOCSIS Cable Device
    (W) WLAN Access Point, (P) Repeater, (S) Station, (O) Other

Device ID           Local Intf     Hold-time  Capability      Port ID
DEP4203706l562G3     Gi1/0/11       180        B,T
4220370l562G3:P1

Total entries displayed: 1
```

注意

为了使所做的修改生效，把 running-config 复制到 startup-config：**copy running-config startup-config**。

8.3. 配置 Cisco Nexus 交换机

8.3.1. 配置主干（trunk）端口

OpenStack Networking 允许实例连接到已存在于物理网络中的 VLAN。*主干（trunk）*这个术语被用来描述一个端口，这个端口允许多个 VLAN 通过它。通过主干，VLAN 就可以分布于包括虚拟交换机在内的多个交换机中。例如，物理网络中的带有 **VLAN110** tag 的网络数据可以到达 Compute 节点，它的 8021q 模块会把带有 tag 的网络数据发送到 vSwitch 中的相关 VLAN 中。

8.3.1.1. 为 Cisco Nexus 交换机配置主干（trunk）端口

如果使用 Cisco Nexus，则需要使用以下配置来允许到 VLAN 110 和 111 的数据可以被发送到您的实例。这个配置假设物理节点已有一个连接到物理交换机中的 **Ethernet1/12** 接口的以太网网线。

注意

以下只是一个示例。如果在没有根据您的具体情况对相关值进行修改的情况下把它们复制到您的交换机配置中，则可能会导致一些意料外的问题。

```
interface Ethernet1/12
  description Trunk to Compute Node
  switchport mode trunk
  switchport trunk allowed vlan 1,110,111
  switchport trunk native vlan 1
end
```

8.3.2. 配置访问端口

因为不是 Compute 节点上的所有网卡都需要处理网络数据，所以不需要配置为允许多个 VLAN 通过。这些端口只需要配置一个 VLAN 就可能实现其它操作需求（如传递管理网络流量或 Block Storage 的数据）。这些端口通常被称为访问端口（access port），它们的配置会比 trunk 端口的配置简单。

8.3.2.1. 为 Cisco Nexus 交换机配置访问端口

继续使用以前的示图，这个例子把 Cisco Nexus 交换机上的 Ethernet1/13 配置为 eth1 的一个访问端口。这个配置假设您的物理节点有一个连接到物理交换机中的 **Ethernet1/13** 接口的以太网网线。

注意

以下只是一个示例。如果在没有根据您的具体情况对相关值进行修改的情况下把它们复制到您的交换机配置中，则可能会导致一些意料外的问题。

```
interface Ethernet1/13
 description Access port for Compute Node
 switchport mode access
 switchport access vlan 200
```

8.3.3. 配置 LACP 端口聚合

LACP 允许您把多个物理网卡结合在一起成为一个单独的逻辑通道。LACP 在 Linux 中也被称为绑定模式 4，它会创建一个动态的绑定来实现负载均衡和容错功能。LACP 需要在物理的 NIC 端和物理的交换机端口端都进行配置。

8.3.3.1. 在网络网卡上配置 LACP

1. 编辑 */home/stack/network-environment.yaml* 文件：

```
- type: linux_bond
  name: bond1
  mtu: 9000
```

```
  bonding_options:{get_param: BondInterfaceOvsOptions};
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000
```

2. 配置 Open vSwitch 网桥来使用 **LACP**：

```
BondInterfaceOvsOptions:
    "mode=802.3ad"
```

如需了解配置网络绑定的信息，请参阅 Director 安装和使用指南。

8.3.3.2. 在 Cisco Nexus 交换机上配置 LACP

在这个例子中，Compute 节点有两个使用 VLAN 100 的网卡：

1. 把 Compute 节点的两个网卡物理地连接到交换机（例如，连接到端口 12 和 13）。

2. 确认 LACP 已被启用：

```
(config)# show feature | include lacp
lacp                  1         enabled
```

3. 把端口 1/12 和 1/13 配置为访问端口，并作为一个频道组的成员：

```
interface Ethernet1/13
 description Access port for Compute Node
 switchport mode access
 switchport access vlan 200
 channel-group 10 mode active

interface Ethernet1/13
 description Access port for Compute Node
 switchport mode access
 switchport access vlan 200
 channel-group 10 mode active
```

8.3.4. 配置 MTU 设置

一些特定的网络流量可能需要对 MTU 的大小进行调整才可以正常工作。例如，特定的 NFS 或 iSCSI 网络数据可能需要设置巨型帧（9000 字节）。

注意

对 MTU 设置的修改需要在数据经过的所有节点上进行，这包括虚拟交换机。如需了解更多与在 OpenStack 环境中修改 MTU 的信息，请参阅 第 9 章 *配置 MTU 设置*。

8.3.4.1. 在 Cisco Nexus 7000 交换机上配置 MTU 设置

在 7000 系列的交换机中，MTU 的设置可以只应用于一个单独的接口。以下命令会把接口 1/12 配置为使用 9000 字节的巨型帧：

```
interface ethernet 1/12
  mtu 9216
  exit
```

8.3.5. 配置 LLDP 发现（LLDP discovery）

`ironic-python-agent` 服务会监听来自于连接的交换机中的 LLDP 数据包。它所收集的信息包括交换机名、端口详情和有效的 VLAN。与 CDP（Cisco Discovery Protocol）相似，LLDP 会在 director 的 **`introspection`（内省）**过程中帮助发现物理硬件。

8.3.5.1. 在 Cisco Nexus 7000 交换机上配置 LLDP

在 Cisco Nexus 7000 系列交换机中可以为单独的接口启用 LLDP：

```
interface ethernet 1/12
  lldp transmit
  lldp receive

interface ethernet 1/13
  lldp transmit
  lldp receive
```

注意

为了使所做的修改生效，把 running-config 复制到 startup-config：**`copy running-config startup-config`**。

8.4. 配置 Cumulus Linux 交换机

8.4.1. 配置主干（trunk）端口

OpenStack Networking 允许实例连接到已存在于物理网络中的 VLAN。*主干（trunk）*这个术语被用来描述一个端口，这个端口允许多个 VLAN 通过它。通过主干，VLAN 就可以分布于包括虚拟交换机在内的多个交换机中。例如，物理网络中的带有 **`VLAN110`** tag 的网络数据可以到达 Compute 节点，它的 8021q 模块会把带有 tag 的网络数据发送到 vSwitch 中的相关 VLAN 中。

8.4.1.1. 为 Cumulus Linux 交换机配置主干（trunk）端口

如果使用 Cumulus Linux 交换机，需要使用以下配置来允许 VLANs 100 和 200 的数据可以发送到您的示例。这个配置假设物理节点已有一个连接到物理交换机中的端口 **`swp1`** 和 **`swp2`** 的收发器。

注意

以下只是一个示例。如果在没有根据您的具体情况对相关值进行修改的情况下把它们复制到您的交换机配置中，则可能会导致一些意料外的问题。

```
auto bridge
iface bridge
  bridge-vlan-aware yes
  bridge-ports glob swp1-2
  bridge-vids 100 200
```

8.4.2. 配置访问端口

因为不是 Compute 节点上的所有网卡都需要处理网络数据，所以不需要配置为允许多个 VLAN 通过。这些端口只需要配置一个 VLAN 就可能实现其它操作需求（如传递管理网络流量或 Block Storage 的数据）。这些端口通常被称为访问端口（access port），它们的配置会比 trunk 端口的配置简单。

8.4.2.1. 为 Cumulus Linux 交换机配置访问端口

继续使用以前的示图，这个例子把 Cumulus Linux 交换机上的 **swp1** 配置为一个访问端口。这个配置假设您的物理节点有一个连接到物理交换机中的接口的以太网网线。Cumulus Linux 使用 **eth** 作为管理接口，使用 **swp** 作为访问/主干接口。

注意

以下只是一个示例。如果在没有根据您的具体情况对相关值进行修改的情况下把它们复制到您的交换机配置中，则可能会导致一些意料外的问题。

```
auto bridge
iface bridge
  bridge-vlan-aware yes
  bridge-ports glob swp1-2
  bridge-vids 100 200


auto swp1
iface swp1
  bridge-access 100


auto swp2
iface swp2
  bridge-access 200
```

8.4.3. 配置 LACP 端口聚合

LACP 允许您把多个物理网卡结合在一起成为一个单独的逻辑通道。LACP 在 Linux 中也被称为绑定模式 4，它会创建一个动态的绑定来实现负载均衡和容错功能。LACP 需要在物理的 NIC 端和物理的交换机端口端都进行配置。

8.4.3.1. 在网络网卡上配置 LACP

Cumulus Linux 中的物理网卡不需要配置。

8.4.3.2. 在 Cumulus Linux 交换机上配置 LACP

在 */etc/network/interfaces* 中为 **bond0** 添加以下内容来配置绑定：

```
auto bond0
iface bond0
   address 10.0.0.1/30
   bond-slaves swp1 swp2 swp3 swp4
```

注意

运行 **sudo ifreload -a** 来重新加载更新的配置以使修改生效

8.4.4. 配置 MTU 设置

一些特定的网络流量可能需要对 MTU 的大小进行调整才可以正常工作。例如，特定的 NFS 或 iSCSI 网络数据可能需要设置巨型帧（9000 字节）。

注意

对 MTU 设置的修改需要在数据经过的所有节点上进行，这包括虚拟交换机。如需了解更多与在 OpenStack 环境中修改 MTU 的信息，请参阅 第 9 章 *配置 MTU 设置*。

8.4.4.1. 在 Cumulus Linux 交换机上配置 MTU 设置

以下例子会在 Cumulus Linux 交换机上启用巨型帧。

```
auto swp1
iface swp1
  mtu 9000
```

运行 **sudo ifreload -a** 来重新加载更新的配置以使修改生效

8.4.5. 配置 LLDP 发现（LLDP discovery）

在默认情况下，LLDP 服务（lldpd）会在交换机引导时以守护进程的形式运行。

运行以下命令查看所有端口/接口附近的所有 LLDP

```
cumulus@switch$ netshow lldp
Local Port  Speed  Mode         Remote Port   Remote Host Summary
----------  ---    ---------    -----  -----  ----------- --------
eth0        10G    Mgmt         ====   swp6   mgmt-sw     IP:
10.0.1.11/24
swp51       10G    Interface/L3 ====   swp1   spine01     IP:
10.0.0.11/32
swp52       10G    Interface/L  ====   swp1   spine02     IP:
10.0.0.11/32
```

8.5. 配置 Extreme Networks EXOS 交换机

8.5.1. 配置主干（trunk）端口

OpenStack Networking 允许实例连接到已存在于物理网络中的 VLAN。*主干（trunk）*这个术语被用来描述一个端口，这个端口允许多个 VLAN 通过它。通过主干，VLAN 就可以分布于包括虚拟交换机在内的多个交换机中。例如，物理网络中的带有 **VLAN110** tag 的网络数据可以到达 Compute 节点，它的 8021q 模块会把带有 tag 的网络数据发送到 vSwitch 中的相关 VLAN 中。

8.5.1.1. 在 Extreme Networks EXOS 交换机上配置主干网络

如果使用 X-670 系列交换机，可能会需要使用以下配置来允许到 VLANs 110 和 111 的数据可以被发送到您的实例。这个配置假设物理节点已有一个连接到物理交换机中的 **24** 接口的以太网网线。在这个例子中，**DATA** 和 **MNGT** 是 VLAN 的名称。

注意

以下只是一个示例。如果在没有根据您的具体情况对相关值进行修改的情况下把它们复制到您的交换机配置中，则可能会导致一些意料外的问题。

```
#create vlan DATA tag 110
#create vlan MNGT tag 111
#configure vlan DATA add ports 24 tagged
#configure vlan MNGT add ports 24 tagged
```

8.5.2. 配置访问端口

因为不是 Compute 节点上的所有网卡都需要处理网络数据，所以不需要配置为允许多个 VLAN 通过。这些端口只需要配置一个 VLAN 就可能实现其它操作需求（如传递管理网络流量或 Block Storage 的数据）。这些端口通常被称为访问端口（access port），它们的配置会比 trunk 端口的配置简单。

8.5.2.1. 为 Extreme Networks EXOS 交换机配置访问端口

继续使用以前的示图，这个例子把 Extreme Networks X-670 系列交换机上的 10 配置为 eth1 的一个访问端口。您可以使用以下配置来允许 VLAN **110** 和 **111** 的数据通过来到达您的实例。这个配置假设您的物理节点有一个连接到物理交换机中的 **10** 接口的以太网网线。

注意

以下只是一个示例。如果在没有根据您的具体情况对相关值进行修改的情况下把它们复制到您的交换机配置中，则可能会导致一些意料外的问题。

```
create vlan VLANNAME tag NUMBER
configure vlan Default delete ports PORTSTRING
configure vlan VLANNAME add ports PORTSTRING untagged
```

例如：

```
#create vlan DATA tag 110
#configure vlan Default delete ports 10
#configure vlan DATA add ports 10 untagged
```

8.5.3. 配置 LACP 端口聚合

8.5.3. 配置 LACP 端口聚合

LACP 允许您把多个物理网卡结合在一起成为一个单独的逻辑通道。LACP 在 Linux 中也被称为绑定模式 4，它会创建一个动态的绑定来实现负载均衡和容错功能。LACP 需要在物理的 NIC 端和物理的交换机端口端都进行配置。

8.5.3.1. 在网络网卡上配置 LACP

1. 编辑 */home/stack/network-environment.yaml* 文件：

```
- type: linux_bond
  name: bond1
  mtu: 9000
  bonding_options:{get_param: BondInterfaceOvsOptions};
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000
```

2. 配置 Open vSwitch 网桥来使用 **LACP**：

```
BondInterfaceOvsOptions:
    "mode=802.3ad"
```

如需了解配置网络绑定的信息，请参阅 Director 安装和使用指南。

8.5.3.2. 在 Extreme Networks EXOS 交换机上配置 LACP

在这个例子中，Compute 节点有两个使用 VLAN 100 的网卡：

```
enable sharing MASTERPORT grouping ALL_LAG_PORTS lacp
configure vlan VLANNAME add ports PORTSTRING tagged
```

例如：

```
#enable sharing 11 grouping 11,12 lacp
#configure vlan DATA add port 11 untagged
```

8.5.4. 配置 MTU 设置

一些特定的网络流量可能需要对 MTU 的大小进行调整才可以正常工作。例如，特定的 NFS 或 iSCSI 网络数据可能需要设置巨型帧（9000 字节）。

注意

对 MTU 设置的修改需要在数据经过的所有节点上进行，这包括虚拟交换机。如需了解更多与在 OpenStack 环境中修改 MTU 的信息，请参阅 第 9 章 *配置 MTU 设置*。

8.5.4.1. 在 Extreme Networks EXOS 交换机上配置 MTU 设置

以下例子会在 Extreme Networks EXOS 交换机上启用巨型帧，并支持转发带有 9000 字节的 IP 数据包：

```
enable jumbo-frame ports PORTSTRING
configure ip-mtu 9000 vlan VLANNAME
```

例如：

```
# enable jumbo-frame ports 11
# configure ip-mtu 9000 vlan DATA
```

8.5.5. 配置 LLDP 发现（LLDP discovery）

`ironic-python-agent` 服务会监听来自于连接的交换机中的 LLDP 数据包。它所收集的信息包括交换机名、端口详情和有效的 VLAN。与 CDP（Cisco Discovery Protocol）相似，LLDP 会在 director 的 **`introspection`（内省）**过程中帮助发现物理硬件。

8.5.5.1. 在 Extreme Networks EXOS 交换机上配置 LLDP 设置

这个例子允许在任何 Extreme Networks EXOS 交换机上配置 LLDP 设置。在这个例子中，**11** 代表端口字符串：

```
enable lldp ports 11
```

8.6. 配置 Juniper EX 系列交换机

8.6.1. 配置主干（trunk）端口

OpenStack Networking 允许实例连接到已存在于物理网络中的 VLAN。*主干（trunk）*这个术语被用来描述一个端口，这个端口允许多个 VLAN 通过它。通过主干，VLAN 就可以分布于包括虚拟交换机在内的多个交换机中。例如，物理网络中的带有 **`VLAN110`** tag 的网络数据可以到达 Compute 节点，它的 8021q 模块会把带有 tag 的网络数据发送到 vSwitch 中的相关 VLAN 中。

8.6.1.1. 在 Juniper EX 系列交换机上配置主干端口

如果使用运行 Juniper JunOS 的 Juniper EX 系列交换机，则可能需要使用以下配置来允许到 **`110`** 和 **`111`** 的数据可以被发送到您的实例。这个配置假设物理节点已有一个连接到物理交换机中的 **`ge-1/0/12`** 接口的以太网网线。

注意

以下只是一个示例。如果在没有根据您的具体情况对相关值进行修改的情况下把它们复制到您的交换机配置中，则可能会导致一些意料外的问题。

```
ge-1/0/12 {
         description Trunk to Compute Node;
             unit 0 {
                 family ethernet-switching {
                     port-mode trunk;
```

```
                    vlan {
                        members [110 111];
                         }
                    native-vlan-id 1;
                }
            }
}
```

8.6.2. 配置访问端口

因为不是 Compute 节点上的所有网卡都需要处理网络数据，所以不需要配置为允许多个 VLAN 通过。这些端口只需要配置一个 VLAN 就可能实现其它操作需求（如传递管理网络流量或 Block Storage 的数据）。这些端口通常被称为访问端口（access port），它们的配置会比 trunk 端口的配置简单。

8.6.2.1. 为 Juniper EX 系列交换机配置访问端口

继续使用以前的示图，这个例子把 Juniper EX 系列交换机上的 ge-1/0/13 配置为 eth1 的一个访问端口。这个配置假设您的物理节点有一个连接到物理交换机中的 **ge-1/0/13** 接口的以太网网线。

注意

以下只是一个示例。如果在没有根据您的具体情况对相关值进行修改的情况下把它们复制到您的交换机配置中，则可能会导致一些意料外的问题。

```
 ge-1/0/13 {
          description Access port for Compute Node
              unit 0 {
                  family ethernet-switching {
                      port-mode access;
                      vlan {
                          members 200;
                           }
                      native-vlan-id 1;
                  }
              }
}
```

8.6.3. 配置 LACP 端口聚合

LACP 允许您把多个物理网卡结合在一起成为一个单独的逻辑通道。LACP 在 Linux 中也被称为绑定模式 4，它会创建一个动态的绑定来实现负载均衡和容错功能。LACP 需要在物理的 NIC 端和物理的交换机端口端都进行配置。

8.6.3.1. 在网络网卡上配置 LACP

1. 编辑 */home/stack/network-environment.yaml* 文件：

```
- type: linux_bond
  name: bond1
  mtu: 9000
  bonding_options:{get_param: BondInterfaceOvsOptions};
```

```
  members:
    - type: interface
      name: nic3
      mtu: 9000
      primary: true
    - type: interface
      name: nic4
      mtu: 9000
```

2. 配置 Open vSwitch 网桥来使用 **LACP**：

```
BondInterfaceOvsOptions:
    "mode=802.3ad"
```

如需了解配置网络绑定的信息，请参阅 Director 安装和使用指南。

8.6.3.2. 在 EX 系列交换机上配置 LACP

在这个例子中，Compute 节点有两个使用 VLAN 100 的网卡：

1. 把 Compute 节点的两个网卡物理地连接到交换机（例如，连接到端口 12 和 13）。

2. 创建端口聚合：

```
chassis {
    aggregated-devices {
        ethernet {
            device-count 1;
        }
    }
}
```

3. 配置交换端口 12 (ge-1/0/12) 和 13 (ge-1/0/13) 加入到端口聚合 **ae1**：

```
interfaces {
    ge-1/0/12 {
        gigether-options {
            802.3ad ae1;
        }
    }
    ge-1/0/13 {
        gigether-options {
            802.3ad ae1;
            }
        }
}
```

4. 在端口聚合 **ae1** 上启用 LACP：

```
interfaces {
    ae1 {
        aggregated-ether-options {
            lacp {
                active;
```

```
            }
        }
    }
}
```

5. 把聚合 **ae1** 添加到 VLAN 100：

```
interfaces {
    ae1 {
        vlan-tagging;
        unit 100 {
            vlan-id 100;
        }
    }
}
```

6. 复查新的端口频道。这个结果输出应该列出新创建的端口聚合 **ae1**，它包括 **ge-1/0/12** 和 **ge-1/0/13**：

```
> show lacp statistics interfaces ae1

Aggregated interface: ae1
LACP Statistics: LACP Rx LACP Tx Unknown Rx Illegal Rx
ge-1/0/12 0 0 0 0
ge-1/0/13 0 0 0 0
```

注意

运行 **commit** 命令来使所做的改变生效。

8.6.4. 配置 MTU 设置

一些特定的网络流量可能需要对 MTU 的大小进行调整才可以正常工作。例如，特定的 NFS 或 iSCSI 网络数据可能需要设置巨型帧（9000 字节）。

注意

对 MTU 设置的修改需要在数据经过的所有节点上进行，这包括虚拟交换机。如需了解更多与在 OpenStack 环境中修改 MTU 的信息，请参阅 第 9 章 *配置 MTU 设置*。

8.6.4.1. 在 Juniper EX 系列交换机上配置 MTU 设置

以下示例在 Juniper EX4200 交换机上启用巨型帧。

1. 对于 Juniper EX 系列交换机，MTU 的设置是针对于独立接口的。以下命令在 ge-1/0/14 和 ge-1/0/15 端口上配置巨型帧：

```
set interfaces ge-1/0/14 mtu 9216
set interfaces ge-1/0/15 mtu 9216
```

注意

运行 **commit** 命令保存所做的改变。

2. 如果使用 LACP 聚合，需要在聚合中设置 MTU 的大小，而不是在成员网卡中进行设置。例如，以下在 **ae1** 聚合中设置 MTU 的大小：

```
set interfaces ae1 mtu 9200
```

8.6.5. 配置 LLDP 发现（LLDP discovery）

ironic-python-agent 服务会监听来自于连接的交换机中的 LLDP 数据包。它所收集的信息包括交换机名、端口详情和有效的 VLAN。与 CDP（Cisco Discovery Protocol）相似，LLDP 会在 director 的 **introspection（内省）**过程中帮助发现物理硬件。

8.6.5.1. 在 Juniper EX 系列交换机上配置 LLDP

您可以在所有接口中启用 LLDP，或只在独立端口上启用：

1. 例如，使用以下方法在 Juniper EX 4200 交换机的所有接口上启用 LLDP：

```
lldp {
 interface all{
  enable;
 }
 }
}
```

2. 或者，只在接口 **ge-1/0/14** 中启用 LLDP：

```
lldp {
 interface ge-1/0/14{
  enable;
 }
 }
}
```

注意

运行 **commit** 命令来使所做的改变生效。

部分 II. 高级配置

以实用手册的形式介绍了高级的 OpenStack Networking 特性。

第 9 章 配置 MTU 设置

9.1. MTU 概述

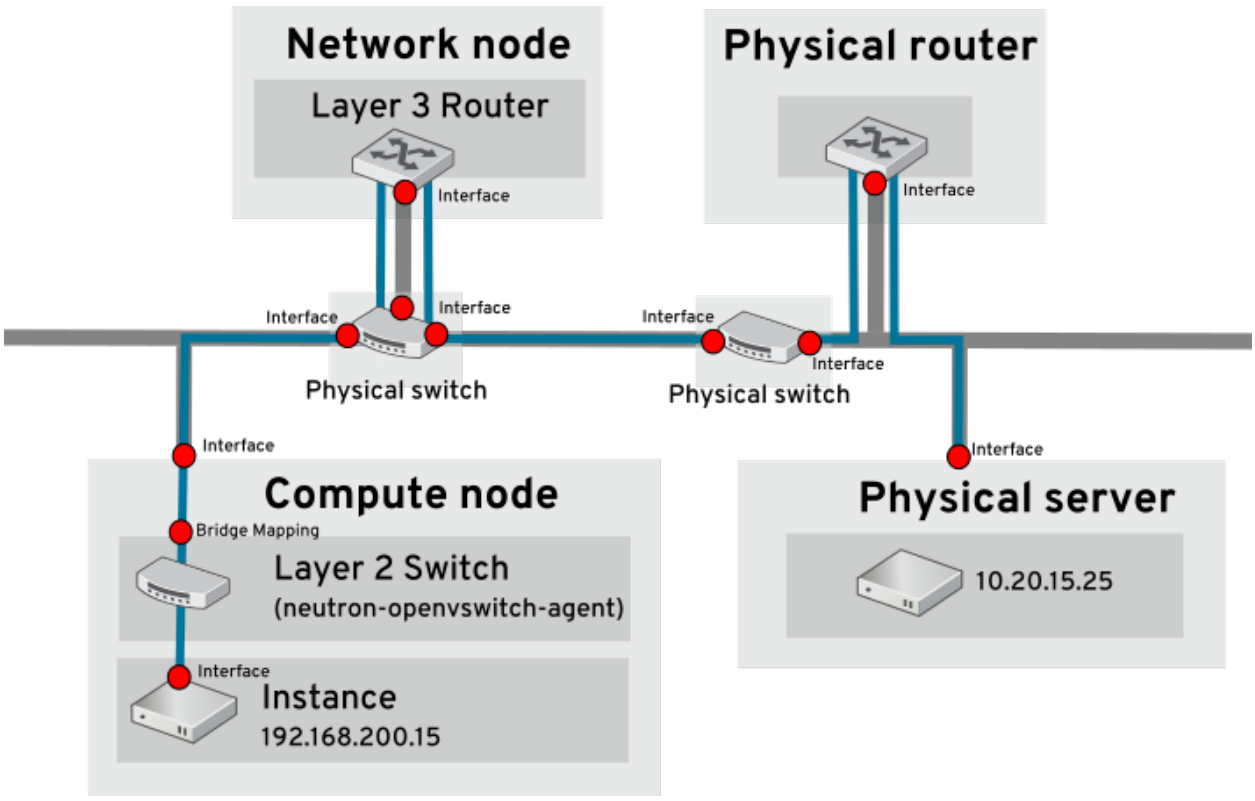
在 Red Hat OpenStack Platform 9 (Mitaka) 中，OpenStack Networking 可以计算出可以被安全应用到实例中的最大 MTU 的大小。MTU 的值设定了可以被传播的一个网络数据包的最大数据量，它需要根据应用程序的实际情况进行适当设置。例如，NFS 共享和 VoIP 应用的 MTU 的值就可能不一样。

注意

在 Red Hat OpenStack Platform 9 (Mitaka) 中，OpenStack Networking 可以计算出可以被安全应用到实例中的最大 MTU 的大小，然后用户可以通过 *neutron net-show* 命令检查这个值。如果被实例支持，所需的 MTU 值可以被广告到 DHCPv4 客户端以实现自动配置。

为了可以正常工作，MTU 的设置在端点到端点间应该保持一致。这意味着，在数据包要经过的所有端点中（包括虚拟机本身、虚拟网络基础架构、物理网络和目标服务器本身），MTU 的设置都需要相同。

例如，下图中的红虚线代表了，为了在一个实例和物理服务器间传输数据包，需要调整 MTU 值的端点。每个需要处理网络数据的接口都需要修改 MTU 值来支持特定的 MTU 大小。如果数据需要在不被分段的情况下从实例 *192.168.200.15* 传送到物理服务器 *10.20.15.25*，则需要这个设置：



不统一的 MTU 值可能会导致多种网络问题，其中最常见的问题就是网络性能的降低。这些问题不太容易被排除，这需要检查每个可能的网络端点是否设置了正确的 MTU 值。

9.1.1. 配置 MTU 广告

MTU 广告（MTU advertisement）可以简化 MTU 配置的过程。它使用 DHCP 配置来设置 MTU，这样最佳的 MTU 的值就可以通过 DHCPv4 广告给所有实例。在 */etc/neutron/neutron.conf* 中启用 MTU 广告功能：

```
advertise_mtu = True
```

租户网络配置的 MTU 选项会被广告到使用 DHCPv4 的实例中。

注意

不是所有的 DHCPv4 客户端都可以自动配置 MTU 的值。

9.1.2. 配置租户网络

在 *Red Hat OpenStack Platform 9* director 中，您可以在网络环境文件中使用一个参数来为所有租户网络设置默认的 MTU 值。这将可以方便地使您的配置与物理 MTU 相一致：

- **NeutronTenantMtu** - 虚拟以太网设备的 MTU 值。如果使用 VXLAN/GRE 隧道，它的值需要最少比物理网络中的 MTU 值小 50 个字节。在默认情况下，它的值是 **1400**。

9.1.3. 在 Director 中配置 MTU 的设置

这个示例展示了如何使用 NIC config 模板设置 MTU 的方法。MTU 需要在网桥、绑定（如果使用）、接口和 VLAN 中设置：

```
            -
              type: ovs_bridge
              name: br-isolated
              use_dhcp: false
              mtu: 9000    # <--- Set MTU
              members:
                -
                  type: ovs_bond
                  name: bond1
                  mtu: 9000    # <--- Set MTU
                  ovs_options: {get_param: BondInterfaceOvsOptions}
                  members:
                    -
                      type: interface
                      name: ens15f0
                      mtu: 9000    # <--- Set MTU
                      primary: true
                    -
                      type: interface
                      name: enp131s0f0
                      mtu: 9000    # <--- Set MTU
                -
                  type: vlan
                  device: bond1
                  vlan_id: {get_param: InternalApiNetworkVlanID}
                  mtu: 9000    # <--- Set MTU
                  addresses:
                  -
                    ip_netmask: {get_param: InternalApiIpSubnet}
                -
                  type: vlan
                  device: bond1
                  mtu: 9000    # <--- Set MTU
```

```
                vlan_id: {get_param: TenantNetworkVlanID}
                addresses:
                -
                  ip_netmask: {get_param: TenantIpSubnet}
```

9.1.4. 检查 MTU 的计算结果

查看计算出的 MTU 值。它代表了计算出的，可以被实例使用的 MTU 的最大值。然后，您可以在网络数据经过的所有接口中配置这个值。

```
# neutron net-show <network>
```

注意

在目前，推荐把实例内的 MTU 值设置为一个小于 1500 字节的值（如 1450 字节）。这个较小的值代表了 Open vSwitch 用于路由目的所要添加的额外的头数据。

第 10 章 配置 QOS

Red Hat OpenStack Platform 9 推出了对网络服务质量（QoS）策略的支持。OpenStack 管理员可以使用这些策略，通过为实例的网络速度设置不同限制来实现提供不同服务级别的功能。当网络流量超过所设定的限制时，相关网络通讯会被忽略。

10.1. QoS 策略范围

QoS 策略可以应用到单独的端口上，也可以应用到一个特定的租户网络中（其中的所有端口都会继承这个策略）。

10.2. QoS 策略管理

QoS 策略可以被动态应用、修改或删除。这个示例手工创建了一个带宽限制规则并被应用到一个端口。

1. 查看租户列表并决定需要创建 QoS 策略的租户 ID：

```
# openstack project list
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| 4b0b98f8c6c040f38ba4f7146e8680f5 | auditors |
| 519e6344f82e4c079c8e2eabb690023b | services |
| 80bf5732752a41128e612fe615c886c6 | demo     |
| 98a2f53c20ce4d50a40dac4a38016c69 | admin    |
+----------------------------------+----------+
```

2. 在 **admin** 租户中创建名为 **bw-limiter** 的 QoS 策略：

```
# neutron qos-policy-create 'bw-limiter' --tenant-id
98a2f53c20ce4d50a40dac4a38016c69
```

3. 为 **bw-limiter** 策略配置策略规则：

```
# neutron qos-bandwidth-limit-rule-create bw-limiter --max_kbps
3000 --max_burst_kbps 300
```

4. 配置一个 neutron 端口来应用 **bw-limiter** 策略：

```
# neutron port-update <port id> --qos-policy bw-limiter
```

5. 查看 QoS 规则。例如：

```
# neutron qos-rule-show 9be535c3-daa2-4d7b-88ea-e8de16

+------------------+----------------------------------+
| Field            | Value                            |
+------------------+----------------------------------+
| id               | 9be535c3-daa2-4d7b-88ea-e8de16   |
| rule_type        | bandwidth_limit                  |
```

```
| description       |                                  |
| max_kbps          | 3000                             |
| max_burst_kbps    | 300                              |
+-------------------+----------------------------------+
```

使用以下值来配置相应的策略算法：

- **max_kbps** - 实例允许发送的最大速率（以 Kbps 为单位）。
- **max_burst_kbps** - 接口可以发送的、超过策略数据的最大数据量（以 Kbps 为单位）。

10.3. QoS 策略的 RBAC

Red Hat OpenStack Platform 9 为 QoS 策略增加了 RBAC（Role-based Access Control - 基于角色的访问控制）。作为结果，现在可以针对于特定项目应用 QoS 策略。

例如，您现在可以为低优先级网络数据设置一个 QoS 策略，并只在特定项目中应用它。以下命令会把以前创建的 *bw-limiter* 策略分配给 *demo* 租户：

```
# neutron rbac-create 'bw-limiter' --type qos-policy --target-tenant
80bf5732752a41128e612fe615c886c6 --action access_as_shared
```

第 11 章 配置网桥映射

本章介绍了如何在 Red Hat OpenStack Platform 中配置*网桥映射*。

11.1. 网桥映射的作用

网桥映射允许供应商网络流量到达相关的物理网络。网络数据会从路由器的 *qg-xxx* 接口离开供应商网络，并到达 **br-int**。**br-int** 和 **br-ex** 间的 veth pair 允许网络数据通过供应商网络的网桥，并到达物理网络。

11.1.1. 配置网桥映射

以下是 **br-int** 和 **br-ex** 间的一个 veth pair 示例：

```
int-br-ex <-> phy-br-ex
```

这个连接在 bridge_mappings 设置中配置。例如：

```
bridge_mappings = physnet1:br-ex,physnet2:br-ex2
```

注意

如果没有 **bridge_mapping** 项，则不存在网络的连接，从而无法和外部网络进行交流。

这个配置的第一个项使用 patch peer 在 **br-int** 和 **br-ex** 间创建了一个连接；第 2 个项为 **br-ex2** 创建了一个 patch peer。

11.1.2. 配置控制器节点

bridge_mappings 的设置需要和控制器节点上的 **network_vlan_ranges** 选项相匹配。对于上面的例子，控制器节点的配置如下：

```
network_vlan_ranges = physnet1,physnet2
```

这些值创建了用来代表相关外部网络的供应商网络，外部网络通过路由器接口与租户网络进行连接。因此，需要在运行路由器的网络节点上配置 **bridge_mappings**。这意味着，路由器上的网络数据可以使用正确的、由供应商网络代表的物理网络（如 **physnet1**）向外发送。

11.1.3. 网络数据传输

除了创建连接外，这个设置还在 **br-int** 和 **br-ex** 上配置了 OVS 流来允许和外部网络进行通讯。每个外部网络由一个内部 VLAN id 代表，VLAN id 被 tag 到路由器的 **qg-xxx** 端口。当数据包到达 **phy-br-ex** 时，**br-ex** 端口会去掉 VLAN tag，并把数据包移到物理接口，然后发送到外部网络。从外部网络返回的数据包会首先到达 **br-ex**，然后使用 **phy-br-ex <→ int-br-ex** 移到 **br-int**。当数据包到达 **int-br-ex** 时，**br-int** 中的另外一个流会向数据包添加内部的 vlan tag。这样，这个数据包就可以被 **qg-xxx** 接受。

11.2. 维护网桥映射

在删除任何映射后，都需要进行一个 patch-port 清理的操作，这保证了错误的项已从网桥配置中删除。这个操作可以通过两个方法进行：

- 手工端口清除 - 需要仔细地手工删除无用的端口。这个方法不需要在停止网络连接的情况下进行。
- 使用 **neutron-ovs-cleanup** 自动进行端口删除 - 自动进行端口删除，但需要停止系统的正常工作，并添加所需的映射信息。如果您的环境允许停止网络连接服务，则可以使用这个方法。

下面给出了这两种方法的使用示例：

11.2.1. 手工进行端口清除

手工端口清除的操作可以在不停止系统运行的情况下，删除不需要的端口。您可以使用它们的命名规则指定这些端口：在 **br-$external_bridge** 中，它们被命名为 "phy-"**$external_bridge**；在 **br-int** 中，它们被命名为 "int-"**$external_bridge**。

这个例子会从 bridge_mappings 中删除一个网桥，并清理相关的端口。**1.** 编辑 **ovs_neutron_plugin.ini**，从 **bridge_mappings** 中删除 **physnet2:br-ex2** 的项：

```
bridge_mappings = physnet1:br-ex,physnet2:br-ex2
```

删除 **physnet2:br-ex2** 的项。**bridge_mappings** 将变为如下所示：

```
bridge_mappings = physnet1:br-ex
```

2. 使用 **ovs-vsctl** 删除与已被删除的 **physnet2:br-ex2** 项相关联的 patch 端口：

```
# ovs-vsctl del-port br-ex2 phy-br-ex2
# ovs-vsctl del-port br-int int-br-ex2
```

注意

如果整个 **bridge_mappings** 项都被删除，或被注释掉，则需要为每个项都运行清除命令。

2. 重启 **neutron-openvswitch-agent**：

```
# service neutron-openvswitch-agent restart
```

11.2.2. 自动进行端口清除

这个操作使用带有 **--ovs_all_ports** 标识的 **neutron-ovs-cleanup** 命令。重启 **neutron** 服务或整个节点来使网桥返回到正常的工作状态。这个过程需要完全停止网络服务。

neutron-ovs-cleanup 命令从所有 OVS 网桥中"拔掉"全部端口（实例、qdhcp/qrouter 以及其它）。使用 **--ovs_all_ports** 标识将会从 **br-int** 上删除所有端口，从 **br-tun** 中清理隧道端点，以及从网桥到网桥的端口连接。另外，物理接口（如 eth0、eth1）也会从网桥（如 br-ex、br-ex2）中删除。因此，在使用 **ovs-vsctl** 把这些端口重新手工添加前，到实例的连接都会丢失：

```
# ovs-vsctl add-port br-ex eth1
```

11.2.2.1. neutron-ovs-cleanup 使用示例：

1. 把 **ovs_neutron_plugin.ini** 中的 bridge_mapping 项进行一个备份。

2. 使用 **--ovs_all_ports** 标识运行 **neutron-ovs-cleanup**。请注意，这一步会导致整个网络停止工作。

```
# /usr/bin/neutron-ovs-cleanup
--config-file /usr/share/neutron/neutron-dist.conf
--config-file /etc/neutron/neutron.conf
--config-file /etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini
--log-file /var/log/neutron/ovs-cleanup.log --ovs_all_ports
```

3. 重启 OpenStack Networking 服务：

```
# systemctl restart neutron-openvswitch-agent
# systemctl restart neutron-l3-agent.service
# systemctl restart neutron-dhcp-agent.service
```

4. 重新把 **bridge_mapping** 项添加到 **ovs_neutron_plugin.ini** 来恢复网络连接。

5. 重启 **neutron-openvswitch-agent** 服务：

```
# systemctl restart neutron-openvswitch-agent
```

注意

当 OVS agent 重启时，它并不会影响到没有出现在 **bridge_mappings** 中的连接。因此，如果您有 **br-int** 连接到 **br-ex2**，**br-ex2** 在它上面有数据流，把它从 **bridge_mappings** 配置中删除（或把它注释掉）不会断开这两个网桥（无论您是否重启了服务或整个节点）。

第 12 章 配置 RBAC

在 OpenStack Networking 中使用 RBAC（Role-based Access Control，基于角色的访问控制）可以对共享 *neutron* 网络进行“颗粒式”的控制。以前，网络只可能在所有租户网络中共享，或不被任何网络共享。现在，OpenStack Networking 使用一个 RBAC 表来控制 *neutron* 网络在不同租户间的共享，管理员可以控制哪些租户有权限把实例附加到网络中。
作为结果，云管理员可以删除一些租户的创建网络的权限，而只允许他们附加那些与他们的项目相关的已存在的网络。

12.1. 创建一个新的 RBAC 策略

以下是如何使用 RBAC 策略来为一个租户赋予访问一个共享网络的步骤示例。

1. 查看所有有效网络的列表：

```
# neutron net-list
+--------------------------------------+-------------+--------
-----------------------------------------------+
| id                                   | name        | subnets
|
+--------------------------------------+-------------+--------
-----------------------------------------------+
| fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 | web-servers | 20512ffe-
ad56-4bb4-b064-2cb18fecc923 192.168.200.0/24 |
| bcc16b34-e33e-445b-9fde-dd491817a48a | private     | 7fe4a05a-
4b81-4a59-8c47-82c965b0e050 10.0.0.0/24      |
| 9b2f4feb-fee8-43da-bb99-032e4aaf3f85 | public      | 2318dc3b-
cff0-43fc-9489-7d4cf48aaab9 172.24.4.224/28  |
+--------------------------------------+-------------+--------
-----------------------------------------------+
```

2. 查看租户列表：

```
# openstack project list
+----------------------------------+----------+
| ID                               | Name     |
+----------------------------------+----------+
| 4b0b98f8c6c040f38ba4f7146e8680f5 | auditors |
| 519e6344f82e4c079c8e2eabb690023b | services |
| 80bf5732752a41128e612fe615c886c6 | demo     |
| 98a2f53c20ce4d50a40dac4a38016c69 | admin    |
+----------------------------------+----------+
```

3. 为 **`web-servers`** 网络创建一个 RBAC，它为 *auditors* tenant (**`4b0b98f8c6c040f38ba4f7146e8680f5`**) 赋予了访问权限：

```
# neutron rbac-create fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 --type
network --target-tenant 4b0b98f8c6c040f38ba4f7146e8680f5 --action
access_as_shared
Created a new rbac_policy:
+---------------+--------------------------------------+
| Field         | Value                                |
+---------------+--------------------------------------+
| action        | access_as_shared                     |
```

```
| id            | 314004d0-2261-4d5e-bda7-0181fcf40709 |
| object_id     | fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 |
| object_type   | network                              |
| target_tenant | 4b0b98f8c6c040f38ba4f7146e8680f5     |
| tenant_id     | 98a2f53c20ce4d50a40dac4a38016c69     |
+---------------+--------------------------------------+
```

作为结果，*auditors* 项目中的用户可以把实例连接到 **web-servers** 网络。

12.2. 检查 RBAC 策略

1. 使用 **neutron rbac-list** 获得已存在 RBAC 策略的 ID：

```
# neutron rbac-list
+--------------------------------------+-------------+--------
------------------------------+
| id                                   | object_type |
object_id                            |
+--------------------------------------+-------------+--------
------------------------------+
| 314004d0-2261-4d5e-bda7-0181fcf40709 | network     | fa9bb72f-
b81a-4572-9c7f-7237e5fcabd3 |
| bbab1cf9-edc5-47f9-aee3-a413bd582c0a | network     | 9b2f4feb-
fee8-43da-bb99-032e4aaf3f85 |
+--------------------------------------+-------------+--------
------------------------------+
```

2. 使用 **neutron rbac-show** 查看特定 RBAC 项的详细信息：

```
# neutron rbac-show 314004d0-2261-4d5e-bda7-0181fcf40709
+---------------+--------------------------------------+
| Field         | Value                                |
+---------------+--------------------------------------+
| action        | access_as_shared                     |
| id            | 314004d0-2261-4d5e-bda7-0181fcf40709 |
| object_id     | fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 |
| object_type   | network                              |
| target_tenant | 4b0b98f8c6c040f38ba4f7146e8680f5     |
| tenant_id     | 98a2f53c20ce4d50a40dac4a38016c69     |
+---------------+--------------------------------------+
```

12.3. 删除 RBAC 策略

1. 使用 **neutron rbac-list** 获得已存在 RBAC 策略的 ID：

```
# neutron rbac-list
+--------------------------------------+-------------+--------
------------------------------+
| id                                   | object_type |
object_id                            |
+--------------------------------------+-------------+--------
------------------------------+
| 314004d0-2261-4d5e-bda7-0181fcf40709 | network     | fa9bb72f-
```

```
b81a-4572-9c7f-7237e5fcabd3 |
| bbab1cf9-edc5-47f9-aee3-a413bd582c0a | network     | 9b2f4feb-
fee8-43da-bb99-032e4aaf3f85 |
+--------------------------------------+-------------+--------
-----------------------------+
```

2. 使用 **neutron rbac-delete** 删除 RBAC（使用它的 ID）：

```
# neutron rbac-delete 314004d0-2261-4d5e-bda7-0181fcf40709
Deleted rbac_policy: 314004d0-2261-4d5e-bda7-0181fcf40709
```

12.4. 外部网络的 RBA

通过使用 **--action access_as_external** 参数，RBAC 可以控制对外部网络（带有网关接口的网络）的访问。

例如，以下步骤为 **web-servers** 网络创建了一个 RBAC，并为 engineering 租户（**c717f263785d4679b16a122516247deb**）赋予了访问权限：

1. 使用 **--action access_as_external** 创建一个新的 RBAC 策略：

```
# neutron rbac-create 6e437ff0-d20f-4483-b627-c3749399bdca --type
network --target-tenant c717f263785d4679b16a122516247deb --action
access_as_external
 Created a new rbac_policy:
+---------------+--------------------------------------+
| Field         | Value                                |
+---------------+--------------------------------------+
| action        | access_as_external                   |
| id            | ddef112a-c092-4ac1-8914-c714a3d3ba08 |
| object_id     | 6e437ff0-d20f-4483-b627-c3749399bdca |
| object_type   | network                              |
| target_tenant | c717f263785d4679b16a122516247deb     |
| tenant_id     | c717f263785d4679b16a122516247deb     |
+---------------+--------------------------------------+
```

2. 作为结果，Engineering 租户中的用户可以查看这个网络，或把实例连接到这个网络：

```
$ neutron net-list
+--------------------------------------+-------------+-------------
----------------------------------------+
| id                                   | name        | subnets
|
+--------------------------------------+-------------+-------------
----------------------------------------+
| 6e437ff0-d20f-4483-b627-c3749399bdca | web-servers | fa273245-1eff-
4830-b40c-57eaeac9b904 192.168.10.0/24 |
+--------------------------------------+-------------+-------------
----------------------------------------+
```

第 13 章 配置分布式虚拟路由（DVR）

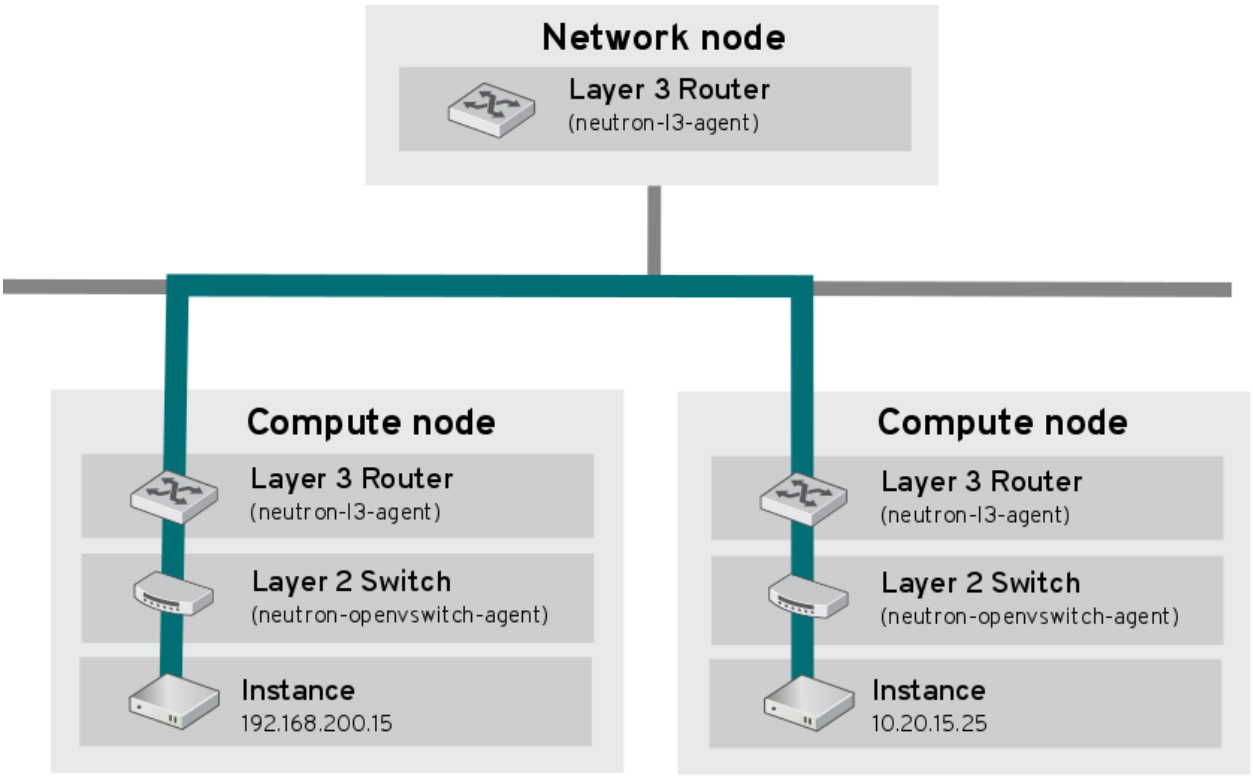
分布式虚拟路由（Distributed Virtual Routing，简称 DVR）允许用户在 Compute 节点上直接放置 L3 路由。这可以在不首先通过一个网络节点路由的情况下，在 Compute 节点间（East-West）进行网络通信。另外，浮动 IP 的命名空间会在相关的 Compute 节点间进行复制，分配了浮动 IP 的实例就可以在不通过网络节点路由的情况下向外部（North-South）发送网络数据。没有浮动 IP 地址的实例仍然需要通过 Network 节点路由 SNAT 网络数据。

Red Hat OpenStack Platform 7（kilo）添加了在使用分布式路由时，对 VLAN 和 VXLAN/GRE 之间连接的支持。现在，DVR 中的 VLAN 和 VXLAN/GRE 通道间可以进行相互连接。

注意

DVR 作为一个技术预览包括在 Red Hat OpenStack Platform 9 中。关于技术预览的支持范围的更多信息，请参考 https://access.redhat.com/support/offerings/techpreview/。

在以下的图例中，独立子网间的实例可以在不通过 Network 节点路由的情况下进行相互通讯：



13.1. 配置 DVR

1. 在 Network 节点中的 **`neutron.conf`** 文件中启用 **`router_distributed`**。这保证了以后所有创建的路由器在默认情况下都是分布式的。

```
router_distributed = True
```

您可以通过修改 **`policy.json`** 文件来改变默认的行为：

```
neutron router-create --distributed=True/False <name>
```

2. 配置第 3 层代理

在 Compute 节点上，编辑 **l3_agent.ini** 文件来启用 DVR：

```
agent_mode = dvr
```

在 Network 节点中的分布式路由器上配置 **dvr_snat**：

```
agent_mode = dvr_snat
```

3. 配置第 2 层 agent

在 Network 节点和 Compute 节点中的 L2 Agent 上启动 DVR 和 L2 population 功能。例如，如果使用 Open vSwitch，编辑 **ovs_neutron_plugin.ini** 文件：

```
enable_distributed_routing = True
l2_population = True
```

4. 在 ML2 上启用 L2 population 机制驱动

在 Controller 上编辑 **ml2_conf.ini** 文件：

```
[ml2]
mechanism_drivers = openvswitch, l2population #Other values may be
listed here as well
```

在 Compute 节点上编辑 **ml2_conf.ini** 文件：

```
[agent]
l2_population = True
```

5. 重启服务以使所做的修改生效：

在 Controller 上重启以下服务：

```
# systemctl restart neutron-server.service
# systemctl restart neutron-l3-agent.service
# systemctl restart neutron-openvswitch-agent.service
```

在 Compute 节点上重启以下服务：

```
# systemctl restart neutron-l3-agent.service
# systemctl restart neutron-metadata-agent
```

注意

当前，还不支持把已存在的、非分布式的路由器转换为 DVR。

第 14 章 配置 BALANCING-AS-A-SERVICE（LBAAS）

负载均衡即服务（Load Balancing-as-a-Service，简称 LBaaS）使 OpenStack Networking 可以在相关的实例中平均分配入站的网络流量。以下介绍了配置 OpenStack Networking 通过Open vSwitch（OVS）或 Linux Bridge 插件来使用 LBaaS 的方法。

Red Hat OpenStack Platform 5 包括了负载均衡即服务（Load Balancing-as-a-Service，简称 LBaaS）功能，它使 OpenStack Networking 可以在相关的实例中平均分配入站的网络流量。这保证了入站的网络流量所带来的工作负载可以以可预测的方式在多个实例中共享，从而达到有效使用系统资源的目的。入站的网络流量以下面的方法之一进行分配：

- **轮转（Round robin）** - 在多个实例间均匀轮转分配请求。
- **源 IP** - 从同一个源 IP 地址发送的请求会被分配到同一个实例。
- **最少连接数量** - 把请求分配到有最少活跃连接数量的实例。

表 1：LBaaS 的特性

表 14.1. LBaaS 的特性

功能	描述
监控程序	LBaaS 通过 ping、TCP、HTTP 和 HTTPS GET 的方法提供了可用性的监控功能。监控程序的主要功能是决定池的成员是否可以处理请求。
管理	LBaaS 由一组工具程序进行管理。管理员可以使用 REST API 进行自动化管理或编写相关的脚本；用户可以通过 CLI（neutron）或 OpenStack dashboard 对负载均衡功能进行管理。
连接限制	入站的网络流量可以通过连接数量进行限制，这将可以用来进行负载的控制，并防止 DoS（Denial of Service）攻击。
会话持久性	LBaaS 可以把入站的请求路由到一个池中的同一个实例中，从而实现了连接会话的持久性。LBaaS 支持基于 cookie 或基于源 IP 地址的路由算法。

注意

LBaaS 当前只支持 IPv4 地址。

14.1. OpenStack Networking 和 LBaaS 拓扑

OpenStack Networking（neutron）服务大体可以被分为两类。

1. - **Neutron API 服务器** - 这个服务运行 OpenStack Networking API 服务器，它为最终用户提供了 API，以及和 OpenStack Networking 进行通讯的服务。这个服务器还负责和底层数据库进行交互来存储和获取租户网络、路由器、负载均衡设备以及其它信息。

2. - **Neutron Agent（代理）** - 为 OpenStack Networking 执行网络功能的服务：

- **neutron-dhcp-agent** - 为租户私人网络管理 DHCP IP 地址。
- **neutron-l3-agent** - 在租户私人网络、外部网络和其它系统间执行第 3 层路由。
- **neutron-lbaas-agent** - 配置由租户创建的 LBaaS 路由器。

14.1.1. 服务的位置

OpenStack Networking 服务可以在同一个物理服务器上运行，也可以分别在专用的服务器上运行。

运行 API 服务的服务器通常被称为 **Controller（控制器）节点**，而运行 the OpenStack Networking agent 的服务器被称为 **Network（网络）节点**。在一个理想的生产环境中，这些组件应该分别运行在独立的专用节点上以获得好的性能和可扩展性。但是，在一个测试环境或一个概念验证（PoC）部署中，可以把这些组件集中在一个节点上运行。本章分别介绍了这两种情况。关于 Controller 节点配置的内容需要在 API 服务器上运行，关于 Network 节点的内容需要在运行 LBaaS agent 的服务器上运行。

注意

如果 Controller 和 Network 节点都在同一个物理节点上运行，则所有这些步骤都需要在这个服务器上运行。

14.2. 配置 LBaaS

这个过程配置 OpenStack Networking 来使用带有 Open vSwitch（OVS）或 Linux Bridge 插件的 LBaaS。在为基于 OVS 的插件（包括 BigSwitch、Floodlight、NEC、NSX 和 Ryu）启用 LBaaS 时需要 Open vSwitch LBaaS 驱动。

注意

在默认情况下， Red Hat OpenStack Plaftform 包括了对 LBaaS 的 *HAProxy* 驱动的支持。如需了解支持的其它服务供应商的驱动信息，请参阅 https://access.redhat.com/certification。

在运行 **neutron-server** 服务的节点上执行这些操作：

在 Controller 节点上（API 服务器）：

1. 通过 **/etc/neutron/neutron_lbaas.conf** 文件中的 service_provider 参数启用 HAProxy 插件。

```
service_provider =
LOADBALANCER:Haproxy:neutron_lbaas.services.loadbalancer.drivers.haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
```

2. 通过设置 **/etc/neutron/neutron.conf** 文件中的 **service_plugin** 值来启用 LBaaS 插件：

-

```
service_plugins = lbaas
```

3. 重启 _neutron-server services 服务以使所做的修改生效。

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

14.2.1. 使用 Dashboard 启用 LBaaS 集成功能

通常情况下，Horizon dashboard 和 neutron API 服务运行在同一个节点上。您可以通过 Dashboard 中的项目界面启用负载均衡功能。在运行 Dashboard（horizon）服务的节点上执行这些步骤：

1. 把 **/etc/openstack-dashboard/local_settings** 文件中的 **enable_lb** 选项设置为 **True**：

```
OPENSTACK_NEUTRON_NETWORK = {'enable_lb': True,
```

2. 重启 *httpd* 服务以使所做的修改生效。

```
# systemctl restart httpd.service
```

现在，您可以在 dashboard 项目中的**网**络列表中看到负载均衡管理选项。

14.3. 配置网络节点（运行 LBaaS Agent 的节点）

1. 在 **/etc/neutron/lbaas_agent.ini** 文件中启用 HAProxy load balancer：

```
device_driver =
neutron.services.loadbalancer.drivers.haproxy.namespace_driver.HaproxyN
SDriver
```

2. 在 **/etc/neutron/lbaas_agent.ini** 中配置 **user_group** 选项

```
# The user group
# user_group = nogroup
user_group = haproxy
```

3. 在 **/etc/neutron/lbaas_agent.ini** 文件中选择所需的驱动：

- 如果使用 Open vSwitch 插件：

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

- 如果使用 Linux Bridge 插件：

```
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
```

4.重启 *neutron-lbaas-agent* 服务以使所做的修改生效。

```
# systemctl restart neutron-lbaas-agent.service
```

第 15 章 带有 IPV6 的租户网络

本章介绍了如何在一个租户网络中实现 IPv6 子网的信息。除了租户网络，director 7.3 还可以为 overcloud 节点配置原生 IPv6 的部署。

Red Hat OpenStack Platform 6 增加了在租户网络中使用 IPv6 的功能。IPv6 子网可以在存在的租户网络中创建，并支持以下地址分配模式：SLAAC（Stateless Address Autoconfiguration）、**Stateful DHCPv6** 和 **Stateless DHCPv6**。本章介绍了 IPv6 子网创建选项，并提供了一个创建示例。

15.1. IPv6 子网选项

IPv6 子网使用 **neutron subnet-create** 命令创建。另外，您还可以指定地址模式和 RA（Router Advertisement）模式。下面介绍了这些选项的组合：

RA 模式	地址模式	结果
ipv6_ra_mode=not set	ipv6-address-mode=slaac	这个实例使用 **SLAAC** 从外部路由（没有被 OpenStack Networking 管理的路由）接收 IPv6 地址。
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateful	这个实例使用 **DHCPv6 stateful** 从 OpenStack Networking（dnsmasq）接收 IPv6 地址和可选的信息。
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateless	这个实例使用 SLAAC 从外部路由接收一个 IPv6 地址，并使用 **DHCPv6 stateless** 从 OpenStack Networking（dnsmasq）接收可选的信息。
ipv6_ra_mode=slaac	ipv6-address-mode=not-set	实例使用 SLAAC 从 OpenStack Networking（**radvd**）接收一个 IPv6 地址。
ipv6_ra_mode=dhcpv6-stateful	ipv6-address-mode=not-set	这个实例使用 **DHCPv6 stateful** 从外部 DHCPv6 服务器接收一个 IPv6 地址和可选的信息。

RA 模式	地址模式	结果
ipv6_ra_mode=dhcpv6-stateless	ipv6-address-mode=not-set	这个实例使用 SLAAC 从 OpenStack Networking（**radvd**）接收一个 IPv6 地址，并使用 **DHCPv6 stateless** 从外部 DHCPv6 服务器接收可选的信息。
ipv6_ra_mode=slaac	ipv6-address-mode=slaac	实例通过 **SLAAC** 从 OpenStack Networking（**radvd**）接收一个 IPv6 地址。
ipv6_ra_mode=dhcpv6-stateful	ipv6-address-mode=dhcpv6-stateful	实例通过 **DHCPv6 stateful** 从 OpenStack Networking（**dnsmasq**）接收一个 IPv6 地址，使用 **DHCPv6 stateful** 从 OpenStack Networking（**dnsmasq**）获得其它可选信息。
ipv6_ra_mode=dhcpv6-stateless	ipv6-address-mode=dhcpv6-stateless	实例通过 **DHCPv6 stateful** 从 OpenStack Networking（**radvd**）接收一个 IPv6 地址，使用 **DHCPv6 stateless** 从 OpenStack Networking（**dnsmasq**）获得其它可选信息。

15.1.1. 使用 Stateful DHCPv6 创建一个 IPv6 子网

这个过程使用以上介绍的设置在一个租户网络中创建一个 IPv6 子网。它的初始步骤是收集所需的租户和网络信息，然后使用这些信息来生成一个子网创建命令。

注意

OpenStack Networking 只支持对 SLAAC 的 EUI-64 IPv6 地址分配。因为主机将可以基于基本的 64 位再加上 MAC 地址来自分配地址，所以这将可以简化 IPv6 网络。使用不同的网络掩码和 SLAAC 的 *address_assign_type* 创建子网将会失败。

1. 获得需要创建 IPv6 子网的项目的租户 id。（这个值在不同 OpenStack 环境中是不同的，因此您的值可能会和这里使用的值不同）。在这个例子中，*QA* 的租户将会接收一个 IPv6 子网。

```
# keystone tenant-list
+----------------------------------+----------+---------+
|                id                |   name   | enabled |
+----------------------------------+----------+---------+
```

```
| 25837c567ed5458fbb441d39862e1399 |    QA    |   True  |
| f59f631a77264a8eb0defc898cb836af |  admin   |   True  |
| 4e2e1951e70643b5af7ed52f3ff36539 |   demo   |   True  |
| 8561dff8310e4cd8be4b6fd03dc8acf5 | services |   True  |
+----------------------------------+----------+---------+
```

2. 获得 OpenStack Networking（neutron）中的所有网络的列表，并记录下包括 IPv6 子网的网络名称。在这个例子中，使用 *database-servers*。

```
# neutron net-list
+--------------------------------------+------------------+-------
-----------------------------------------------------+
| id                                   | name             | subnets
|
+--------------------------------------+------------------+-------
-----------------------------------------------------+
| 8357062a-0dc2-4146-8a7f-d2575165e363 | private          | c17f74c4-
db41-4538-af40-48670069af70 10.0.0.0/24            |
| 31d61f7d-287e-4ada-ac29-ed7017a54542 | public           | 303ced03-
6019-4e79-a21c-1942a460b920 172.24.4.224/28        |
| 6aff6826-4278-4a35-b74d-b0ca0cbba340 | database-servers |
|
+--------------------------------------+------------------+-------
-----------------------------------------------------+
```

3. 使用从以上步骤中获得的 *QA* **tenant-id**（**25837c567ed5458fbb441d39862e1399**）生成网络命令。另外，还需要包括 IPv6 子网的目标网络名称。在这个例子中，使用 *database-servers* 网络：

```
# neutron subnet-create --ip-version 6 --ipv6_address_mode=dhcpv6-
stateful --tenant-id 25837c567ed5458fbb441d39862e1399 database-servers
fdf8:f53b:82e4::53/125

Created a new subnet:
+-------------------+-------------------------------------------------
----------------+
| Field             | Value
|
+-------------------+-------------------------------------------------
----------------+
| allocation_pools  | {"start": "fdf8:f53b:82e4::52", "end":
"fdf8:f53b:82e4::56"} |
| cidr              | fdf8:f53b:82e4::53/125
|
| dns_nameservers   |
|
| enable_dhcp       | True
|
| gateway_ip        | fdf8:f53b:82e4::51
|
| host_routes       |
|
| id                | cdfc3398-997b-46eb-9db1-ebbd88f7de05
|
| ip_version        | 6
|
| ipv6_address_mode | dhcpv6-stateful
```

```
|
| ipv6_ra_mode      |
|
| name              |
|
| network_id        | 6aff6826-4278-4a35-b74d-b0ca0cbba340
|
| tenant_id         | 25837c567ed5458fbb441d39862e1399
|
+-------------------+-------------------------------------------
----------------+
```

4. 查看网络列表来验证这个配置。请注意，*database-servers* 项中的内容反映了新创建的 IPv6 子网：

```
# neutron net-list
+--------------------------------------+------------------+--------
-------------------------------------------------+
| id                                   | name             | subnets
|
+--------------------------------------+------------------+--------
-------------------------------------------------+
| 6aff6826-4278-4a35-b74d-b0ca0cbba340 | database-servers | cdfc3398-
997b-46eb-9db1-ebbd88f7de05 fdf8:f53b:82e4::50/125 |
| 8357062a-0dc2-4146-8a7f-d2575165e363 | private          | c17f74c4-
db41-4538-af40-48670069af70 10.0.0.0/24            |
| 31d61f7d-287e-4ada-ac29-ed7017a54542 | public           | 303ced03-
6019-4e79-a21c-1942a460b920 172.24.4.224/28        |
+--------------------------------------+------------------+--------
-------------------------------------------------+
```

作为这个配置的结果，*QA* 租户创建的实例在加入到 *database-servers* 子网中时可以接收一个 DHCP IPv6 地址：

```
# nova list
+--------------------------------------+------------+--------+-----
-------+-------------+--------------------------------------+
| ID                                   | Name       | Status | Task
State | Power State | Networks                             |
+--------------------------------------+------------+--------+-----
-------+-------------+--------------------------------------+
| fad04b7a-75b5-4f96-aed9-b40654b56e03 | corp-vm-01 | ACTIVE | -
| Running     | database-servers=fdf8:f53b:82e4::52 |
+--------------------------------------+------------+--------+-----
-------+-------------+--------------------------------------+
```

第 16 章 管理租户配额

本章介绍了为 OpenStack Networking 组件管理租户/项目配额的方法。

OpenStack Networking（neutron）支持使用配额来限制租户/项目创建的资源数量。例如，您可以通过修改 **neutron.conf** 文件中的 **quota_router** 值来限制一个租户可以创建的路由器的数量：

```
quota_router = 10
```

每个租户最多可以有 10 路由器。

不同的网络组件还会有其专用的配额设置：

16.1. L3 配额选项

L3 网络可用的配额选项：**quota_floatingip** - 每个租户允许使用的浮动 IP 数量。**quota_network** - 每个租户允许使用的网络数量。**quota_port** - 每个租户允许使用的端口数量。**quota_router** - 每个租户允许使用的路由器数量。**quota_subnet** - 每个租户允许使用的子网数量。**quota_vip** - 每个租户允许使用的 vip 数量。

16.2. 防火墙配额选项

用于防火墙管理的配额选项：**quota_firewall** - 每个租户所允许的防火墙数量。**quota_firewall_policy** - 每个租户所允许的防火墙策略数量。**quota_firewall_rule** - 每个租户所运行的防火墙规则数量。

16.3. 安全组配额选项

用于管理允许的安全组数量的配额选项：**quota_security_group** - 每个租户所允许的安全组数量。**quota_security_group_rule** - 每个租户所允许的安全组规则数量。

16.4. 管理配额选项

管理员可以考虑使用的配额选项：**default_quota** - 每个租户所允许的默认资源数量。**quota_health_monitor** - 每个租户所允许的健康监测程序的数量。健康监测程序不会消耗资源，但是，因为 OpenStack Networking 后端会把成员看做为资源消费者，所以健康监测程序有配额选项。**quota_member** - 每个租户所允许的池成员的数量。池成员不会消耗资源，但是，因为 OpenStack Networking 后端会把成员看做为资源消费者，所以它有配额选项。**quota_pool** - 每个租户所允许的池数量。

第 17 章 配置 FIREWALL-AS-A-SERVICE（FWAAS）

Firewall-as-a-Service (FWaaS) 插件为 OpenStack Networking (neutron) 添加了边界防火墙（perimeter firewall）管理功能。FWaaS 使用 iptables 在一个项目的所有虚拟路由上应用防火墙规则，并支持在一个项目中使用一个防火墙策略和逻辑防火墙实例。

FWaaS 在网络边界进行操作，它会对 OpenStack Networking (neutron) 的路由进行过滤。这一点和安全组有所不同，安全组在实例一级进行操作。

注意

FWaaS 当前还是一个技术预览，我们不推荐您在未经测试的情况下使用它。

以下示图展示了 *VM2* 实例的出站和入站的网络数据：

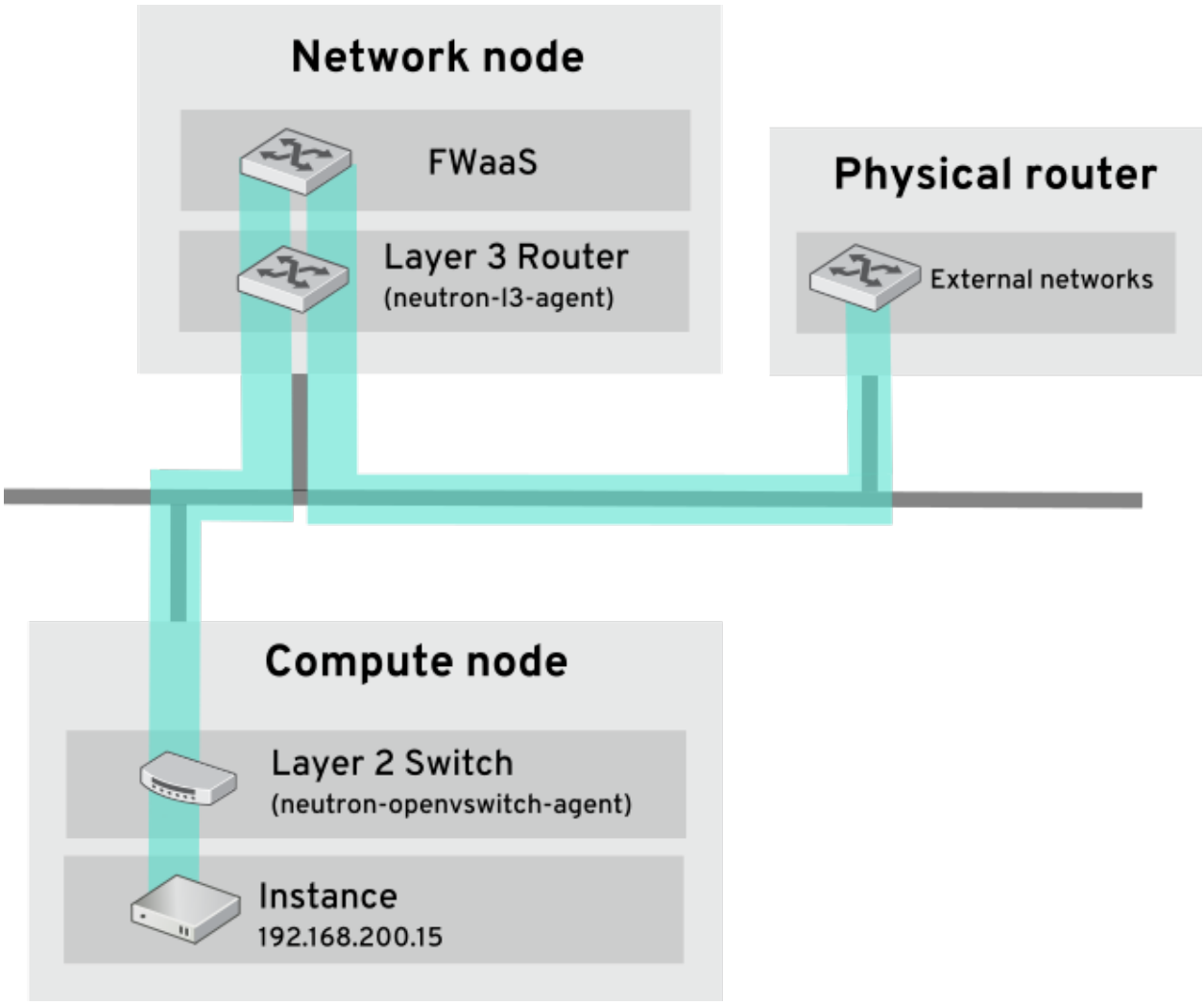


图 1. FWaaS 结构

17.1. 启用 FWaaS

1. 安装 FWaaS 软件包：

```
# yum install openstack-neutron-fwaas python-neutron-fwaas
```

2. 在 **neutron.conf** 文件中启用 FWaaS 插件：

```
service_plugins = neutron.services.firewall.fwaas_plugin.FirewallPlugin
```

3. 在 **fwaas_driver.ini** 文件中配置 FWaaS：

```
[fwaas]
driver =
neutron.services.firewall.drivers.linux.iptables_fwaas.IptablesFwaasDri
ver
enabled = True

[service_providers]
service_provider =
LOADBALANCER:Haproxy:neutron_lbaas.services.loadbalancer.drivers.haprox
y.plugin_driver.HaproxyOnHostPluginDriver:default
```

4. FWaaS 管理选项包括在 OpenStack 的 dashboard 中。在 **local_settings.py** 文件中（通常位于 Controller 节点上）启用这个选项：

```
/usr/share/openstack-
dashboard/openstack_dashboard/local/local_settings.py
'enable_firewall' = True
```

5. 重启 **neutron-server** 服务以使所做的改变生效：

```
# systemctl restart neutron-server
```

17.2. 配置 FWaaS

先创建防火墙规则以及包括它们的防火墙策略，然后再创建防火墙来应用这个防火墙策略：

1. 创建一个防火墙规则：

```
$ neutron firewall-rule-create --protocol <tcp|udp|icmp|any> --
destination-port <port-range> --action <allow|deny>
```

CLI 需要一个协议值，如果防火墙规则与协议无关，则可以使用*任何*值。

2. 创建一个防火墙策略：

```
$ neutron firewall-policy-create --firewall-rules "<firewall-rule IDs
or names separated by space>" myfirewallpolicy
```

以上指定的规则顺序非常重要。您可以先创建一个空的防火墙规则，以后再添加规则（使用 update 操作来添加多个规则，或使用 insert-rule 操作来添加一个规则）。

请注意： FWaaS 总会在每个策略的最后添加一个默认禁用所有网络数据的规则。因此，没有任何规则的防火墙策略会在默认情况下阻止所有网络流量。

17.3. 创建一个防火墙

```
$ neutron firewall-create <firewall-policy-uuid>
```

在 OpenStack Networking 路由器被创建、接口被添加前，防火墙会处于 PENDING_CREATE 状态。

第 18 章 配置 ALLOWED-ADDRESS-PAIRS

allowed-address-pairs 允许您指定 mac_address/ip_address（CIDR）对，使它们可以在不考虑子网的情况下通过一个端口。这会启用对一些协议的使用，如 VRRP，它会在两个实例间浮动一个 IP 地址，从而实现快速故障切换功能。

注意

当前，只有以下插件支持 allowed-address-pairs 扩展：ML2、Open vSwitch 和 VMware NSX。

18.1. 基本的 allowed-address-pairs 操作

创建一个带有 allowed-address-pairs 的端口：

```
# neutron port-create net1 --allowed-address-pairs type=dict list=true
mac_address=<mac_address>,ip_address=<ip_cidr>
```

18.2. 添加 allowed-address-pairs

```
# neutron port-update <port-uuid> --allowed-address-pairs type=dict
list=true mac_address=<mac_address>,ip_address=<ip_cidr>
```

注意

OpenStack Networking 不允许设置和一个端口的 **mac_address** 和 **ip_address** 相同的 allowed-address-pair。这是因为，带有这个 **mac_address** 和 **ip_address** 的网络流量已被允许通过这个端口，所以这样的配置不会起任何作用。

第 19 章 配置第 3 层高可用性

本章解释了第 3 层高可用性功能在一个 OpenStack Networking 环境中所起的作用，并且包括了使用这个功能保护网络中的虚拟路由器的步骤。

19.1. 没有 HA 功能的 OpenStack Networking

一个不带有任何高可用性功能的 OpenStack Networking 环境会在物理节点出现问题时变得非常脆弱。

在一个典型的环境中，租户会创建虚拟路由器，而它们会被调度到物理 L3 agent 节点上运行。当一个 L3 agent 节点出现问题时，依赖它的虚拟机将失去和外部网络的连接；浮动 IP 地址也将无效。

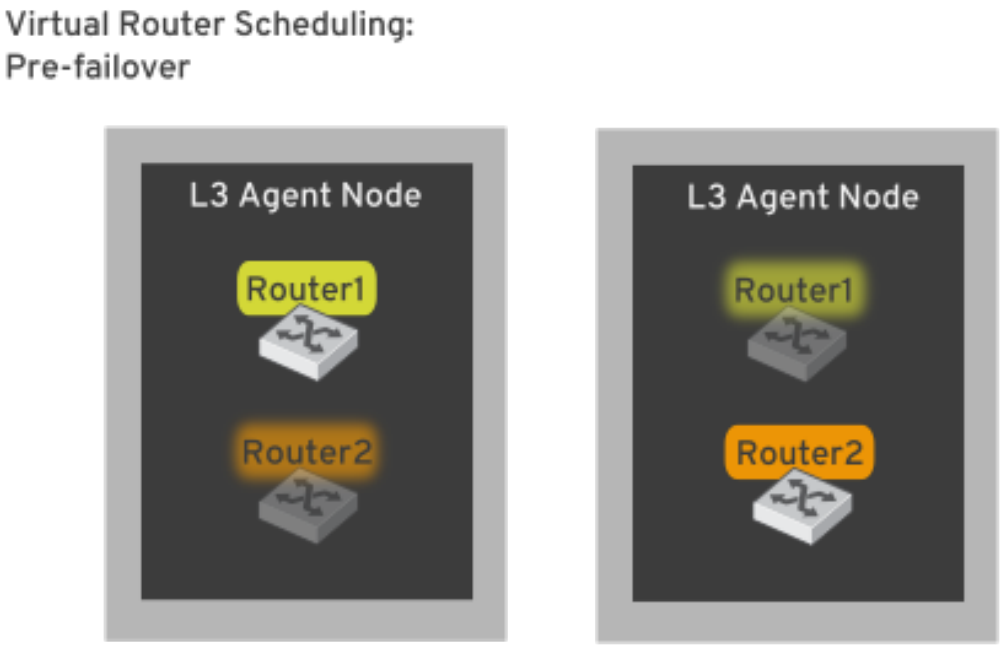
19.2. 第 3 层高可用性概述

“主动/被动”式高可用性配置使用工业标准的 VRRP（在 RFC 3768 中定义）保护租户的路由器和浮动 IP 地址。虚拟路由器会在多个 OpenStack Networking 节点上随机调度，其中的一个是 *active（活跃的）*，其它作为 *standby（备份）*的角色。

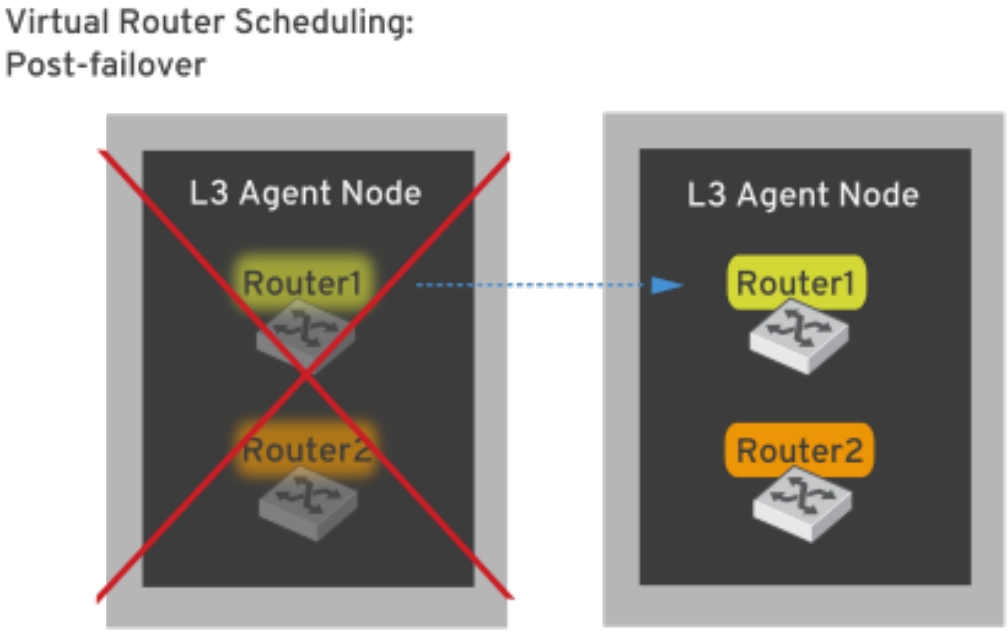
注意

为了成功部署第 3 层高可用性功能，需要有冗余的 OpenStack Networking 节点，这些节点必须具备相似的配置（浮动 IP 范围、到外部网络的访问）。

在下图中，活跃的 Router1 和 Router2 分别运行于独立的物理 L3 agent 节点上。第 3 层高可用性已在相关的节点上调度了备份虚拟路由器，当物理节点出现问题时，会继续提供相关的服务：



当 L3 agent 节点出现问题时，第 3 层高可用性会把受影响的虚拟路由器和浮动 IP 地址重新调度到一个可以工作的节点上：



在发生故障转移操作时，使用浮动 IP 的 TCP 会话不会受到影响，它们会被迁移到一个新的 L3 节点上。只有 SNAT 的网络流量会受到故障转移操作的影响。

L3 agent 本身也会通过 active/active HA 的模式被保护。

19.2.1. 故障转移的条件

Layer 3 高可用性功能会在出现以下情况时自动重新调度被保护的资源：

- 因为硬件故障造成 L3 agent 关机或无电源供应。
- L3 agent 节点从物理网络中被隔离并无法被连接。

注意

手工停止 L3 agent 服务不会导致故障转移操作的发生。

19.3. 租户的考虑因素

第 3 层高可用性配置在后台发生，对租户是不可见的。租户可以继续象往常一样创建和管理虚拟路由器，但在设计您的第 3 层高可用性功能时需要考虑以下问题：

- 第 3 层高可用性功能支持每个租户最多有 255 个虚拟路由。
- 内部的 VRRP 信息会在为每个项目自动创建的一个独立的内部网络中传输。这个过程对用户是透明的。

19.4. 后台的变化

现在，Neutron API 已被更新，管理员可以在创建路由器时设置 **--ha=True/False** 标识来覆盖 neutron.conf 文件中的默认 l3_ha 配置。下节介绍了进行这个配置的步骤。

19.4.1. 对 neutron-server 的改变

- 无论 OpenStack Networking 使用什么调度程序（random 或 leastrouter），第 3 层高可用性功能都随机分配活跃的角色。
- 修改数据库的 schema 来支持为虚拟路由器分配 VIP。
- 根据以上的介绍创建一个传输网络来处理第 3 层高可用性网络流量。

19.4.2. 对 L3 agent 的改变

- 添加一个新的 keepalived manager 来提供负载均衡和 HA 的功能。
- 把 IP 地址转换为 VIP。

19.5. 配置步骤

以下步骤在 OpenStack Networking 和 L3 agent 节点上启用第 3 层高可用性功能。

19.6. 配置 OpenStack Networking 节点

1. 在 neutron.conf 文件中配置第 3 层高可用性功能。启用 L3 HA，并定义保护每个虚拟路由器所需的 L3 agent 节点的数量：

```
l3_ha = True
max_l3_agents_per_router = 2
min_l3_agents_per_router = 2
```

其中：

- **l3_ha** - 如果它被设置为 True，以后创建的所有虚拟路由器在默认情况下都是 HA 路由器，而不是一般的路由器。管理员可以通过以下方法覆盖每个路由器中的这个设置：

```
# neutron router-create --ha=<True | False> routerName
```

- **max_l3_agents_per_router** - 把它的值设置为您的部署中的最小网络节点数量值和总共的网络节点数量值之间的值。例如，您共有 4 个 OpenStack Networking 节点，并把这个值设为 2，则每个 HA 虚拟路由器只有 2 个 L3 agent 来保护它：一个是活跃的，一个用于备份。另外，每次当一个新的 L3 agent 节点被部署时，额外的备份虚拟路由器都会被调度，直到达到了 max_l3_agents_per_router 指定的值。这样，您就可以通过添加新的 L3 agent 来扩展备份路由器的数量。
- **min_l3_agents_per_router** - 这个设置确保了 HA 规则在生效。它会在虚拟路由器创建的过程中被设置，来确保有足够的 L3 Agent 节点来实现高可用性的功能。例如，您有 2 个网络节点，一个节点因为某些原因变为不可用，则此时将无法创建新的路由器，因为在创建 HA 路由器时需要满足最少的活跃 L3 agent 数量要求。

2. 重启 **neutron-server** 服务以使所做的修改生效：

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

19.7. 检查您的配置

在虚拟路由器的命名空间内运行 ip address 命令，一个 HA 设备应该包括在返回结果中（带有 *ha-* 前缀）。

```
# ip netns exec qrouter-b30064f9-414e-4c98-ab42-646197c74020 ip address
<snip>
2794: ha-45249562-ec: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state DOWN group default
link/ether 12:34:56:78:2b:5d brd ff:ff:ff:ff:ff:ff
inet 169.254.0.2/24 brd 169.254.0.255 scope global ha-54b92d86-4f
```

现在，第 3 层高可用性功能已被启用，在某个节点出现问题时，虚拟路由器和浮动 IP 地址会受到保护。

第 20 章 SR-IOV 对虚拟网络的支持

从 RHEL OpenStack Platform 6 开始，扩展了SR-IOV（single root I/O virtualization - 单引导 I/O 虚拟化）对虚拟机网络的支持。这意味着，OpenStack 可以不再需要虚拟网桥，而是把物理 NIC 的功能直接扩展到实例中。另外，通过对 IEEE 802.1br 的支持，虚拟 NIC 可以与物理交换机进行集成，并由它进行管理。

20.1. 在 RHEL OpenStack Platform 中配置 SR-IOV

本章包含配置 SR-IOV 以将物理 NIC 传递到虚拟实例的过程。这些步骤假设系统包括了一个 Controller 节点、一个 OpenStack Networking（neutron） 节点和多个 Compute（nova）节点。

请注意：当已存在了适当的 L2 配置（例如，flat 或 VLAN），使用 SR-IOV 端口的实例与使用普通端口（如连接到 Open vSwitch 网桥）的实例间就可以彼此进行通讯。当前，在相同 Compute 节点上的使用 SR-IOV 端口的实例与使用普通 vSwitch 端口的实例间的通讯有一个限制：如果它们在网卡上共享相同的 PF（Physical Function），则无法进行通讯。

20.2. 在 Compute 节点上创建虚拟功能

在带有支持硬件的所有 Compute 节点上执行这些步骤。

注意： 如需了解更多所支持硬件的信息，请参阅相关文档。

这个步骤会配置一个系统来传递一个 *Intel 82576* 网络设备。同时，虚拟功能（Virtual Functions）也会被创建，实例可以使用它们来进行 SR-IOV 到设备的访问。

1. 确认 **Intel VT-d** 或 **AMD IOMMU** 已在系统的 BIOS 中启用。请参阅机器的 BIOS 配置菜单，或其它相关信息。

2. 确保在操作系统中启用 *Intel VT-d* 或 *AMD IOMMU*：

- 对于 *Intel VT-d* 系统，请参阅这里。
- 对于 *AMD IOMMU* 系统，请参阅这里。

3. 运行 *lspci* 命令确保网络设备可以被系统识别：

```
[root@compute ~]# lspci | grep 82576
```

网络设备包括在结果中：

```
03:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
03:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
```

4. 执行以下步骤来在 Compute 节点上激活 Virtual Functions：

4a.删除内核模块。在下一步中会对模块进行重新配置：

```
[root@compute ~]# modprobe -r igb
```

请注意：在第 4 步中，需要使用支持 SRIOV 的 NIC 所使用的模块，而不是使用其它 NIC 的 **`igb`**（如 **`ixgbe`** 或 **`mlx4_core`**）。运行 ethtool 命令来确认驱动。在这个例子中，em1 是我们需要使用的 PF（Physical Function）：

```
[root@compute ~]# ethtool -i em1 | grep ^driver
```

4b. 启动模块时 max_vfs 的值为 7（或不多于被支持的最多数量）。

```
[root@compute ~]# modprobe igb max_vfs=7
```

4c. 使 Virtual Functions 有持久性：

```
[root@compute ~]# echo "options igb max_vfs=7"
>>/etc/modprobe.d/igb.conf
```

请注意： 对于 Red Hat Enterprise Linux 7，为了使配置具有持久性，在进行完第 4 步后根据 rebuild the initial ramdisk image 中介绍的内容进行相关操作。

请注意：在 *4c.* 和 *4d.* 中进行的设置持久性如下：**modprobe** 命令在使用相同内核模块的所有 NIC 上启用 Virtual Function，并在系统重新引导后仍然可以保持这个设置。虽然可以只在特定 NIC 上启用 VF，但这可能会造成一些问题。例如，使用以下命令来在 **enp4s0f1** 接口上启用 VF：

```
# echo 7 > /sys/class/net/enp4s0f1/device/sriov_numvfs
```

当系统重启后，这个设备将不会被保留。解决这个问题的一个方法是，把这个设置添加到 *rc.local* 中，但这个方法仍然有以下限制：

```
# chmod +x /etc/rc.d/rc.local
# echo "echo 7 > /sys/class/net/enp4s0f1/device/sriov_numvfs" >>
/etc/rc.local
```

请注意：因为额外的 systemd，Red Hat Enterprise Linux 将会并行启动服务，而不是依次启动它们。这意味着，*rc.local* 在引导过程中被执行的位置是不固定的。因此，一些不可预见的情况可能会出现，我们不推荐使用这个方法。

4d. 把 *intel_iommu=pt* 和 *igb.max_vfs=7* 参数添加到内核命令行来在内核中激活 Intel VT-d。如果您希望一直使用这个方式引导内核，可以修改当前的设置；或者，您可以创建一个带有这些参数的自定义菜单，这样，您的系统就可以在默认情况下以这种方式启动，并且可以在需要时，不使用这些参数启动内核。

- 要修改当前内核命令行参数，运行以下命令：

```
[root@compute ~]# grubby --update-kernel=ALL --args="intel_iommu=pt
igb.max_vfs=7"
```

如需了解更多使用 *grubby* 的信息，请参阅系统管理指南中的 Configuring GRUB 2 Using the grubby Tool。

请注意：如果使用 *Dell Power Edge R630* 节点，则需要使用 **intel_iommu=on** 而不是 **intel_iommu=pt**。您可以通过 *grubby* 启用它：

```
# grubby --update-kernel=ALL --args="intel_iommu=on"
```

- 创建一个自定义菜单项：

i. 在 *grub* 中找到默认的项：

```
[root@compute ~]# grub2-editenv list
saved_entry=Red Hat Enterprise Linux Server (3.10.0-123.9.2.el7.x86_64)
7.0 (Maipo)
```

ii. a. 把所需的 *menuentry* 项的内容（以 "menuentry" 开始，以 "}" 结束。它在开始部分包括了前一步命令显示的 saved_entry 的值）从 */boot/grub2/grub.cfg* 复制到 */etc/grub.d/40_custom*。**b.** 在 *menuentry* 后修改标题 **c.** 在以 linux16 开始的行的最后添加 *intel_iommu=on*。

例如：

```
menuentry 'Red Hat Enterprise Linux Server, with Linux 3.10.0-
123.el7.x86_64 - SRIOV' --class red --class gnu-linux --class gnu --
class os --unrestricted $menuentry_id_option 'gnulinux-3.10.0-
123.el7.x86_64-advanced-4718717c-73ad-4f5f-800f-f415adfccd01' {
    load_video
    set gfxpayload=keep
    insmod gzio
    insmod part_msdos
    insmod ext2
    set root='hd0,msdos2'
    if [ x$feature_platform_search_hint = xy ]; then
      search --no-floppy --fs-uuid --set=root --hint-bios=hd0,msdos2 -
-hint-efi=hd0,msdos2 --hint-baremetal=ahci0,msdos2 --hint='hd0,msdos2'
5edd1db4-1ebc-465c-8212-552a9c97456e
    else
      search --no-floppy --fs-uuid --set=root 5edd1db4-1ebc-465c-8212-
552a9c97456e
    fi
    linux16 /vmlinuz-3.10.0-123.el7.x86_64 root=UUID=4718717c-73ad-
4f5f-800f-f415adfccd01 ro vconsole.font=latarcyrheb-sun16 biosdevname=0
crashkernel=auto  vconsole.keymap=us nofb console=ttyS0,115200
LANG=en_US.UTF-8 intel_iommu=pt igb.max_vfs=7
    initrd16 /initramfs-3.10.0-123.el7.x86_64.img
}
```

iii. 更新 *grub.cfg* 以使配置文件中的修改生效：

```
[root@compute ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

iv. 修改默认的项：

```
[root@compute ~]# grub2-set-default 'Red Hat Enterprise Linux Server,
with Linux 3.10.0-123.el7.x86_64 - SRIOV'
```

v. 创建 *dist.conf* 配置文件。

请注意：在执行这个步骤前，请重新查看描述 allow_unsafe_interrupts 的段落：*检查 allow_unsafe_interrupts 的设置*。

```
[root@compute ~]# echo "options vfio_iommu_type1
allow_unsafe_interrupts=1" > /etc/modprobe.d/dist.conf
```

5. 重启服务器以使新内核参数生效：

```
[root@compute ~]# systemctl reboot
```

6. 查看 Compute 节点上的 SR-IOV 内核模块。运行 *lsmod* 以确认模块已被加载：

```
[root@compute ~]#  lsmod |grep igb
```

这个经过过滤的结果应该包括所需的模块：

```
igb    87592  0
dca    6708    1 igb
```

7. 记录下您的网卡的 PCI 厂商 ID（格式是 vendor_id:product_id）。运行带有 *-nn* 选项的 *lspci* 命令可以得到这个值。例如：

```
[root@compute ~]# lspci  -nn | grep -i 82576
05:00.0 Ethernet controller [0200]: Intel Corporation 82576 Gigabit
Network Connection [8086:10c9] (rev 01)
05:00.1 Ethernet controller [0200]: Intel Corporation 82576 Gigabit
Network Connection [8086:10c9] (rev 01)
05:10.0 Ethernet controller [0200]: Intel Corporation 82576 Virtual
Function [8086:10ca] (rev 01)
```

请注意：根据您的网卡硬件的具体情况，这个参数可能会不同。

8. 使用 *lspci* 命令查看新创建的 VF：

```
[root@compute ~]# lspci | grep 82576
```

这个结果会包括设备和 Virtual Function：

```
0b:00.0 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection (rev 01)
0b:00.1 Ethernet controller: Intel Corporation 82576 Gigabit Network
Connection(rev 01)
0b:10.0 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:10.1 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:10.2 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:10.3 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:10.4 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:10.5 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:10.6 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:10.7 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:11.0 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:11.1 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:11.2 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:11.3 Ethernet controller: Intel Corporation 82576 Virtual Function
```

```
(rev 01)
0b:11.4 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
0b:11.5 Ethernet controller: Intel Corporation 82576 Virtual Function
(rev 01)
```

20.3. 在 Network 节点上配置 SR-IOV

OpenStack Networking（neutron）使用 ML2 机制驱动来支持 SR-IOV。在 Network 网络上执行这些步骤来配置 SR-IOV 驱动。

1. 在 */etc/neutron/plugins/ml2/openvswitch_agent.ini* 文件中启用 *sriovnicswitch*。例如，这个配置启用了 SR-IOV 机制驱动和 Open vSwitch。

请注意：*sriovnicswitch* 不支持 *DHCP Agent* 的当前接口驱动，因此在使用 *sriovnicswitch* 时需要 *openvswitch*（或其它支持 VLAN 的机制驱动）。

```
[ml2]
tenant_network_types = vlan
type_drivers = vlan
mechanism_drivers = openvswitch, sriovnicswitch
[ml2_type_vlan]
network_vlan_ranges = physnet1:15:20
```

- network_vlan_ranges - 在这个例子中，*physnet1* 被作为网络标签使用，VLAN 的范围是 *15-20*。

请注意：*sriovnicswitch* 当前只支持 *flat* 和 *vlan* 驱动。

2. 可选 - 如果您需要 VF 连接状态和 admin 状态管理，而且您的厂商支持这些功能，请在 */etc/neutron/plugins/ml2/sriov_agent.ini* 文件中启用这个选项：

```
[root@network ~]# openstack-config --set
/etc/neutron/plugins/ml2/sriov_agent.ini ml2_sriov agent_required True
```

3. 可选 - 支持的 vendor_id/product_id 组合是 *15b3:1004, 8086:10ca*。如果网卡厂商的产品 ID 和这个不同，则需要指定它们。例如：

```
[ml2_sriov]
supported_pci_vendor_devs = 15b3:1004,8086:10ca
```

4. 配置 *neutron-server.service* 来使用 *ml2_conf_sriov.ini* 文件。例如：

```
[root@network ~]# vi /usr/lib/systemd/system/neutron-server.service

[Service]
Type=notify
User=neutron
ExecStart=/usr/bin/neutron-server --config-file
/usr/share/neutron/neutron-dist.conf --config-file
/etc/neutron/neutron.conf --config-file
/etc/neutron/plugins/ml2/openvswitch_agent.ini --config-file
/etc/neutron/plugins/ml2/sriov_agent.ini  --log-file
/var/log/neutron/server.log
```

5. 重启 *neutron-server* 服务来使配置生效：

```
[root@network ~]# systemctl restart neutron-server.service
```

20.4. 在 Controller 节点上配置 SR-IOV

1. 为了正确调度 SR-IOV 设备，Compute 调度程序需要使用带有 *PciPassthroughFilter* 过滤的 *FilterScheduler*。在 Controller 节点上的 *nova.conf* 文件中应用这个配置。例如：

```
scheduler_available_filters=nova.scheduler.filters.all_filters
scheduler_default_filters=RetryFilter,AvailabilityZoneFilter,RamFilter,
ComputeFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter,CoreFilte
r, PciPassthroughFilter
```

2. 重启 Compute 调度器以使修改生效：

```
[root@compute ~]# systemctl restart openstack-nova-scheduler.service
```

20.5. 在 Compute 节点上配置 SR-IOV

在所有 Compute 节点上，为每个物理网络关联有效的 VF：

1. 在 *nova.conf* 文件中定义这些项。在这个例子中，把设备名为 *enp5s0f1* 的 VF 网络与网络标签为 *physnet1* 的物理网络相关联。其中由 *physical_network* 所指定的网络标签是以前在 *network_vlan_ranges* 中配置的。

```
pci_passthrough_whitelist={"devname": "enp5s0f1",
"physical_network":"physnet1"}
```

下面的配置会把厂商 ID 为 *8086* 的 PF 网络与网络标签为 *physnet1*（由 *physical_network* 指定）的物理网络进行关联：~ pci_passthrough_whitelist = \{"vendor_id": "8086","product_id": "10ac", "physical_network":"physnet1"} ~

PCI passthrough whitelist 项的格式如下：

```
["device_id": "<id>",] ["product_id": "<id>",]
["address": "[[[[<domain>]:]<bus>]:][<slot>][.[<function>]]" |
"devname": "Ethernet Interface Name",]
"physical_network":"Network label string"
```

- **id** - *id* 的值可以是通配符（*），或一个有效的 device/product id。您可以使用 *lspci* 列出有效的设备名。
- **address** - *address* 的格式与使用带有 *-s* 选项的 *lspci* 命令输出中的格式相同。
- **devname** - *devname* 是一个有效的 PCI 设备名称。您可以使用 *ifconfig -a* 列出全部有效的名称。这个项需要和与一个 vNIC 相关联的 PF 或 VF 值相对应。如果设备由代表一个 SR-IOV PF 的地址或 devname 来定义，则 PF 下的所有 VF 必须和这个项匹配。另外，一个项可以关联0 个或多个标签。
- **physical_network** - 在使用 SR-IOV 网络时，"physical_network" 被用来指定设备附加到的物理网络。

您可以为每个主机配置多个 whitelist 项。*device_id*、*product_id*、以及 *address* 或 *devname* 的值将会和查询 libvirt 所返回的 PCI 设备进行匹配。

2. 重启 *nova-compute* 服务以使所做的改变生效：

```
[root@compute ~]# systemctl restart openstack-nova-compute
```

20.6. 启用 OpenStack Networking SR-IOV agent

可选的 OpenStack Networking SR-IOV agent 启用了对 admin_state 端口的管理。这个 agent 与网络适配器进行集成，允许管理员打开/关闭 VF 的管理状态。

另外，如果 *agent_required=True* 已在 OpenStack Networking（neutron）服务器上配置，则需要在每个 Compute 节点上都运行 OpenStack Networking SR-IOV Agent。

请注意：当前，不是所有网卡厂商都支持使用这个 agent 管理端口状态。

1. 安装 *sriov-nic-agent* 软件包以便进行以下步骤：

```
[root@compute ~]# yum install openstack-neutron-sriov-nic-agent
```

2. 在 */etc/neutron/plugins/ml2/openvswitch_agent.ini* 文件中启用 *NoopFirewallDriver*：

```
[root@compute ~]# openstack-config --set
/etc/neutron/plugins/ml2/openvswitch_agent.ini securitygroup
firewall_driver neutron.agent.firewall.NoopFirewallDriver
```

3. 在 */etc/neutron/plugins/ml2/sriov_agent.ini* 文件中添加映射信息。在这个例子中，*physnet1* 是物理网络、*enp4s0f1* 是 physical function。把 *exclude_devices* 设为空来允许 agent 管理所有相关的 VF。

```
[sriov_nic]
physical_device_mappings = physnet1:enp4s0f1
exclude_devices =
```

4. *可选操作* - 为了在 agent 配置中排除特定的 VF，在 *sriov_nic* 项中列出要排除的 VF。例如：

```
exclude_devices = eth1:0000:07:00.2; 0000:07:00.3, eth2:0000:05:00.1;
0000:05:00.2
```

5. 配置 *neutron-sriov-nic-agent.service* 来使用 *ml2_conf_sriov.ini* 文件。例如：

```
[root@compute ~]# vi /usr/lib/systemd/system/neutron-sriov-nic-
agent.service

[Service]
Type=simple
User=neutron
ExecStart=/usr/bin/neutron-sriov-nic-agent --config-file
/usr/share/neutron/neutron-dist.conf --config-file
/etc/neutron/neutron.conf --log-file /var/log/neutron/sriov-nic-
agent.log --config-file /etc/neutron/plugins/ml2/sriov_agent.ini
```

6. 启动 OpenStack Networking SR-IOV agent：

```
[root@compute ~]# systemctl enable neutron-sriov-nic-agent.service
[root@compute ~]# systemctl start neutron-sriov-nic-agent.service
```

20.7. 配置一个实例来使用 SR-IOV 端口

在这个例子中，SR-IOV 端口被添加到 *web* 网络。

1. 获得有效的网络列表

```
[root@network ~]# neutron net-list
+--------------------------------------+---------+----------------
----------------------------------+
| id                                   | name    | subnets
|
+--------------------------------------+---------+----------------
----------------------------------+
| 3c97eb09-957d-4ed7-b80e-6f052082b0f9 | corp    | 78328449-796b-49cc-
96a8-1daba7a910be 172.24.4.224/28 |
| 721d555e-c2e8-4988-a66f-f7cbe493afdb | web     | 140e936e-0081-4412-
a5ef-d05bacf3d1d7 10.0.0.0/24     |
+--------------------------------------+---------+----------------
----------------------------------+
```

这个结果列出了已在 OpenStack Networking 中创建的网络，以及子网的详情。

2. 在 *web* 网络中创建端口

```
[root@network ~]# neutron port-create web --name sr-iov --binding:vnic-
type direct
Created a new port:
+-----------------------+----------------------------------------
--------------------------------------+
| Field                 | Value
|
+-----------------------+----------------------------------------
--------------------------------------+
| admin_state_up        | True
|
| allowed_address_pairs |
|
| binding:host_id       |
|
| binding:profile       | {}
|
| binding:vif_details   | {}
|
| binding:vif_type      | unbound
|
| binding:vnic_type     | normal
|
| device_id             |
|
| device_owner          |
|
| fixed_ips             | {"subnet_id": "140e936e-0081-4412-a5ef-
d05bacf3d1d7", "ip_address": "10.0.0.2"} |
| id                    | a2122b4d-c9a9-4a40-9b67-ca514ea10a1b
|
| mac_address           | fa:16:3e:b1:53:b3
```

```
|
| name                  | sr-iov
|
| network_id            | 721d555e-c2e8-4988-a66f-f7cbe493afdb
|
| security_groups       | 3f06b19d-ec28-427b-8ec7-db2699c63e3d
|
| status                | DOWN
|
| tenant_id             | 7981849293f24ed48ed19f3f30e69690
|
+-----------------------+------------------------------------------
--------------------------------------+
```

3. 使用新端口创建一个实例。创建一个名为 *webserver01* 的新实例，配置它来使用新端口（端口 ID 是前一个输出中的 *id* 项中的值）：

请注意：您可以使用 *glance image-list* 命令来获得有效镜像的列表，以及它们的 UUID。

```
[root@compute ~]# nova boot --flavor m1.tiny --image 59a66200-45d2-
4b21-982b-d06bc26ff2d0  --nic port-id=a2122b4d-c9a9-4a40-9b67-
ca514ea10a1b  webserver01
```

新实例 *webserver01* 被创建，并配置为使用 SR-IOV 端口。

20.8. 检查 allow_unsafe_interrupts 设置

为了完全把带有分配设备的客户机与主机分离，需要平台对中断重映射功能的支持。如果不支持这个功能，主机可能会受到来自于恶意客户机上的中断注入攻击（interrupt injection attack）。而在一个客户端可以被完全信任的环境中，管理员可能会允许使用 *allow_unsafe_interrupts* 选项进行 PCI 设备的分配。您需要根据具体情况检查是否在主机上启用 *allow_unsafe_interrupts*。如果主机上的 IOMMU 支持中断重映射功能，则不需要启用这个选项。

1. 使用 *dmesg* 检查您的主机是否支持 IOMMU 中断重映射功能：

```
[root@compute ~]# dmesg |grep ecap
```

当 ecap *(0xf020ff → …1111)* 的第 3 位是 *1* 时，意味着 IOMMU 支持中断重映射。

2. 确认 IRQ 重映射是否已被启用：

```
[root@compute ~]# dmesg |grep "Enabled IRQ"
[    0.033413] Enabled IRQ remapping in x2apic mode
```

请注意：如需手工禁用 "IRQ remapping"，把 *intremap=off* 添加到 *grub.conf* 文件中。

3. 如果主机的 IOMMU 不支持中断重映射，您需要在 kvm 模块中启用 *allow_unsafe_assigned_interrupts=1*。

20.9. 额外需要考虑的因素

- 在选择 vNIC 类型时请注意，当前还不支持 *vnic_type=macvtap*。
- 不支持带有附加了 SR-IOV 实例的 VM 迁移。
- 当前，安全组不能在启用了 SR-IOV 的端口中使用。