



# Red Hat OpenStack Platform 16.2

## 网络指南

Red Hat OpenStack Platform 网络的高级指南



# Red Hat OpenStack Platform 16.2 网络指南

---

Red Hat OpenStack Platform 网络的高级指南

OpenStack Team  
rhos-docs@redhat.com

## 法律通告

Copyright © 2024 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, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

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

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

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

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

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

The OpenStack<sup>®</sup> 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

用于常见 OpenStack 网络任务的手册。

# 目录

前言 .....	6
使开源包含更多 .....	7
对红帽文档提供反馈 .....	8
<b>第 1 章 OPENSTACK 网络简介 .....</b>	<b>9</b>
1.1. 管理 RHOSP 网络	9
1.2. 网络服务组件	10
1.3. 模块层 2 (ML2)网络	11
1.4. ML2 网络类型	11
1.5. 模块第 2 层(ML2)机制驱动程序	12
1.6. OPEN VSWITCH	13
1.7. 打开虚拟网络(OVN)	13
1.8. 模块层 2 (ML2)类型和机制驱动程序兼容性	13
1.9. RHOSP 网络服务的扩展驱动程序	14
<b>第 2 章 使用 ML2/OVN .....</b>	<b>15</b>
2.1. RHOSP OVN 架构中的组件列表	15
2.2. ML2/OVN 数据库	16
2.3. COMPUTE 节点上的 OVN-CONTROLLER 服务	17
2.4. COMPUTE 节点上的 OVN 元数据代理	17
2.5. OVN 可组合服务	17
2.6. OVN 第 3 层高可用性	18
2.7. OVN 和 OVS 机制驱动程序中的功能支持	19
2.8. 使用 ML2/OVN 对非安全端口的限制	20
2.9. ML2/OVS 到 ML2/OVN 原位迁移：验证和禁止的情况	20
2.10. 在新的 RHOSP 16.2 部署中使用 ML2/OVS 而不是默认的 ML2/OVN	21
2.11. 升级后保留 ML2/OVS，而不是默认的 ML2/OVN	22
2.12. 使用 ML2/OVN 部署自定义角色	22
2.13. 带有 ML2/OVN 和原生 OVN DHCP 的 SR-IOV	26
<b>第 3 章 管理项目网络 .....</b>	<b>27</b>
3.1. VLAN 计划	27
3.2. 网络流量类型	27
3.3. IP 地址消耗	28
3.4. 虚拟网络	29
3.5. 添加网络路由	29
3.6. 网络计划示例	29
3.7. 创建网络	30
3.8. 使用子网	32
3.9. 创建子网	32
3.10. 添加路由器	33
3.11. 清除所有资源并删除项目	34
3.12. 删除路由器	34
3.13. 删除子网	34
3.14. 删除网络	35
<b>第 4 章 将虚拟机实例连接到物理网络 .....</b>	<b>36</b>
4.1. OPENSTACK 网络拓扑概述	36
4.2. OPENSTACK 网络服务的放置	36
4.3. 配置扁平提供商网络	36
4.4. 扁平提供商网络数据包流的工作方式？	39

4.5. 对扁平提供商网络上的实例物理网络连接进行故障排除	42
4.6. 配置 VLAN 提供商网络	44
4.7. VLAN 提供商网络数据包流的工作方式？	47
4.8. VLAN 提供商网络上的实例物理网络连接故障排除	50
4.9. 在 ML2/OVS 部署中为提供商网络启用多播侦听	52
4.10. 在 ML2/OVN 部署中启用多播	54
4.11. 启用计算元数据访问	55
4.12. 浮动 IP 地址	56
<b>第 5 章 管理浮动 IP 地址</b>	<b>57</b>
5.1. 创建浮动 IP 池	57
5.2. 分配特定的浮动 IP	57
5.3. 创建高级网络	59
5.4. 分配随机浮动 IP	59
5.5. 创建多个浮动 IP 池	61
5.6. 配置浮动 IP 端口转发	62
5.7. 为浮动 IP 创建端口转发	63
5.8. 桥接物理网络	65
5.9. 添加接口	66
5.10. 删除接口	66
<b>第 6 章 网络故障排除</b>	<b>67</b>
6.1. 基本 PING 测试	67
6.2. 查看当前端口状态	69
6.3. 与 VLAN 提供商网络的连接故障排除	70
6.4. 查看 VLAN 配置和日志文件	70
6.5. 在 ML2/OVN 命名空间中执行基本 ICMP 测试	71
6.6. 从项目网络内进行故障排除(ML2/OVS)	72
6.7. 在命名空间中执行高级 ICMP 测试(ML2/OVS)	74
6.8. 为 OVN 故障排除命令创建别名	74
6.9. 监控 OVN 逻辑流	76
6.10. 监控 OPENFLOWS	79
6.11. 验证 ML2/OVN 部署	80
6.12. 为 ML2/OVN 设置日志记录模式	82
6.13. 修复无法在边缘站点上注册的 OVN 控制器	83
6.14. ML2/OVN 日志文件	84
<b>第 7 章 为 OPENSTACK 网络配置物理交换机</b>	<b>86</b>
7.1. 规划您的物理网络环境	86
7.2. 配置 CISCO CATALYST 交换机	86
7.3. 配置 CISCO NEXUS 交换机	92
7.4. 配置 CUMULUS LINUX 交换机	95
7.5. 配置 EXTREME EXOS 交换机	98
7.6. 配置 JUNIPER EX 系列交换机	101
<b>第 8 章 配置最大传输单元(MTU)设置</b>	<b>107</b>
8.1. MTU 概述	107
8.2. 在 DIRECTOR 中配置 MTU 设置	107
8.3. 查看生成的 MTU 计算	108
<b>第 9 章 使用服务质量(QOS)策略来管理数据流量</b>	<b>109</b>
9.1. QOS 规则	109
9.2. 为 QOS 策略配置网络服务	110
9.3. 使用 QOS 策略控制最小带宽	115

9.4. 使用 QOS 策略限制网络流量	122
9.5. 使用 KIOSK 标记 QOS 策略来优先考虑网络流量	126
9.6. 使用网络服务 RBAC 将 QOS 策略应用到项目	128
<b>第 10 章 配置网桥映射</b>	<b>129</b>
10.1. 网桥映射概述	129
10.2. 流量流	129
10.3. 配置网桥映射	129
10.4. 为 OVS 维护网桥映射	131
<b>第 11 章 VLAN 感知实例</b>	<b>134</b>
11.1. VLAN 中继和 VLAN 透明网络	134
11.2. 在 ML2/OVN 部署中启用 VLAN 透明性	134
11.3. 查看中继插件	136
11.4. 创建中继连接	136
11.5. 在中继中添加子端口	138
11.6. 配置实例以使用中继	139
11.7. 配置网络服务 RPC 超时	141
11.8. 了解中继状态	142
<b>第 12 章 配置 RBAC 策略</b>	<b>143</b>
12.1. RBAC 策略概述	143
12.2. 创建 RBAC 策略	143
12.3. 查看 RBAC 策略	144
12.4. 删除 RBAC 策略	144
12.5. 为外部网络授予 RBAC 策略访问权限	145
<b>第 13 章 配置分布式虚拟路由(DVR)</b>	<b>146</b>
13.1. 了解分布式虚拟路由(DVR)	146
13.2. DVR 概述	147
13.3. DVR 已知问题和注意事项	147
13.4. 支持的路由架构	148
13.5. 使用 ML2 OVS 部署 DVR	148
13.6. 将集中式路由器迁移到分布式路由	149
13.7. 部署禁用分布式虚拟路由(DVR)的 ML2/OVN OPENSTACK	150
<b>第 14 章 使用 IPV6 的项目网络</b>	<b>151</b>
14.1. IPV6 子网选项	151
14.2. 使用 STATEFUL DHCPV6 创建 IPV6 子网	152
<b>第 15 章 管理项目配额</b>	<b>155</b>
15.1. 配置项目配额	155
15.2. L3 配额选项	155
15.3. 防火墙配额选项	155
15.4. 安全组配额选项	155
15.5. 管理配额选项	156
<b>第 16 章 部署路由的供应商网络</b>	<b>157</b>
16.1. 路由供应商网络的优点	157
16.2. 路由供应商网络的基本信息	157
16.3. 路由供应商网络的限制	157
16.4. 准备路由的提供商网络	158
16.5. 创建路由的提供商网络	160
16.6. 将非路由网络迁移到路由的提供商网络	166

<b>第 17 章 配置允许的地址对</b> .....	<b>169</b>
17.1. 允许的地址对概述	169
17.2. 创建端口并允许一个地址对	169
17.3. 添加允许的地址对	170
<b>第 18 章 日志记录安全组操作</b> .....	<b>171</b>
18.1. 验证是否启用了安全组日志记录	172
18.2. 为安全组创建日志对象	172
18.3. 列出和查看安全组的日志对象	173
18.4. 为安全组启用和禁用日志对象	173
18.5. 为安全组重命名日志对象	174
18.6. 删除安全组的日志对象	174
18.7. 访问安全组日志内容	174
18.8. 安全组日志内容示例	174
18.9. 为安全组日志记录调整速率和突发限制	175
<b>第 19 章 常见的管理网络任务</b> .....	<b>176</b>
19.1. 配置 L2 填充驱动程序	176
19.2. 调整 KEEPALIVED 以避免 VRRP 数据包丢失	178
19.3. 指定 DNS 分配给端口的名称	179
19.4. 为端口分配 DHCP 属性	182
19.5. 载入内核模块	184
19.6. 配置共享安全组	185
<b>第 20 章 配置第 3 层高可用性(HA)</b> .....	<b>187</b>
20.1. 没有高可用性(HA)的 RHOSP 网络服务	187
20.2. 第 3 层高可用性概述(HA)	187
20.3. 第 3 层高可用性(HA)故障转移条件	187
20.4. 第 3 层高可用性(HA)的项目注意事项	188
20.5. RHOSP 网络服务的高可用性(HA)更改	188
20.6. 在 RHOSP 网络服务节点上启用第 3 层高可用性(HA)	188
20.7. 查看高可用性(HA) RHOSP 网络服务节点配置	190
<b>第 21 章 替换 NETWORKER 节点</b> .....	<b>191</b>
21.1. 准备替换网络节点	191
21.2. 替换网络器节点	192
21.3. 重新调度节点并清理网络服务	195
<b>第 22 章 使用可用区使网络资源高度可用</b> .....	<b>197</b>
22.1. 关于网络服务可用区	197
22.2. 为 ML2/OVS 配置网络服务可用区	197
22.3. 使用 ML2/OVN 配置网络服务可用区	200
22.4. 手动将可用区分配给网络和路由器	203
<b>第 23 章 使用标签识别虚拟设备</b> .....	<b>206</b>
23.1. 标记虚拟设备	206





## 前言



### 注意

您不能在实例创建过程中将基于角色的访问控制(RBAC)共享安全组直接应用到实例。要将 RBAC 共享安全组应用到实例，必须首先创建端口，将共享安全组应用到该端口，然后将该端口分配给实例。请参阅 [将安全组添加到端口](#)。

---

## 使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 [CTO Chris Wright 的信息](#)。

## 对红帽文档提供反馈

我们感谢您对文档提供反馈信息。与我们分享您的成功秘诀。

### 在 JIRA 中提供文档反馈

使用 [Create Issue](#) 表单对文档提供反馈。JIRA 问题将在 Red Hat OpenStack Platform Jira 项目中创建，您可以在其中跟踪您的反馈进度。

1. 确保您已登录到 JIRA。如果您没有 JIRA 帐户，请创建一个帐户来提交反馈。
2. 点击以下链接打开 **Create Issue** 页面：[Create Issue](#)
3. 完成 **Summary** 和 **Description** 字段。在 **Description** 字段中，包含文档 URL、章节或章节号以及问题的详细描述。不要修改表单中的任何其他字段。
4. 点 **Create**。

# 第 1 章 OPENSTACK 网络简介

Networking 服务(neutron)是 Red Hat OpenStack Platform (RHOSP)的软件定义型网络(SDN)组件。RHOSP 网络服务管理虚拟机实例的内部和外部流量，并提供路由、分段、DHCP 和元数据等核心服务。它为虚拟网络功能提供 API，以及交换机、路由器、端口和防火墙的管理。

## 1.1. 管理 RHOSP 网络

通过 Red Hat OpenStack Platform (RHOSP)网络服务(neutron)，您可以有效地满足您的站点网络目标。您可以：

- **提供项目中虚拟机实例的连接。**

项目网络主要启用常规（非特权）项目，在不涉及管理员的情况下管理网络。这些网络完全虚拟，需要虚拟路由器与其他项目网络和外部网络（如互联网）交互。项目网络通常还会为实例提供 DHCP 和元数据服务。RHOSP 支持以下项目网络类型：flat、VLAN、VXLAN、GRE 和 GENEVE。

如需更多信息，[请参阅管理项目网络。](#)

- **将虚拟机实例连接到项目外的网络。**

提供商网络提供如项目网络的连接。但是，只有管理（非特权）用户可以管理这些网络，因为它们与物理网络基础架构进行接口。RHOSP 支持以下供应商网络类型：扁平化和 VLAN。

在项目网络内，您可以使用浮动 IP 地址池或单个浮动 IP 地址将入口流量定向到虚拟机实例。使用网桥映射，您可以将物理网络名称（接口标签）与通过 OVS 或 OVN 创建的网桥关联，以允许提供商网络流量访问物理网络。

如需更多信息，[请参阅将虚拟机实例连接到物理网络。](#)

- **创建为边缘计算优化的网络。**

Operator 可以创建通常在边缘部署中使用的路由供应商网络，它依赖于多个第 2 层网络段，而不是仅有一个网段的传统网络。

路由提供商网络为最终用户简化云，因为它们只看到一个网络。对于云操作员，路由供应商网络提供可扩展和容错能力。例如，如果发生重大错误，则只有一个片段会受到影响，而不是整个网络失败。

如需更多信息，[请参阅部署路由的供应商网络。](#)

- **将您的网络资源高度可用。**

您可以使用可用区(AZ)和虚拟路由器冗余协议(VRRP)保持网络资源高度可用。operator 组附加到不同 AZ 上不同电源源的网络节点。接下来，Operator 调度关键服务，如 DHCP、L3、FW 等，使其位于单独的 AZ 上。

RHOSP 使用 VRRP 使项目路由器和浮动 IP 地址高度可用。集中式路由，分布式虚拟路由(DVR)根据 VRRP 提供替代路由设计，该设计将部署 L3 代理并在每个 Compute 节点上调度路由器。

如需更多信息，[请参阅使用可用区使网络资源高度可用。](#)

- **在端口级别保护您的网络。**

安全组为虚拟防火墙规则提供一个容器，用于控制入口（绑定到实例）和出口（从实例绑定）网络流量（通过端口级别出站）。安全组使用默认拒绝策略，仅包含允许特定流量的规则。每个端口都可以以补充的方式引用一个或多个安全组。防火墙驱动程序将安全组规则转换为底层数据包过滤技术的配置，如 iptables。

如需更多信息，[请参阅配置共享安全组。](#)

- **管理端口流量。**

通过允许的地址对，您可以识别特定的 MAC 地址、IP 地址或两者，以允许网络流量通过端口传递，而不考虑子网。当您定义允许的地址对时，您可以使用 VRRP（虚拟路由器冗余协议）的协议，该协议在两个虚拟机实例之间浮点 IP 地址以启用快速数据平面故障转移。

如需更多信息，请参[阅配置允许的地址对](#)。

- **优化大型覆盖网络。**

使用 L2 Population 驱动程序，您可以启用广播、多播和单播流量，以便在大型覆盖网络上扩展。

如需更多信息，请参[阅配置 L2 填充驱动程序](#)。

- **为虚拟机实例上的流量设置入口和出口限制。**

您可以使用服务质量(QoS)策略为实例提供不同的服务级别，以将速率限制应用到出口和入口流量。您可以将 QoS 策略应用到单独的端口。您还可以将 QoS 策略应用到项目网络，其中没有附加特定策略的端口会继承策略。

如需更多信息，请参[阅配置服务质量\(QoS\)策略](#)。

- **管理 RHOSP 项目可以创建的网络资源量。**

借助网络服务配额选项，您可以设置用户可以创建的网络资源项目数量的限值。这包括端口、子网、网络等资源。

如需更多信息，请参[阅管理项目配额](#)。

- **优化用于网络功能虚拟化(SVVP)的虚拟机实例。**

实例可以通过单个虚拟 NIC 发送和接收 VLAN 标记的流量。这对希望 VLAN 标记流量的 NFV 应用程序(VNF)特别有用，允许单个虚拟 NIC 为多个客户或服务提供服务。

在 VLAN 透明网络中，您可以在虚拟机实例中设置 VLAN 标记。VLAN 标签通过网络传输，并由同一 VLAN 上的虚拟机实例消耗，并被其他实例和设备忽略。VLAN 中继通过将 VLAN 合并到单个中继端口来支持 VLAN 感知实例。

如需更多信息，请参[阅 VLAN 感知实例](#)。

- **控制哪些项目可以将实例附加到共享网络。**

在 RHOSP 网络服务中使用基于角色的访问控制(RBAC)策略，云管理员可以删除某些项目创建网络的能力，并可以让它们附加到与其项目对应的预先存在的网络。

如需更多信息，请参[阅配置 RBAC 策略](#)。

## 1.2. 网络服务组件

Red Hat OpenStack Platform (RHOSP)网络服务(neutron)包括以下组件：

- **API Server**

RHOSP 网络 API 包括对第 2 层网络和 IP 地址管理(IPAM)的支持，以及第 3 层路由器构造的扩展，它允许第 2 层网络和网关到外部网络之间的路由。RHOSP 网络包括一个增加的插件列表，它允许与各种商业和开源网络技术互操作性，包括路由器、交换机、虚拟交换机和软件定义网络(SDN)控制器。

- **模块层 2 (ML2)插件和代理**

ML2 插件和拔出端口，创建网络或子网，并提供 IP 地址。

- **消息传递队列**

接受和路由代理之间的 RPC 请求，以完成 API 操作。消息队列在 ML2 插件中使用，用于在每个虚拟机监控程序上运行的 neutron 服务器和 neutron 代理之间的 RPC，用于 Open vSwitch 和 Linux 网桥的 ML2 机制驱动程序。

### 1.3. 模块层 2 (ML2)网络

模块层 2 (ML2)是 Red Hat OpenStack Platform (RHOSP)网络核心插件。ML2 模块设计通过机制驱动程序支持混合网络技术的并发操作。Open Virtual Network (OVN)是 ML2 使用的默认机制驱动程序。

ML2 框架区分了可配置的两​​种驱动程序：

#### 类型驱动程序

定义从技术上实现 RHOSP 网络的方式。

每种可用网络类型都由 ML2 类型驱动程序管理，它们维护任何所需的特定类型网络状态。验证提供商网络的特定于类型的信息，类型驱动程序负责在项目网络中分配空闲片段。类型驱动程序的示例包括 GENEVE、GRE、VXLAN 等。

#### 机制驱动程序

定义访问特定类型的 RHOSP 网络的机制。

机制驱动程序使用类型驱动程序建立的信息，并将其应用到已启用的网络机制。机制驱动程序示例是 Open Virtual Networking (OVN)和 Open vSwitch (OVS)。

机制驱动程序可以使用 L2 代理，并使用 RPC 直接与外部设备或控制器交互。您可以同时使用多种机制，并同时键入驱动程序来访问同一虚拟网络的不同端口。

#### 其他资源

- [第 1.8 节 “模块层 2 \(ML2\)类型和机制驱动程序兼容性”](#)

### 1.4. ML2 网络类型

您可以同时运行多个网络片段。ML2 支持多个网络段的使用和互连。您不必将端口绑定到网络段，因为 ML2 将端口绑定到具有连接性的端口。根据机制驱动程序，ML2 支持以下网络片段类型：

- flat
- VLAN
- GENEVE 隧道
- VXLAN 和 GRE 隧道

#### flat

所有虚拟机 (VM) 实例都位于同一网络中，也可以与主机共享。没有 VLAN 标记或其他网络分离。

#### VLAN

使用 RHOSP 网络用户可以使用 VLAN ID (802.1Q 标记)创建多个供应商或项目网络，对应于物理网络中存在的 VLAN。这允许实例在环境中相互通信。它们还可以与同一第 2 层 VLAN 上的专用服务器、防火墙、负载均衡器和其他网络基础架构通信。

您可以使用 VLAN 来分段在同一交换机上运行的计算机的网络流量。这意味着，您可以通过将端口配置为不同的网络成员来以逻辑方式划分交换机，以便出于安全原因，您可以使用这些端口来分隔流量。

例如，如果您的交换机总有 24 个端口，您可以将端口 1-6 分配给 VLAN200，并将端口 7-18 分配给 VLAN201。因此，连接到 VLAN200 的计算机与 VLAN201 上的计算机完全分开；它们无法直接通信，如果需要，流量必须通过路由器传递，就像它们是两个独立的物理交换机一样。防火墙也可用于管理哪些 VLAN 可以相互通信。

## GENEVE 隧道

GENEVE 识别并适应网络虚拟化中不同设备的功能和需求。它为隧道提供框架，而不是针对整个系统的规定。Geneve 定义封装期间添加的元数据内容，并尝试适应各种虚拟化场景。它使用 UDP 作为其传输协议，并且使用可扩展选项标头的大小动态。Geneve 支持单播、多播和广播。GENEVE 类型驱动程序与 ML2/OVN 机制驱动程序兼容。

## VXLAN 和 GRE 隧道

VXLAN 和 GRE 使用网络覆盖来支持实例间的私有通信。需要 RHOSP 网络路由器，以便流量在 GRE 或 VXLAN 项目网络外到达流量。还需要路由器才能将网络连接的项目网络与外部网络（包括互联网）连接；路由器能够使用浮动 IP 地址直接从外部网络连接实例。VXLAN 和 GRE 类型驱动程序与 ML2/OVS 机制驱动程序兼容。

## 其他资源

- [第 1.8 节 “模块层 2 \(ML2\)类型和机制驱动程序兼容性”](#)

## 1.5. 模块第 2 层(ML2)机制驱动程序

模块层 2 (ML2)插件作为带有通用代码库的机制实施。这种方法能够重复使用代码，并消除了代码维护和测试的许多复杂性。

您可以使用编排服务(heat)参数启用机制驱动程序，即 **NeutronMechanismDrivers**。以下是 heat 自定义环境文件的示例：

```
parameter_defaults:
  ...
  NeutronMechanismDrivers: ansible,ovn,baremetal
  ...
```

指定机制驱动程序的顺序很重要。在前面的示例中，如果要使用 baremetal 机制驱动程序绑定端口，则必须在 **ansible** 前指定 **baremetal**。否则，ansible 驱动程序将绑定端口，因为它在 **NeutronMechanismDrivers** 的值列表前面是 **baremetal**。

从 RHOSP 15 开始，红帽选择 ML2/OVN 作为所有新部署的默认机制驱动程序，因为它为大多数客户提供了 ML2/OVS 机制驱动程序的即时优势。这些优点会随每个发行版本一起实现，同时我们继续增强并改进 ML2/OVN 功能集。

通过 RHOSP 17 发行版本提供对已弃用的 ML2/OVS 机制驱动程序的支持。在此期间，ML2/OVS 驱动程序处于维护模式，接收程序错误修复和正常支持，大多数新功能开发发生在 ML2/OVN 机制驱动程序中。

在 RHOSP 18.0 中，红帽计划完全删除 ML2/OVS 机制驱动程序并停止支持它。

如果您的现有 Red Hat OpenStack Platform (RHOSP)部署使用 ML2/OVS 机制驱动程序，请开始评估计划迁移到机制驱动程序。RHOSP 16.2 支持迁移，并将在 RHOSP 17.1 中被支持。RHOSP 17.0 中包括了迁移工具用于测试目的。

在尝试从 ML2/OVS 迁移到 ML2/OVN 之前，红帽要求您提交一个主动支持问题单。红帽不支持在没有主动支持问题单的情况下进行迁移。请参阅 [如何在 Red Hat OpenStack Platform 上为计划活动打开主动问题单？](#)



## 其他资源

- [将网络服务迁移到 ML2 OVN 机制驱动程序](#)
- [Red Hat OpenStack Platform 中的组件、插件和驱动程序支持中的 Neutron](#)
- [高级 Overcloud 自定义指南中的环境文件。](#)
- [高级 Overcloud 自定义指南中的创建 overcloud 中包括环境文件](#)

## 1.6. OPEN VSWITCH

Open vSwitch (OVS)是一个软件定义的网络(SDN)虚拟交换机，类似于 Linux 软件网桥。OVS 为虚拟网络提供交换服务，支持行业标准 OpenFlow 和 sFlow。OVS 也可以使用第 2 层功能（如 STP、LACP 和 802.1Q VLAN 标记）与物理交换机集成。Open vSwitch 版本 1.11.0-1.el6 或更高版本也支持使用 VXLAN 和 GRE 进行隧道。

有关网络接口绑定的更多信息，请参阅[高级 Overcloud 自定义指南中的网络接口绑定](#)。



### 注意

为降低 OVS 中网络循环的风险，只能有一个接口或单个绑定作为给定网桥的成员。如果需要多个绑定或接口，可以配置多个网桥。

## 1.7. 打开虚拟网络(OVN)

Open Virtual Network (OVN)是一个系统，支持虚拟机和容器环境中的逻辑网络抽象。有时为 Open vSwitch 称为开源虚拟网络，OVN 会补充 OVS 的现有功能，以添加逻辑网络抽象的原生支持，如逻辑 L2 和 L3 覆盖、安全组和服务，如 DHCP。

物理网络包含物理有线、交换机和路由器。虚拟网络将物理网络扩展到虚拟机监控程序或容器平台，将虚拟机或容器桥接到物理网络中。OVN 逻辑网络是在软件中实施的，它通过隧道或其他封装从物理网络中计算。这允许逻辑网络中使用的 IP 和其他地址空间与物理网络中使用的 IP 和其他地址空间重叠，而不会导致冲突。可以在不考虑其运行的物理网络拓扑的情况下排列逻辑网络拓扑。因此，作为逻辑网络一部分的虚拟机可以在不中断的情况下从一个物理机器迁移到另一个物理机器。

封装层可防止连接到逻辑网络的虚拟机和容器与物理网络上的节点通信。对于集群虚拟机和容器，这可能可以接受甚至需要这样做，但在很多情况下，虚拟机和容器需要连接到物理网络。OVN 提供多种形式的网关来实现这一目的。OVN 部署由多个组件组成：

### 云管理系统(CMS)

通过管理 OVN 逻辑网络元素并将 OVN 集成到物理网络中，并将 OVN 逻辑网络基础架构连接到物理网络元素。一些示例包括 OpenStack 和 OpenShift。

### OVN 数据库

存储代表 OVN 逻辑和物理网络的数据。

### hypervisor

在物理或虚拟机上运行 Open vSwitch，并将 OVN 逻辑网络转换为 OpenFlow。

### 网关

通过在隧道和物理网络基础架构间转发数据包，将基于隧道的 OVN 逻辑网络扩展到物理网络中。

## 1.8. 模块层 2 (ML2)类型和机制驱动程序兼容性

在规划 Red Hat OpenStack Platform (RHOSP) 数据网络时，请参考下表，以确定每个修改层 2 (ML2) 机制驱动程序支持的网络类型。

表 1.1. ML2 机制驱动程序支持的网络类型

机制驱动程序	支持这些类型驱动程序				
	flat	GRE	VLAN	VXLAN	GENEVE
打开虚拟网络(OVN)	是	否	是	是 [1]	是
Open vSwitch (OVS)	是	是	是	是	否

[1] ML2/OVN VXLAN 支持仅限于每个网络的 4096 个网络和 4096 端口。另外，依赖入口端口的 ACL 无法用于 ML2/OVN 和 VXLAN，因为入口端口没有被传递。

## 1.9. RHOSP 网络服务的扩展驱动程序

Red Hat OpenStack Platform (RHOSP) 网络服务(neutron)可扩展。扩展用于两个目的：它们允许在 API 中引入新功能，而无需更改版本，它们允许引入特定于供应商的 niche 功能。应用程序可以通过对 `/extensions` URI 执行 GET 来以编程方式列出可用扩展。请注意，这是一个版本化请求；也就是说，一个 API 版本中可用的扩展可能在另一个 API 版本中不可用。

ML2 插件还支持扩展驱动程序，允许其他可插拔驱动程序扩展 ML2 插件中用于网络对象实施的核心资源。扩展驱动程序示例包括支持 QoS、端口安全性等。

## 第 2 章 使用 ML2/OVN

Red Hat OpenStack Platform (RHOSP) 网络由 Networking 服务(neutron)管理。网络服务的核心是 Modular Layer 2 (ML2) 插件，RHOSP ML2 插件的默认机制驱动程序是 Open Virtual Networking (OVN) 机制驱动程序。

较早的 RHOSP 版本默认使用 Open vSwitch (OVS) 机制驱动程序，但红帽建议为大多数部署推荐 ML2/OVN 机制驱动程序。

如果您从 RHOSP 13 ML2/OVS 部署升级到 RHOSP 16，红帽建议升级后从 ML2/OVS 迁移到 ML2/OVN。在某些情况下，ML2/OVN 可能无法满足您的要求。在这些情况下，您可以使用 ML2/OVS 部署 RHOSP。

### 2.1. RHOSP OVN 架构中的组件列表

RHOSP OVN 架构将 OVS Modular Layer 2 (ML2) 机制驱动程序替换为 OVN ML2 机制驱动程序来支持网络 API。OVN 为 Red Hat OpenStack 平台提供网络服务。

如图 2.1 所示，OVN 架构由以下组件和服务组成：

#### ML2 插件带有 OVN 机制驱动程序

ML2 插件将 OpenStack 特定的网络配置转换为平台中立的 OVN 逻辑网络配置。它通常在 Controller 节点上运行。

#### OVN 北向(NB)数据库(ovn-nb)

此数据库存储 OVN ML2 插件的逻辑 OVN 网络配置。它通常在 Controller 节点上运行，并侦听 TCP 端口 **6641**。

#### OVN 北向服务(ovn-northd)

此服务将逻辑网络配置从 OVN NB 数据库转换为逻辑数据路径流，并在 OVN 南向数据库中填充它们。它通常在 Controller 节点上运行。

#### OVN 南向(SB)数据库(ovn-sb)

这个数据库存储转换的逻辑数据路径流。它通常在 Controller 节点上运行，并侦听 TCP 端口 **6642**。

#### OVN 控制器(ovn-controller)

此控制器连接到 OVN SB 数据库，并充当 Open vSwitch 控制器来控制 and 监控网络流量。它在所有定义 **OS::Tripleo::Services::OVNController** 的 Compute 和网关节点上运行。

#### OVN 元数据代理(ovn-metadata-agent)

此代理创建 **haproxy** 实例，用于管理用于代理元数据 API 请求的 OVS 接口、网络命名空间和 HAProxy 进程。代理在所有定义 **OS::Tripleo::Services::OVNMetadataAgent** 的 Compute 和网关节点上运行。

#### OVS 数据库服务器(OVSDB)

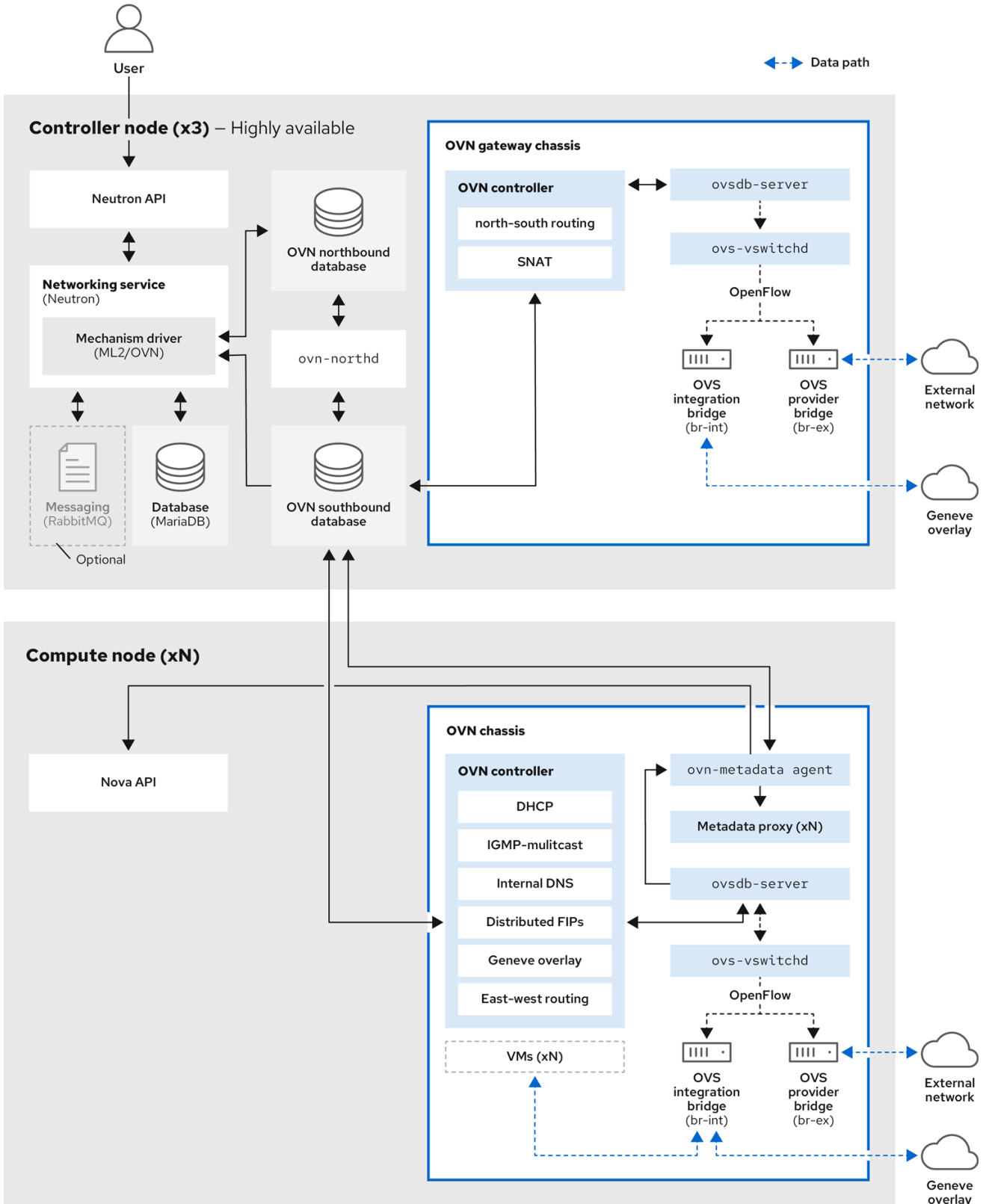
托管 OVN 北向和南向数据库。还与 **ovs-vswitchd** 交互，以托管 OVS 数据库 **conf.db**。



#### 注意

NB 数据库的 schema 文件位于 **/usr/share/ovn/ovn-nb.ovsschema** 中，SB 数据库架构文件位于 **/usr/share/ovn/ovn-sb.ovsschema** 中。

图 2.1. RHOSP 环境中的 OVN 架构



329\_OpenStack\_0923

## 2.2. ML2/OVN 数据库

在 Red Hat OpenStack Platform ML2/OVN 部署中，网络配置信息通过共享分布式数据库在进程之间传递。您可以检查这些数据库以验证网络的状态并发现问题。

## OVN 北向数据库

北向数据库(**OVN\_Northbound**)充当 OVN 和云管理系统（如 Red Hat OpenStack Platform (RHOSP)）之间的接口。RHOSP 生成北向数据库的内容。

北向数据库包含网络的当前状态，以逻辑端口、逻辑交换机、逻辑路由器等的形式显示。每个 RHOSP 网络服务(neutron)对象在北向数据库的表中表示。

## OVN 南向数据库

南向数据库(**OVN\_Southbound**)包含 OVN 系统的逻辑和物理配置状态，以支持虚拟网络抽象。**ovn-controller** 使用此数据库中的信息配置 OVS 以满足网络服务(neutron)要求。

## 2.3. COMPUTE 节点上的 OVN-CONTROLLER 服务

**ovn-controller** 服务在每个 Compute 节点上运行，并连接到 OVN 南向(SB)数据库服务器，以检索逻辑流。**ovn-controller** 将这些逻辑流转换为物理 OpenFlow 流，并将流添加到 OVS 网桥(**br-int**)。要与 **ovs-vsitchd** 进行通信并安装 OpenFlow 流，**ovn-controller** 使用在 **ovn-controller** 启动时传递的 UNIX 套接字路径（如 **unix:/var/run/openvswitch/db.sock**）连接到本地 **ovsdb-server**（托管 **conf.db**）。

**ovn-controller** 服务需要在 **Open\_vSwitch** 表的 **external\_ids** 列中需要特定的键值对；**puppet-ovn** 使用 **puppet-vswitch** 来填充这些字段。以下示例显示了 **puppet-vswitch** 在 **external\_ids** 列中配置的键值对：

```
hostname=<HOST NAME>
ovn-encap-ip=<IP OF THE NODE>
ovn-encap-type=geneve
ovn-remote=tcp:OVN_DBS_VIP:6642
```

## 2.4. COMPUTE 节点上的 OVN 元数据代理

OVN 元数据代理在 **tripleo-heat-templates/deployment/ovn-metadata-container-puppet.yaml** 文件中配置，并通过 **OS::TripleO::Services::OVNMetadataAgent** 包含在默认的 Compute 角色中。因此，带有默认参数的 OVN 元数据代理作为 OVN 部署的一部分被部署。

OpenStack 虚拟机实例通过链接 IP 地址访问网络元数据服务：169.254.169.254。**neutron-ovn-metadata-agent** 能够访问存在计算元数据 API 的主机网络。每个 HAProxy 都位于无法访问适当的主机网络的网络命名空间中。HAProxy 将必要的标头添加到元数据 API 请求中，然后将请求通过 UNIX 域套接字转发到 **neutron-ovn-metadata-agent**。

OVN 网络服务为每个虚拟网络创建一个唯一网络命名空间，以启用元数据服务。Compute 节点上的实例访问的每个网络都有对应的元数据命名空间(ovnmeta-<network\_uuid>)。

## 2.5. OVN 可组合服务

Red Hat OpenStack Platform 通常由预定义角色中的节点组成，如 Controller 角色、Compute 角色和不同的存储角色类型。每个默认角色都包含一组在核心 heat 模板集合中定义的服务。

在默认的 Red Hat OpenStack (RHOSP)部署中，ML2/OVN 可组合服务在 Controller 节点上运行。您可以选择创建自定义 Networker 角色，并在专用 Networker 节点上运行 OVN 可组合服务。

OVN 可组合服务 **ovn-dbs** 部署到名为 **ovn-dbs-bundle** 的容器中。在默认安装 **ovn-dbs** 中包含在 Controller 角色中，并在 Controller 节点上运行。由于服务可组合使用，您可以将其分配到另一个角色，如 Networker 角色。

如果您将 OVN 可组合服务分配给另一个角色，请确保该服务与 pacemaker 服务位于同一个节点上，该服务控制 OVN 数据库容器。

### 相关信息

- [使用 ML2/OVN 部署自定义角色](#)
- [带有 ML2/OVN 和原生 OVN DHCP 的 SR-IOV](#)

## 2.6. OVN 第 3 层高可用性

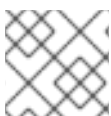
OVN 支持第 3 层高可用性(L3 HA)，无需任何特殊配置。



### 注意

在创建路由器时，请勿使用 `--ha` 选项，因为 OVN 路由器默认具有高可用性。**OpenStack router 创建包含 `--ha` 选项的命令会失败。**

OVN 自动将路由器端口调度到所有可用网关节点，它们可作为指定外部网络上的 L3 网关。OVN L3 HA 使用 OVN **Logical\_Router\_Port** 表中的 **gateway\_chassis** 列。大多数功能都由带有捆绑的 `active_passive` 输出的 OpenFlow 规则管理。**ovn-controller** 处理地址解析协议(ARP)响应器和路由器启用和禁用。为 FIP 和路由器外部地址获取 ARP 也会定期由 **ovn-controller** 发送。



### 注意

L3HA 使用 OVN 将路由器恢复回原始网关节点，以避免任何节点成为瓶颈。

### BFD 监控

OVN 使用双向转发检测(BFD)协议来监控网关节点的可用性。此协议封装在从节点到节点的 Geneve 隧道之上。

每个网关节点监控部署中星号拓扑中的所有其他网关节点。网关节点还监控计算节点，使网关启用和禁用数据包的路由和 ARP 响应和公告。

每个计算节点使用 BFD 监控每个网关节点，并通过给定路由器的活跃网关节点自动窃取外部流量，如源和目标网络地址转换(SNAT 和 VNI)。Compute 节点不需要监控其他计算节点。



### 注意

没有检测到外部网络失败，因为 ML2-OVS 配置会发生。

OVN 的 L3 HA 支持以下故障模式：

- 网关节点与网络断开连接（隧道接口）。
- **ovs-vswitchd** 停止(**ovs-switchd** 负责 BFD 信号)
- **OVN-controller** 停止(**ovn-controller** 将自身作为注册的节点删除)。



### 注意

这个 BFD 监控机制仅适用于链接失败，不适用于路由失败。

## 2.7. OVN 和 OVS 机制驱动程序中的功能支持

检查 Red Hat OpenStack Platform (RHOSP)功能的可用性，作为 OVS 到 OVN 机制驱动程序迁移计划的一部分。

功能	OVN RHOS P 16.2	OVN RHOS P 17.1	OVS RHOS P 16.2	OVS RHOS P 17.1	附加信息
使用 OVN DHCP 置备裸机	否	否	是	是	OVN 上的内置 DHCP 服务器无法置备裸机节点。它无法为 provisioning 网络提供 DHCP。Chainbooting iPXE 需要标记(dnsmasq 中的-- <b>dhcp-match</b> )，这在 OVN DHCP 服务器中不被支持。See <a href="https://bugzilla.redhat.com/show_bug.cgi?id=1622154">https://bugzilla.redhat.com/show_bug.cgi?id=1622154</a> .
VLAN 项目（租户网络）端口的 VF（直接）端口的北/南路由	否	否	是	是	核心 OVN 限制。请参阅 <a href="https://bugs.launchpad.net/neutron/+bug/1875852">https://bugs.launchpad.net/neutron/+bug/1875852</a> 。
内部 DNS 记录的反向 DNS	否	是	是	是	See <a href="https://bugzilla.redhat.com/show_bug.cgi?id=2211426">https://bugzilla.redhat.com/show_bug.cgi?id=2211426</a> .
隔离网络的内部 DNS 解析	否	否	是	是	OVN 不支持隔离网络的内部 DNS 解析，因为它不为 DNS 服务分配端口。这不会影响 OVS 部署，因为 OVS 使用 dnsmasq。请参阅 <a href="https://issues.redhat.com/browse/OSP-25661">https://issues.redhat.com/browse/OSP-25661</a> 。
安全组日志记录	技术预览	是	否	否	RHOSP 不支持使用 OVS 机制驱动程序的安全组日志记录。
无状态安全组	否	是	否	否	请参阅 <a href="#">配置安全组</a> 。
负载均衡服务分布式虚拟路由(DVR)	是	是	否	否	OVS 机制驱动程序通过 Controller 或网络节点路由负载均衡服务流量，即使启用了 DVR。OVN 机制驱动程序通过 Compute 节点直接路由负载均衡服务流量。
IPv6 DVR	是	是	否	否	使用 OVS 机制驱动程序时，RHOSP 不会将 IPv6 流量分发到 Compute 节点，即使启用了 DVR。所有入口/出口流量都通过集中式 Controller 或 Network 节点。如果您需要 IPv6 DVR，请使用 OVN 机制驱动程序。

功能	OVN RHOS P 16.2	OVN RHOS P 17.1	OVS RHOS P 16.2	OVS RHOS P 17.1	附加信息
DVR 和第 3 层高可用性(L3 HA)	是	是	否	否	使用 OVS 机制驱动程序的 RHOSP 部署不支持 DVR 与 L3 HA 结合使用。如果您将 DVR 与 RHOSP director 搭配使用，则禁用 L3 HA。这意味着网络服务仍然在网络节点上调度路由器，并在 L3 代理间共享它们。但是，如果一个代理失败，由此代理托管的所有路由器也会失败。这只会影响 SNAT 流量。红帽建议在这种情况下使用 <b>allow_automatic_l3agent_failover</b> 功能，以便在一个网络节点失败时，路由器会重新调度到不同的节点。

## 2.8. 使用 ML2/OVN 对非安全端口的限制

如果您在使用默认 ML2/OVN 机制驱动程序的 Red Hat Open Stack Platform (RHOSP)部署中禁用端口安全插件扩展以及大量端口，则端口可能无法访问。

在某些大型 ML2/OVN RHOSP 部署中，ML2/OVN 中的流链限制可能会丢弃指向禁用安全插件的端口的 ARP 请求。

ML2/OVN 可以支持的实际逻辑交换机端口数量没有记录的最大限制，但限制大约有 4,000 个端口。

贡献大约限制的属性是 ML2/OVN 生成的 OpenFlow 管道重新提交的数量，并更改整个逻辑拓扑。

## 2.9. ML2/OVS 到 ML2/OVN 原位迁移：验证和禁止的情况

红帽会继续测试和优化原位迁移场景。与红帽大客户经理或全球专业服务合作，以确定您的 OVS 部署是否满足有效原位升级场景的条件。

### 2.9.1. 验证 ML2/OVS 到 ML2/OVN 迁移场景

#### DVR 到 DVR

启动：使用 DVR 的 OVS 的 RHOSP 16.1.1 或更高版本。  
end: 与 DVR 的 OVN 相同的 RHOSP 版本和发行版本。

SR-IOV 不在启动环境中或迁移后添加。

#### 集中式路由 + 使用虚拟功能(VF)端口的 SR-IOV

启动：使用 OVS（无 DVR）和 SR-IOV 的 RHOSP 16.1.1 或更高版本。  
end: 与 OVN（无 DVR）和 SR-IOV 相同的 RHOSP 版本和发行版本。

使用 SR-IOV 虚拟功能(VF)端口的工作负载。SR-IOV 物理功能(PF)端口会导致迁移失败。

### 2.9.2. ML2/OVS 到 ML2/OVN 的原位迁移尚未被验证

在红帽宣布根本问题解决之前，您无法在以下情况下执行从 ML2/OVS 到 ML2/OVN 的原位升级。



## OVS 部署使用网络功能虚拟化(SVVP)

红帽支持使用 ML2/OVN 和 NFV 的新部署，但尚未成功测试 ML2/OVS 和 NFV 部署迁移到 ML2/OVN。要跟踪此问题的进度，请参阅 [https://bugzilla.redhat.com/show\\_bug.cgi?id=1925290](https://bugzilla.redhat.com/show_bug.cgi?id=1925290)。

## 带有物理功能(PF)端口的 SR-IOV

当任何工作负载使用 SR-IOV PF 端口时，迁移测试会失败。要跟踪此问题的进度，请参阅 [https://bugzilla.redhat.com/show\\_bug.cgi?id=1879546](https://bugzilla.redhat.com/show_bug.cgi?id=1879546)。

## OVS 使用中继端口

如果您的 ML2/OVS 部署使用中继端口，请不要执行 ML2/OVS 到 ML2/OVN 迁移。迁移不会在 OVN 环境中正确设置中继端口。要跟踪此问题的进度，请参阅 [https://bugzilla.redhat.com/show\\_bug.cgi?id=1857652](https://bugzilla.redhat.com/show_bug.cgi?id=1857652)。

## 带有 VLAN 项目（租户）网络的 DVR

不要使用 DVR 和 VLAN 项目网络迁移到 ML2/OVN。您可以使用集中式路由迁移到 ML2/OVN。要跟踪此问题的进度，请参阅 [https://bugzilla.redhat.com/show\\_bug.cgi?id=1766930](https://bugzilla.redhat.com/show_bug.cgi?id=1766930)。

### 2.9.3. ML2/OVS 到 ML2/OVN 原位迁移和安全组规则

确保原始 ML2/OVS 部署中的任何自定义安全组规则都与目标 ML2/OVN 部署兼容。

例如，默认安全组包含允许出口到 DHCP 服务器的规则。如果您在 ML2/OVS 部署中删除了这些规则，ML2/OVS 会自动添加允许到 DHCP 服务器的隐式规则。ML2/OVN 不支持这些隐式规则，因此在目标 ML2/OVN 环境中，DHCP 和元数据流量无法访问 DHCP 服务器，实例将无法引导。在这种情况下，要恢复 DHCP 访问，您可以添加以下规则：

```
# Allow VM to contact dhcp server (ipv4)
openstack security group rule create --egress --ethertype IPv4 --protocol udp --dst-port 67
${SEC_GROUP_ID}
# Allow VM to contact metadata server (ipv4)
openstack security group rule create --egress --ethertype IPv4 --protocol tcp --remote-ip
169.254.169.254 ${SEC_GROUP_ID}

# Allow VM to contact dhcp server (ipv6, non-slaac). Be aware that the remote-ip may vary
depending on your use case!
openstack security group rule create --egress --ethertype IPv6 --protocol udp --dst-port 547 --
remote-ip ff02::1:2 ${SEC_GROUP_ID}
# Allow VM to contact metadata server (ipv6)
openstack security group rule create --egress --ethertype IPv6 --protocol tcp --remote-ip
fe80::a9fe:a9fe ${SEC_GROUP_ID}
```

## 2.10. 在新的 RHOSP 16.2 部署中使用 ML2/OVS 而不是默认的 ML2/OVN

在 Red Hat OpenStack Platform (RHOSP) 16.0 及之后的版本部署中，带有 Open Virtual Network (ML2/OVN) 的 Modular Layer 2 插件是 RHOSP 网络服务的默认机制驱动程序。如果应用程序需要 ML2/OVS 机制驱动程序，您可以更改此设置。

### 流程

1. 以 **stack** 用户身份登录 undercloud。
2. 在模板文件 `/home/stack/templates/containers-prepare-parameter.yaml` 中，使用 **ovs** 而不是 **ovn** 作为 **neutron\_driver** 参数的值：

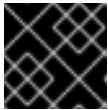
```
parameter_defaults:
  ContainerImagePrepare:
    - set:
      ...
      neutron_driver: ovs
```

3. 在环境文件中，`/usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovs.yaml`，确保 `NeutronNetworkType` 参数包含 `vxlan` 或 `gre` 而不是 `geneve`。

### 示例

```
parameter_defaults:
  ...
  NeutronNetworkType: 'vxlan'
```

4. 运行 `openstack overcloud deploy` 命令，并包含您修改的核心 heat 模板、环境文件和您修改的文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

```
$ openstack overcloud deploy --templates \
-e <your_environment_files> \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/ \
neutron-ovs.yaml \
-e /home/stack/templates/containers-prepare-parameter.yaml \
```

### 其他资源

- [高级 Overcloud 自定义指南中的环境文件](#)。
- [高级 Overcloud 自定义指南中的创建 overcloud 中包括环境文件](#)

## 2.11. 升级后保留 ML2/OVS，而不是默认的 ML2/OVN

在 Red Hat OpenStack Platform (RHOSP) 16.0 及之后的版本部署中，带有 Open Virtual Network (ML2/OVN) 的 Modular Layer 2 插件是 RHOSP 网络服务的默认机制驱动程序。如果您从使用 ML2/OVS 的较早版本的 RHOSP 升级，您可以在升级后从 ML2/OVN 迁移到 ML2/OVS。

如果在升级后选择继续使用 ML2/OVS，请按照红帽的升级过程操作，且不执行 ML2/OVS-to-ML2/OVN 迁移。

### 其他资源

- [升级 框架\(13 到 16.2\) 指南](#)
- [将网络服务迁移到 ML2 OVN 机制驱动程序](#)

## 2.12. 使用 ML2/OVN 部署自定义角色

在默认的 Red Hat OpenStack (RHOSP)部署中，ML2/OVN 可组合服务在 Controller 节点上运行。您可以选择使用受支持的自定义角色，如以下示例所述。

## Networker

在专用网络器节点上运行 OVN 可组合服务。

### 带有 SR-IOV 的 Networker

使用 SR-IOV 在专用 networker 节点上运行 OVN 可组合服务。

### 使用 SR-IOV 的控制器

在支持 SR-IOV 的控制器节点上运行 OVN 可组合服务。

您还可以生成自己的自定义角色。

## 限制

以下限制适用于本发行版本中将 SR-IOV 与 ML2/OVN 和原生 OVN DHCP 搭配使用。

- 所有外部端口都调度到单个网关节点上，因为所有端口只有一个 HA Chassis 组。
- VLAN 租户网络上的 VF (direct)端口上的北路由无法用于 SR-IOV，因为外部端口没有与逻辑路由器的网关端口在一起。请参阅 <https://bugs.launchpad.net/neutron/+bug/1875852>。

## 先决条件

- 您知道如何部署自定义角色。有关更多信息，请参阅 *高级 OpenStack 自定义指南* 中的 [可组合服务和自定义角色](#)。

## 流程

1. 以 **stack** 用户身份登录 undercloud 主机，再提供 **stackrc** 文件。

```
$ source stackrc
```

2. 选择适合您的部署的自定义角色文件。如果它是您需要，直接在 `deploy` 命令中使用它。或者，您可以生成组合其他自定义角色文件自己的自定义角色文件。

Deployment	角色	角色文件
Networker 角色	Networker	<b>Networker.yaml</b>
带有 SR-IOV 的 Networker 角色	NetworkerSriov	<b>NetworkerSriov.yaml</b>
与 SR-IOV 共存控制和网络器	ControllerSriov	<b>ControllerSriov.yaml</b>

3. [可选] 生成一个新的自定义角色数据文件，该文件将其中一个自定义角色文件与其他自定义角色文件相结合。按照 *高级 OpenStack 自定义指南* 中的 [创建 roles\\_data 文件](#) 中的说明进行操作。根据您的部署包括适当的源角色文件。
4. [可选] 要为角色识别特定节点，您可以创建特定的硬件类别，并将类别分配到特定的节点。然后，使用环境文件为角色定义类别，并指定节点数。有关更多信息，请参阅 *高级 OpenStack 自定义指南* 中的 [创建新角色](#) 中的示例。

5. 根据您的部署创建一个环境文件。

Deployment	环境文件示例
Networker 角色	neutron-ovn-dvr-ha.yaml
带有 SR-IOV 的 Networker 角色	ovn-sriov.yaml

6. 根据您的部署包括以下设置。

Deployment	设置
Networker 角色	<pre> ControllerParameters:   OVNCMSOptions: "" ControllerSriovParameters:   OVNCMSOptions: "" NetworkerParameters:   OVNCMSOptions: "enable-chassis-as-gw" NetworkerSriovParameters:   OVNCMSOptions: "" </pre>
带有 SR-IOV 的 Networker 角色	<pre> OS::TripleO::Services::NeutronDhcpAgent: OS::Heat::None  ControllerParameters:   OVNCMSOptions: "" ControllerSriovParameters:   OVNCMSOptions: "" NetworkerParameters:   OVNCMSOptions: "" NetworkerSriovParameters:   OVNCMSOptions: "enable-chassis-as-gw" </pre>
与 SR-IOV 共存控制和网络器	<pre> OS::TripleO::Services::NeutronDhcpAgent: OS::Heat::None  ControllerParameters:   OVNCMSOptions: "" ControllerSriovParameters:   OVNCMSOptions: "enable-chassis-as-gw" NetworkerParameters:   OVNCMSOptions: "" NetworkerSriovParameters:   OVNCMSOptions: "" </pre>

7. 部署 overcloud。使用 **-e** 选项在部署命令中包含环境文件。在部署命令中包含自定义角色数据文件，以及 **-r** 选项。例如：**-r Networker.yaml** 或 **-r mycustomrolesfile.yaml**。

## 验证步骤 - OVN 部署

1. 以 overcloud SSH 用户身份登录 Controller 或 Networker 节点，默认为 **heat-admin**。

### 示例

```
ssh heat-admin@controller-0
```

2. 确保 **ovn\_metadata\_agent** 在 Controller 和 Networker 节点上运行。

```
$ sudo podman ps | grep ovn_metadata
```

### 输出示例

```
a65125d9588d undercloud-0.ctlplane.localdomain:8787/rh-osbs/rhosp16-openstack-
neutron-metadata-agent-ovn:16.2_20200813.1 kolla_start 23 hours ago Up 21 hours
ago ovn_metadata_agent
```

3. 确保带有 OVN 服务或专用网络器节点的 Controller 节点已配置为 OVS 的网关。

```
$ sudo ovs-vsctl get Open_Vswitch . external_ids:ovn-cms-options
```

### 输出示例

```
...
enable-chassis-as-gw
...
```

## 验证步骤 - SR-IOV 部署

1. 以 overcloud SSH 用户身份登录 Compute 节点，默认为 **heat-admin**。

### 示例

```
ssh heat-admin@compute-0
```

2. 确保 **neutron\_sriov\_agent** 在 Compute 节点上运行。

```
$ sudo podman ps | grep neutron_sriov_agent
```

### 输出示例

```
f54cbbf4523a undercloud-0.ctlplane.localdomain:8787/rh-osbs/rhosp16-openstack-neutron-
sriov-agent:16.2_20200813.1
kolla_start 23 hours ago Up 21 hours ago neutron_sriov_agent
```

3. 确保成功检测到 network-available SR-IOV NIC。

```
$ sudo podman exec -uroot galera-bundle-podman-0 mysql nova -e 'select
hypervisor_hostname,pci_stats from compute_nodes;'
```

### 输出示例

```
computesriov-1.localdomain {... {"dev_type": "type-PF", "physical_network": "datacentre",  
"trusted": "true"}, "count": 1}, ... {"dev_type": "type-VF", "physical_network": "datacentre",  
"trusted": "true", "parent_ifname": "enp7s0f3"}, "count": 5}, ...}  
computesriov-0.localdomain {... {"dev_type": "type-PF", "physical_network": "datacentre",  
"trusted": "true"}, "count": 1}, ... {"dev_type": "type-VF", "physical_network": "datacentre",  
"trusted": "true", "parent_ifname": "enp7s0f3"}, "count": 5}, ...}
```

## 其他资源

- *高级 Overcloud 自定义指南*中的[可组合服务和自定义角色](#)。

## 2.13. 带有 ML2/OVN 和原生 OVN DHCP 的 SR-IOV

您可以部署自定义角色，以便在带有原生 OVN DHCP 的 ML2/OVN 部署中使用 SR-IOV。请参阅[第 2.12 节“使用 ML2/OVN 部署自定义角色”](#)。

### 限制

以下限制适用于本发行版本中将 SR-IOV 与 ML2/OVN 和原生 OVN DHCP 搭配使用。

- 所有外部端口都调度到单个网关节点上，因为所有端口只有一个 HA Chassis 组。
- VLAN 租户网络上的 VF (direct) 端口上的北路由无法用于 SR-IOV，因为外部端口没有与逻辑路由器的网关端口在一起。请参阅 <https://bugs.launchpad.net/neutron/+bug/1875852>。

## 其他资源

- *高级 Overcloud 自定义指南*中的[可组合服务和自定义角色](#)。

## 第 3 章 管理项目网络

项目网络可帮助您隔离云计算的网络流量。创建项目网络的步骤包括规划和创建网络，以及添加子网和路由器。

### 3.1. VLAN 计划

在规划 Red Hat OpenStack Platform 部署时，您可以从多个子网开始，您可以从中分配单个 IP 地址。当使用多个子网时，您可以将系统间的流量隔离到 VLAN。

例如，您的管理或 API 流量与服务 Web 流量的系统不同。VLAN 间的流量通过路由器传输，您可以在其中实施防火墙来管理流量流。

您必须规划您的 VLAN 作为整个计划的一部分，其中包括部署中各种类型虚拟网络资源的流量隔离、高可用性和 IP 地址利用率。

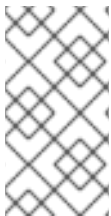


#### 注意

单一网络中的最大 VLAN 数量，或者在一个 OVS 代理中，网络节点的最大 VLAN 数量为 4094。如果您需要超过最大 VLAN 数量，您可以创建多个提供商网络(VXLAN 网络)和多个网络节点，每个网络对应一个。每个节点最多可包含 4094 个专用网络。

### 3.2. 网络流量类型

您可以为您要托管的不同类型的网络流量分配单独的 VLAN。例如，您可以为每个类型的网络都有单独的 VLAN。只有外部网络必须路由到外部物理网络。在本发行版本中，director 提供 DHCP 服务。



#### 注意

您不需要本节中的所有隔离的 VLAN 以用于每个 OpenStack 部署。例如，如果您的云用户不需要根据需要创建临时虚拟网络，您可能不需要项目网络。如果您希望每个虚拟机直接连接到与任何其他物理系统相同的交换机，请将您的 Compute 节点直接连接到提供商网络，并将您的实例配置为使用该提供商网络。

- **置备网络** - 此 VLAN 专用于通过 PXE 引导使用 director 部署新节点。OpenStack Orchestration (heat)将 OpenStack 安装到 overcloud 裸机服务器上。这些服务器附加到物理网络，以便从 undercloud 基础架构接收平台安装镜像。
- **内部 API 网络** - OpenStack 服务使用内部 API 网络进行通信，包括 API 通信、RPC 消息和数据库通信。此外，此网络用于控制器节点之间的操作消息。在规划 IP 地址分配时，请注意每个 API 服务都需要自己的 IP 地址。特别是，您必须为以下每个服务规划 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

使用高可用性时，Pacemaker 会在物理节点之间移动 VIP 地址。

- **存储** - 块存储、NFS、iSCSI 和其他存储服务。出于性能的原因，将此网络隔离为单独的物理以太网链接。
- **存储管理** - OpenStack Object Storage (swift) 使用此网络在参与副本节点之间同步数据对象。代理服务充当用户请求和底层存储层之间的中间接口。代理接收传入的请求，并找到所需的副本来检索请求的数据。使用 Ceph 后端的服务通过存储管理网络连接，因为它们不直接与 Ceph 交互，而是使用前端服务。请注意，RBD 驱动程序是一个例外；此流量直接连接到 Ceph。
- **项目网络** - Neutron 使用 VLAN 隔离（每个项目网络是网络 VLAN）或利用 VXLAN 或 GRE 进行隧道，为各个项目提供自己的网络。网络流量在每个项目网络中隔离。每个项目网络都有一个与其关联的 IP 子网，多个项目网络可能会使用相同的地址。
- **External** - 外部网络托管公共 API 端点和到 Dashboard (horizon) 的连接。您还可以将此网络用于 SNAT。在生产部署中，通常将单独的网络用于浮动 IP 地址和 NAT。
- **提供商网络** - 使用提供商网络将实例附加到现有网络基础架构。您可以使用提供商网络直接映射到数据中心中的现有物理网络，使用扁平网络或 VLAN 标签。这允许实例与 OpenStack 网络基础架构外部的系统共享相同的第 2 层网络。

## 3.3. IP 地址消耗

以下系统使用您分配的范围内的 IP 地址：

- **物理节点** - 每个物理 NIC 都需要一个 IP 地址。将物理 NIC 专用于特定功能是常见的做法。例如，将管理和 NFS 流量分配给不同的物理 NIC，有时有多个 NIC 连接到不同的交换机，以实现冗余目的。



- **用于高可用性的虚拟 IP (VIP)** - 计划为每个控制器节点共享的每个网络在一个或多个 VIP 之间分配。

### 3.4. 虚拟网络

以下虚拟资源消耗 OpenStack 网络中的 IP 地址。这些资源被视为云基础架构的本地，不需要外部物理网络中的系统访问这些资源：

- **项目网络** - 每个项目网络需要一个子网，它可用于为实例分配 IP 地址。
- **虚拟路由器** - 每个路由器接口插入到子网都需要一个 IP 地址。如果要使用 DHCP，每个路由器接口需要两个 IP 地址。
- **实例** - 每个实例需要一个托管实例的项目子网的地址。如果需要入口流量，则必须从指定的外部网络为实例分配一个浮动 IP 地址。
- **管理流量** - 包括 OpenStack 服务和 API 流量。所有服务共享少量 VIP。API、RPC 和数据库服务在内部 API VIP 上进行通信。

### 3.5. 添加网络路由

要允许流量被路由到新网络或从新网络路由，您必须将其子网作为接口添加到现有的虚拟路由器：

1. 在控制面板中，选择 **Project > Network > Routers**
2. 在 **Routers** 列表中选择您的虚拟路由器名称，然后单击 **Add Interface**。  
在子网列表中，选择您的新子网的名称。您可以选择在此字段中为接口指定 IP 地址。
3. 点 **Add Interface**。  
您的网络中的实例现在可以与子网外的系统通信。

### 3.6. 网络计划示例

本例展示了多个容纳多个子网的网络，每个子网被分配一个 IP 地址范围：

表 3.1. 子网计划示例

子网名称	地址范围	地址数	子网掩码
Provisioning 网络	192.168.100.1 - 192.168.100.250	250	255.255.255.0
内部 API 网络	172.16.1.10 - 172.16.1.250	241	255.255.255.0
存储	172.16.2.10 - 172.16.2.250	241	255.255.255.0
存储管理	172.16.3.10 - 172.16.3.250	241	255.255.255.0
租户网络(GRE/VXLAN)	172.16.4.10 - 172.16.4.250	241	255.255.255.0
外部网络( incl. 浮动 IP)	10.1.2.10 - 10.1.3.222	469	255.255.254.0

子网名称	地址范围	地址数	子网掩码
提供商网络 (infrastructure)	10.10.3.10 - 10.10.3.250	241	255.255.252.0

### 3.7. 创建网络

创建一个网络，以便您的实例可以相互通信，并使用 DHCP 接收 IP 地址。有关外部网络连接的更多信息，请参阅 [桥接物理网络](#)。

在创建网络时，务必要知道网络可以托管多个子网。如果您打算在同一网络中托管不同的系统，并且首选在它们之间进行隔离，这非常有用。例如，您可以指定只有一个子网中只有 webserver 流量，而数据库流量则遍历另一个子网。子网相互隔离，希望与其他子网通信的实例都必须由路由器指示其流量。考虑将需要大量流量的系统放在同一子网中，以便它们不需要路由，并避免后续的延迟和负载。

1. 在控制面板中，选择 **Project > Network > Networks**
2. 点 **+Create Network** 并指定以下值：

字段	描述
网络名称	描述性名称，基于网络要执行的角色。如果要将网络与外部 VLAN 集成，请考虑将 VLAN ID 号附加到名称中。例如， <b>webservers_122</b> ，如果在这个子网中托管 HTTP Web 服务器，您的 VLAN 标签为 <b>122</b> 。或者，如果您打算 <b>将网络流量保持私有，而不将网络与外部网络集成</b> ，则可以 <b>使用内部</b> 内部流量。
Admin State	控制网络是否立即可用。使用此字段以 Down 状态创建网络，其中逻辑存在但不活跃。如果您不打算立即将网络输入到生产中，这非常有用。
创建子网	决定是否创建子网。例如，如果您打算将这个网络保留为没有网络连接的占位符，您可能不希望创建子网。

3. 点 **Next** 按钮，然后在 **Subnet** 选项卡中指定以下值：

字段	描述
子网名称	为子网输入一个描述性名称。
网络地址	以 CIDR 格式输入地址，其中包含一个值中的 IP 地址范围和子网掩码。要确定地址，计算子网掩码中屏蔽的位数，并将该值附加到 IP 地址范围中。例如，子网掩码 255.255.255.0 具有 24 个屏蔽的位。要将这个掩码与 IPv4 地址范围 192.168.122.0 搭配使用，请指定地址 192.168.122.0/24。

字段	描述
IP 版本	指定互联网协议版本，其中有效类型为 IPv4 或 IPv6。Network Address 字段中的 IP 地址范围必须与您选择的版本匹配。
网关 IP	默认网关的路由器接口的 IP 地址。此地址是路由目标为外部位置的任何流量的下一跳，且必须在 Network Address 字段中指定的范围内。例如，如果您的 CIDR 网络地址是 192.168.122.0/24，则您的默认网关可能是 192.168.122.1。
禁用网关	禁用转发并隔离子网。

#### 4. 点 Next 指定 DHCP 选项：

- **启用 DHCP** - 为此子网启用 DHCP 服务。您可以使用 DHCP 自动将 IP 设置分发到您的实例。
- **IPv6 地址** - 配置模式.如果创建 IPv6 网络，您必须指定如何分配 IPv6 地址和其他信息：
  - **no Options specified** - 如果您要手动设置 IP 地址，或者使用非 OpenStack 感知方法进行地址分配，请选择这个选项。
  - **ACTIVE (Stateless Address Autoconfiguration)** - 实例根据从 OpenStack 网络路由器发送的路由器广告(RA)消息生成 IPv6 地址。使用此配置创建 OpenStack 网络子网，并将 ra\_mode 设置为 slaac，并将 address\_mode 设置为 slaac。
  - **DHCPv6 有状态** - 实例从 OpenStack 网络 DHCPv6 服务接收 IPv6 地址以及附加选项（如 DNS）。使用此配置创建将 ra\_mode 设置为 dhcpv6-stateful 的子网，并将 address\_mode 设置为 dhcpv6-stateful。
  - **DHCPv6 无状态** - 实例根据从 OpenStack 网络路由器发送的路由器广告(RA)消息生成 IPv6 地址。从 OpenStack Networking DHCPv6 服务分配其他选项（如 DNS）。使用此配置创建将 ra\_mode 设置为 dhcpv6-stateless 的子网，并将 address\_mode 设置为 dhcpv6-stateless。
- **分配池** - 您希望 DHCP 分配的 IP 地址范围。例如，value 192.168.22.100,192.168.22.150 将该范围内的所有地址视为可用于分配。
- **DNS 名称服务器** - 网络中可用的 DNS 服务器的 IP 地址。DHCP 将这些地址分发到实例，以进行名称解析。



#### 重要

对于 DNS 等战略性服务，最好不要在云中托管这些服务。例如，如果您的云托管 DNS 和云变得不可操作，DNS 不可用，云组件无法相互查找。

- **主机路由** - 静态主机路由。首先，以 CIDR 格式指定目的地网络，后跟要用于路由的下一跳（例如 192.168.23.0/24, 10.1.31.1）。如果需要向实例分发静态路由，请提供这个值。

#### 5. 点 Create。

您可以在 **Networks** 选项卡中查看完整的网络。您还可以根据需要点 **Edit** 更改任何选项。在创建实例时，您可以将它们配置为使用其子网，并接收任何指定的 DHCP 选项。

### 3.8. 使用子网

使用子网为实例授予网络连接。每个实例作为实例创建过程的一部分被分配到子网，因此务必要考虑实例正确放置最适合其连接要求。

您只能在预先存在的网络中创建子网。请记住，OpenStack 网络中的项目网络可以托管多个子网。如果您打算在同一网络中托管不同的系统，并且首选在它们之间进行隔离，这非常有用。

例如，您可以指定只有一个子网中只有 webserver 流量，而数据库流量则遍历另一个子网。

子网相互隔离，希望与其他子网通信的实例都必须由路由器指示其流量。因此，您可以通过对同一子网中的系统分组需要相互间高的流量，从而减少网络延迟和负载。

### 3.9. 创建子网

要创建子网，请按照以下步骤执行：

1. 在控制面板中，选择 **Project > Network > Networks**，然后在 **Networks** 视图中点击您的网络名称。
2. 点 **Create Subnet**，并指定以下值：

字段	描述
子网名称	描述性子网名称。
网络地址	CIDR 格式的地址，它包含一个值中的 IP 地址范围和子网掩码。要确定 CIDR 地址，计算子网掩码中屏蔽的位数，并将该值附加到 IP 地址范围中。例如，子网掩码 255.255.255.0 具有 24 个屏蔽的位。要将这个掩码与 IPv4 地址范围 192.168.122.0 搭配使用，请指定地址 192.168.122.0/24。
IP 版本	互联网协议版本，其中有效类型为 IPv4 或 IPv6。Network Address 字段中的 IP 地址范围必须与您选择的协议版本匹配。
网关 IP	默认网关的路由器接口的 IP 地址。此地址是路由目标为外部位置的任何流量的下一跳，且必须在 Network Address 字段中指定的范围内。例如，如果您的 CIDR 网络地址是 192.168.122.0/24，则您的默认网关可能是 192.168.122.1。
禁用网关	禁用转发并隔离子网。

3. 点 **Next** 指定 DHCP 选项：

- **启用 DHCP** - 为此子网启用 DHCP 服务。您可以使用 DHCP 自动将 IP 设置分发到您的实例。
- **IPv6 地址** - 配置模式。如果创建 IPv6 网络，您必须指定如何分配 IPv6 地址和其他信息：

- **no Options specified** - 如果您要手动设置 IP 地址，或者使用非 OpenStack 感知方法进行地址分配，请选择这个选项。
  - **ACTIVE (Stateless Address Autoconfiguration)** - 实例根据从 OpenStack 网络路由器发送的路由器广告(RA)消息生成 IPv6 地址。使用此配置创建 OpenStack 网络子网，并将 ra\_mode 设置为 slaac，并将 address\_mode 设置为 slaac。
  - **DHCPv6 有状态** - 实例从 OpenStack 网络 DHCPv6 服务接收 IPv6 地址以及附加选项（如 DNS）。使用此配置创建将 ra\_mode 设置为 dhcpv6-stateful 的子网，并将 address\_mode 设置为 dhcpv6-stateful。
  - **DHCPv6 无状态** - 实例根据从 OpenStack 网络路由器发送的路由器广告(RA)消息生成 IPv6 地址。从 OpenStack Networking DHCPv6 服务分配其他选项（如 DNS）。使用此配置创建将 ra\_mode 设置为 dhcpv6-stateless 的子网，并将 address\_mode 设置为 dhcpv6-stateless。
- **分配池** - 您希望 DHCP 分配的 IP 地址范围。例如，value 192.168.22.100,192.168.22.150 将该范围内的所有地址视为可用于分配。
  - **DNS 名称服务器** - 网络中可用的 DNS 服务器的 IP 地址。DHCP 将这些地址分发到实例，以进行名称解析。
  - **主机路由** - 静态主机路由。首先，以 CIDR 格式指定目的地网络，后跟要用于路由的下一跳（例如 192.168.23.0/24, 10.1.31.1）。如果需要向实例分发静态路由，请提供这个值。

#### 4. 点 **Create**。

您可以在 **Subnets** 列表中查看子网。您还可以根据需要在 **Edit** 更改任何选项。在创建实例时，您可以将它们配置为使用其子网，并接收任何指定的 DHCP 选项。

### 3.10. 添加路由器

OpenStack 网络使用基于 SDN 的虚拟路由器提供路由服务。路由器是实例与外部子网通信的要求，包括在物理网络中。路由器和子网使用接口进行连接，每个子网需要自己的接口到路由器。

路由器的默认网关定义了路由器收到的任何流量的下一跳。它的网络通常配置为使用虚拟网桥将流量路由到外部物理网络。

要创建路由器，请完成以下步骤：

1. 在控制面板中，选择 **Project > Network > Routers**，然后点 **Create Router**。
2. 为新路由器输入一个描述性名称，然后单击 **Create router**。
3. 单击 **Routers** 列表中新路由器条目旁边的 **Set Gateway**。
4. 在 **External Network** 列表中，指定要接收目标用于外部位置的的网络。
5. 单击 **Set Gateway**。

添加路由器后，您必须配置您创建的任何子网，以使用此路由器发送流量。您可以通过在子网和路由器之间创建接口来实现此目的。

## 重要

子网的默认路由不能被覆盖。删除子网的默认路由时，L3 代理也会自动删除路由器命名空间中的对应路由，网络流量也无法从关联的子网流向和移除。如果删除了现有路由器命名空间路由，以解决此问题，请执行以下步骤：

1. 解除子网上的所有浮动 IP。
2. 将路由器从子网分离。
3. 将路由器重新附加到子网。
4. 重新附加所有浮动 IP。

### 3.11. 清除所有资源并删除项目

使用 `openstack project purge` 命令删除属于特定项目的所有资源，以及删除项目。

例如，要清除 `test-project` 项目的资源，然后删除项目，请运行以下命令：

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

# openstack project purge --project 02e501908c5b438dbc73536c10c9aac0
```

### 3.12. 删除路由器

如果没有连接的接口，您可以删除路由器。

要删除其接口并删除路由器，请完成以下步骤：

1. 在控制面板中，选择 **Project > Network > Routers**，然后点击您要删除的路由器的名称。
2. 选择 **内部接口类型的接口**，然后单击 **Delete Interfaces**。
3. 从 Routers 列表中，选择目标路由器，再点 **Delete Routers**。

### 3.13. 删除子网

如果不再使用子网，可以删除它。但是，如果任何实例仍配置为使用子网，删除尝试会失败，仪表板会显示错误消息。

完成以下步骤以删除网络中的特定子网：

1. 在控制面板中，选择 **Project > Network > Networks**
2. 点网络的名称。

3. 选择目标子网，然后单击 **Delete Subnets**。

## 3.14. 删除网络

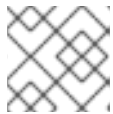
在一些情况下，需要删除之前创建的网络，可能作为内务处理或作为停用过程的一部分。您必须首先删除或分离任何网络仍在使用的接口，然后才能成功删除网络。

要删除项目中的网络以及任何依赖接口，请完成以下步骤：

1. 在控制面板中，选择 **Project > Network > Networks** 删除与目标网络子网关联的所有路由器接口。

要删除接口，请通过点击**网络列表**中的目标网络找到您要删除的网络的 ID 号，查看 ID 字段。与网络关联的所有子网在 **Network ID** 字段中共享这个值。

2. 导航到 **Project > Network > Routers** 在 **Routers** 列表中点虚拟路由器的名称，并找到附加到您要删除的子网的接口。  
您可以通过作为网关 IP 的 IP 地址将这个子网与其他子网区分开。您可以通过确保接口的网络 ID 与您在上一步中记录的 ID 匹配，从而进一步验证区别。
3. 点您要删除的接口的 **Delete Interface** 按钮。
4. 选择 **Project > Network > Networks** 然后点击您的网络名称。
5. 单击您要删除的子网的 **Delete subnet** 按钮。



### 注意

如果您仍然无法在此时删除子网，请确保还没有被任何实例使用。

6. 选择 **Project > Network > Networks** 然后选择您要删除的网络。
7. 单击 **Delete Networks**。

## 第 4 章 将虚拟机实例连接到物理网络

您可以使用扁平 VLAN 提供商网络将虚拟机实例直接连接到外部网络。

### 4.1. OPENSTACK 网络拓扑概述

OpenStack Networking (neutron)有两个类别的服务分布在多个节点类型中。

- **Neutron 服务器** - 此服务运行 OpenStack Networking API 服务器，为最终用户和服务提供 API，以便与 OpenStack 网络交互。此服务器也与底层数据库集成，以存储和检索项目网络、路由器和负载均衡器的详细信息。
- **Neutron 代理** - 这些服务是为 OpenStack 网络执行网络功能：
  - **neutron-dhcp-agent** - 管理项目专用网络的 DHCP IP 地址。
  - **neutron-l3-agent** - 在项目专用网络、外部网络等之间执行第 3 层路由。
- **Compute 节点** - 此节点托管运行虚拟机的虚拟机监控程序，也称为实例。Compute 节点必须直接连接到网络，以便为实例提供外部连接。此节点通常是 I2 代理运行的位置，如 **neutron-openvswitch-agent**。

#### 其他资源

- [第 4.2 节 “OpenStack 网络服务的放置”](#)

### 4.2. OPENSTACK 网络服务的放置

OpenStack 网络服务可以在同一个物理服务器或单独的专用服务器上运行，这些服务器根据其角色命名：

- *Controller 节点* - 运行 API 服务的服务器。
- *网络节点* - 运行 OpenStack 网络代理的服务器。
- *Compute 节点* - 托管实例的虚拟机监控程序服务器。

本章中的步骤适用于包含这三个节点类型的环境。如果您的部署在同一物理节点上同时具有 Controller 和网络节点角色，则必须从该服务器的两个部分执行这些步骤。这也适用于一个高可用性(HA)环境，所有这三个节点都可能运行 Controller 节点和网络节点服务(HA)。因此，您必须完成所有三个节点上适用于 Controller 和网络节点的小节中的步骤。

#### 其他资源

- [第 4.1 节 “OpenStack 网络拓扑概述”](#)

### 4.3. 配置扁平提供商网络

您可以使用扁平提供商网络将实例直接连接到外部网络。如果您有多个物理网络和单独的物理接口，并且希望将每个 Compute 和网络节点连接到这些外部网络，这将非常有用。

#### 先决条件

- 您有多个物理网络。  
本例分别使用名为 **physnet1** 的物理网络，以及 **physnet2**。



- 您有单独的物理接口。  
这个示例分别使用单独的物理接口 **eth0** 和 **eth1**。

## 流程

1. 在 undercloud 主机上，以 stack 用户身份登录，创建自定义 YAML 环境文件。

### 示例

```
$ vi /home/stack/templates/my-modules-environment.yaml
```

### 提示

Red Hat OpenStack Platform 编排服务(heat)使用一组名为 *template* 的计划来安装和配置您的环境。您可以使用自定义环境文件来自定义 overcloud 的各个方面，这是为 workflow 模板 提供自定义的特殊模板。

2. 在 **parameter\_defaults** 下的 YAML 环境文件中，使用 **NeutronBridgeMappings** 指定用于访问外部网络的 OVS 网桥。

### 示例

```
parameter_defaults:
  NeutronBridgeMappings: 'physnet1:br-net1,physnet2:br-net2'
```

3. 在 Controller 和 Compute 节点的自定义 NIC 配置模板中，使用附加的接口配置网桥。

### 示例

```
...
- type: ovs_bridge
  name: br-net1
  mtu: 1500
  use_dhcp: false
  members:
  - type: interface
    name: eth0
    mtu: 1500
    use_dhcp: false
    primary: true
- type: ovs_bridge
  name: br-net2
  mtu: 1500
  use_dhcp: false
  members:
  - type: interface
    name: eth1
    mtu: 1500
    use_dhcp: false
    primary: true
...
```

- 运行 **openstack overcloud deploy** 命令，并包含模板和环境文件，包括此修改后的自定义 NIC 模板和新环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

### 示例

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-neutron-
environment.yaml
```

### 验证

- 创建一个外部网络(**public1**)作为扁平网络，并将它与配置的物理网络(**physnet1**)关联。将它配置为共享网络（使用 **--share**），使其他用户创建直接连接到外部网络的虚拟机实例。

### 示例

```
# openstack network create --share --provider-network-type flat --provider-physical-network
physnet1 --external public01
```

- 使用 **openstack subnet create** 命令创建子网(**public\_subnet**)。

### 示例

```
# openstack subnet create --no-dhcp --allocation-pool
start=192.168.100.20,end=192.168.100.100 --gateway 192.168.100.1 --network public01
public_subnet
```

- 创建虚拟机实例，并将其直接连接到新创建的外部网络。

### 示例

```
$ openstack server create --image rhel --flavor my_flavor --network public01 my_instance
```

### 其他资源

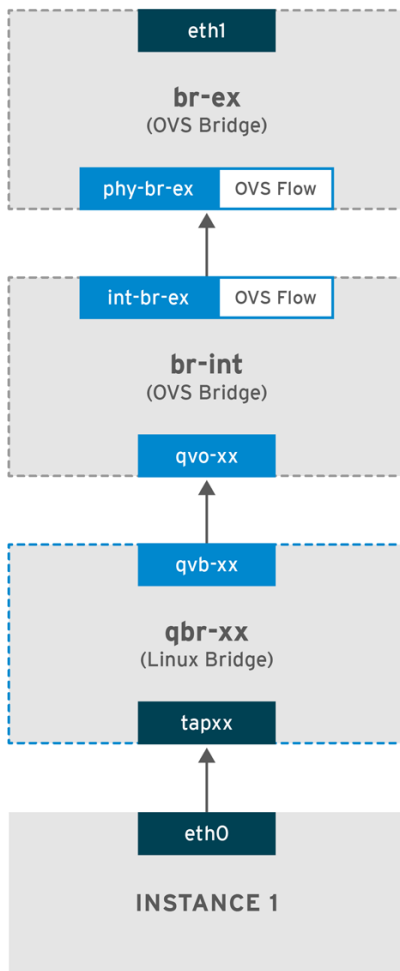
- 高级 *Overcloud 自定义指南* 中的 [自定义网络接口模板](#)。
- 高级 *Overcloud 自定义指南* 中的 [环境文件](#)。
- 高级 *Overcloud 自定义指南* 中的 [创建 overcloud 中包括环境文件](#)
- 命令行界面参考中的 [network create](#)
- Command Line Interface Reference* 中的 [subnet create](#)
- Command Line Interface Reference* 中的 [server create](#)

## 4.4. 扁平提供商网络数据包流的工作方式？

本节论述了如何将流量流和来自具有扁平供应商网络配置的实例的流量。

### 扁平提供商网络中传出流量流

下图描述了离开实例的流量的数据包流，并直接指向外部网络。配置 **br-ex** 外部网桥后，将物理接口添加到网桥，并将实例生成至 Compute 节点，生成的接口和网桥配置类似于下图中的配置（如果使用 `iptables_hybrid` 防火墙驱动程序）：



OPENSTACK\_450456\_0617

1. 数据包离开实例的 **eth0** 接口，并到达 linux 网桥 **qbr-xx**。
2. 网桥 **qbr-xx** 连接到 **br-int**，使用 veth 对 **qvb-xx <-> qvo-xxx**。这是因为网桥用于应用安全组定义的入站/出站防火墙规则。
3. 接口 **qvb-xx** 连接到 **qbr-xx** linux 网桥，**qvoxx** 连接到 **br-int** Open vSwitch (OVS) 网桥。

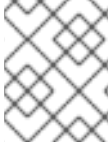
'qbr-xx'Linux 网桥配置示例：

```
# brctl show
qbr269d4d73-e7 8000.061943266ebb no qvb269d4d73-e7
tap269d4d73-e7
```

br-int 上的 qvo-xx 配置：

```
# ovs-vsctl show
```

```
Bridge br-int
  fail_mode: secure
  Interface "qvo63599ba-8f"
  Port "qvo269d4d73-e7"
    tag: 5
    Interface "qvo269d4d73-e7"
```



### 注意

端口 **qvo-xx** 使用与扁平提供商网络关联的内部 VLAN 标签标记。在本例中，VLAN 标签为 **5**。当数据包到达 **qvo-xx** 时，VLAN 标签将附加到数据包标头中。

数据包然后被移到 **br-ex** OVS 网桥，使用 patch-peer **int-br-ex <-> phy-br-ex**。

**br-int** 上的 patch-peer 配置示例：

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

**br-ex** 上的 patch-peer 配置示例：

```
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
```

当此数据包在 **br-ex** 上到达 **phy-br-ex** 时，**br-ex** 中的 OVS 流会剥离 VLAN 标签(5)并将其转发到物理接口。

在以下示例中，输出显示 **phy-br-ex** 的端口号为 **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
```

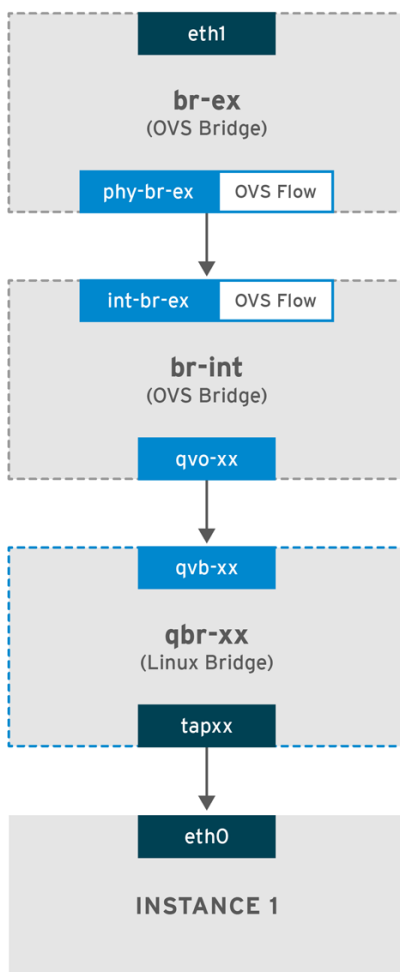
以下输出显示了到达 **phy-br-ex (in\_port=2)** 的任何数据包，其 VLAN 标签为 **5 (dl\_vlan=5)**。此外，**br-ex** 中的 OVS 流可剥离 VLAN 标签，并将数据包转发到物理接口。

```
# 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
```

如果物理接口是另一个 VLAN 标记的接口，则物理接口会将标签添加到数据包中。

## 扁平提供商网络中传入流量流

本节包含来自外部网络传入流量流的信息，直到它到达实例的接口。



OPENSTACK\_450456\_0617

1. 传入流量到达物理节点上的 **eth1**。
2. 数据包传递到 **br-ex** 网桥。
3. 数据包通过 patch-peer **phy-br-ex <--> int-br-ex** 移到 **br-int**。

在以下示例中，**int-br-ex** 使用端口号 **15**。请参阅包含 **15 (int-br-ex)** 的条目：

```
# 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 上的流量流

1. 当数据包到达 **int-br-ex** 时，**br-int** 网桥内的 OVS 流规则会导致数据包添加内部 VLAN 标签 **5**。请参阅 `operations =mod_vlan_vid:5` 条目：

```
# 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. 第二条规则管理到达 `int-br-ex` (`in_port=15`)，没有 VLAN 标签(`vlan_tci=0x0000`)的数据包)：该规则将 VLAN 标签 5 添加到数据包(`action=mod_vlan_vid:5,`)，并将其转发到 **HQxxx**。
3. 在剥离 VLAN 标签后，**qvovxxx** 接受数据包并将其转发到 **qvbxx**。
4. 数据包然后到达实例。



### 注意

VLAN 标签 5 是一个在带有扁平提供商网络的测试 Compute 节点上使用的示例 VLAN；该值由 **neutron-openvswitch-agent** 自动分配。对于您自己的扁平提供商网络，这个值可能会有所不同，并且可能因两个单独的 Compute 节点上的同一网络而异。

### 其他资源

- [第 4.5 节 “对扁平提供商网络上的实例物理网络连接进行故障排除”](#)

## 4.5. 对扁平提供商网络上的实例物理网络连接进行故障排除

"扁平提供商网络数据包流工作方式中提供的输出提供了足够的调试信息，以对扁平提供商网络进行故障排除。以下步骤包含有关故障排除过程的更多信息。

### 流程

1. 查看 **bridge\_mappings**。  
验证您使用的物理网络名称是否与 **bridge\_mapping** 配置的内容一致。

### 示例

在本例中，物理网络名称为 **physnet1**。

```
$ openstack network show provider-flat
```

### 输出示例

```
...
| provider:physical_network | physnet1
...
```

### 示例

在本例中，**bridge\_mapping** 配置的内容也是 **physnet1**：

```
$ grep bridge_mapping /etc/neutron/plugins/ml2/openvswitch_agent.ini
```

### 输出示例

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

2. 检查网络配置。  
确认网络已创建为**外部**，并使用**扁平**类型：

### 示例

在本例中，查询有关网络 **provider-flat** 的详情：

```
$ openstack network show provider-flat
```

### 输出示例

```
...
| provider:network_type | flat |
| router:external      | True |
...
```

3. 检查 patch-peer。  
验证 **br-int** 和 **br-ex** 是否使用 patch-peer **int-br-ex <--> phy-br-ex** 连接。

```
$ ovs-vsctl show
```

### 输出示例

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

### 输出示例

在 **br-ex** 上配置 patch-peer：

```

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

```

如果 **bridge\_mapping** 在 `/etc/neutron/plugins/ml2/openvswitch_agent.ini` 中正确配置，这个连接会在重启 **neutron-openvswitch-agent** 服务时被创建。

如果在重启该服务后没有创建连接，请重新检查 **bridge\_mapping** 设置。

#### 4. 检查网络流。

运行 **ovs-ofctl dump-flows br-ex** 和 **ovs-ofctl dump-flows br-int**，并检查流是否删除传出数据包的内部 VLAN ID，并为传入的数据包添加 VLAN ID。当您在特定 Compute 节点上向这个网络生成实例时，首先添加此流。

- a. 如果在生成实例后没有创建此流，请验证网络是否创建为 **flat**，为 **external**，且 **physical\_network** 名称是正确的。此外，查看 **bridge\_mapping** 设置。
- b. 最后，检查 **ifcfg-br-ex** 和 **ifcfg-ethx** 配置。确保 **ethX** 被添加到 **br-ex** 中的端口，并且 **ifcfg-br-ex** 和 **ifcfg-ethx** 在 **ip a** 的输出中有一个 **UP** 标志。

#### 输出示例

以下输出显示 **eth1** 是 **br-ex** 中的端口：

```

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

```

#### 示例

以下示例演示了 **eth1** 配置为 OVS 端口，并且内核知道从接口传输所有数据包并将其发送到 OVS 网桥 **br-ex**。这可以在条目 (**master ovs-system**) 中观察到。

```

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

```

#### 其他资源

- [第 4.4 节 “扁平提供商网络数据包流的工作方式？”](#)
- [配置网桥映射](#)

## 4.6. 配置 VLAN 提供商网络



当您将在单个 NIC 上的多个 VLAN 标记接口连接到多个提供商网络时，这些新的 VLAN 提供商网络可将虚拟机实例直接连接到外部网络。

### 先决条件

- 您有一个物理网络，其范围为 VLAN。  
本例使用名为 **physnet1** 的物理网络，其范围为 VLAN，**171-172**。
- 您的网络和 Compute 节点使用物理接口连接到物理网络。  
本例使用网络节点和 Compute 节点，它们连接到物理网络 **physnet1**，使用物理接口 **eth1**。
- 这些接口连接到的交换机端口必须配置为中继所需的 VLAN 范围。

### 流程

1. 在 undercloud 主机上，以 stack 用户身份登录，创建自定义 YAML 环境文件。

#### 示例

```
$ vi /home/stack/templates/my-modules-environment.yaml
```

#### 提示

Red Hat OpenStack Platform 编排服务(heat)使用一组名为 *template* 的计划来安装和配置您的环境。您可以使用自定义环境文件来自定义 overcloud 的各个方面，这是为 workflow 模板 提供自定义的特殊模板。

2. 在 **parameter\_defaults** 下的 YAML 环境文件中，使用 **NeutronTypeDrivers** 指定您的网络类型驱动程序。

#### 示例

```
parameter_defaults:
  NeutronTypeDrivers: vxlan,flat,vlan
```

3. 配置 **NeutronNetworkVLANRanges** 设置，以反映正在使用的物理网络和 VLAN 范围：

#### 示例

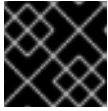
```
parameter_defaults:
  NeutronTypeDrivers: 'vxlan,flat,vlan'
  NeutronNetworkVLANRanges: 'physnet1:171:172'
```

4. 创建外部网络网桥(*br-ex*)，并将端口(*eth1*)与其关联。  
这个示例将 *eth1* 配置为使用 *br-ex*：

#### 示例

```
parameter_defaults:
  NeutronTypeDrivers: 'vxlan,flat,vlan'
  NeutronNetworkVLANRanges: 'physnet1:171:172'
  NeutronBridgeMappings: 'datacentre:br-ex,tenant:br-int'
```

- 运行 **openstack overcloud deploy** 命令，并包含核心模板和环境文件，包括此新环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

### 示例

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-neutron-
environment.yaml
```

### 验证

- 创建外部网络作为类型 **vlan**，并将它们与配置的 **physical\_network** 关联。  
运行以下命令创建两个网络：一个用于 VLAN 171，另一个用于 VLAN 172：

### 示例

```
$ openstack network create \
--provider-network-type vlan \
--provider-physical-network physnet1 \
--provider-segment 171 \
provider-vlan171
```

```
$ openstack network create \
--provider-network-type vlan \
--provider-physical-network physnet1 \
--provider-segment 172 \
provider-vlan172
```

- 创建多个子网，并将其配置为使用外部网络。  
您可以使用 **openstack subnet create** 或控制面板来创建这些子网。确保从网络管理员收到的外部子网详情正确与每个 VLAN 关联。

在本例中，VLAN 171 使用子网 **10.65.217.0/24**，VLAN 172 使用 **10.65.218.0/24**：

### 示例

```
$ openstack subnet create \
--network provider-vlan171 \
--subnet-range 10.65.217.0/24 \
--dhcp \
--gateway 10.65.217.254 \
subnet-provider-171
```

```
$ openstack subnet create \
--network provider-vlan172 \
--subnet-range 10.65.218.0/24 \
--dhcp \
--gateway 10.65.218.254 \
subnet-provider-172
```

## 其他资源

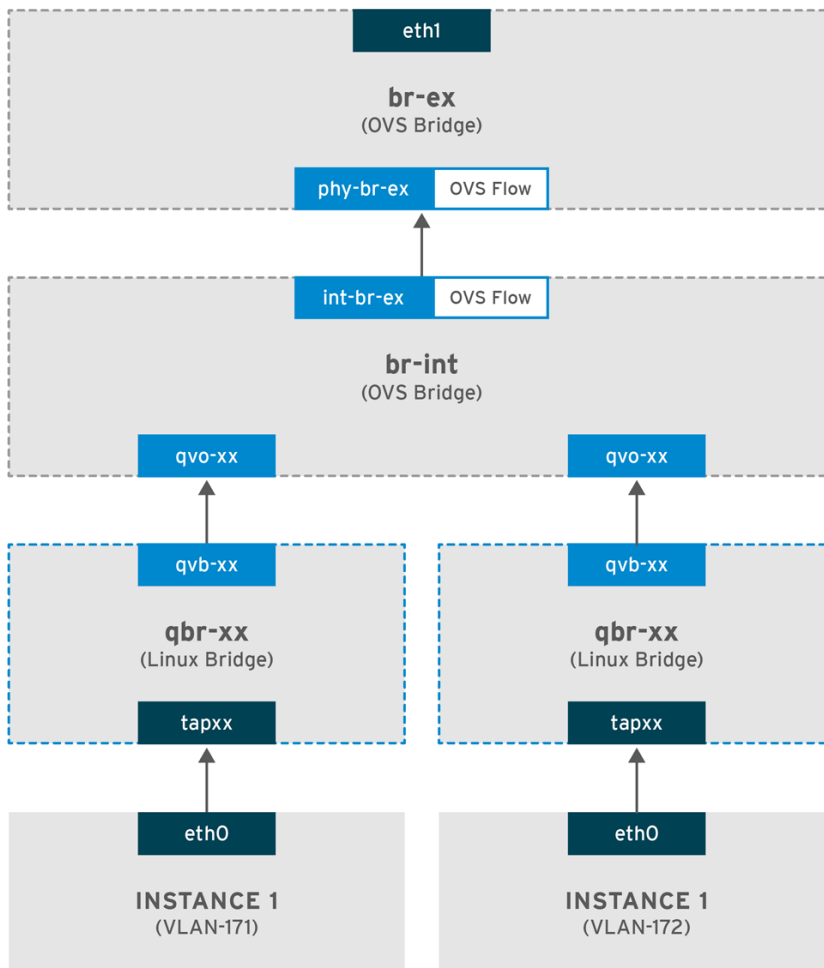
- [高级 OpenStack 自定义指南中的自定义网络接口模板。](#)
- [高级 OpenStack 自定义指南中的环境文件。](#)
- [高级 OpenStack 自定义指南中的创建 overcloud 中包括环境文件](#)
- [命令行界面参考中的 network create](#)
- [Command Line Interface Reference 中的 subnet create](#)

## 4.7. VLAN 提供商网络数据包流的工作方式？

本节论述了如何将流量流和来自 VLAN 供应商网络配置的实例的流量。

### VLAN 提供商网络中传出流量流

下图描述了离开实例的流量的数据包流，并直接指向 VLAN 提供商网络。本例使用附加到两个 VLAN 网络(171 和 172)的两个实例。配置 `br-ex` 后，向其添加一个物理接口，并将实例生成给 Compute 节点，生成的接口和网桥配置类似于下图中的配置：



OPENSTACK\_450456\_0617

1. 离开实例的 `eth0` 接口的数据包会到达连接到实例的 linux 网桥 `qbr-xx`。
2. `evince-xx` 连接到 `br-int`，它利用 veth 对 `<->HQxxx`。
3. `slirp-xx` 连接到 linux 网桥 `xx-xx`，`strat-xx` 连接到 Open vSwitch 网桥 `br-int`。

## Linux 网桥上的 *qbr-xx* 配置示例。

这个示例有两个实例和两个对应的 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 *qvoxx* 配置：

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

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

- **Tailoringxx** 使用与 VLAN 提供商网络关联的内部 VLAN 标签标记。在本例中，内部 VLAN 标签 2 与 VLAN 提供商网络 **provider-171** 关联，VLAN 标签 3 与 VLAN 提供商网络 **provider-172** 关联。当数据包到达 *qvoxx* 时，此 VLAN 标签将添加到数据包标头中。
- 数据包然后被移到 *br-ex* OVS 网桥，使用 patch-peer **int-br-ex** ↔ **phy-br-ex**。 *br-int* 上的 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}
```

*br-ex* 上补丁 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
```

- 当此数据包在 *br-ex* 上到达 *phy-br-ex* 时，*br-ex* 中的 OVS 流会将内部 VLAN 标签替换为与 VLAN 提供商网络关联的实际 VLAN 标签。

以下命令显示 *phy-br-ex* 的端口号为 **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
```

以下命令显示到达 phy-br-ex (**in\_port=4**)的任何数据包，它们具有 VLAN 标签 2 (**dl\_vlan=2**)。Open vSwitch 将 VLAN 标签替换为 171 (**action=mod\_vlan\_vid:171**,the the)，并将数据包转发到物理接口。命令还显示到达 phy-br-ex (**in\_port=4**)的任何数据包，它们具有 VLAN 标签 3 (**dl\_vlan=3**)。Open vSwitch 将 VLAN 标签替换为 172 (**actions=mod\_vlan\_vid:172,NORMAL**)，并将数据包转发到物理接口。neutron-openvswitch-agent 添加这些规则。

```
# 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，此数据包被转发到物理接口 *eth1*。

## VLAN 提供商网络中传入流量流

以下示例流在 Compute 节点上测试，使用 VLAN 标签 2 作为提供商网络 provider-171，以及 VLAN tag 3 用于提供商网络 provider-172。流使用网桥 br-int 上的端口 18。

您的 VLAN 提供商网络可能需要不同的配置。此外，网络的配置要求可能因两个不同的 Compute 节点而异。

以下命令显示 *int-br-ex*，端口号为 18：

```
# 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
```

以下命令显示 br-int 上的流规则。

```
# 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
```

### 传入流示例

本例演示了以下 br-int OVS 流：

```
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
```

- 来自外部网络的 VLAN 标签 172 的数据包通过物理节点上的 *eth1* 到达 *br-ex* 网桥。
- 数据包通过 patch-peer **phy-br-ex <-> int-br-ex** 移到 *br-int*。
- 数据包与流的标准匹配(**in\_port=18,dl\_vlan=172**)。
- 流操作 (**action=mod\_vlan\_vid:3**, the the VLAN tag 172 替换为内部 VLAN 标签 3, 并使用正常第 2 层处理将数据包转发到实例)。

### 其他资源

- [第 4.4 节 “扁平提供商网络数据包流的工作方式？”](#)

## 4.8. VLAN 提供商网络上的实例物理网络连接故障排除

当对 VLAN 提供商网络的连接进行故障排除时，请参阅“VLAN 供应商网络数据包流如何工作”中所述的数据包流。另外，请查看以下配置选项：

### 流程

1. 验证 **bridge\_mapping** 配置中使用的物理网络名称与物理网络名称匹配。

#### 示例

```
$ openstack network show provider-vlan171
```

#### 输出示例

```
...
| provider:physical_network | physnet1
...
```

#### 示例

```
$ grep bridge_mapping /etc/neutron/plugins/ml2/openvswitch_agent.ini
```

#### 输出示例

在这个示例输出中，物理网络名称 **physnet1** 与 **bridge\_mapping** 配置中使用的名称匹配：

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

2. 确认网络已创建为**外部**，类型为 **vlan**，并使用正确的 **segmentation\_id** 值：

### 示例

```
$ openstack network show provider-vlan171
```

### 输出示例

```
...
| provider:network_type | vlan |
| provider:physical_network | physnet1 |
| provider:segmentation_id | 171 |
...
```

3. 检查 patch-peer。  
验证 **br-int** 和 **br-ex** 是否使用 patch-peer **int-br-ex <--> phy-br-ex** 连接。

```
$ ovs-vsctl show
```

此连接是在重启 **neutron-openvswitch-agent** 时创建的，只要 **bridge\_mapping** 在 **/etc/neutron/plugins/ml2/openvswitch\_agent.ini** 中正确配置。

如果在重启该服务后没有创建此设置，则重新检查 **bridge\_mapping** 设置。

4. 检查网络流。
  - a. 要查看传出数据包的流，请运行 **ovs-ofctl dump-flows br-ex** 和 **ovs-ofctl dump-flows br-int**，并验证流是否将内部 VLAN ID 映射到外部 VLAN ID (**segmentation\_id**)。
  - b. 对于传入的数据包，将外部 VLAN ID 映射到内部 VLAN ID。  
当您首次向这个网络生成实例时，neutron OVS 代理会添加此流。
  - c. 如果在生成实例后没有创建此流，请确保将网络创建为 **vlan**，是**外部的**，并且 **physical\_network** 名称正确。另外，重新检查 **bridge\_mapping** 设置。
  - d. 最后，重新检查 **ifcfg-br-ex** 和 **ifcfg-ethx** 配置。  
确保 **br-ex** 包含端口 **ethX**，并且 **ifcfg-br-ex** 和 **ifcfg-ethx** 在 **ip a** 命令的输出中有一个 **UP** 标志。

### 示例

```
$ ovs-vsctl show
```

在这个示例输出中，**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"
```

### 示例

```
$ ip a
```

### 输出示例

在本例中，**eth1** 已添加为端口，并且内核被配置为将接口中的所有数据包移到 OVS 网桥 **br-ex**。这通过条目 **master ovs-system** 来演示。

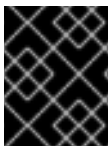
```
5: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP qlen 1000
```

### 其他资源

- [第 4.7 节 “VLAN 提供商网络数据包流的工作方式？”](#)

## 4.9. 在 ML2/OVS 部署中为提供商网络启用多播侦听

要防止对 Red Hat OpenStack Platform (RHOSP) 提供商网络中的每个端口进行多播数据包，您必须启用多播侦听。在使用带有 Open vSwitch 机制驱动程序 (ML2/OVS) 的 Modular Layer 2 插件的 RHOSP 部署中，您可以在 YAML 格式的环境文件中声明 RHOSP Orchestration (heat) NeutronEnable **IgmpSnooping** 参数。



### 重要

在将它应用到生产环境之前，您应该彻底测试并了解任何多播侦听配置。错误配置可能会破坏多播或导致不正确的网络行为。

### 先决条件

- 您的配置必须只使用 ML2/OVS 提供商网络。
- 您的物理路由器还必须启用 IGMP 侦听功能。  
也就是说，物理路由器必须在提供商网络上发送 IGMP 查询数据包，以便从多播组成员请求常规 IGMP 报告，以便在 OVS 中维护 snooping 缓存（用于物理网络）。
- 必须放置 RHOSP 网络服务安全组规则，以允许对虚拟机实例（或禁用端口安全）入站 IGMP。在本例中，为 **ping\_ssh** 安全组创建了一个规则：

### 示例

```
$ openstack security group rule create --protocol igmp --ingress ping_ssh
```

### 流程

1. 在 undercloud 主机上，以 stack 用户身份登录，创建自定义 YAML 环境文件。

### 示例

```
$ vi /home/stack/templates/my-ovs-environment.yaml
```



## 提示

编排服务(heat)使用一组名为 `template` 的计划来安装和配置您的环境。您可以使用自定义环境文件自定义 `overcloud` 的各个方面，这是为 `heat` 模板提供自定义的特殊模板。

2. 在 `parameter_defaults` 下的 YAML 环境文件中，将 `NeutronEnableIcmpSnooping` 设置为 `true`。

```
parameter_defaults:
  NeutronEnableIcmpSnooping: true
  ...
```



### 重要

确保您在冒号(:)和 `true` 之间添加空格字符。

3. 运行 `openstack overcloud deploy` 命令，并包含核心 `heat` 模板、环境文件和新的自定义环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

## 示例

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-ovs-
environment.yaml
```

## 验证

- 验证多播 snooping 是否已启用。

## 示例

```
# sudo ovs-vsctl list bridge br-int
```

## 输出示例

```
...
mcast_snooping_enable: true
...
other_config: {mac-table-size="50000", mcast-snooping-disable-flood-unregistered=True}
...
```

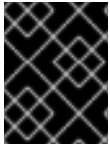
## 其他资源

- *Red Hat OpenStack Platform* 中的组件、插件和驱动程序支持中的 [Neutron](#)
- *高级 Overcloud 自定义指南* 中的 [环境文件](#)。

- [高级 Overcloud 自定义指南](#)中的[创建 overcloud](#) 中包括环境文件
- [Overcloud 参数指南](#)中的 [networking\(neutron\)](#) 参数
- [创建和管理实例指南](#)中的[创建安全组](#)

## 4.10. 在 ML2/OVN 部署中启用多播

要支持多播流量，请修改部署的安全配置，以允许多播流量访问多播组中的虚拟机(VM)实例。要防止多播流量激增，请启用 IGMP 侦听。



### 重要

在将它应用到生产环境之前，测试并了解任何多播侦听配置。错误配置可能会破坏多播或导致不正确的网络行为。

### 先决条件

- 具有 ML2/OVN 机制驱动程序的 OpenStack 部署。

### 流程

1. 配置安全性，以允许到适当虚拟机实例的多播流量。例如，创建一对安全组规则来允许来自 IGMP querier 的 IGMP 流量进入并退出虚拟机实例，以及第三个规则来允许多播流量。

#### 示例

安全组 `mySG` 允许 IGMP 流量进入并退出虚拟机实例。

```
openstack security group rule create --protocol igmp --ingress mySG
openstack security group rule create --protocol igmp --egress mySG
```

另一个规则允许多播流量访问虚拟机实例。

```
openstack security group rule create --protocol udp mySG
```

作为设置安全组规则的替代选择，一些运算符选择选择性地禁用网络上的端口安全性。如果您选择禁用端口安全性，请考虑并计划任何相关的安全风险。

2. 在 `undercloud` 节点上的环境文件中设置 `heat` 参数 **NeutronEnableIcmpSnooping: True**。例如，将以下行添加到 `ovn-extras.yaml` 中。

#### 示例

```
parameter_defaults:
  NeutronEnableIcmpSnooping: True
```

3. 将环境文件包含在 **openstack overcloud deploy** 命令中，以及任何与您环境相关的其他环境文件，并部署 overcloud。

```
$ openstack overcloud deploy \
--templates \
...
```

```
-e <other_overcloud_environment_files> \
-e ovn-extras.yaml \
...
```

将 `<other_overcloud_environment_files>` 替换为作为现有部署一部分的环境文件列表。

## 验证

1. 验证多播 snooping 是否已启用。列出北向数据库 Logical\_Switch 表。

```
$ ovn-nbctl list Logical_Switch
```

### 输出示例

```
_uuid      : d6a2fbcd-aaa4-4b9e-8274-184238d66a15
other_config : {mcast_flood_unregistered="false", mcast_snoop="true"}
...
```

Networking Service (neutron) `igmp_snooping_enable` 配置转换为 OVN 北向数据库中 Logical\_Switch 表的 `other_config` 列中设置的 `mcast_snooping_enable` 选项。请注意, `mcast_flood_unregistered` 始终为 "false"。

2. 显示 IGMP 组。

```
$ ovn-sbctl list IGMP_group
```

### 输出示例

```
_uuid      : 2d6cae4c-bd82-4b31-9c63-2d17cbeadc4e
address    : "225.0.0.120"
chassis    : 34e25681-f73f-43ac-a3a4-7da2a710ecd3
datapath   : eaf0f5cc-a2c8-4c30-8def-2bc1ec9dcabc
ports     : [5eaf9dd5-eae5-4749-ac60-4c1451901c56, 8a69efc5-38c5-48fb-bbab-30f2bf9b8d45]
...
```

## 其他资源

- [Red Hat OpenStack Platform 中的组件、插件和驱动程序支持中的 Neutron](#)
- [高级 Overcloud 自定义指南中的环境文件](#)。
- [高级 Overcloud 自定义指南中的创建 overcloud 中包括环境文件](#)

## 4.11. 启用计算元数据访问

如本章中所述的实例直接附加到提供商网络, 并将外部路由器配置为其默认网关。不使用 OpenStack Networking (neutron) 路由器。这意味着 `neutron` 路由器无法用于将实例的元数据请求代理到 `nova-metadata` 服务器, 这可能会在运行 `cloud-init` 时失败。但是, 可以通过将 `dhcp` 代理配置为代理元数据请求来解决这个问题。您可以在 `/etc/neutron/dhcp_agent.ini` 中启用此功能。例如:

```
enable_isolated_metadata = True
```

-

## 4.12. 浮动 IP 地址

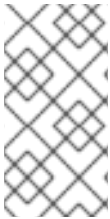
您可以使用同一网络为实例分配浮动 IP 地址，即使浮动 IP 已与专用网络关联。您从这个网络中作为浮动 IP 分配的地址会绑定到网络节点上的 `qrouter-xxx` 命名空间，并对关联的私有 IP 地址执行 `DNAT-SNAT`。相反，您为直接外部网络访问分配的 IP 地址直接在实例内绑定，并允许实例直接与外部网络通信。

## 第 5 章 管理浮动 IP 地址

除了具有私有、固定 IP 地址外，虚拟机实例还可以有一个公共或浮动 IP 地址来与其他网络通信。本节中的信息论述了如何使用 Red Hat OpenStack Platform (RHOSP) 网络服务(neutron)创建和管理浮动 IP。

### 5.1. 创建浮动 IP 池

您可以使用浮动 IP 地址将入口流量定向到 OpenStack 实例。首先，您必须定义一个有效路由的外部 IP 地址池，然后您可以动态分配给实例。OpenStack 网络将用于该浮动 IP 的所有传入的流量路由到您与浮动 IP 关联的实例。



#### 注意

OpenStack 网络以 CIDR 格式从同一 IP 范围分配浮动 IP 地址分配给所有项目（租户）。因此，所有项目都可以消耗每个浮动 IP 子网的浮动 IP。您可以使用特定项目的配额来管理此行为。例如，您可以将 **ProjectA** 和 **ProjectB** 的默认值设置为 **10**，同时将 **ProjectC** 的配额设置为 **0**。

#### 流程

- 创建外部子网时，您还可以定义浮动 IP 分配池。

```
$ openstack subnet create --no-dhcp --allocation-pool
start=IP_ADDRESS,end=IP_ADDRESS --gateway IP_ADDRESS --network
SUBNET_RANGE NETWORK_NAME
```

如果子网仅托管浮动 IP 地址，请考虑使用 **openstack subnet create** 命令中的 **--no-dhcp** 选项禁用 DHCP 分配。

#### 示例

```
$ openstack subnet create --no-dhcp --allocation_pool
start=192.168.100.20,end=192.168.100.100 --gateway 192.168.100.1 --network
192.168.100.0/24 public
```

#### 验证

- 您可以通过为实例分配随机浮动 IP 来验证池是否已正确配置。（请参见后续链接。）

#### 其他资源

- *Command Line Interface Reference* 中的 [subnet create](#)
- [分配随机浮动 IP](#)

### 5.2. 分配特定的浮动 IP

您可以为虚拟机实例分配特定的浮动 IP 地址。

#### 流程

- 使用 **openstack server add floating ip** 命令将浮动 IP 地址分配给实例。

## 示例

```
$ openstack server add floating ip prod-serv1 192.0.2.200
```

## 验证步骤

- 使用 **openstack server show** 命令确认您的浮动 IP 与您的实例相关联。

## 示例

```
$ openstack server show prod-serv1
```

## 输出示例

```
+-----+-----+
| Field          | Value                               |
+-----+-----+
| OS-DCF:diskConfig | MANUAL                               |
| OS-EXT-AZ:availability_zone | nova                               |
| OS-EXT-STS:power_state | Running                             |
| OS-EXT-STS:task_state | None                                 |
| OS-EXT-STS:vm_state | active                               |
| OS-SRV-USG:launched_at | 2021-08-11T14:45:37.000000         |
| OS-SRV-USG:terminated_at | None                                 |
| accessIPv4      |                                       |
| accessIPv6      |                                       |
| addresses       | public=198.51.100.56,192.0.2.200   |
| config_drive    |                                       |
| created         | 2021-08-11T14:44:54Z               |
| flavor          | review-ephemeral                   |
|                 | (8130dd45-78f6-44dc-8173-4d6426b8e520) |
| hostId         | 2308c8d8f60ed5394b1525122fb5bf8ea55c78b8 |
|                 | 0ec6157eca4488c9                   |
| id             | aef3ca09-887d-4d20-872d-1d1b49081958 |
| image          | rhel8                               |
|                 | (20724bfe-93a9-4341-a5a3-78b37b3a5dfb) |
| key_name       | example-keypair                     |
| name           | prod-serv1                           |
| progress       | 0                                    |
| project_id     | bd7a8c4a19424cf09a82627566b434fa   |
| properties     |                                       |
| security_groups | name='default'                       |
| status         | ACTIVE                               |
| updated        | 2021-08-11T14:45:37Z               |
| user_id        | 4b7e19a0d723310fd92911eb2fe59743a3a5cd32 |
|                 | 45f76ffcd91096196f646b5           |
| volumes_attached |                                       |
+-----+-----+
```

## 其他资源

- *Command Line Interface Reference* 中的 [server add floating ip](#)

- [命令行界面参考中的 server show](#)
- [分配随机浮动 IP](#)

## 5.3. 创建高级网络

在 Dashboard 中从 **Admin** 视图创建网络时，管理员可使用高级网络选项。使用这些选项指定项目，并定义您要使用的网络类型。

### 流程

1. 在仪表板中，选择 **Admin > Networks > Create Network > Project**
2. 选择您要使用 **Project** 下拉列表托管新网络的项目。
3. 查看 **Provider Network Type** 中的选项：
  - **Local** - 流量保留在本地计算主机上，并有效地与任何外部网络隔离。
  - **flat** - 流量保留在单一网络中，也可以与主机共享。没有 VLAN 标记或其他网络隔离。
  - **vlan** - 使用与物理网络中存在的 **VLAN** 对应的 VLAN ID 创建网络。此选项允许实例与同一第 2 层 VLAN 上的系统通信。
  - **GRE** - 使用跨多个节点的网络覆盖，以实现实例之间的私有通信。出口覆盖的流量必须路由。
  - **VXLAN** - 与 GRE 类似，并使用网络覆盖跨越多个节点来跨实例之间的私有通信。出口覆盖的流量必须路由。
4. 点 **Create Network**。  
查看 **Project Network Topology**，以验证网络是否已成功创建。

### 其他资源

- [分配特定的浮动 IP](#)
- [分配随机浮动 IP](#)

## 5.4. 分配随机浮动 IP

您可以从外部 IP 地址池动态为虚拟机实例分配浮动 IP 地址。

### 先决条件

- 可路由的外部 IP 地址池。  
更多信息请参阅 [第 5.1 节“创建浮动 IP 池”](#)。

### 流程

1. 输入以下命令从池中分配浮动 IP 地址。在本例中，网络名为 **public**。

#### 示例

```
$ openstack floating ip create public
```

## 输出示例

在以下示例中，新分配的浮动 IP 是 **192.0.2.200**。您可以将其分配给实例。

```

+-----+-----+-----+-----+
| Field          | Value                               |
+-----+-----+-----+-----+
| fixed_ip_address | None                                 |
| floating_ip_address | 192.0.2.200                         |
| floating_network_id | f0dcc603-f693-4258-a940-0a31fd4b80d9 |
| id              | 6352284c-c5df-4792-b168-e6f6348e2620 |
| port_id         | None                                 |
| router_id       | None                                 |
| status          | ACTIVE                              |
+-----+-----+-----+-----+

```

2. 输入以下命令查找您的实例：

```
$ openstack server list
```

## 输出示例

```

+-----+-----+-----+-----+-----+-----+
| ID          | Name      | Status | Networks | Image | Flavor |
+-----+-----+-----+-----+-----+-----+
| aef3ca09-88 | prod-serv1 | ACTIVE | public=198. | rhel8 | review- |
| 7d-4d20-872 |           |       | 51.100.56  |       | ephemeral |
| d-1d1b49081 |           |       |           |       |           |
| 958         |           |       |           |       |           |
|           |           |       |           |       |           |
+-----+-----+-----+-----+-----+-----+

```

3. 将实例名称或 ID 与浮动 IP 关联。

## 示例

```
$ openstack server add floating ip prod-serv1 192.0.2.200
```

## 验证步骤

- 输入以下命令确认您的浮动 IP 与您的实例相关联。

## 示例

```
$ openstack server show prod-serv1
```

## 输出示例

```

+-----+-----+-----+-----+
| Field          | Value                               |
+-----+-----+-----+-----+
| OS-DCF:diskConfig | MANUAL                              |
+-----+-----+-----+-----+

```



```

| OS-EXT-AZ:availability_zone | nova |
| OS-EXT-STS:power_state | Running |
| OS-EXT-STS:task_state | None |
| OS-EXT-STS:vm_state | active |
| OS-SRV-USG:launched_at | 2021-08-11T14:45:37.000000 |
| OS-SRV-USG:terminated_at | None |
| accessIPv4 | |
| accessIPv6 | |
| addresses | public=198.51.100.56,192.0.2.200 |
| config_drive | |
| created | 2021-08-11T14:44:54Z |
| flavor | review-ephemeral |
| | (8130dd45-78f6-44dc-8173-4d6426b8e520) |
| hostId | 2308c8d8f60ed5394b1525122fb5bf8ea55c78b8 |
| | 0ec6157eca4488c9 |
| id | aef3ca09-887d-4d20-872d-1d1b49081958 |
| image | rhel8 |
| | (20724bfe-93a9-4341-a5a3-78b37b3a5dfb) |
| key_name | example-keypair |
| name | prod-serv1 |
| progress | 0 |
| project_id | bd7a8c4a19424cf09a82627566b434fa |
| properties | |
| security_groups | name='default' |
| status | ACTIVE |
| updated | 2021-08-11T14:45:37Z |
| user_id | 4b7e19a0d723310fd92911eb2fe59743a3a5cd32 |
| | 45f76ffced91096196f646b5 |
| volumes_attached | |
+-----+

```

## 其他资源

- [命令行界面参考中的 floating ip create](#)
- [Command Line Interface Reference](#) 中的 [server add floating ip](#)
- [命令行界面参考中的 server show](#)
- [创建浮动 IP 池](#)

## 5.5. 创建多个浮动 IP 池

OpenStack 网络支持每个 L3 代理有一个浮动 IP 池。因此，您必须扩展 L3 代理以创建额外的浮动 IP 池。

### 流程

- 确保在 `/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf` 中，对于环境中只有一个 L3 代理，则属性 `handle_internal_only_routers` 被设置为 `True`。此选项将 L3 代理配置为仅管理非外部路由器。

## 其他资源

- [创建浮动 IP 池](#)
- [分配随机浮动 IP](#)

## 5.6. 配置浮动 IP 端口转发

要让用户为浮动 IP 设置端口转发，您必须启用 Red Hat OpenStack Platform (RHOSP)网络服务 (neutron) port\_forwarding' 服务插件。

### 先决条件

- 您必须具有 RHOSP 管理员特权。
- **port\_forwarding** 服务插件要求您设置 **路由器服务** 插件。

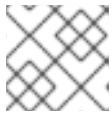
### 流程

1. 以 stack 用户身份登录 undercloud 主机。
2. 查找 stackrc undercloud 凭证文件：

```
$ source ~/stackrc
```

3. 在自定义环境 YAML 文件中，设置 **port\_forwarding** 服务插件：

```
parameter_defaults:
  NeutronPluginExtensions: "router,port_forwarding"
```



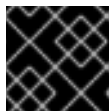
#### 注意

**port\_forwarding** 服务插件要求您设置 **路由器服务** 插件。

4. 如果您将 ML2/OVS 机制驱动程序与网络服务搭配使用，还必须为 OVS L3 代理设置 **port\_forwarding** 扩展：

```
parameter_defaults:
  NeutronPluginExtensions: "router,port_forwarding"
  NeutronL3AgentExtensions: "port_forwarding"
```

5. 部署 overcloud，并包含核心 heat 模板、环境文件和新的自定义环境文件。



#### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

```
$ openstack overcloud deploy --templates \
-e <your_environment_files> \
-e /home/stack/templates/my-environment.yaml
```

RHOSP 用户现在可以为浮动 IP 设置端口转发。如需更多信息，请参阅 [第 5.7 节“为浮动 IP 创建端口转发”](#)。

## 验证

1. 提供 overcloud 凭据文件。

### 示例

```
$ source ~/overcloudrc
```

2. 确保网络服务已成功载入 **port\_forwarding** 和**路由器** 服务插件：

```
$ openstack extension list --network -c Name -c Alias --max-width 74 |\
grep -i -e 'Neutron L3 Router' -i -e floating-ip-port-forwarding
```

### 输出示例

成功验证会生成类似如下的输出：

```
| Floating IP Port Forwarding | floating-ip-port-forwarding |
| Neutron L3 Router          | router                        |
```

## 其他资源

- 使用 *director* 安装和管理 Red Hat OpenStack Platform 中的 环境文件 [https://access.redhat.com/documentation/zh-cn/red\\_hat\\_openstack\\_platform/16.2/html/installing\\_and\\_managing\\_red\\_hat\\_openstack\\_platform\\_heat-templates#con\\_environment-files\\_understanding-heat-templates](https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/16.2/html/installing_and_managing_red_hat_openstack_platform_heat-templates#con_environment-files_understanding-heat-templates)
- 使用 *director* 安装和管理 Red Hat OpenStack Platform 指南中的创建 overcloud 中的环境文件。

## 5.7. 为浮动 IP 创建端口转发

您可以使用 Red Hat OpenStack Platform Networking 服务(neutron)为浮动 IP 设置端口转发。

### 先决条件

- 网络服务必须在加载 **port\_forwarding** 服务插件的情况下运行。如需更多信息，请参阅 [第 5.6 节“配置浮动 IP 端口转发”](#)。

### 流程

1. 提供您的凭据文件。

### 示例

```
$ source ~/overcloudrc
```

2. 使用以下命令为浮动 IP 创建端口转发：

```
$ openstack floating ip port forwarding create \
--internal-ip-address <internal-ip-address> \
--port <port> \
--internal-protocol-port <port-number> \
```

```
--external-protocol-port <port-number> \
--protocol <protocol> \
<floating-ip>
```

- 将 **<internal-ip-address>** 替换为内部目标 IP 地址。  
这是与运行应用的实例关联的 IP 地址。
- 将 **<port>** 替换为实例所附加的网络服务端口的名称或 ID。
- 将 **--internal-protocol-port** 中的 **<port-number>** 替换为内部目标端口号。  
这是应用程序在实例中使用的端口号。
- 将 **--external-protocol-port** 中的 **<port-number>** 替换为外部源端口号。  
这是在 RHOSP 云外部运行的应用程序的端口号。
- 将 **<protocol>** 替换为接收端口转发流量的应用程序所使用的协议，如 TCP 或 UDP。
- 将 **<floating-ip>** 替换为您要转发的端口流量的浮动 IP。

### 示例

本例为附加到浮动 IP **198.51.100.47** 的实例创建端口。浮动 IP 使用网络服务端口 **1adfdb09-e8c6-4708-b5aa-11f50fc22d62**。当网络服务检测到传入 **198.51.100.47:80** 的外部流量时，它将流量转发到内部 IP 地址 **203.0.113.107**，在 TCP 端口 **8080** 上：

```
$ openstack floating ip port forwarding create \
--internal-ip-address 203.0.113.107 \
--port 1adfdb09-e8c6-4708-b5aa-11f50fc22d62 \
--internal-protocol-port 8080 \
--external-protocol-port 80 \
--protocol tcp \
198.51.100.47
```

### 验证

- 确认网络服务已经为浮动 IP 端口建立了转发。

### 示例

以下示例验证浮动 IP **198.51.100.47** 的端口转发是否成功：

```
$ openstack floating ip port forwarding list 198.51.100.47 --max-width 74
```

### 输出示例

输出显示，发送到 TCP 端口 80 上的浮动 IP **198.51.100.47** 的流量转发到实例上的端口 **8080**，其内部地址为 **203.0.113.107**：

```
+-----+-----+-----+-----+-----+-----+-----+
+
| ID      | Internal Port ID | Internal IP Address | Internal Port | External Port | Protocol |
| Description |
+-----+-----+-----+-----+-----+-----+-----+
+
| 5cf204c7 | 1adfdb09-e8c6-47 | 203.0.113.107      | 8080 | 80 | tcp |
| -6825-45 | 08-b5aa-11f50fc2 |                    |      |    |     |
```

```

| de-84ec- | 2d62      |      |      |      |      |      |
| 2eb507be |      |      |      |      |      |      |
| 543e     |      |      |      |      |      |      |
+-----+-----+-----+-----+-----+-----+-----+
+

```

## 其他资源

- [命令行界面参考中的 floating ip 端口转发创建](#)

## 5.8. 桥接物理网络

将您的虚拟网络桥接到物理网络，以启用与虚拟实例的连接。

在此过程中，物理接口 **eth0** 示例映射到网桥 **br-ex**；虚拟网桥充当物理网络和任何虚拟网络之间的中间。

因此，遍历 **eth0** 的所有流量都使用配置的 Open vSwitch 访问实例。

要将物理 NIC 映射到虚拟 Open vSwitch 网桥，请完成以下步骤：

### 流程

1. 在文本编辑器中打开 **/etc/sysconfig/network-scripts/ifcfg-eth0**，并使用站点上网络的值更新以下参数：

- IPADDR
  - 子网掩码网关
  - DNS1（名称服务器）
- 下面是一个示例：

```

# vi /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
ONBOOT=yes

```

2. 在文本编辑器中打开 **/etc/sysconfig/network-scripts/ifcfg-br-ex**，并使用之前分配给 eth0 的 IP 地址值更新虚拟网桥参数：

```

# 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，您可以将浮动 IP 地址分配给实例，并将其提供给物理网络。

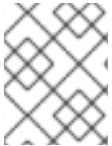
## 其他资源

- [配置网桥映射](#)

## 5.9. 添加接口

您可以使用接口将路由器与子网互连，以便路由器可以将实例发送到其中间子网外目的地的任何流量。

要添加路由器接口并将新接口连接到子网，请完成以下步骤：



### 注意

此流程使用 Network Topology 功能。使用此功能，您可以在执行网络管理任务的同时查看所有虚拟路由器和网络的图形表示。

1. 在仪表板中，选择 **Project > Network > Network Topology**。
2. 找到您要管理的路由器，将鼠标悬停在上面，将鼠标悬停在上面，然后单击 **Add Interface**。
3. 指定您要连接到路由器的子网。  
您还可以指定 IP 地址。该地址可用于测试和故障排除目的，因为成功对此接口的 ping 表示流量按预期路由。
4. 点 **Add interface**。  
Network Topology 图自动更新，以反映路由器和子网之间的新接口连接。

## 5.10. 删除接口

如果您不再需要路由器为子网定向到子网的流量，您可以删除子网的接口。

要删除接口，请完成以下步骤：

1. 在控制面板中，选择 **Project > Network > Routers**。
2. 单击托管您要删除的接口的路由器的名称。
3. 选择接口类型(**内部接口**)，然后单击 **Delete Interfaces**。

## 第 6 章 网络故障排除

Red Hat OpenStack Platform 中网络连接故障排除的诊断过程与物理网络的诊断过程类似。如果使用 VLAN，您可以将虚拟基础架构视为物理网络的中继扩展，而不是由谁独立的环境。ML2/OVS 网络和默认 ML2/OVN 网络故障排除之间存在一些区别。

### 6.1. 基本 PING 测试

`ping` 命令是分析网络连接问题的有用工具。结果充当网络连接的基本指标，但可能无法完全排除所有连接问题，比如防火墙阻止实际应用程序流量。`ping` 命令将流量发送到特定的目的地，然后报告尝试是否成功。



#### 注意

`ping` 命令是 ICMP 操作。要使用 `ping`，您必须允许 ICMP 流量遍历任何中间防火墙。

从存在网络问题的机器运行时，`ping` 测试最有用，因此如果机器看起来完全离线，可能需要通过 VNC 管理控制台连接到命令行。

例如，以下 `ping test` 命令会验证网络基础架构的多个层才能成功；名称解析、IP 路由和网络切换必须可以正常工作：

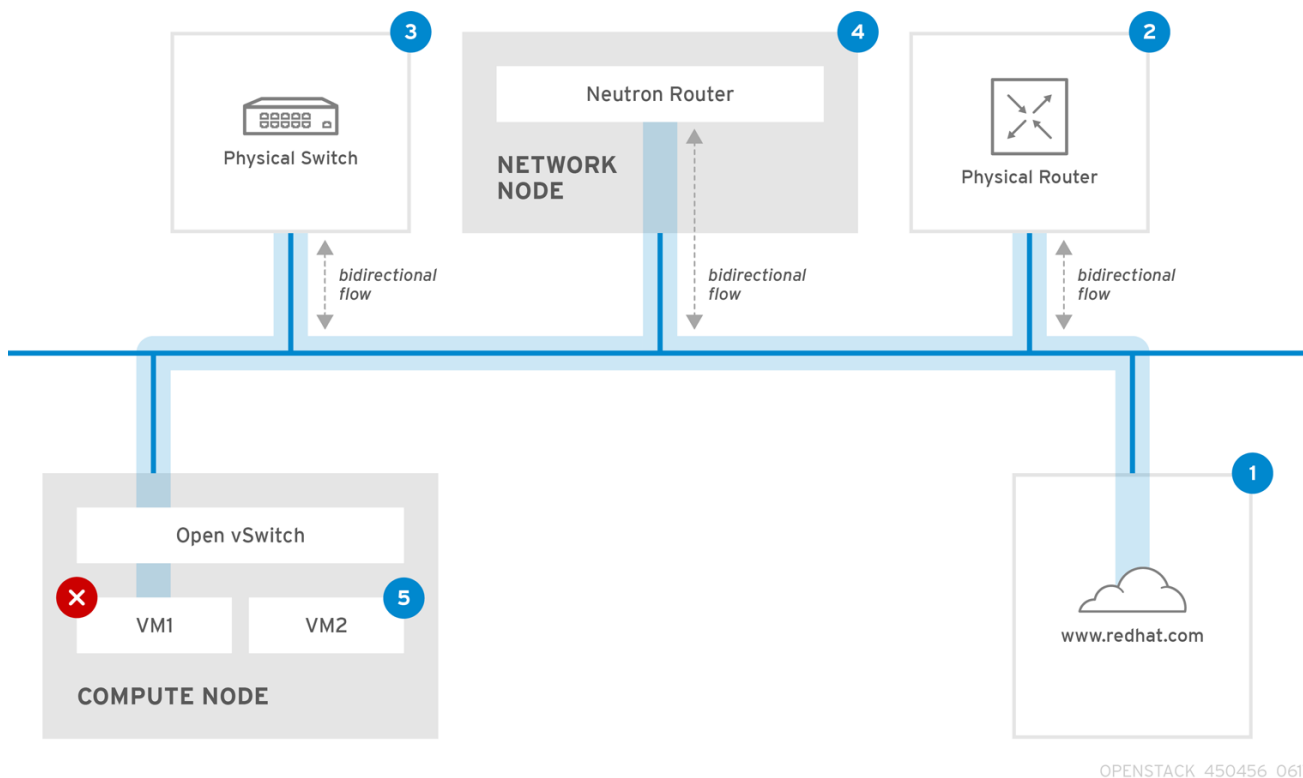
```
$ ping www.example.com
```

```
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
```

您可以使用 `Ctrl-c` 终止 `ping` 命令，之后会显示结果摘要。零百分比的数据包丢失表示连接稳定且没有超时。

```
--- 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
```

`ping` 测试的结果可能会非常困难，具体取决于您测试的目标。例如，下图 VM1 遇到某种形式的连接问题。可能的目的地以蓝色的形式编号，结果来自成功或失败的结果：



1. **互联网** - 常见的第一步是将 ping 测试发送到互联网位置，如 www.example.com。

- **成功**：此测试表示计算机和互联网之间的所有各种网络点都正常运行。这包括虚拟网络和物理网络基础架构。
- **失败**：对距离互联网位置的 ping 测试可能会失败。如果您的网络中的其他机器可以成功 ping 互联网，这证明互联网连接正常工作，且问题可能会在本地机器配置中成功。

2. **物理路由器** - 网络管理员指定将流量定向到外部目的地的路由器接口。

- **成功**：对物理路由器进行 ping 测试可以确定本地网络和底层交换机是否正常工作。这些数据不会遍历路由器，因此它们不会证明默认网关中是否存在路由问题。
- **失败**：这表示 VM1 和默认网关间的问题。路由器/切换可能会停机，或者您可能使用不正确的默认网关。将配置与您知道的其他服务器上正常工作的配置进行比较。尝试 ping 本地网络中的其他服务器。

3. **Neutron 路由器** - 这是 Red Hat OpenStack Platform 用来指示虚拟机流量的虚拟 SDN（软件定义网络）路由器。

- **成功**：防火墙允许 ICMP 流量，网络节点在线。
- **失败**：确认实例安全组中是否允许 ICMP 流量。检查网络节点是否已在线，确认所有必需的服务是否正在运行，并查看 L3 代理日志(/var/log/neutron/l3-agent.log)。

4. **物理交换机** - 物理交换机管理同一物理网络中节点之间的流量。

- **成功**：虚拟机发送到物理交换机的流量必须通过虚拟网络基础架构传递，这表示此片段正常工作。
- **失败**：检查物理交换机端口是否已配置为中继所需的 VLAN。

5. **VM2** - 在同一 Compute 节点上 ping 同一子网中的虚拟机。

- **成功**：VM1 上的 NIC 驱动程序和基本 IP 配置可以正常工作。



- **失败**：验证 VM1 上的网络配置。或者，VM2 上的防火墙可能只是阻止 ping 流量。此外，验证虚拟切换配置并查看 Open vSwitch 日志文件。

## 6.2. 查看当前端口状态

基本故障排除任务是创建附加到路由器的所有端口的清单，并确定端口状态(**DOWN** 或 **ACTIVE**)。

### 流程

1. 要查看附加到名为 **r1** 的路由器的所有端口，请运行以下命令：

```
# openstack port list --router r1
```

### 输出示例

```
+-----+-----+-----+-----+
+-----+
| id           | name | mac_address   | fixed_ips
+-----+-----+-----+-----+
| b58d26f0-cc03-43c1-ab23-ccdb1018252a |   | fa:16:3e:94:a7:df | {"subnet_id": "a592fdbababd-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. 要查看每个端口的详情，请运行以下命令：包括您要查看的端口的端口 ID。其结果包括端口状态，在以下示例中表示为 **ACTIVE** 状态：

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

### 输出示例

```
+-----+-----+-----+-----+
+
| 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:vnictype | normal
| device_id      | 49c6ebdc-0e62-49ad-a9ca-58cea464472f
| device_owner   | network:router_interface
| extra_dhcp_opts |
| fixed_ips      | {"subnet_id": "a592fdbababd-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 |
+-----+-----+
+

```

3. 为每个端口执行第 2 步以确定其状态。

### 6.3. 与 VLAN 提供商网络的连接故障排除

OpenStack 网络可以将 VLAN 网络中继到 SDN 交换机。支持 VLAN 标记的提供商网络意味着虚拟实例可以与物理网络中的服务器子网集成。

#### 流程

1. 使用 **ping <gateway-IP-address>** ping 网关。

考虑使用这些命令创建网络的示例：

```

# openstack network create --provider-network-type vlan --provider-physical-network phy-
eno1 --provider-segment 120 provider
# openstack subnet create --no-dhcp --allocation-pool
start=192.168.120.1,end=192.168.120.153 --gateway 192.168.120.254 --network provider
public_subnet

```

在本例中，网关 IP 地址为 **192.168.120.254**。

```
$ ping 192.168.120.254
```

2. 如果 ping 失败，请执行以下操作：
  - a. 确认您有关联 VLAN 的网络流。  
VLAN ID 可能尚未设置。在本例中，OpenStack 网络配置为将 VLAN 120 中继到提供商网络。（请参阅第 1 步中的 `--provider:segmentation_id=120`。）
  - b. 使用命令 **ovs-ofctl dump-flows <bridge-name>** 来确认网桥接口上的 VLAN 流。  
在本例中，网桥名为 **br-ex**：

```

# 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

```

### 6.4. 查看 VLAN 配置和日志文件

要帮助验证或排除部署，您可以：

- 验证 Red Hat Openstack Platform (RHOSP)网络服务(neutron)代理的注册和状态。
- 验证网络配置值，如 VLAN 范围。

## 流程

1. 使用 **openstack network agent list** 命令来验证 RHOSP Networking 服务代理是否已启动，并使用正确的主机名注册。

```
(overcloud)[stack@undercloud~]$ openstack network 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            |
+-----+-----+-----+-----+-----+
```

2. 查看 `/var/log/containers/neutron/openvswitch-agent.log`。查找确认创建过程使用了 **ovs-ofctl** 命令来配置 VLAN 中继。
3. 验证 `/etc/neutron/l3_agent.ini` 文件中的 **external\_network\_bridge**。如果在 **external\_network\_bridge** 参数中有一个硬编码的值，则无法将提供商网络与 L3-agent 搭配使用，您无法创建必要的流。**external\_network\_bridge** 值的格式必须是 `'external_network_bridge = '''`。
4. 检查 `/etc/neutron/plugin.ini` 文件中的 **network\_vlan\_ranges** 值。对于提供商网络，不要指定数字 VLAN ID。仅在使用 VLAN 隔离项目网络时指定 ID。
5. 验证 **OVS 代理配置文件网桥映射**，以确认映射到 **phy-eno1** 的网桥是否存在，并正确连接到 **eno1**。

## 6.5. 在 ML2/OVN 命名空间中执行基本 ICMP 测试

作为基本的故障排除步骤，您可以尝试从同一第 2 层网络上的 OVN 元数据接口 ping 实例。

### 先决条件

- RHOSP 部署，使用 ML2/OVN 作为网络服务(neutron)默认机制驱动程序。

### 流程

1. 使用您的 Red Hat OpenStack Platform 凭据登录 overcloud。
2. 运行 **openstack server list** 命令以获取虚拟机实例的名称。
3. 运行 **openstack server show** 命令，以确定在其上运行实例的 Compute 节点。

### 示例

```
$ openstack server show my_instance -c OS-EXT-SRV-ATTR:host \
-c addresses
```

### 输出示例

```
+-----+-----+
| Field          | Value                                     |
+-----+-----+
| OS-EXT-SRV-ATTR:host | compute0.ctlplane.example.com          |
| addresses         | finance-network1=192.0.2.2; provider-  |
|                   | storage=198.51.100.13                  |
+-----+-----+
```

4. 登录 Compute 节点主机。

### 示例

```
$ ssh heat-admin@compute0.ctlplane
```

5. 运行 **ip netns list** 命令来查看 OVN 元数据命名空间。

### 输出示例

```
ovnmeta-07384836-6ab1-4539-b23a-c581cf072011 (id: 1)
ovnmeta-df9c28ea-c93a-4a60-b913-1e611d6f15aa (id: 0)
```

6. 使用元数据命名空间运行 **ip netns exec** 命令，以 ping 关联的网络。

### 示例

```
$ sudo ip netns exec ovnmeta-df9c28ea-c93a-4a60-b913-1e611d6f15aa \
ping 192.0.2.2
```

### 输出示例

```
PING 192.0.2.2 (192.0.2.2) 56(84) bytes of data.
64 bytes from 192.0.2.2: icmp_seq=1 ttl=64 time=0.470 ms
64 bytes from 192.0.2.2: icmp_seq=2 ttl=64 time=0.483 ms
64 bytes from 192.0.2.2: icmp_seq=3 ttl=64 time=0.183 ms
64 bytes from 192.0.2.2: icmp_seq=4 ttl=64 time=0.296 ms
64 bytes from 192.0.2.2: icmp_seq=5 ttl=64 time=0.307 ms
^C
--- 192.0.2.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 122ms
rtt min/avg/max/mdev = 0.183/0.347/0.483/0.116 ms
```

### 其他资源

- 命令行界面参考中的 [server show](#)

## 6.6. 从项目网络内进行故障排除(ML2/OVS)

在 Red Hat Openstack Platform (RHOSP) ML2/OVS 网络中，所有项目流量都包含在网络命名空间中，以便项目可以在不相互干扰的情况下配置网络。例如，网络命名空间允许不同的项目具有相同的子网范围 192.168.1.1/24，而不会干扰它们。

## 先决条件

- RHOSP 部署，使用 ML2/OVS 作为网络服务(neutron)默认机制驱动程序。

## 流程

1. 使用 **openstack network list** 命令列出所有项目网络，以确定哪个网络命名空间包含网络：

```
$ openstack network list
```

在此输出中，请注意 **web-servers** 网络的 ID (**9cb32fe0-d7fb-432c-b116-f483c6497b08**)。命令将网络 ID 附加到网络命名空间，这样您可以在下一步中识别命名空间。

## 输出示例

```
+-----+-----+-----+
| 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 |
+-----+-----+-----+
```

2. 使用 **ip netns list** 命令列出所有网络命名空间：

```
# ip netns list
```

输出中包含与 **web-servers** 网络 ID 匹配的命名空间。

在这个输出中，命名空间为 **qdhcp-9cb32fe0-d7fb-432c-b116-f483c6497b08**。

## 输出示例

```
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
```

3. 通过在命名空间中运行命令（使用 **ip netns exec <namespace>** 作为前缀）检查 **web-servers** 网络的配置。  
本例中使用了 **route -n** 命令。

## 示例

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

### 输出示例

```
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
```

## 6.7. 在命名空间中执行高级 ICMP 测试(ML2/OVS)

您可以使用 **tcpdump** 和 **ping** 命令的组合对 Red Hat Openstack Platform (RHOSP) ML2/OVS 网络进行故障排除。

### 先决条件

- RHOSP 部署，使用 ML2/OVS 作为网络服务(neutron)默认机制驱动程序。

### 流程

1. 使用 **tcpdump** 命令捕获 ICMP 流量：

#### 示例

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

2. 在一个单独的命令行窗口中，对外部网络执行 ping 测试：

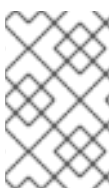
#### 示例

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

3. 在运行 **tcpdump** 会话的终端中，观察 ping 测试的详细结果。

### 输出示例

```
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

当您对流量执行 **tcpdump** 分析时，您会看到对路由器接口的响应数据包标题，而不是与虚拟机实例的响应。这是预期的行为，因为 **qrouter** 在返回数据包上执行目标网络地址转换 (DNAT)。

## 6.8. 为 OVN 故障排除命令创建别名

您可以在 **ovn\_controller** 容器中运行 OVN 命令，如 **ovn-nbctl show**。容器在 Controller 节点和 Compute 节点上运行。要简化对命令的访问，请创建并提供定义别名的脚本。

### 先决条件

- 使用 ML2/OVN 作为默认机制驱动程序部署的 Red Hat OpenStack Platform。

### 流程

1. 以具有访问 OVN 容器所需的权限的用户身份登录到 Controller 主机。

#### 示例

```
$ ssh heat-admin@controller-0.ctlplane
```

2. 创建一个 shell 脚本文件，其中包含您要运行的 **ovn** 命令。

#### 示例

```
vi ~/bin/ovn-alias.sh
```

3. 添加 **ovn** 命令并保存脚本文件。

#### 示例

在这个示例中，**ovn-sbctl**、**ovn-nbctl** 和 **ovn-trace** 命令已添加到别名文件中：

```
EXTERNAL_ID=\
$(sudo ovs-vsctl get open . external_ids:ovn-remote | awk -F: '{print $2}')
export NBDB=tcp:${EXTERNAL_ID}:6641
export SBDB=tcp:${EXTERNAL_ID}:6642

alias ovn-sbctl="sudo podman exec ovn_controller ovn-sbctl --db=$SBDB"
alias ovn-nbctl="sudo podman exec ovn_controller ovn-nbctl --db=$NBDB"
alias ovn-trace="sudo podman exec ovn_controller ovn-trace --db=$SBDB"
```

4. 在 Compute 主机上重复此流程中的步骤。

### 验证

1. 查找脚本文件。

#### 示例

```
# source ovn-alias.sh
```

2. 运行命令以确认脚本文件是否正常工作。

#### 示例

```
# ovn-nbctl show
```

#### 输出示例

```

switch 26ce22db-1795-41bd-b561-9827cbd81778 (neutron-f8e79863-6c58-43d0-8f7d-
8ec4a423e13b) (aka internal_network)
  port 1913c3ae-8475-4b60-a479-df7bcce8d9c8
    addresses: ["fa:16:3e:33:c1:fc 192.168.254.76"]
  port 1aabaee3-b944-4da2-bf0a-573215d3f3d9
    addresses: ["fa:16:3e:16:cb:ce 192.168.254.74"]
  port 7e000980-59f9-4a0f-b76a-4fdf4e86f27b
    type: localport
    addresses: ["fa:16:3e:c9:30:ed 192.168.254.2"]

```

## 其他资源

- **OVN-nbctl --help** 命令
- **OVN-sbctl --help** 命令
- **OVN-trace --help** 命令

## 6.9. 监控 OVN 逻辑流

OVN 使用作为优先级、匹配和操作的流表的逻辑流。这些逻辑流被分发到在每个 Red Hat Openstack Platform (RHOSP) Compute 节点上运行的 **ovn-controller**。在 Controller 节点上使用 **ovn-sbctl lflow-list** 命令来查看完整的逻辑流集合。

### 先决条件

- 使用 ML2/OVN 作为网络服务(neutron)默认机制驱动程序的 RHOSP 部署。
- 为 OVN 数据库命令创建别名文件。  
请参阅 [第 6.8 节“为 OVN 故障排除命令创建别名”](#)。

### 流程

1. 以具有访问 OVN 容器所需的权限的用户身份登录到 Controller 主机。

#### 示例

```
$ ssh heat-admin@controller-0.ctlplane
```

2. 提供 OVN 数据库命令的别名文件。  
更多信息请参阅 [第 6.8 节“为 OVN 故障排除命令创建别名”](#)。

#### 示例

```
source ~/ovn-alias.sh
```

3. 查看逻辑流：

```
$ ovn-sbctl lflow-list
```

4. 检查输出。

#### 输出示例



```

Datapath: "sw0" (d7bf4a7b-e915-4502-8f9d-5995d33f5d10) Pipeline: ingress
  table=0 (ls_in_port_sec_l2 ), priority=100 , match=(eth.src[40]), action=(drop;)
  table=0 (ls_in_port_sec_l2 ), priority=100 , match=(vlan.present), action=(drop;)
  table=0 (ls_in_port_sec_l2 ), priority=50 , match=(inport == "sw0-port1" && eth.src ==
{00:00:00:00:00:01}), action=(next;)
  table=0 (ls_in_port_sec_l2 ), priority=50 , match=(inport == "sw0-port2" && eth.src ==
{00:00:00:00:00:02}), action=(next;)
  table=1 (ls_in_port_sec_ip ), priority=0 , match=(1), action=(next;)
  table=2 (ls_in_port_sec_nd ), priority=90 , match=(inport == "sw0-port1" && eth.src ==
00:00:00:00:00:01 && arp.sha == 00:00:00:00:00:01), action=(next;)
  table=2 (ls_in_port_sec_nd ), priority=90 , match=(inport == "sw0-port1" && eth.src ==
00:00:00:00:00:01 && ip6 && nd && ((nd.sll == 00:00:00:00:00:00 || nd.sll ==
00:00:00:00:00:01) || ((nd.tll == 00:00:00:00:00:00 || nd.tll == 00:00:00:00:00:01))))), action=
(next;)
  table=2 (ls_in_port_sec_nd ), priority=90 , match=(inport == "sw0-port2" && eth.src ==
00:00:00:00:00:02 && arp.sha == 00:00:00:00:00:02), action=(next;)
  table=2 (ls_in_port_sec_nd ), priority=90 , match=(inport == "sw0-port2" && eth.src ==
00:00:00:00:00:02 && ip6 && nd && ((nd.sll == 00:00:00:00:00:00 || nd.sll ==
00:00:00:00:00:02) || ((nd.tll == 00:00:00:00:00:00 || nd.tll == 00:00:00:00:00:02))))), action=
(next;)
  table=2 (ls_in_port_sec_nd ), priority=80 , match=(inport == "sw0-port1" && (arp || nd)),
action=(drop;)
  table=2 (ls_in_port_sec_nd ), priority=80 , match=(inport == "sw0-port2" && (arp || nd)),
action=(drop;)
  table=2 (ls_in_port_sec_nd ), priority=0 , match=(1), action=(next;)
  table=3 (ls_in_pre_acl ), priority=0, match=(1), action=(next;)
  table=4 (ls_in_pre_lb ), priority=0 , match=(1), action=(next;)
  table=5 (ls_in_pre_stateful ), priority=100 , match=(reg0[0] == 1), action=(ct_next;)
  table=5 (ls_in_pre_stateful ), priority=0 , match=(1), action=(next;)
  table=6 (ls_in_acl ), priority=0 , match=(1), action=(next;)
  table=7 (ls_in_qos_mark ), priority=0 , match=(1), action=(next;)
  table=8 (ls_in_lb ), priority=0 , match=(1), action=(next;)
  table=9 (ls_in_stateful ), priority=100 , match=(reg0[1] == 1), action=
(ct_commit(ct_label=0/1); next;)
  table=9 (ls_in_stateful ), priority=100 , match=(reg0[2] == 1), action=(ct_lb;)
  table=9 (ls_in_stateful ), priority=0 , match=(1), action=(next;)
  table=10(ls_in_arp_rsp ), priority=0 , match=(1), action=(next;)
  table=11(ls_in_dhcp_options ), priority=0 , match=(1), action=(next;)
  table=12(ls_in_dhcp_response), priority=0 , match=(1), action=(next;)
  table=13(ls_in_l2_lkup ), priority=100 , match=(eth.mcast), action=(output =
" _MC_flood"; output;)
  table=13(ls_in_l2_lkup ), priority=50 , match=(eth.dst == 00:00:00:00:00:01), action=
(output = "sw0-port1"; output;)
  table=13(ls_in_l2_lkup ), priority=50 , match=(eth.dst == 00:00:00:00:00:02), action=
(output = "sw0-port2"; output;)
Datapath: "sw0" (d7bf4a7b-e915-4502-8f9d-5995d33f5d10) Pipeline: egress
  table=0 (ls_out_pre_lb ), priority=0 , match=(1), action=(next;)
  table=1 (ls_out_pre_acl ), priority=0 , match=(1), action=(next;)
  table=2 (ls_out_pre_stateful), priority=100 , match=(reg0[0] == 1), action=(ct_next;)
  table=2 (ls_out_pre_stateful), priority=0 , match=(1), action=(next;)
  table=3 (ls_out_lb ), priority=0 , match=(1), action=(next;)
  table=4 (ls_out_acl ), priority=0 , match=(1), action=(next;)
  table=5 (ls_out_qos_mark ), priority=0 , match=(1), action=(next;)
  table=6 (ls_out_stateful ), priority=100 , match=(reg0[1] == 1), action=
(ct_commit(ct_label=0/1); next;)
  table=6 (ls_out_stateful ), priority=100 , match=(reg0[2] == 1), action=(ct_lb;)

```

```

table=6 (ls_out_stateful ), priority=0 , match=(1), action=(next;)
table=7 (ls_out_port_sec_ip ), priority=0 , match=(1), action=(next;)
table=8 (ls_out_port_sec_l2 ), priority=100 , match=(eth.mcast), action=(output;)
table=8 (ls_out_port_sec_l2 ), priority=50 , match=(output == "sw0-port1" && eth.dst ==
{00:00:00:00:00:01}), action=(output;)
table=8 (ls_out_port_sec_l2 ), priority=50 , match=(output == "sw0-port2" && eth.dst ==
{00:00:00:00:00:02}), action=(output;)

```

OVN 和 OpenFlow 之间的主要区别包括：

- OVN 端口是位于网络中的某个位置的逻辑实体，而不是单一交换机上的物理端口。
  - OVN 为管道中的每个表指定一个名称，除了其数字外。名称描述了管道中该阶段的目的。
  - OVN 匹配语法支持复杂的布尔值表达式。
  - OVN 逻辑流中支持的操作会超过 OpenFlow 的操作。您可以在 OVN 逻辑流语法中实施更高级别的功能，如 DHCP。
5. 运行 OVN trace。

**ovn-trace** 命令可以模拟数据包如何通过 OVN 逻辑流传输，或者帮助您确定数据包被丢弃的原因。使用以下参数提供 **ovn-trace** 命令：

#### DATAPATH

模拟数据包启动的逻辑交换机或逻辑路由器。

#### MICROFLOW

模拟的数据包，使用 **ovn-sb** 数据库使用的语法。

#### 示例

本例在模拟的数据包中显示 **--minimal** 输出选项，并显示数据包到达其目的地：

```
$ ovn-trace --minimal sw0 'inport == "sw0-port1" && eth.src == 00:00:00:00:00:01 &&
eth.dst == 00:00:00:00:00:02'
```

#### 输出示例

```
#
reg14=0x1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,dl_type=0x
0000
output("sw0-port2");
```

#### 示例

如需了解更多详细信息，这个相同的模拟数据包的 **--summary** 输出会显示完整的执行管道：

```
$ ovn-trace --summary sw0 'inport == "sw0-port1" && eth.src == 00:00:00:00:00:01 &&
eth.dst == 00:00:00:00:00:02'
```

#### 输出示例

示例输出显示：

- 数据包从 **sw0-port1** 端口输入 **sw0** 网络并运行 ingress 管道。

- **output** 变量设置为 **sw0-port2**，表示此数据包的预期目的地为 **sw0-port2**。
- 数据包是来自 ingress 管道的输出，它会将其引入 **sw0** 的出口管道，并将 **output** 变量设置为 **sw0-port2**。
- 输出操作在出口管道中执行，它将数据包输出到 **output** 变量的当前值，即 **sw0-port2**。

```
#
reg14=0x1,vlan_tci=0x0000,dl_src=00:00:00:00:00:01,dl_dst=00:00:00:00:00:02,dl_type
=0x0000
ingress(dp="sw0", inport="sw0-port1") {
    output = "sw0-port2";
    output;
    egress(dp="sw0", inport="sw0-port1", output="sw0-port2") {
        output;
        /* output to "sw0-port2", type "" */;
    };
};
```

### 其他资源

- [第 6.8 节 “为 OVN 故障排除命令创建别名”](#)
- **OVN-sbctl --help** 命令
- **OVN-trace --help** 命令

## 6.10. 监控 OPENFLOWS

您可以使用 **ovs-ofctl dump-flows** 命令监控 Red Hat Openstack Platform (RHOSP) 网络中逻辑交换机上的 OpenFlow 流。

### 先决条件

- 使用 ML2/OVN 作为网络服务(neutron)默认机制驱动程序的 RHOSP 部署。

### 流程

1. 以具有访问 OVN 容器所需的权限的用户身份登录到 Controller 主机。

#### 示例

```
$ ssh heat-admin@controller-0.ctlplane
```

2. 运行 **ovs-ofctl dump-flows** 命令。

#### 示例

```
$ sudo ovs-ofctl dump-flows br-int
```

3. 检查输出，类似于以下输出。

#### 输出示例

```
$ ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=72.132s, table=0, n_packets=0, n_bytes=0, idle_age=72,
  priority=10,in_port=1,dl_src=00:00:00:00:00:01 actions=resubmit(,1)
  cookie=0x0, duration=60.565s, table=0, n_packets=0, n_bytes=0, idle_age=60,
  priority=10,in_port=2,dl_src=00:00:00:00:00:02 actions=resubmit(,1)
  cookie=0x0, duration=28.127s, table=0, n_packets=0, n_bytes=0, idle_age=28, priority=0
  actions=drop
  cookie=0x0, duration=13.887s, table=1, n_packets=0, n_bytes=0, idle_age=13,
  priority=0,in_port=1 actions=output:2
  cookie=0x0, duration=4.023s, table=1, n_packets=0, n_bytes=0, idle_age=4,
  priority=0,in_port=2 actions=output:1
```

## 其他资源

- **ovs-ofctl --help** 命令

## 6.11. 验证 ML2/OVN 部署

验证 Red Hat OpenStack Platform (RHOSP)部署上的 ML2/OVN 网络包括创建测试网络和子网，并执行诊断任务，如验证 specific 容器是否正在运行。

### 先决条件

- RHOSP 的新部署，使用 ML2/OVN 作为网络服务(neutron)默认机制驱动程序。
- 为 OVN 数据库命令创建别名文件。  
请参阅 [第 6.8 节“为 OVN 故障排除命令创建别名”](#)。

### 流程

1. 创建测试网络和子网。

```
NETWORK_ID=\
$(openstack network create internal_network | awk '/^ id/ {print $4}')

openstack subnet create internal_subnet \
--network $NETWORK_ID \
--dns-nameserver 8.8.8.8 \
--subnet-range 192.168.254.0/24
```

如果您遇到错误，请执行以下步骤。

2. 验证相关容器是否在 Controller 主机上运行：
  - a. 以具有访问 OVN 容器所需的权限的用户身份登录到 Controller 主机。

### 示例

```
$ ssh heat-admin@controller-0.ctlplane
```

- b. 输入以下命令：

```
sudo podman ps -a --format="{{.Names}}"|grep ovn
```

如以下示例所示，输出应该列出 OVN 容器：

### 输出示例

```
ovn-dbs-bundle-podman-0
ovn_dbs_init_bundle
ovn_controller
```

3. 验证相关容器是否在 Compute 主机上运行：

- a. 以具有访问 OVN 容器所需的权限的用户身份登录到 Compute 主机。

### 示例

```
$ ssh heat-admin@compute-0.ctlplane
```

- b. 输入以下命令：

```
$ sudo podman ps -a --format="{{.Names}}"|grep ovn
```

如以下示例所示，输出应该列出 OVN 容器：

### 输出示例

```
ovn_controller
ovn_metadata_agent
neutron-haproxy-ovnmeta-26f493a7-1141-416a-9288-f08ff45fccac
neutron-haproxy-ovnmeta-b81bd1f1-0ff4-4142-8706-0f295db3c3af
```

4. 检查日志文件中的错误消息。

```
grep -r ERR /var/log/containers/openvswitch/ /var/log/containers/neutron/
```

5. 提供别名文件以运行 OVN 数据库命令。

更多信息请参阅 [第 6.8 节“为 OVN 故障排除命令创建别名”](#)。

### 示例

```
$ source ~/ovn-alias.sh
```

6. 查询 northbound 和 southbound 数据库以检查响应。

### 示例

```
# ovn-nbctl show
# ovn-sbctl show
```

7. 尝试从同一第 2 层网络上的 OVN 元数据接口 ping 实例。

更多信息请参阅 [第 6.5 节“在 ML2/OVN 命名空间中执行基本 ICMP 测试”](#)。

8. 如果您需要联系红帽以获取支持，请执行此红帽解决方案中介绍的步骤，[如何收集红帽支持所需的所有日志以调查 OpenStack 问题](#)。

## 其他资源

- 命令行界面参考中的 [network create](#)
- *Command Line Interface Reference* 中的 [subnet create](#)
- [第 6.8 节 “为 OVN 故障排除命令创建别名”](#)
- **OVN-nbctl --help** 命令
- **OVN-sbctl --help** 命令

## 6.12. 为 ML2/OVN 设置日志记录模式

将 ML2/OVN 日志记录设置为 debug 模式以提供额外的故障排除信息。将 logging Back 设置为 info 模式，以便在不需要额外的调试信息时使用较少的磁盘空间。

### 先决条件

- 使用 ML2/OVN 作为默认机制驱动程序部署的 Red Hat OpenStack Platform。

### 流程

1. 登录到 Controller 或 Compute 节点，您要以具有访问 OVN 容器所需的权限的用户身份设置日志记录模式。

#### 示例

```
$ ssh heat-admin@controller-0.ctlplane
```

2. 设置 ML2/OVN 日志记录模式。

#### 调试日志记录模式

```
$ sudo podman exec -it ovn_controller ovn-appctl -t ovn-controller vlog/set dbg
```

#### info 日志记录模式

```
$ sudo podman exec -it ovn_controller ovn-appctl -t ovn-controller vlog/set info
```

### 验证

- 确认 **ovn-controller** 容器日志现在包含调试信息：

```
$ sudo grep DBG /var/log/containers/openvswitch/ovn-controller.log
```

#### 输出示例

您应该看到包含字符串 **|DBG|** 的最新日志消息：

```
2022-09-29T20:52:54.638Z|00170|vconn(ovn_pinctrl0)|DBG|unix:/var/run/openvswitch/br-int.mgmt: received: OFPT_ECHO_REQUEST (OF1.5) (xid=0x0): 0 bytes of payload
2022-09-29T20:52:54.638Z|00171|vconn(ovn_pinctrl0)|DBG|unix:/var/run/openvswitch/br-
```

```
int.mgmt: sent (Success): OFPT_ECHO_REPLY (OF1.5) (xid=0x0): 0 bytes of payload
```

- 确认 ovn-controller 容器日志包含类似如下的字符串：

```
...received request vlog/set["info"], id=0
```

## 其他资源

- [第 6.14 节 “ML2/OVN 日志文件”](#)

## 6.13. 修复无法在边缘站点上注册的 OVN 控制器

### 问题

Red Hat OpenStack Platform (RHOSP)边缘站点上的 OVN 控制器无法注册。



### 注意

这个错误可能会在从 *早期* RHOSP 版本-RHOSP 16.1.7 及更早版本或 RHOSP 16.2.0 更新的 RHOSP 16.2 ML2/OVN 部署中发生。

### 错误示例

遇到的错误类似如下：

```
2021-04-12T09:14:48.994Z|04754|ovsdb_idl|WARN|transaction error: {"details": "Transaction causes multiple rows in \"Encap\" table to have identical values (geneve and \"10.14.2.7\") for index on columns \"type\" and \"ip\". First row, with UUID 3973cad5-eb8a-4f29-85c3-c105d861c0e0, was inserted by this transaction. Second row, with UUID f06b71a8-4162-475b-8542-d27db3a9097a, existed in the database before this transaction and was not modified by the transaction.", "error": "constraint violation"}
```

### 原因

如果 **ovn-controller** 进程替换了主机名，它会注册另一个机箱条目，其中包括另一个 encap 条目。如需更多信息，请参阅 [BZ#1948472](#)。

### 解决方案

按照以下步骤解决这个问题：

1. 如果您还没有，请为此流程稍后使用的必要 OVN 数据库命令创建别名。  
如需更多信息，请参阅 [OVN 故障排除命令创建别名](#)。
2. 以具有访问 OVN 容器所需的权限的用户身份登录到 Controller 主机。

### 示例

```
$ ssh heat-admin@controller-0.ctlplane
```

3. 从 `/var/log/containers/openvswitch/ovn-controller.log` 获取 IP 地址
4. 确认 IP 地址正确：

```
ovn-sbctl list encap |grep -a3 <IP address from ovn-controller.log>
```

5. 删除包含 IP 地址的机箱：

```
ovn-sbctl chassis-del <chassis-id>
```

6. 检查 **Chassis\_Private** 表以确认已删除 chassis：

```
ovn-sbctl find Chassis_private chassis=""
```

7. 如果报告任何条目，使用以下命令删除它们：

```
$ ovn-sbctl destroy Chassis_Private <listed_id>
```

8. 重启以下容器：

- **tripleo\_ovn\_controller**
- **tripleo\_ovn\_metadata\_agent**

```
$ sudo systemctl restart tripleo_ovn_controller
$ sudo systemctl restart tripleo_ovn_metadata_agent
```

验证

- 确认 OVN 代理正在运行：

```
$ openstack network agent list -c "Agent Type" -c State -c Binary
```

输出示例

```
+-----+-----+-----+
| Agent Type          | State | Binary          |
+-----+-----+-----+
| OVN Controller Gateway agent | UP    | ovn-controller  |
| OVN Controller Gateway agent | UP    | ovn-controller  |
| OVN Controller agent      | UP    | ovn-controller  |
| OVN Metadata agent       | UP    | neutron-ovn-metadata-agent |
| OVN Controller Gateway agent | UP    | ovn-controller  |
+-----+-----+-----+
```

### 6.14. ML2/OVN 日志文件

日志文件跟踪与 ML2/OVN 机制驱动程序的部署和操作相关的事件。

表 6.1. 每个节点的 ML2/OVN 日志文件

节点	Log	路径 /var/log/containers/openvswi tch...
Controller, Compute, Networking	OVS 北向数据库服务器	.../ovn-controller.log



节点	Log	路径 /var/log/containers/openvswi tch...
Controller	OVS 北向数据库服务器	.../ovsdb-server-nb.log
Controller	OVS 南向数据库服务器	.../ovsdb-server-sb.log
Controller	OVN 北向数据库服务器	.../ovn-northd.log

## 第 7 章 为 OPENSTACK 网络配置物理交换机

本章记录了 OpenStack 网络所需的常见物理交换机配置步骤。为特定交换机包括特定于供应商的配置。

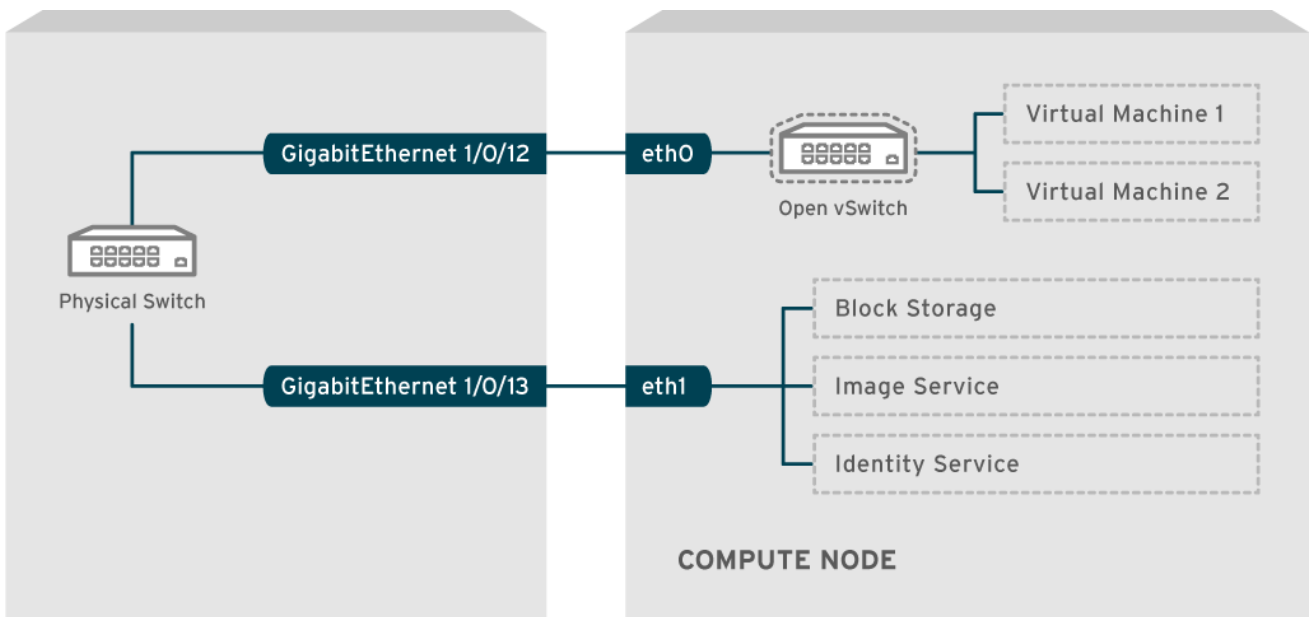
### 7.1. 规划您的物理网络环境

OpenStack 节点上的物理网络适配器具有不同类型的网络流量，如实例流量、存储数据或身份验证请求。这些 NIC 执行的流量类型会影响您必须在物理交换机上配置端口。

首先，您必须决定要执行哪个类型的流量的其他 Compute 节点的物理 NIC。然后，当 NIC 线到物理交换机端口时，您必须配置交换机端口以允许中继或常规流量。

例如，下图描述了有两个 NIC 为 eth0 和 eth1 的 Compute 节点。每个 NIC 都连接到物理交换机上的 Gigabit Ethernet 端口，eth0 执行实例流量，eth1 为 OpenStack 服务提供连接：

图 7.1. 网络布局示例



OPENSTACK\_377160\_1115



#### 注意

此图不包含容错所需的额外冗余 NIC。

#### 其他资源

高级 Overcloud 自定义指南中的[网络接口绑定](#)

### 7.2. 配置 CISCO CATALYST 交换机

#### 7.2.1. 关于中继端口

通过 OpenStack 网络，您可以将实例连接到物理网络中已存在的 VLAN。术语 *中继* 用于描述允许多个 VLAN 遍历同一端口的端口。使用这些端口，VLAN 可以跨越多个交换机，包括虚拟交换机。例如，物理网络中标记为 VLAN110 的流量到达 Compute 节点，其中 8021q 模块将标记的流量定向到 vSwitch 上的相应 VLAN。

## 7.2.2. 为 Cisco Catalyst 交换机配置中继端口

- 如果使用运行 Cisco IOS 的 Cisco Catalyst 交换机，您可以使用以下配置语法来允许 VLAN 110 和 111 的流量传递给您的实例。  
在此配置中，假定您的物理节点有一个以太网电缆连接到物理交换机上的 GigabitEthernet1/0/12 接口 GigabitEthernet1/0/12。



### 重要

这些值是示例。您必须更改本例中的值以匹配环境中的值。在不调整的情况下将这些值复制并粘贴到交换机配置中，可能会导致意外中断。

```
interface GigabitEthernet1/0/12
description Trunk to Compute Node
spanning-tree portfast trunk
switchport trunk encapsulation dot1q
switchport mode trunk
switchport trunk native vlan 2
switchport trunk allowed vlan 2,110,111
```

使用以下列表了解这些参数：

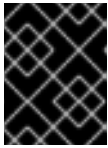
字段	描述
<b>interface GigabitEthernet1/0/12</b>	X 节点连接到的 NIC 的交换机端口。确保将 <b>GigabitEthernet1/0/12</b> 值替换为您的环境的正确端口值。使用 show interface 命令查看端口列表。
<b>Compute 节点截断的描述</b>	可用于识别此接口的唯一描述性值。
<b>spanning-tree portfast trunk</b>	如果您的环境使用 STP，请将此值设置为指示此端口用于中继流量的端口。
<b>switchport trunk 封装点 1q</b>	启用 802.1q 中继标准（而不是 ISL）。这个值因交换机支持的配置而异。
<b>switchport 模式中继</b>	将此端口配置为中继端口，而不是访问端口，这意味着它允许 VLAN 流量传递给虚拟交换机。
<b>switchport trunk native vlan 2</b>	设置一个原生 VLAN，以指示发送未标记（非 VLAN）流量的交换机。
<b>switchport trunk 允许 vlan 2,110,111</b>	定义允许通过中继的 VLAN。

## 7.2.3. 关于访问端口

不是 Compute 节点上的所有 NIC 都执行实例流量，因此您不需要配置所有 NIC 来允许多个 VLAN 传递。访问端口只需要一个 VLAN，并可能满足其他操作要求，如传输管理流量或块存储数据。这些端口通常称为访问端口，通常需要比中继端口更简单的配置。

## 7.2.4. 为 Cisco Catalyst 交换机配置访问端口

- 使用图 7.1 “网络布局示例” 图中的示例，GigabitEthernet1/0/13（在 Cisco Catalyst 交换机中）被配置为 **eth1** 的访问端口。  
在此配置中，您的物理节点有一个以太网电缆连接到物理交换机上的 GigabitEthernet1/0/12 接口 GigabitEthernet1/0/12。



### 重要

这些值是示例。您必须更改本例中的值以匹配环境中的值。在不调整的情况下将这些值复制并粘贴到交换机配置中，可能会导致意外中断。

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

这些设置如下所述：

字段	描述
<b>interface GigabitEthernet1/0/13</b>	X 节点连接到的 NIC 的交换机端口。确保将 <b>GigabitEthernet1/0/12</b> 值替换为您的环境的正确端口值。使用 <code>show interface</code> 命令查看端口列表。
<b>Compute 节点的描述访问端口</b>	可用于识别此接口的唯一描述性值。
<b>switchport 模式访问</b>	将此端口配置为访问端口，而不是中继端口。
<b>switchport access vlan 200</b>	配置端口以允许 VLAN 200 上的流量。您必须使用此 VLAN 的 IP 地址配置您的 Compute 节点。
<b>spanning-tree portfast</b>	如果使用 STP，则将此值设置为指示 STP 不尝试将其初始化为中继，从而加快初始连接期间的端口握手（如服务器重启）。

## 7.2.5. 关于 LACP 端口聚合

您可以使用链路聚合控制协议(LACP)将多个物理 NIC 组合在一起组成单个逻辑频道。LACP 也称为 802.3ad（或 Linux 中的绑定模式 4），LACP 为负载平衡和容错创建一个动态绑定。您必须在物理 NIC 和物理交换机端口上配置 LACP。

### 其他资源

高级 *Overcloud* 自定义指南中的[网络接口绑定](#)。

## 7.2.6. 在物理 NIC 中配置 LACP

您可以在物理 NIC 上配置链路聚合控制协议(LACP)。

## 流程

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. 将 Open vSwitch 网桥配置为使用 LACP：

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

## 其他资源

高级 *Overcloud* 自定义指南中的[网络接口绑定](#)

### 7.2.7. 为 Cisco Catalyst 交换机配置 LACP

在本例中，Compute 节点有两个使用 VLAN 100 的 NIC：

## 流程

1. Compute 节点上的物理连接 NIC 到交换机（例如，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 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
```

- 查看您的新端口频道。生成的输出列出了新的 port-channel **Po1**，成员端口 **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

请记住，通过将 running-config 复制到 startup-config: **copy running-config startup-config** 来应用更改。

## 7.2.8. 关于 MTU 设置

您必须为某些类型的网络流量调整 MTU 大小。例如，某些 NFS 或 iSCSI 流量需要巨型帧(9000 字节)。



### 注意

您必须在流量要通过的所有跃点（包括任何虚拟交换机）上的端到端更改 MTU 设置。

### 其他资源

- [配置最大传输单元\(MTU\)设置](#)

## 7.2.9. 为 Cisco Catalyst 交换机配置 MTU 设置

完成本示例流程中的步骤，在 Cisco Catalyst 3750 交换机上启用巨型帧。

- 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 设置在 3750 交换机上更改交换机范围，而不适用于单个接口。运行以下命令，将交换机配置为使用 9000 字节的巨型帧。如果您的交换机支持此功能，您可能希望为单个接口配置 MTU 设置。

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

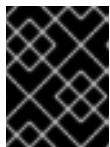
sw01(config)# system mtu jumbo 9000
Changes to the system jumbo MTU will not take effect until the next reload is done
```



### 注意

请记住，通过将 running-config 复制到 startup-config: **copy running-config startup-config** 来保存您的更改。

3. 重新加载开关以应用更改。



### 重要

重新载入交换机会导致对依赖交换机的任何设备造成网络中断。因此，仅在调度的维护期间重新载入交换机。

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

4. 在交换机重新加载后，确认新的巨型 MTU 大小。确切的输出可能因您的交换机模型而异。例如，**系统 MTU** 可能会应用到非千兆接口，**Jumbo MTU** 可能会描述所有 Gigabit 接口。

```
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
```

## 7.2.10. 关于 LLDP 发现

**ironic-python-agent** 服务侦听来自连接的交换机的 LLDP 数据包。收集的信息可以包含交换机名称、端口详情和可用 VLAN。与 Cisco Discovery Protocol (CDP) 类似，g LLDP 有助于在 director 内省过程中发现物理硬件。

## 7.2.11. 为 Cisco Catalyst 交换机配置 LLDP

### 流程

1. 运行 **lldp run** 命令，在您的 Cisco Catalyst 交换机上全局启用 LLDP：

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

sw01(config)# lldp run
```

## 2. 查看任何与 LLDP 兼容的设备：

```
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
DEP42037061562G3  Gi1/0/11   180       B,T        422037061562G3:P1

Total entries displayed: 1
```



### 注意

请记住，通过将 `running-config` 复制到 `startup-config`: **copy running-config startup-config** 来保存您的更改。

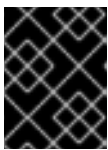
## 7.3. 配置 CISCO NEXUS 交换机

### 7.3.1. 关于中继端口

通过 OpenStack 网络，您可以将实例连接到物理网络中已存在的 VLAN。术语 *中继* 用于描述允许多个 VLAN 遍历同一端口的端口。使用这些端口，VLAN 可以跨越多个交换机，包括虚拟交换机。例如，物理网络中标记为 VLAN110 的流量到达 Compute 节点，其中 8021q 模块将标记的流量定向到 vSwitch 上的相应 VLAN。

### 7.3.2. 为 Cisco Nexus 交换机配置中继端口

- 如果使用 Cisco Nexus，您可以使用以下配置语法来允许 VLAN 110 和 111 的流量传递给您的实例。  
此配置假设您的物理节点有一个以太网电缆连接到物理交换机上的接口 **Ethernet1/12**。



### 重要

这些值是示例。您必须更改本例中的值以匹配环境中的值。在不调整的情况下将这些值复制并粘贴到交换机配置中，可能会导致意外中断。

```
interface Ethernet1/12
  description Trunk to Compute Node
  switchport mode trunk
  switchport trunk allowed vlan 2,110,111
  switchport trunk native vlan 2
end
```

### 7.3.3. 关于访问端口

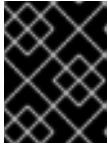
不是 Compute 节点上的所有 NIC 都执行实例流量，因此您不需要配置所有 NIC 来允许多个 VLAN 传递。访问端口只需要一个 VLAN，并可能满足其他操作要求，如传输管理流量或块存储数据。这些端口通常称为访问端口，通常需要比中继端口更简单的配置。



### 7.3.4. 为 Cisco Nexus 交换机配置访问端口

#### 流程

- 使用 图 7.1 “网络布局示例” 图中的示例，Ethernet1/13（在 Cisco Nexus 交换机中）被配置为 eth1 的访问端口。此配置假设您的物理节点有一个以太网电缆连接到物理交换机上的接口 Ethernet1/13。



#### 重要

这些值是示例。您必须更改本例中的值以匹配环境中的值。在不调整的情况下将这些值复制并粘贴到交换机配置中，可能会导致意外中断。

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

### 7.3.5. 关于 LACP 端口聚合

您可以使用链路聚合控制协议(LACP)将多个物理 NIC 组合在一起组成单个逻辑频道。LACP 也称为 802.3ad（或 Linux 中的绑定模式 4），LACP 为负载平衡和容错创建一个动态绑定。您必须在物理 NIC 和物理交换机端口上配置 LACP。

#### 其他资源

高级 *Overcloud 自定义指南* 中的 [网络接口绑定](#)。

### 7.3.6. 在物理 NIC 中配置 LACP

您可以在物理 NIC 上配置链路聚合控制协议(LACP)。

#### 流程

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. 将 Open vSwitch 网桥配置为使用 LACP：

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

## 其他资源

高级 *Overcloud* 自定义指南中的[网络接口绑定](#)

### 7.3.7. 为 Cisco Nexus 交换机配置 LACP

在本例中，Compute 节点有两个使用 VLAN 100 的 NIC：

#### 流程

1. 物理将 Compute 节点 NIC 连接到交换机（例如，端口 12 和 13）。
2. 确认启用了 LACP：

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

3. 将端口 1/12 和 1/13 配置为访问端口，并作为频道组的成员。  
根据您的部署，您可以部署中继接口，而不是访问接口。

例如，对于 Cisco UCI，NIC 是虚拟接口，因此您可能更喜欢专门配置访问端口。通常，这些接口包含 VLAN 标记配置。

```
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
```



#### 注意

当您使用 PXE 在 Cisco 交换机上置备节点时，您可能需要设置选项 **no lacp graceful-convergence**，且没有 **lacp suspend-individual** 来启动端口并引导服务器。如需更多信息，请参阅 Cisco 交换机文档。

### 7.3.8. 关于 MTU 设置

您必须为某些类型的网络流量调整 MTU 大小。例如，某些 NFS 或 iSCSI 流量需要巨型帧(9000 字节)。



#### 注意

您必须在流量要通过的所有跃点（包括任何虚拟交换机）上的端到端更改 MTU 设置。

## 其他资源

- [配置最大传输单元\(MTU\)设置](#)

### 7.3.9. 为 Cisco Nexus 7000 交换机配置 MTU 设置

将 MTU 设置应用到 7000-series 交换机上的单个接口。

#### 流程

- 运行以下命令将接口 1/12 配置为使用 9000 字节的巨型帧：

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

### 7.3.10. 关于 LLDP 发现

**ironic-python-agent** 服务侦听来自连接的交换机的 LLDP 数据包。收集的信息可以包含交换机名称、端口详情和可用 VLAN。与 Cisco Discovery Protocol (CDP) 类似，g LLDP 有助于在 director 内省过程中发现物理硬件。

### 7.3.11. 为 Cisco Nexus 7000 交换机配置 LLDP

#### 流程

- 您可以为 Cisco Nexus 7000-series 交换机上的各个接口启用 LLDP：

```
interface ethernet 1/12
  lldp transmit
  lldp receive
  no lacp suspend-individual
  no lacp graceful-convergence

interface ethernet 1/13
  lldp transmit
  lldp receive
  no lacp suspend-individual
  no lacp graceful-convergence
```



#### 注意

请记住，通过将 running-config 复制到 startup-config: **copy running-config startup-config** 来保存您的更改。

## 7.4. 配置 CUMULUS LINUX 交换机

### 7.4.1. 关于中继端口

通过 OpenStack 网络，您可以将实例连接到物理网络中已存在的 VLAN。术语 *中继* 用于描述允许多个 VLAN 遍历同一端口的端口。使用这些端口，VLAN 可以跨越多个交换机，包括虚拟交换机。例如，物理网络中标记为 VLAN110 的流量到达 Compute 节点，其中 8021q 模块将标记的流量定向到 vSwitch 上的相应 VLAN。

### 7.4.2. 为 Cumulus Linux 交换机配置中继端口

此配置假设您的物理节点已转换，来切换物理交换机上的端口 swp1 和 swp2。



### 重要

这些值是示例。您必须更改本例中的值以匹配环境中的值。在不调整的情况下将这些值复制并粘贴到交换机配置中，可能会导致意外中断。

### 流程

- 使用以下配置语法，允许 VLAN 100 和 200 的流量传递给您的实例。

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

### 7.4.3. 关于访问端口

不是 Compute 节点上的所有 NIC 都执行实例流量，因此您不需要配置所有 NIC 来允许多个 VLAN 传递。访问端口只需要一个 VLAN，并可能满足其他操作要求，如传输管理流量或块存储数据。这些端口通常称为访问端口，通常需要比中继端口更简单的配置。

### 7.4.4. 为 Cumulus Linux 交换机配置访问端口

此配置假设您的物理节点有一个以太网电缆连接到物理交换机上的接口。Cumulus Linux 交换机将 **eth** 用于管理界面，**swp** 用于访问/中继端口。



### 重要

这些值是示例。您必须更改本例中的值以匹配环境中的值。在不调整的情况下将这些值复制并粘贴到交换机配置中，可能会导致意外中断。

### 流程

- 使用 [图 7.1 “网络布局示例”](#) 图中的示例，**swp1**（在 Cumulus Linux 交换机中）被配置为访问端口。

```
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
```

### 7.4.5. 关于 LACP 端口聚合

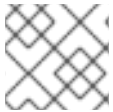
您可以使用链路聚合控制协议(LACP)将多个物理 NIC 组合在一起组成单个逻辑频道。LACP 也称为 802.3ad (或 Linux 中的绑定模式 4)，LACP 为负载平衡和容错创建一个动态绑定。您必须在物理 NIC 和物理交换机端口上配置 LACP。

#### 其他资源

高级 *Overcloud* 自定义指南中的[网络接口绑定](#)。

### 7.4.6. 关于 MTU 设置

您必须为某些类型的网络流量调整 MTU 大小。例如，某些 NFS 或 iSCSI 流量需要巨型帧(9000 字节)。



#### 注意

您必须在流量要通过的所有跃点（包括任何虚拟交换机）上的端到端更改 MTU 设置。

#### 其他资源

- [配置最大传输单元\(MTU\)设置](#)

### 7.4.7. 为 Cumulus Linux 交换机配置 MTU 设置

#### 流程

- 这个示例在您的 Cumulus Linux 交换机上启用巨型帧。

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



#### 注意

请记住，通过重新载入更新的配置来应用您的更改：**sudo ifreload -a**

### 7.4.8. 关于 LLDP 发现

**ironic-python-agent** 服务侦听来自连接的交换机的 LLDP 数据包。收集的信息可以包含交换机名称、端口详情和可用 VLAN。与 Cisco Discovery Protocol (CDP)类似，g LLDP 有助于在 director 内省过程中发现物理硬件。

### 7.4.9. 为 Cumulus Linux 交换机配置 LLDP

默认情况下，LLDP 服务 lldpd 作为守护进程运行，在切换引导时启动。

#### 流程

- 要查看所有端口/接口上的所有 LLDP 邻居，请运行以下命令：

```
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
```

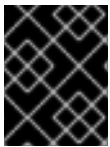
## 7.5. 配置 EXTREME EXOS 交换机

### 7.5.1. 关于中继端口

通过 OpenStack 网络，您可以将实例连接到物理网络中已存在的 VLAN。术语 *中继* 用于描述允许多个 VLAN 遍历同一端口的端口。使用这些端口，VLAN 可以跨越多个交换机，包括虚拟交换机。例如，物理网络中标记为 VLAN110 的流量到达 Compute 节点，其中 8021q 模块将标记的流量定向到 vSwitch 上的相应 VLAN。

### 7.5.2. 在 Extreme Networks EXOS 交换机中配置中继端口

如果使用 X-670 系列开关，请参考以下示例，以允许 VLAN 110 和 111 的流量传递给您的实例。



#### 重要

这些值是示例。您必须更改本例中的值以匹配环境中的值。在不调整的情况下将这些值复制并粘贴到交换机配置中，可能会导致意外中断。

#### 流程

- 此配置假设您的物理节点有一个以太网电缆连接到物理交换机上的接口 24。在本例中，DATA 和 MNGT 是 VLAN 名称。

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

### 7.5.3. 关于访问端口

不是 Compute 节点上的所有 NIC 都执行实例流量，因此您不需要配置所有 NIC 来允许多个 VLAN 传递。访问端口只需要一个 VLAN，并可能满足其他操作要求，如传输管理流量或块存储数据。这些端口通常称为访问端口，通常需要比中继端口更简单的配置。

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

此配置假设您的物理节点有一个以太网电缆连接到物理交换机上的接口 10。



#### 重要

这些值是示例。您必须更改本例中的值以匹配环境中的值。在不调整的情况下将这些值复制并粘贴到交换机配置中，可能会导致意外中断。

#### 流程

- 在此配置中，在 Extreme Networks X-670 系列交换机上，**10** 用作 **eth1** 的访问端口。

```
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
```

### 7.5.5. 关于 LACP 端口聚合

您可以使用链路聚合控制协议(LACP)将多个物理 NIC 组合在一起组成单个逻辑频道。LACP 也称为 802.3ad（或 Linux 中的绑定模式 4），LACP 为负载平衡和容错创建一个动态绑定。您必须在物理 NIC 和物理交换机端口上配置 LACP。

#### 其他资源

高级 *Overcloud* 自定义指南中的[网络接口绑定](#)。

### 7.5.6. 在物理 NIC 中配置 LACP

您可以在物理 NIC 上配置链路聚合控制协议(LACP)。

#### 流程

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. 将 Open vSwitch 网桥配置为使用 LACP：

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

#### 其他资源

高级 *Overcloud* 自定义指南中的[网络接口绑定](#)

### 7.5.7. 在 Extreme Networks EXOS 交换机中配置 LACP

...

## 流程

- 在本例中，Compute 节点有两个使用 VLAN 100 的 NIC：

```
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
```

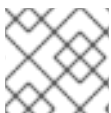


### 注意

您可能需要调整 LACP 协商脚本中的超时时间。如需更多信息，请参阅 [https://gtacknowledge.extremenetworks.com/articles/How\\_To/LACP-configured-ports-interfere-with-PXE-DHCP-on-servers](https://gtacknowledge.extremenetworks.com/articles/How_To/LACP-configured-ports-interfere-with-PXE-DHCP-on-servers)

## 7.5.8. 关于 MTU 设置

您必须为某些类型的网络流量调整 MTU 大小。例如，某些 NFS 或 iSCSI 流量需要巨型帧(9000 字节)。



### 注意

您必须在流量要通过的所有跃点（包括任何虚拟交换机）上的端到端更改 MTU 设置。

## 其他资源

- [配置最大传输单元\(MTU\)设置](#)

## 7.5.9. 在 Extreme Networks EXOS 交换机中配置 MTU 设置

### 流程

- 在本示例中运行命令以在 Extreme Networks EXOS 交换机上启用巨型帧，并配置支持使用 9000 字节转发 IP 数据包：

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

### 示例

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

## 7.5.10. 关于 LLDP 发现

**ironic-python-agent** 服务侦听来自连接的交换机的 LLDP 数据包。收集的信息可以包含交换机名称、端口详情和可用 VLAN。与 Cisco Discovery Protocol (CDP)类似，g LLDP 有助于在 director 内省过程中发现物理硬件。

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



## 流程

- 在本例中，在 Extreme Networks EXOS 交换机上启用了 LLDP。11 代表端口字符串：

```
enable lldp ports 11
```

## 7.6. 配置 JUNIPER EX 系列交换机

### 7.6.1. 关于中继端口

通过 OpenStack 网络，您可以将实例连接到物理网络中已存在的 VLAN。术语 *中继* 用于描述允许多个 VLAN 遍历同一端口的端口。使用这些端口，VLAN 可以跨越多个交换机，包括虚拟交换机。例如，物理网络中标记为 VLAN110 的流量到达 Compute 节点，其中 8021q 模块将标记的流量定向到 vSwitch 上的相应 VLAN。

### 7.6.2. 为 Juniper EX 系列交换机配置中继端口

#### 流程

- 如果使用运行 Juniper JunOS 的 Juniper EX 系列交换机，请使用以下配置语法来允许 VLAN 110 和 111 的流量传递给您的实例。  
此配置假设您的物理节点有一个以太网电缆连接到物理交换机上的 ge-1/0/12 接口。



#### 重要

这些值是示例。您必须更改本例中的值以匹配环境中的值。在不调整的情况下将这些值复制并粘贴到交换机配置中，可能会导致意外中断。

```
ge-1/0/12 {
  description Trunk to Compute Node;
  unit 0 {
    family ethernet-switching {
      port-mode trunk;
      vlan {
        members [110 111];
      }
      native-vlan-id 2;
    }
  }
}
```

### 7.6.3. 关于访问端口

不是 Compute 节点上的所有 NIC 都执行实例流量，因此您不需要配置所有 NIC 来允许多个 VLAN 传递。访问端口只需要一个 VLAN，并可能满足其他操作要求，如传输管理流量或块存储数据。这些端口通常称为访问端口，通常需要比中继端口更简单的配置。

### 7.6.4. 为 Juniper EX 系列交换机配置访问端口

例如，Juniper EX 系列交换机显示 **ge-1/0/13** 作为 **eth1** 的访问端口。

+



## 重要

这些值是示例。您必须更改本例中的值以匹配环境中的值。在不调整的情况下将这些值复制并粘贴到交换机配置中，可能会导致意外中断。

## 流程

此配置假设您的物理节点有一个以太网电缆连接到物理交换机上的 **ge-1/0/13** 接口。

+

```

ge-1/0/13 {
  description Access port for Compute Node
  unit 0 {
    family ethernet-switching {
      port-mode access;
      vlan {
        members 200;
      }
      native-vlan-id 2;
    }
  }
}

```

### 7.6.5. 关于 LACP 端口聚合

您可以使用链路聚合控制协议(LACP)将多个物理 NIC 组合在一起组成单个逻辑频道。LACP 也称为 802.3ad（或 Linux 中的绑定模式 4），LACP 为负载平衡和容错创建一个动态绑定。您必须在物理 NIC 和物理交换机端口上配置 LACP。

#### 其他资源

高级 *Overcloud* 自定义指南中的[网络接口绑定](#)。

### 7.6.6. 在物理 NIC 中配置 LACP

您可以在物理 NIC 上配置链路聚合控制协议(LACP)。

## 流程

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. 将 Open vSwitch 网桥配置为使用 LACP :

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

## 其他资源

高级 Overcloud 自定义指南中的[网络接口绑定](#)

### 7.6.7. 为 Juniper EX 系列交换机配置 LACP

在本例中，Compute 节点有两个使用 VLAN 100 的 NIC。

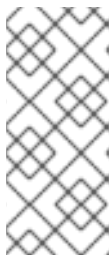
## 流程

1. 物理将 Compute 节点的两个 NIC 连接到交换机（例如，端口 12 和 13）。
2. 创建端口聚合：

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

3. 配置交换机端口 12 (ge-1/0/12 和 13 (ge-1/0/13))来加入端口聚合 **ae1** :

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



## 注意

对于 Red Hat OpenStack Platform director 部署，若要 PXE 从绑定引导，您必须将其中一个绑定成员配置为 lACP 强制启动，使得只有一个绑定成员在内省和第一次引导过程中启动。使用 lACP 强制配置的绑定成员必须与在 *instackenv.json* 中 MAC 地址的绑定成员相同(ironic 已知的 MAC 地址必须是使用 force-up 配置的同 MAC 地址)。

4. 在端口聚合 **ae1** 上启用 LACP :

```
interfaces {
```

```

ae1 {
  aggregated-ether-options {
    lacp {
      active;
    }
  }
}

```

5. 将聚合 **ae1** 添加到 VLAN 100 中：

```

interfaces {
  ae1 {
    vlan-tagging;
    native-vlan-id 2;
    unit 100 {
      vlan-id 100;
    }
  }
}

```

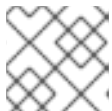
6. 查看您的新端口频道。生成的输出列出了新的端口聚合 **ae1**，其成员端口为 **ge-1/0/12** 和 **ge-1/0/13**：

```

> 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

请记住，通过运行 **commit** 命令应用您的更改。

## 7.6.8. 关于 MTU 设置

您必须为某些类型的网络流量调整 MTU 大小。例如，某些 NFS 或 iSCSI 流量需要巨型帧(9000 字节)。



### 注意

您必须在流量要通过的所有跃点（包括任何虚拟交换机）上的端到端更改 MTU 设置。

### 其他资源

- [配置最大传输单元\(MTU\)设置](#)

## 7.6.9. 为 Juniper EX 系列交换机配置 MTU 设置

这个示例在 Juniper EX4200 交换机上启用巨型帧。



## 注意

MTU 值会根据您是否使用 Juniper 或 Cisco 设备的不同而有所不同。例如，对于 Cisco，Juniper 上的 **9216** 等于 **9202**。额外的字节用于 L2 标头，其中 Cisco 会自动将此值添加到指定的 MTU 值，但在使用 Juniper 时可用的 MTU 将小于指定的 14 字节。因此，为了支持 VLAN 上 **9000** 的 MTU，必须在 Juniper 上配置 **9014** 的 MTU。

## 流程

1. 对于 Juniper EX 系列交换机，为单个接口设置 MTU 设置。这些命令在 **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

请记住，通过运行 **commit** 命令保存您的更改。

2. 如果使用 LACP 聚合，则需要其中设置 MTU 大小，而不是在成员 NIC 中设置。例如，此设置为 ae1 聚合配置 MTU 大小：

```
set interfaces ae1 mtu 9216
```

### 7.6.10. 关于 LLDP 发现

**ironic-python-agent** 服务侦听来自连接的交换机的 LLDP 数据包。收集的信息可以包含交换机名称、端口详情和可用 VLAN。与 Cisco Discovery Protocol (CDP) 类似，g LLDP 有助于在 director 内省过程中发现物理硬件。

### 7.6.11. 为 Juniper EX 系列交换机配置 LLDP

您可以全局启用所有接口的 LLDP，或只为单个接口启用 LLDP。

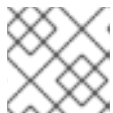
## 流程

- 在 Juniper EX 4200 交换机中使用以下太多启用 LLDP：

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

- 使用以下内容作为单个接口 **ge-1/0/14** 启用 LLDP：

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



### 注意

请记住，通过运行 **commit** 命令应用您的更改。

## 第 8 章 配置最大传输单元(MTU)设置

### 8.1. MTU 概述

OpenStack 网络可以计算您可以安全地应用到实例的最大最大传输单元(MTU)大小。MTU 值指定单个网络数据包可以传输的最大数据量；这个数字是根据应用程序最合适的大小的变量。例如，NFS 共享可能需要不同的 MTU 大小与 IANA 应用程序的不同 MTU 大小。

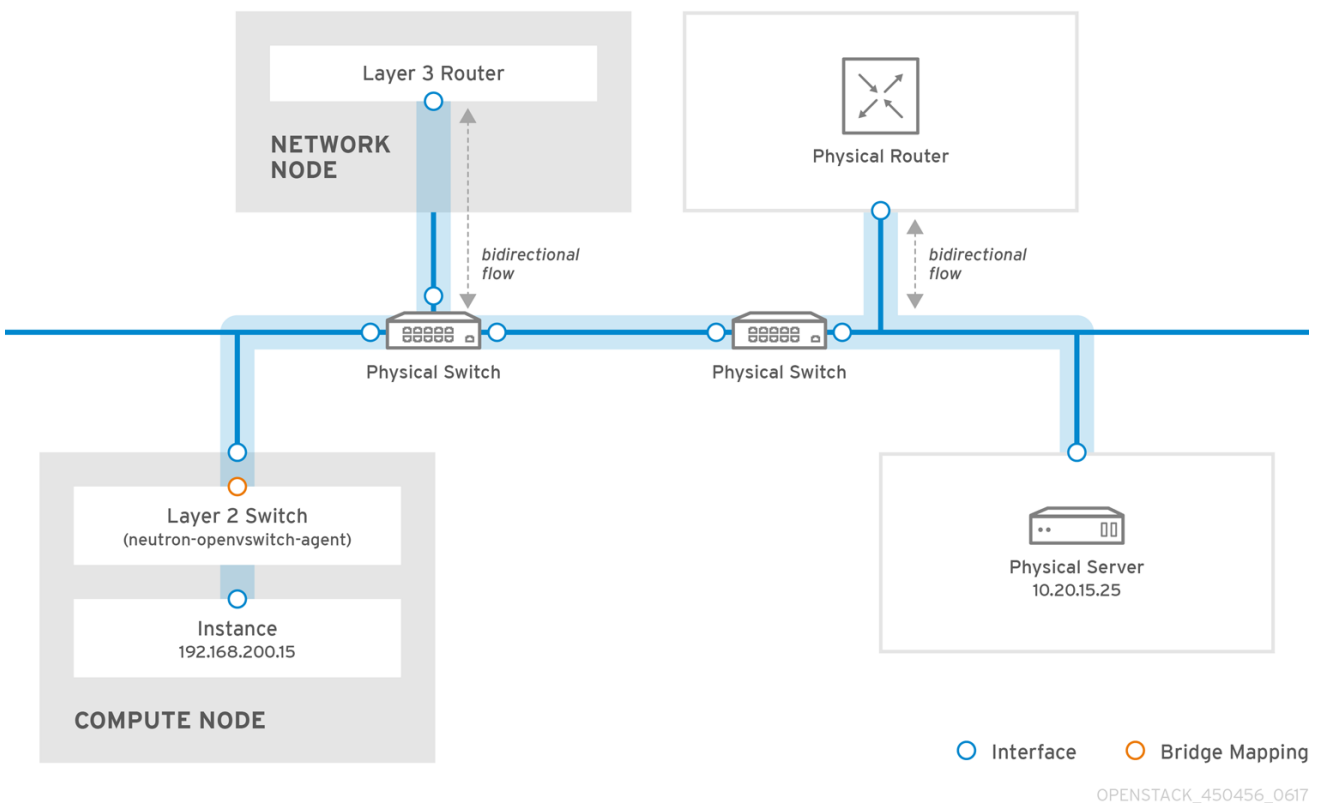


#### 注意

您可以使用 `openstack network show <network_name>` 命令查看 OpenStack 网络计算的最大可能 MTU 值。`net-mtu` 是一个 neutron API 扩展，在某些实现中不存在。您需要的 MTU 值可以公告到 DHCPv4 客户端，如果实例支持，以及通过路由器广告(RA)数据包的 IPv6 客户端。要发送路由器广告，必须将网络附加到路由器。

您必须从端到端配置 MTU 设置。这意味着 MTU 设置在每个数据包通过时必须相同，包括虚拟机、虚拟网络基础架构、物理网络和目标服务器。

例如，下图中的 circles 表示必须为实例和物理服务器之间的流量调整 MTU 值的各种点。您必须为处理网络流量的非常接口更改 MTU 值，以适应特定 MTU 大小的数据包。如果流量从实例 `192.168.200.15` 到物理服务器 `10.20.15.25`，则需要此项：



不一致的 MTU 值可能会导致几个网络问题，最常见的是随机数据包丢失，从而导致连接丢弃和减慢网络性能。这些问题会出现问题，因为您必须识别并检查每个可能的网络点以确保它具有正确的 MTU 值。

### 8.2. 在 DIRECTOR 中配置 MTU 设置

本例演示了如何使用 NIC 配置模板设置 MTU。您必须在网桥、绑定（如果适用）、接口和 VLAN 上设置 MTU：

```

-
  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}

```

### 8.3. 查看生成的 MTU 计算

您可以查看计算的 MTU 值，这是实例可以使用的最大可能 MTU 值。使用这个计算的 MTU 值来配置网络流量路径中涉及的所有接口。

```
# openstack network show <network>
```



## 第 9 章 使用服务质量(QoS)策略来管理数据流量

您可以使用服务质量(QoS)策略为虚拟机实例提供不同的服务级别，以将速率限制应用到 Red Hat OpenStack Platform (RHOSP)网络上的出口和入口流量。

您可以将 QoS 策略应用到单独的端口，或将 QoS 策略应用到项目网络，其中没有附加特定策略的端口会继承策略。



### 注意

DHCP 和内部路由器端口等内部网络拥有的端口不包括在网络策略应用程序中。

您可以动态应用、修改或删除 QoS 策略。但是，为了保证最小带宽 QoS 策略，您只能在没有实例使用该策略分配的任何端口时才应用修改。

### 9.1. QoS 规则

您可以配置以下规则类型，以在 Red Hat OpenStack Platform (RHOSP) Networking 服务(neutron)中定义服务质量(QoS)策略：

#### Minimum bandwidth (minimum\_bandwidth)

对某些类型的流量提供最小带宽限制。如果实施，则会将最佳努力减少到应用该规则的每个端口的指定带宽。

#### 带宽限制(bandwidth\_limit)

对网络、端口、浮动 IP 和路由器网关 IP 提供带宽限制。如果实现，则超过指定速率的任何流量都会被丢弃。

#### DSCP marking (dscp\_marking)

使用不同的服务代码点(DSCP)值标记网络流量。

可在各种上下文中强制执行 QoS 策略，包括虚拟机实例放置、浮动 IP 分配和网关 IP 分配。

根据强制上下文以及您使用的机制驱动程序，QoS 规则会影响出口流量（从实例上传）、入口流量（下载到实例）或两者。

表 9.1. 按驱动程序支持的流量方向（所有 QoS 规则类型）

规则 [8]	按机制驱动程序支持的流量方向		
	ML2/OVS	ML2/SR-IOV	ML2/OVN
最小带宽	仅出口 [4][5]	仅限出口	目前，不支持 [6]
带宽限制	Egress [1][2] 和 ingress	仅出口 [3]	Egress 和 ingress
账单标记	仅限出口	N/A	仅限 Egress[7]

[1] OVS 出口带宽限制在 sVirt 接口中执行，并且流量策略，而不是流量 shaping。

[2] 在 RHOSP 16.2.2 及更高版本中，通过使用 **ip link** 命令应用网络接口中的 QoS 策略，在硬件卸载端口上支持 OVS 出口带宽限制。

[3] 机制驱动程序忽略 **max-burst-kbits** 参数，因为它们不支持它。

[4] 规则只适用于非隧道网络：扁平化和 VLAN。

[5] 硬件卸载端口上支持 OVS 出口最小带宽，方法是使用 **ip link** 命令应用网络接口中的 QoS 策略。

[6] [https://bugzilla.redhat.com/show\\_bug.cgi?id=2060310](https://bugzilla.redhat.com/show_bug.cgi?id=2060310)

[7] ML2/OVN 不支持在隧道协议中的 DSCP 标记。

[8] RHOSP 不支持中继端口的 QoS。

表 9.2. 用于放置报告和调度的驱动程序支持的流量方向（仅限最小带宽）

强制类型	按方向机制驱动程序支持的流量		
	ML2/OVS	ML2/SR-IOV	ML2/OVN
Placement	Egress 和 ingress	Egress 和 ingress	目前，不支持

表 9.3. 支持驱动程序用于强制类型的流量方向（仅限带宽限制）

强制类型	按机制驱动程序支持的流量方向	
	ML2/OVS	ML2/OVN
浮动 IP	Egress 和 ingress	Egress 和 ingress
网关 IP	Egress 和 ingress	目前，不支持 [1]

[1] [https://bugzilla.redhat.com/show\\_bug.cgi?id=2064185](https://bugzilla.redhat.com/show_bug.cgi?id=2064185)

## 其他资源

- [创建并应用带宽限制 QoS 策略和规则](#)
- [创建并应用保证的最小带宽 QoS 策略和规则](#)
- [为出口流量创建并应用与 QoS 标记的 QoS 策略和规则](#)

## 9.2. 为 QOS 策略配置网络服务

Red Hat OpenStack Platform (RHOSP) 网络服务(neutron)中的服务质量功能是通过 **qos** 服务插件提供的。使用 ML2/OVS 和 ML2/OVN 机制驱动程序时，**qos** 默认会被加载。但是，对于 ML2/SR-IOV，这不是 true。

将 **qos** 服务插件与 ML2/OVS 和 ML2/SR-IOV 机制驱动程序搭配使用时，还必须在其相应的代理上加载 **qos** 扩展。

以下列表总结了为 QoS 配置网络服务必须执行的任务。任务详情遵循此列表：

- 对于所有 QoS 策略类型：
  - 添加 **qos** 服务插件。
  - 为代理添加 **qos** 扩展（仅限 OVS 和 SR-IOV）。
- 仅使用最小带宽策略调度虚拟机实例的额外任务：
  - 如果 hypervisor 名称与 Compute 服务(nova)使用的名称不同，请指定它。
  - 为每个 Compute 节点上的相关代理配置资源供应商入口和出口带宽。
  - （可选）将 **vnic\_types** 标记为不受支持。
- 在仅使用 ML/OVS 和隧道的系统中标记策略的附加任务：
  - 将 **dscp\_inherit** 设置为 **true**。

### 先决条件

- 您必须是具有访问 RHOSP undercloud 的 **stack** 用户。

### 流程

1. 以 **stack** 用户身份登录 undercloud 主机。
2. 查找 undercloud 凭证文件：

```
$ source ~/stackrc
```

3. 确认 **qos** 服务插件尚未加载。

```
$ openstack network qos policy list
```

如果没有加载 **qos** 服务插件，您会收到 **ResourceNotFound** 错误。如果没有收到错误，则插件会被加载，您不需要执行本主题中的步骤。

4. 创建 YAML 自定义环境文件。

### 示例

```
$ vi /home/stack/templates/my-neutron-environment.yaml
```

5. 您的环境文件必须包含 keywords **parameter\_defaults**。在 **parameter\_defaults** 下面的新行中，将 **qos** 添加到 **NeutronServicePlugins** 参数：

```
parameter_defaults:
  NeutronServicePlugins: "qos"
```

6. 如果使用 ML2/OVS 和 ML2/SR-IOV 机制驱动程序，则必须分别使用 **NeutronAgentExtensions** 或 **NeutronSriovAgentExtensions** 变量在代理上加载 **qos** 扩展：

- ML2/OVS

-

```
parameter_defaults:
  NeutronServicePlugins: "qos"
  NeutronAgentExtensions: "qos"
```

- ML2/SR-IOV

```
parameter_defaults:
  NeutronServicePlugins: "qos"
  NeutronSriovAgentExtensions: "qos"
```

7. 如果要使用最小带宽 QoS 策略调度虚拟机实例，还必须执行以下操作：

- a. 将 **放置** 添加到插件列表中，并确保列表还包括 **qos**：

```
parameter_defaults:
  NeutronServicePlugins: "qos,placement"
```

- b. 如果虚拟机监控程序名称与计算服务(nova)使用的规范管理程序名称匹配，请跳至第 7.iii 步。

如果虚拟机监控程序名称与计算服务使用的规范管理程序名称不匹配，请使用 **resource\_provider\_default\_hypervisor** 指定替代的 hypervisor 名称：

- ML2/OVS

```
parameter_defaults:
  NeutronServicePlugins: "qos,placement"
  ExtraConfig:
    Neutron::agents::ml2::ovs::resource_provider_default_hypervisor: %
    {hiera('fqdn_canonical')}
```

- ML2/SR-IOV

```
parameter_defaults:
  NeutronServicePlugins: "qos,placement"
  ExtraConfig:
    Neutron::agents::ml2::sriov::resource_provider_default_hypervisor: %
    {hiera('fqdn_canonical')}
```

**重要**

设置替代管理程序名称的另一种方法是使用 **resource\_provider\_hypervisor** :

- o ML2/OVS

```
parameter_defaults:
  ExtraConfig:
```

```
Neutron::agents::ml2::ovs::resource_provider_hypervisors:"ens5:%
{hiera('fqdn_canonical')},ens6:%{hiera('fqdn_canonical')}}"
```

- o ML2/SR-IOV

```
parameter_defaults:
  ExtraConfig:
```

```
Neutron::agents::ml2::sriov::resource_provider_hypervisors:
  "ens5:%{hiera('fqdn_canonical')},ens6:%
  {hiera('fqdn_canonical')}}"
```

c. 为每个需要提供最小带宽的 Compute 节点上的相关代理配置资源供应商入口和出口带宽。您可以使用以下格式配置出口、入口或两者：

- 仅配置出口带宽，以 kbps 为单位：

```
NeutronOvsResourceProviderBandwidths: <bridge0>:<egress_kbps>,<bridge1>:
<egress_kbps>,...,<bridgeN>:<egress_kbps>
```

- 仅配置入口带宽，以 kbps 为单位：

```
NeutronOvsResourceProviderBandwidths: <bridge0>:<ingress_kbps>,<bridge1>:
<ingress_kbps>,...,<bridgeN>:<ingress_kbps>
```

- 以 kbps 为单位配置出口和入口带宽：

```
NeutronOvsResourceProviderBandwidths: <bridge0>:<egress_kbps>:
<ingress_kbps>,<bridge1>:<egress_kbps>:<ingress_kbps>,...,<bridgeN>:
<egress_kbps>:<ingress_kbps>
```

**示例 - OVS 代理**

要为 OVS 代理配置资源供应商入口和出口带宽，请在网络环境文件中添加以下配置：

```
parameter_defaults:
  ...
  NeutronBridgeMappings: physnet0:br-physnet0
  NeutronOvsResourceProviderBandwidths: br-physnet0:10000000:10000000
```

**示例 - SRIOV 代理**

要为 SRIOV 代理配置资源供应商入口和出口带宽，请在网络环境文件中添加以下配置：

```
parameter_defaults:
```

```
...
NeutronML2PhysicalNetworkMtus: physnet0:1500,physnet1:1500
NeutronSriovResourceProviderBandwidths:
ens5:40000000:40000000,ens6:40000000:40000000
```

- d. 可选：当多个 ML2 机制驱动程序默认支持且在放置服务中跟踪多个代理时，将 **vnic\_types** 标记为不被支持，请在环境文件中添加以下配置：

#### 示例 - OVS 代理

```
parameter_defaults:
...
NeutronOvsVnicTypeBlacklist: direct
```

#### 示例 - SRIOV 代理

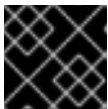
```
parameter_defaults:
...
NeutronSriovVnicTypeBlacklist: direct
```

8. 如果要创建 kiosk 标记策略，并将 ML2/OVS 与隧道协议(VXLAN 或 GRE)搭配使用，在 **NeutronAgentExtensions** 下添加以下行：

```
parameter_defaults:
...
ControllerExtraConfig:
  neutron::config::server_config:
    agent/dscp_inherit:
      value: true
```

当 **dscp\_inherit** 为 **true** 时，网络服务会将内部标头的 DSCP 值复制到外部标头。

9. 运行部署命令，并包含核心 heat 模板、其他环境文件和这种新的自定义环境文件。



#### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

#### 示例

```
$ openstack overcloud deploy --templates \
-e <other_environment_files> \
-e /home/stack/templates/my-neutron-environment.yaml
```

#### 验证

- 确认 **qos** 服务插件已加载：

```
$ openstack network qos policy list
```

如果载入了 **qos** 服务插件，则不会收到 **ResourceNotFound** 错误。

## 其他资源

- [RHOSP 网络服务的扩展驱动程序](#)
- [高级 Overcloud 自定义指南中的环境文件](https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/16.2/html/advanced_overcloud_customization/assembly_underst_heat-templates#con_environment-files_understanding-heat-templates)
- [高级 Overcloud 自定义指南中的 创建 overcloud 中包括环境文件](#)
- [第 9.3.1 节 “使用网络服务后端强制强制最小带宽”](#)
- [第 9.3.2 节 “使用最小带宽 QoS 策略调度实例”](#)
- [第 9.4 节 “使用 QoS 策略限制网络流量”](#)
- [第 9.5 节 “使用 kiosk 标记 QoS 策略来优先考虑网络流量”](#)

### 9.3. 使用 QoS 策略控制最小带宽

对于 Red Hat OpenStack Platform (RHOSP)网络服务(neutron)，可以在两个不同的上下文中强制实施保证最小带宽 QoS 规则：网络服务后端强制和资源分配调度强制。

网络后端 ML2/OVS 或 ML2/SR-IOV 会尝试保证应用该规则的每个端口都小于指定的网络带宽。

当您使用资源分配调度带宽强制时，计算服务(nova)仅将虚拟机实例放在支持最小带宽的主机上。

您可以使用网络服务后端强制、资源分配调度实施或两者都应用 QoS 最小带宽规则。

下表标识支持最小带宽 QoS 策略的 Modular Layer 2 (ML2)机制驱动程序。

**表 9.4. 支持最小带宽 QoS 的 ML2 机制驱动程序**

ML2 机制驱动程序	Agent	VNIC 类型
ML2/SR-IOV	sriovnicswitch	direct
ML2/OVS	openvswitch	normal

#### 其他资源

- [第 9.3.1 节 “使用网络服务后端强制强制最小带宽”](#)
- [第 9.3.2 节 “使用最小带宽 QoS 策略调度实例”](#)

#### 9.3.1. 使用网络服务后端强制强制最小带宽

您可以通过将 Red Hat OpenStack Platform (RHOSP)服务质量(QoS)策略应用到端口来确保网络流量的最小带宽。这些端口必须由扁平或 VLAN 物理网络支持。



#### 注意

目前，带有 Open Virtual Network 机制驱动程序(ML2/OVN)的 Modular Layer 2 插件不支持最小带宽 QoS 规则。

## 先决条件

- RHOSP Networking 服务(neutron)必须载入 **qos** 服务插件。（这是默认值。）
- 不要将具有和没有带宽保证的端口混合到同一物理接口，因为这可能导致在不保证的情况下将所需的资源（打包）拒绝端口。

## 提示

创建主机聚合以将带宽保证与那些端口分开，而无需带宽保证。

## 流程

1. 提供您的凭据文件。

### 示例

```
$ source ~/overcloudrc
```

2. 确认 **qos** 服务插件已加载到网络服务中：

```
$ openstack network qos policy list
```

如果没有加载 **qos** 服务插件，则您会收到 **ResourceNotFound** 错误，且您必须加载 **qos** 服务插件，然后才能继续。更多信息请参阅 [第 9.2 节“为 QoS 策略配置网络服务”](#)。

3. 识别您要为其创建 QoS 策略的项目 ID：

```
$ openstack project list
```

### 输出示例

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

4. 使用上一步中的项目 ID，为项目创建一个 QoS 策略。

### 示例

在本例中，为 **admin** 项目创建一个名为 **guaranteed\_min\_bw** 的 QoS 策略：

```
$ openstack network qos policy create --share \
--project 98a2f53c20ce4d50a40dac4a38016c69 guaranteed_min_bw
```

5. 配置策略的规则。

### 示例



在本例中，为名为 **guaranteed\_min\_bw** 的策略创建最小带宽为 **40000000** kbps 的入口和出口的 QoS 规则：

```
$ openstack network qos rule create \
--type minimum-bandwidth --min-kbps 40000000 \
--ingress guaranteed_min_bw

$ openstack network qos rule create \
--type minimum-bandwidth --min-kbps 40000000 \
--egress guaranteed_min_bw
```

6. 配置端口以将策略应用到。

### 示例

在本例中，**guaranteed\_min\_bw** 策略应用于端口 ID **56x9aiw1-2v74-144x-c2q8-ed8w423a6s12**：

```
$ openstack port set --qos-policy guaranteed_min_bw \
56x9aiw1-2v74-144x-c2q8-ed8w423a6s12
```

### 验证

- **ML2/SR-IOV**

使用 root 访问权限，登录 Compute 节点，并显示物理功能中保存的虚拟功能的详细信息。

### 示例

```
# ip -details link show enp4s0f1
```

### 输出示例

```
50: enp4s0f1: <BROADCAST,MULTICAST,SLAVE,UP,LOWER_UP> mtu 9000 qdisc mq
master mx-bond state UP mode DEFAULT group default qlen 1000
    link/ether 98:03:9b:9d:73:74 brd ff:ff:ff:ff:ff:ff permaddr 98:03:9b:9d:73:75 promiscuity 0
    minmtu 68 maxmtu 9978
    bond_slave state BACKUP mii_status UP link_failure_count 0 perm_hwaddr
98:03:9b:9d:73:75 queue_id 0 addrngenmode eui64 numtxqueues 320 numrxqueues 40
    gso_max_size 65536 gso_max_segs 65535 portname p1 switchid 74739d00039b0398
    vf 0 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
    vf 1 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
    vf 2 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
    vf 3 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
    vf 4 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
    vf 5 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
    vf 6 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
    vf 7 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
```

```
trust off, query_rss off
  vf 8 link/ether fa:16:3e:2a:d2:7f brd ff:ff:ff:ff:ff:ff, tx rate 999 (Mbps), max_tx_rate
999Mbps, spoof checking off, link-state disable, trust off, query_rss off
  vf 9 link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff, spoof checking off, link-state disable,
trust off, query_rss off
```

- **ML2/OVS**

使用 root 访问权限，登录到计算节点，显示物理网桥接口上的 **tc** 规则和类。

### 示例

```
# tc class show dev mx-bond
```

### 输出示例

```
class htb 1:11 parent 1:ffff prio 0 rate 4Gbit ceil 34359Mbit burst 9000b cburst 8589b
class htb 1:1 parent 1:ffff prio 0 rate 72Kbit ceil 34359Mbit burst 9063b cburst 8589b
class htb 1:ffff root rate 34359Mbit ceil 34359Mbit burst 8589b cburst 8589b
```

### 其他资源

- 命令行界面参考中的 [network qos policy create](#)
- 命令行界面参考中的 [network qos 规则创建](#)
- 命令行界面参考中的 [端口集](#)

## 9.3.2. 使用最小带宽 QoS 策略调度实例

您可以将最小带宽 QoS 策略应用到端口，以确保生成其 Red Hat OpenStack Platform (RHOSP) 虚拟机实例的主机具有最低网络带宽。

### 先决条件

- RHOSP Networking 服务(neutron)必须载入 **qos** 和 **placement** 服务插件。默认情况下加载 **qos** 服务插件。
- 网络服务必须支持以下 API 扩展：
  - **agent-resources-synced**
  - **port-resource-request**
  - **qos-bw-minimum-ingress**
- 您必须使用 ML2/OVS 或 ML2/SR-IOV 机制驱动程序。
- 只有当没有使用该策略的任何端口的实例时，才能修改最小带宽 QoS 策略。如果端口绑定，网络服务将无法更新放置 API 使用量信息。
- 放置服务必须支持微版本 1.29。
- Compute 服务(nova)必须支持 microversion 2.72。

## 流程

1. 提供您的凭据文件。

### 示例

```
$ source ~/overcloudrc
```

2. 确认 **qos** 服务插件已加载到网络服务中：

```
$ openstack network qos policy list
```

如果没有加载 **qos** 服务插件，则您会收到 **ResourceNotFound** 错误，且您必须加载 **qos** 服务插件，然后才能继续。更多信息请参阅 [第 9.2 节“为 QoS 策略配置网络服务”](#)。

3. 识别您要为其创建 QoS 策略的项目 ID：

```
$ openstack project list
```

### 输出示例

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

4. 使用上一步中的项目 ID，为项目创建一个 QoS 策略。

### 示例

在本例中，为 **admin** 项目创建一个名为 **guaranteed\_min\_bw** 的 QoS 策略：

```
$ openstack network qos policy create --share \
--project 98a2f53c20ce4d50a40dac4a38016c69 guaranteed_min_bw
```

5. 配置策略的规则。

### 示例

在本例中，为名为 **guaranteed\_min\_bw** 的策略创建最小带宽为 **40000000** kbps 的入口和出口的 QoS 规则：

```
$ openstack network qos rule create \
--type minimum-bandwidth --min-kbps 40000000 \
--ingress guaranteed_min_bw
$ openstack network qos rule create \
--type minimum-bandwidth --min-kbps 40000000 \
--egress guaranteed_min_bw
```

6. 配置端口以将策略应用到。

## 示例

在本例中，**guaranteed\_min\_bw** 策略应用于端口 ID **56x9aiw1-2v74-144x-c2q8-ed8w423a6s12**：

```
$ openstack port set --qos-policy guaranteed_min_bw \
  56x9aiw1-2v74-144x-c2q8-ed8w423a6s12
```

## 验证

1. 以 stack 用户身份登录 undercloud 主机。
2. 查找 undercloud 凭证文件：

```
$ source ~/stackrc
```

3. 列出所有可用资源供应商：

```
$ openstack --os-placement-api-version 1.17 resource provider list
```

## 输出示例

```
+-----+-----+-----+-----+
| uuid          | name          | generation |
| root_provider_uuid | parent_provider_uuid |          |
+-----+-----+-----+-----+
| 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | dell-r730-014.localdomain |          |
28 | 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | None |          |
| 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | dell-r730-063.localdomain |          |
18 | 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | None |          |
| e2f5082a-c965-55db-acb3-8daf9857c721 | dell-r730-063.localdomain:NIC Switch agent |          |
| 0 | 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | 6b15ddce-13cf-4c85-a58f- |
baec5b57ab52 |
| d2fb0ef4-2f45-53a8-88be-113b3e64ba1b | dell-r730-014.localdomain:NIC Switch agent |          |
| 0 | 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | 31d3d88b-bc3a-41cd-9dc0- |
fda54028a882 |
| f1ca35e2-47ad-53a0-9058-390ade93b73e | dell-r730-063.localdomain:NIC Switch |          |
agent:enp6s0f1 | 13 | 6b15ddce-13cf-4c85-a58f-baec5b57ab52 | e2f5082a-c965-55db- |
acb3-8daf9857c721 |
| e518d381-d590-5767-8f34-c20def34b252 | dell-r730-014.localdomain:NIC Switch |          |
agent:enp6s0f1 | 19 | 31d3d88b-bc3a-41cd-9dc0-fda54028a882 | d2fb0ef4-2f45-53a8- |
88be-113b3e64ba1b |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

4. 检查特定资源提供的带宽。

```
(undercloud)$ openstack --os-placement-api-version 1.17 \
  resource provider inventory list <rp_uuid>
```

## 示例

在本例中，由主机 **dell-r730-014** 上的接口 **enp6s0f1** 提供的带宽被检查，使用资源供应商 UUID **e518d381-d590-5767-8f34-c20def34b252**：

```
[stack@dell-r730-014 nova]$ openstack --os-placement-api-version 1.17 \
resource provider inventory list e518d381-d590-5767-8f34-c20def34b252
```

### 输出示例

```
+-----+-----+-----+-----+-----+-----+
| resource_class      | allocation_ratio | min_unit | max_unit | reserved | step_size |
total |
+-----+-----+-----+-----+-----+-----+
| NET_BW_EGR_KILOBIT_PER_SEC |      1.0 |      1 | 2147483647 |      0 |      1 |
10000000 |
| NET_BW_IGR_KILOBIT_PER_SEC |      1.0 |      1 | 2147483647 |      0 |      1 |
10000000 |
+-----+-----+-----+-----+-----+-----+
```

5. 要在实例运行时检查资源供应商的声明，请运行以下命令：

```
(undercloud)$ openstack --os-placement-api-version 1.17 \
resource provider show --allocations <rp_uuid>
```

### 示例

在本例中，针对资源提供程序的声明在主机 (**dell-r730-014**) 上进行检查，使用资源提供程序 UUID **e518d381-d590-5767-8f34-c20def34b252**：

```
[stack@dell-r730-014 nova]$ openstack --os-placement-api-version 1.17 resource provider
show --allocations e518d381-d590-5767-8f34-c20def34b252 -f value -c allocations
```

### 输出示例

```
{3cbb9e07-90a8-4154-8acd-b6ec2f894a83: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, 8848b88b-4464-443f-bf33-5d4e49fd6204: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, 9a29e946-698b-4731-bc28-89368073be1a: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, a6c83b86-9139-4e98-9341-dc76065136cc: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 3000000, NET_BW_IGR_KILOBIT_PER_SEC:
3000000}}, da60e33f-156e-47be-a632-870172ec5483: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 1000000, NET_BW_IGR_KILOBIT_PER_SEC:
1000000}}, eb582a0e-8274-4f21-9890-9a0d55114663: {resources:
{NET_BW_EGR_KILOBIT_PER_SEC: 3000000, NET_BW_IGR_KILOBIT_PER_SEC:
3000000}}}
```

### 其他资源

- 命令行界面参考中的 [network qos policy create](#)
- 命令行界面参考中的 [network qos 规则创建](#)

- 命令行界面参考中的 [端口集](#)

## 9.4. 使用 QoS 策略限制网络流量

您可以创建 Red Hat OpenStack Platform (RHOSP)网络服务(neutron)质量服务质量(QoS)策略，该策略限制 RHOSP 网络、端口或浮动 IP 上的带宽，并丢弃任何超过指定速率的流量。

### 先决条件

- 网络服务必须加载 **qos** 服务插件。（插件默认会被加载。）

### 流程

1. 提供您的凭据文件。

#### 示例

```
$ source ~/overcloudrc
```

2. 确认 **qos** 服务插件已加载到网络服务中：

```
$ openstack network qos policy list
```

如果没有加载 **qos** 服务插件，则您会收到 **ResourceNotFound** 错误，且您必须加载 **qos** 服务插件，然后才能继续。更多信息请参阅 [第 9.2 节“为 QoS 策略配置网络服务”](#)。

3. 识别您要为其创建 QoS 策略的项目 ID：

```
$ openstack project list
```

#### 输出示例

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

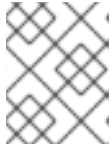
4. 使用上一步中的项目 ID，为项目创建一个 QoS 策略。

#### 示例

在本例中，为 **admin** 项目创建一个名为 **bw-limiter** 的 QoS 策略：

```
$ openstack network qos policy create --share --project
98a2f53c20ce4d50a40dac4a38016c69 bw-limiter
```

5. 配置策略的规则。



## 注意

只要每个规则的类型或方向不同，您可以在策略中添加多个规则。例如，您可以指定两个带宽限制规则，一个用于出口，另一个带有 ingress 方向。

## 示例

在本例中，为名为 **bw-limiter** 的策略创建 QoS 入口和出口规则，带宽限制为 **50000** kbps，最大突发大小为 **50000** kbps：

```
$ openstack network qos rule create --type bandwidth-limit \
  --max-kbps 50000 --max-burst-kbits 50000 --ingress bw-limiter

$ openstack network qos rule create --type bandwidth-limit \
  --max-kbps 50000 --max-burst-kbits 50000 --egress bw-limiter
```

- 您可以创建附加策略的端口，或者将策略附加到预先存在的端口。

## 示例 - 创建附加策略的端口

在本例中，策略 **bw-limiter** 与端口 **port2** 关联：

```
$ openstack port create --qos-policy bw-limiter --network private port2
```

## 输出示例

```
+-----+-----+
| Field          | Value                                     |
+-----+-----+
| admin_state_up | UP                                       |
| allowed_address_pairs |                                         |
| binding_host_id |                                         |
| binding_profile |                                         |
| binding_vif_details |                                         |
| binding_vif_type | unbound                                 |
| binding_vnic_type | normal                                  |
| created_at      | 2022-07-04T19:20:24Z                   |
| data_plane_status | None                                    |
| description     |                                         |
| device_id       |                                         |
| device_owner    |                                         |
| dns_assignment  | None                                    |
| dns_name        | None                                    |
| extra_dhcp_opts |                                         |
| fixed_ips       | ip_address='192.0.2.210', subnet_id='292f8c-...' |
| id              | f51562ee-da8d-42de-9578-f6f5cb248226   |
| ip_address      | None                                    |
| mac_address     | fa:16:3e:d9:f2:ba                       |
| name            | port2                                    |
| network_id      | 55dc2f70-0f92-4002-b343-ca34277b0234   |
| option_name     | None                                    |
| option_value    | None                                    |
| port_security_enabled | False                                  |
| project_id      | 98a2f53c20ce4d50a40dac4a38016c69       |
| qos_policy_id   | 8491547e-add1-4c6c-a50e-42121237256c   |
```

```

| revision_number | 6 |
| security_group_ids | 0531cc1a-19d1-4cc7-ada5-49f8b08245be |
| status | DOWN |
| subnet_id | None |
| tags | [] |
| trunk_details | None |
| updated_at | 2022-07-04T19:23:00Z |
+-----+-----+

```

### 示例 - 将策略附加到预先存在的端口

在本例中，策略 **bw-limiter** 与 **port1** 关联：

```
$ openstack port set --qos-policy bw-limiter port1
```

### 验证

- 确认将限制策略应用到端口。
  - 获取策略 ID。

#### 示例

在本例中，查询 QoS 策略 **bw-limiter**：

```
$ openstack network qos policy show bw-limiter
```

#### 输出示例

```

+-----+-----+
| Field      | Value |
+-----+-----+
| description |      |
| id         | 8491547e-add1-4c6c-a50e-42121237256c |
| is_default | False |
| name       | bw-limiter |
| project_id | 98a2f53c20ce4d50a40dac4a38016c69 |
| revision_number | 4 |
| rules      | [{u'max_kbps': 50000, u'direction': u'egress', |
|            | u'type': u'bandwidth_limit', |
|            | u'id': u'0db48906-a762-4d32-8694-3f65214c34a6', |
|            | u'max_burst_kbps': 50000, |
|            | u'qos_policy_id': u'8491547e-add1-4c6c-a50e-42121237256c'}, |
|            | [{u'max_kbps': 50000, u'direction': u'ingress', |
|            | u'type': u'bandwidth_limit', |
|            | u'id': u'faabef24-e23a-4fdf-8e92-f8cb66998834', |
|            | u'max_burst_kbps': 50000, |
|            | u'qos_policy_id': u'8491547e-add1-4c6c-a50e-42121237256c'}] |
| shared     | False |
+-----+-----+

```

- 查询端口，并确认其策略 ID 与上一步中获取的策略 ID 匹配。

#### 示例



在本例中，查询 **port1**：

```
$ openstack port show port1
```

### 输出示例

```
+-----+-----+
| Field          | Value                                          |
+-----+-----+
| admin_state_up | UP                                            |
| allowed_address_pairs | ip_address='192.0.2.128', mac_address='fa:16:3e:e1:eb:73' |
|
| binding_host_id | compute-2.redhat.local                      |
| binding_profile |                                              |
| binding_vif_details | port_filter='True'                          |
| binding_vif_type | ovs                                          |
| binding_vnic_type | normal                                       |
| created_at      | 2022-07-04T19:07:56                        |
| data_plane_status | None                                         |
| description     |                                              |
| device_id      | 53abd2c4-955d-4b44-b6ad-f106e3f15df0      |
| device_owner   | compute:nova                               |
| dns_assignment | fqdn='host-192-0-2-213.openstacklocal.', hostname='my-host3',
|
|                | ip_address='192.0.2.213'                    |
| dns_domain    | None                                         |
| dns_name      |                                              |
| extra_dhcp_opts |                                              |
| fixed_ips     | ip_address='192.0.2..213', subnet_id='641d1db2-3b40-437b-b87b-
63 |
|                | 079a7063ca'                                |
|                | ip_address='2001:db8:0:f868:f816:3eff:fee1:eb73', subnet_id='c7ed0 |
|                | 70a-d2ee-4380-baab-6978932a7dcc'           |
| id           | 56x9aiw1-2v74-144x-c2q8-ed8w423a6s12      |
| location     | cloud="", project.domain_id=, project.domain_name=, project.id='7c |
|                | b99d752fdb4944a2208ec9ee019226', project.name=,
region_name='regio |
|                | nOne', zone=                               |
| mac_address   | fa:16:3e:e1:eb:73                          |
| name         | port2                                        |
| network_id    | 55dc2f70-0f92-4002-b343-ca34277b0234      |
| port_security_enabled | True                                         |
| project_id    | 98a2f53c20ce4d50a40dac4a38016c69          |
| propagate_uplink_status | None                                         |
| qos_policy_id | 8491547e-add1-4c6c-a50e-42121237256c      |
| resource_request | None                                         |
| revision_number | 6                                            |
| security_group_ids | 4cdeb836-b5fd-441e-bd01-498d758704fd      |
| status       | ACTIVE                                       |
| tags         |                                              |
| trunk_details | None                                         |
| updated_at   | 2022-07-04T19:11:41Z                      |
+-----+-----+
```

- [命令行界面参考中的 network qos 规则创建](#)
- [命令行界面参考中的 network qos 规则](#)
- [命令行界面参考中的 network qos rule delete](#)
- [命令行界面参考中的 Network qos 规则列表](#)

## 9.5. 使用 KIOSK 标记 QOS 策略来优先考虑网络流量

您可以通过在 IP 标头中嵌入相关值，使用区分服务代码点(DSCP)在 Red Hat OpenStack Platform (RHOSP)网络上实现服务质量(QoS)策略。RHOSP Networking 服务(neutron)QoS 策略可以使用 DSCP 标记来管理 neutron 端口和网络上的出口流量。

### 先决条件

- 网络服务必须加载 **qos** 服务插件。（这是默认值。）
- 您必须使用 ML2/OVS 或 ML2/OVN 机制驱动程序。

### 流程

1. 提供您的凭据文件。

#### 示例

```
$ source ~/overcloudrc
```

2. 确认 **qos** 服务插件已加载到网络服务中：

```
$ openstack network qos policy list
```

如果没有加载 **qos** 服务插件，则您会收到 **ResourceNotFound** 错误，而且您必须配置网络服务，然后才能继续。更多信息请参阅 [第 9.2 节“为 QoS 策略配置网络服务”](#)。

3. 识别您要为其创建 QoS 策略的项目 ID：

```
$ openstack project list
```

#### 输出示例

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

4. 使用上一步中的项目 ID，为项目创建一个 QoS 策略。

#### 示例

在本例中，为 **admin** 项目创建一个名为 **qos-web-servers** 的 QoS 策略：

```
openstack network qos policy create --project 98a2f53c20ce4d50a40dac4a38016c69 qos-
web-servers
```

5. 创建 kiosk 规则，并将其应用到策略。

### 示例

在这个示例中，DSCP 规则使用 DSCP 标记 **18** 创建，并应用到 **qos-web-servers** 策略：

```
openstack network qos rule create --type dscp-marking --dscp-mark 18 qos-web-servers
```

### 输出示例

```
Created a new dscp_marking_rule:
+-----+-----+
| Field | Value |
+-----+-----+
| dscp_mark | 18 |
| id | d7f976ec-7fab-4e60-af70-f59bf88198e6 |
+-----+-----+
```

6. 您可以更改分配给规则的 CRUD 值。

### 示例

在本例中，对于规则 **d7f976ec-7fab-4e60-af70-f59bf88198e6**，在 **qos-web-servers** 策略中，为 rule 改为 22：

```
$ openstack network qos rule set --dscp-mark 22 qos-web-servers d7f976ec-7fab-4e60-af70-
f59bf88198e6
```

7. 您可以删除 kiosk 规则。

### 示例

在本例中，DSCP 规则 **d7f976ec-7fab-4e60-af70-f59bf88198e6**（在 **qos-web-servers** 策略中）被删除：

```
$ openstack network qos rule delete qos-web-servers d7f976ec-7fab-4e60-af70-
f59bf88198e6
```

## 验证

- 确认 DSCP 规则已应用到 QoS 策略。

### 示例

在本例中，**d7f976ec-7fab-4e60-af70-f59bf88198e6** 应用到 QoS 策略 **qos-web-servers**：

```
$ openstack network qos rule list qos-web-servers
```

### 输出示例

```
+-----+-----+
| dscp_mark | id |
+-----+-----+
|    18 | d7f976ec-7fab-4e60-af70-f59bf88198e6 |
+-----+-----+
```

## 其他资源

- [命令行界面参考中的 network qos 规则创建](#)
- [命令行界面参考中的 network qos 规则](#)
- [命令行界面参考中的 network qos rule delete](#)
- [命令行界面参考中的 Network qos 规则列表](#)

## 9.6. 使用网络服务 RBAC 将 QoS 策略应用到项目

使用 Red Hat OpenStack Platform (RHOSP)网络服务(neutron), 您可以为服务质量(QoS)策略添加基于角色的访问控制(RBAC)。因此, 您可以将 QoS 策略应用到单独的项目。

### 先决条件

- 您必须有一个或多个 QoS 策略。

### 流程

- 创建与特定 QoS 策略关联的 RHOSP 网络服务 RBAC 策略, 并将其分配给特定的项目 :

```
$ openstack network rbac create --type qos_policy --target-project <project_name |
project_ID> --action access_as_shared <QoS_policy_name | QoS_policy_ID>
```

### 示例

例如, 您可能有一个 QoS 策略, 允许较低优先级网络流量, 名为 **bw-limiter**。使用 RHOSP 网络服务 RBAC 策略, 您可以将 QoS 策略应用到特定的项目 :

```
$ openstack network rbac create --type qos_policy --target-project
80bf5732752a41128e612fe615c886c6 --action access_as_shared bw-limiter
```

## 其他资源

- [命令行界面参考中的 network rbac create](#)
- [第 9.3.1 节 “使用网络服务后端强制强制最小带宽”](#)
- [第 9.3.2 节 “使用最小带宽 QoS 策略调度实例”](#)
- [第 9.4 节 “使用 QoS 策略限制网络流量”](#)
- [第 9.5 节 “使用 kiosk 标记 QoS 策略来优先考虑网络流量”](#)

## 第 10 章 配置网桥映射

在 Red Hat OpenStack Platform (RHOSP) 中，网桥映射将物理网络名称（接口标签）与使用 Modular Layer 2 插件机制驱动程序 Open vSwitch (OVS) 或 Open Virtual Network (OVN) 创建的网桥关联。RHOSP Networking 服务(neutron)使用网桥映射来允许提供商网络流量访问物理网络。

本节中包含的主题有：

- [第 10.1 节 “网桥映射概述”](#)
- [第 10.2 节 “流量流”](#)
- [第 10.3 节 “配置网桥映射”](#)
- [第 10.4 节 “为 OVS 维护网桥映射”](#)
- [第 10.4.1 节 “手动清理 OVS 补丁端口”](#)
- [第 10.4.2 节 “自动清理 OVS 补丁端口”](#)

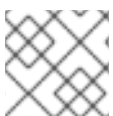
### 10.1. 网桥映射概述

在 Red Hat OpenStack Platform (RHOSP) 网络服务(neutron)中，您可以使用桥接映射来允许提供商网络流量访问物理网络。流量从路由器的 **qg-xxx** 接口离开提供商网络，并到达中间网桥(**br-int**)。

数据路径的下一部分因部署使用哪个机制驱动程序而异：

- ML2/OVS：**br-int** 和 **br-ex** 之间的补丁端口允许流量通过提供商网络的网桥和向外物理网络。
- ML2/OVN：网络服务仅在虚拟机绑定到虚拟机监控程序且虚拟机需要端口时在虚拟机监控程序上创建一个补丁端口。

您可以在调度路由器的网络节点上配置网桥映射。路由器流量可以使用正确的物理网络进行出口，如提供商网络所示。



#### 注意

网络服务只支持每个物理网络的一个网桥。不要将多个物理网络映射到同一网桥。

### 10.2. 流量流

每个外部网络都由一个内部 VLAN ID 表示，它标记为路由器 **qg-xxx** 端口。当数据包到达 **phy-br-ex** 时，**br-ex** 端口会剥离 VLAN 标签，并将数据包移到物理接口，然后移到外部网络。

来自外部网络的返回数据包到达 **br-ex**，并使用 **phy-br-ex <-> int-br-ex** 移到 **br-int**。当数据包通过 **br-ex** 到 **br-int** 时，数据包的外部 VLAN ID 替换为 **br-int** 中的内部 VLAN 标签，这将允许 **qg-xxx** 接受数据包。

如果是出口数据包，数据包的内部 VLAN 标签将被替换为 **br-ex** 中的外部 VLAN 标签（或者在 **NeutronNetworkVLANRanges** 参数中定义的外部网桥）。

### 10.3. 配置网桥映射

要修改 Red Hat OpenStack Platform (RHOSP) 网络服务(neutron)将提供商网络与物理网络连接的网桥映射，您需要修改必要的 heat 参数并重新部署 overcloud。

## 先决条件

- 您必须能够以 **stack** 用户身份访问 undercloud 主机。
- 您必须在调度路由器的网络节点上配置网桥映射。
- 您还必须为您的 Compute 节点配置网桥映射。

## 流程

1. 以 stack 用户身份登录 undercloud 主机。
2. 查找 undercloud 凭证文件：

```
$ source ~/stackrc
```

3. 创建自定义 YAML 环境文件。

### 示例

```
$ vi /home/stack/templates/my_bridge_mappings.yaml
```

4. 您的环境文件必须包含 keywords **parameter\_defaults**。添加 **NeutronBridgeMappings** heat 参数，其值在 **parameter\_defaults** 关键字后适合您的站点。

### 示例

在本例中，**NeutronBridgeMappings** 参数分别关联物理名称 **datacentre** 和 **租户**，网桥 **br-ex** 和 **br-tenant**。

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"
```



### 注意

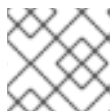
如果没有使用 **NeutronBridgeMappings** 参数，默认会将主机上的外部网桥(br-ex)映射到物理名称(datacentre)。

5. 如果您使用扁平网络，请使用 **NeutronFlatNetworks** 参数添加其名称。

### 示例

在本例中，参数将物理名称 **datacentre** 与网桥 **br-ex** 关联，并将物理名称 **租户** 与网桥 **br-tenant** 相关联。

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"
  NeutronFlatNetworks: "my_flat_network"
```



### 注意

如果没有使用 **NeutronFlatNetworks** 参数，则默认为 **datacentre**。

- 如果您使用 VLAN 网络，请使用 **NeutronNetworkVLANRanges** 参数指定网络名称以及它访问的 VLAN 范围。

### 示例

在本例中，**NeutronNetworkVLANRanges** 参数为 **tenant** 网络指定 VLAN 范围 **1 - 1000**：

```
parameter_defaults:
  NeutronBridgeMappings: "datacentre:br-ex,tenant:br-tenant"
  NeutronNetworkVLANRanges: "tenant:1:1000"
```

- 运行部署命令，包括核心 heat 模板、环境文件和新的自定义环境文件。

```
$ openstack overcloud deploy --templates \
  -e <your_environment_files> \
  -e /home/stack/templates/my_bridge_mappings.yaml
```

- 执行以下步骤：
  - 使用网络 VLAN 范围，创建代表相应外部网络的提供商网络。（在创建 neutron 提供商网络或浮动 IP 网络时使用物理名称。）
  - 使用路由器接口将外部网络连接到您的项目网络。

### 其他资源

- 高级 *Overcloud 自定义指南* 中的 [网络环境参数](#)
- 高级 *Overcloud 自定义指南* 中的 [创建 overcloud 中包括环境文件](#)

## 10.4. 为 OVS 维护网桥映射

删除任何 OVS 网桥映射后，您必须执行后续的清理，以确保网桥配置清除任何关联的补丁端口条目。您可以使用以下方法执行此操作：

- 手动端口清理 - 需要仔细删除多余的补丁端口。不需要中断网络连接。
- 自动端口清理 - 执行自动清理，但需要中断，并且要求重新添加所需的网桥映射。当可以容忍网络连接中断时，在调度的维护窗口期间选择这个选项。



### 注意

删除 OVN 网桥映射时，OVN 控制器会自动清理任何关联的补丁端口。

### 10.4.1. 手动清理 OVS 补丁端口

删除任何 OVS 网桥映射后，您还必须删除关联的补丁端口。

#### 先决条件

- 要清理的补丁端口必须是 Open Virtual Switch (OVS) 端口。
- 执行手动补丁端口清理 **不需要** 系统中断。
- 您可以通过其命名约定来识别要清理的补丁端口：

- 在 **br- $\$$ external\_bridge** 补丁端口中，名为 **phy- $\langle$ external bridge name $\rangle$** （如 phy-br-ex2）。
- 在 **br-int** 补丁端口中，名为 **int- $\langle$ external bridge name $\rangle$** （例如 int-br-ex2）。

## 流程

1. 使用 **ovs-vsctl** 删除与已删除网桥映射条目关联的 OVS 补丁端口：

```
# ovs-vsctl del-port br-ex2 datacentre
# ovs-vsctl del-port br-tenant tenant
```

2. 重启 **neutron-openvswitch-agent**：

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

### 10.4.2. 自动清理 OVS 补丁端口

删除任何 OVS 网桥映射后，您还必须删除关联的补丁端口。



#### 注意

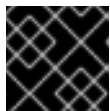
删除 OVN 网桥映射时，OVN 控制器会自动清理任何关联的补丁端口。

## 先决条件

- 要清理的补丁端口必须是 Open Virtual Switch (OVS) 端口。
- 使用 **neutron-ovs-cleanup** 命令自动清理补丁端口会导致网络连接中断，且应仅在调度的维护窗口期间执行。
- 使用标志 **--ovs\_all\_ports** 从 **br-int** 中删除所有 patch 端口，清理来自 **br-tun** 的隧道结束，以及从网桥到网桥的 patch 端口。
- **neutron-ovs-cleanup** 命令从所有 OVS 网桥中拔出所有补丁端口 (instances、qdhcp/qrouter 等)。

## 流程

1. 使用 **--ovs\_all\_ports** 标志运行 **neutron-ovs-cleanup** 命令。



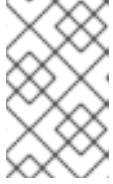
#### 重要

执行此步骤将导致网络中断。

```
# /usr/bin/neutron-ovs-cleanup
--config-file /etc/neutron/plugins/ml2/openvswitch_agent.ini
--log-file /var/log/neutron/ovs-cleanup.log --ovs_all_ports
```

2. 通过重新部署 overcloud 来恢复连接。  
重新运行 **openstack overcloud deploy** 命令时，您的网桥映射值会被重新应用。





## 注意

重启后，OVS 代理不会影响 bridge\_mappings 中不存在的任何连接。因此，如果您有连接到 **br-ex2** 的 **br-int**，并且 **br-ex2** 上有一些数据流，在重启 OVS 代理或节点时，从 bridge\_mappings 配置中删除 **br-int** 不会断开这两个网桥。

## 其他资源

- 高级 Overcloud 自定义指南中的 [网络环境参数](#)
- 高级 Overcloud 自定义指南中的 [创建 overcloud 中包括环境文件](#)

## 第 11 章 VLAN 感知实例

### 11.1. VLAN 中继和 VLAN 透明网络

虚拟机实例可以通过单个虚拟 NIC 发送和接收 VLAN 标记的流量。这对希望 VLAN 标记流量的 NFV 应用程序(VNF)特别有用，允许单个虚拟 NIC 为多个客户或服务提供服务。

在 ML2/OVN 部署中，您可以使用 VLAN 透明网络支持 VLAN 感知实例。作为 ML2/OVN 或 ML2/OVS 部署的替代选择，您可以使用中继来支持 VLAN 感知实例。

在 VLAN 透明网络中，您可以在虚拟机实例中设置 VLAN 标记。VLAN 标签通过网络传输，并由同一 VLAN 上的实例消耗，并被其他实例和设备忽略。在 VLAN 透明网络中，VLAN 在实例中管理。您不需要在 OpenStack Networking Service (neutron)中设置 VLAN。

VLAN 中继通过将 VLAN 合并到单个中继端口来支持 VLAN 感知实例。例如，项目数据网络可以使用 VLAN 或隧道(VXLAN、GRE 或 Geneve)分段，而实例会看到标记为 VLAN ID 的流量。在将网络数据包注入实例之前，网络数据包会立即标记，不需要在整个网络中标记它们。

下表比较 VLAN 透明网络和 VLAN 中继的某些功能。

	透明	中继
机制驱动程序支持	ML2/OVN	ML2/OVN, ML2/OVS
管理的 VLAN 设置	虚拟机实例	OpenStack Networking Service (neutron)
IP 分配	在虚拟机实例中配置	由 DHCP 分配
VLAN ID	灵活。您可以在实例中设置 VLAN ID	已修复。实例必须使用中继中配置的 VLAN ID

### 11.2. 在 ML2/OVN 部署中启用 VLAN 透明性

如果您需要在虚拟机(VM)实例之间发送 VLAN 标记的流量，请启用 VLAN 透明数据。在 VLAN 透明网络中，您可以直接在虚拟机中配置 VLANS，而无需在 neutron 中配置它们。

#### 先决条件

- 部署 Red Hat OpenStack Platform 16.1 或更高版本，使用 ML2/OVN 作为机制驱动程序。
- VLAN 或 Geneve 类型的提供商网络。不要在带有扁平类型提供商网络的部署中使用 VLAN 透明。
- 确保外部交换机支持两个 VLAN 上使用 ethertype 0x8100 的 802.1q VLAN 堆栈。OVN VLAN 透明不支持将外部供应商 VLAN ethertype 设置为 0x88A8 或 0x9100 的 802.1ad QinQ。
- 您必须具有 RHOSP 管理员特权。

#### 流程

1. 以 stack 用户身份登录 undercloud 主机。

2. 查找 stackrc undercloud 凭证文件：

```
$ source ~/stackrc
```

3. 在 undercloud 节点上的环境文件中，将 **EnableVLANTransparency** 参数设置为 **true**。例如，将以下行添加到 **ovn-extras.yaml** 中：

```
parameter_defaults:
  EnableVLANTransparency: true
```

4. 在 **openstack overcloud deploy** 命令中包含环境文件以及与您环境相关的任何其他环境文件并部署 overcloud。

```
$ openstack overcloud deploy \
--templates \
...
-e <other_overcloud_environment_files> \
-e ovn-extras.yaml \
...
```

将 **<other\_overcloud\_environment\_files>** 替换为作为现有部署一部分的环境文件列表。

5. 使用 **--transparent-vlan** 参数创建网络。

#### 示例

```
$ openstack network create network-name --transparent-vlan
```

6. 在每个参与的虚拟机上设置一个 VLAN 接口。  
将接口 MTU 设置为小于 underlay 网络的 MTU 的 4 字节，以适应 VLAN 透明所需的额外标记。例如，如果 lay 网络 MTU 是 1500，请将接口 MTU 设置为 1496。

以下示例命令在 **eth0** 中添加一个 VLAN 接口，其 MTU 为 1496。VLAN 为 50，接口名称为 **vlan50**：

#### 示例

```
$ ip link add link eth0 name vlan50 type vlan id 50 mtu 1496
$ ip link set vlan50 up
$ ip addr add 192.128.111.3/24 dev vlan50
```

7. 对于您在虚拟机内在 VLAN 接口（第 4 步）中创建的 IP 地址，选择这些替代方案之一：

- 在虚拟机端口上设置允许的地址对。

#### 示例

以下示例在端口 **fv82gwk3-qq2e-yu93-go31-56w7sf476mm0** 上设置了允许的地址对，并使用 **192.128.111.3**，并选择性地添加 MAC 地址 **00:40:96:a8:45:c4**：

```
$ openstack port set --allowed-address \
ip-address=192.128.111.3,mac-address=00:40:96:a8:45:c4 \
fv82gwk3-qq2e-yu93-go31-56w7sf476mm0
```

- 禁用端口上的端口安全性。  
当无法列出允许的地址对中所有可能的组合时，禁用端口安全性会提供实际的替代方法。

### 示例

以下示例禁用端口 **fv82gwk3-qq2e-yu93-go31-56w7sf476mm0** 的端口安全性：

```
$ openstack port set --no-security-group \
--disable-port-security \
fv82gwk3-qq2e-yu93-go31-56w7sf476mm0
```

### 验证

1. 使用 vlan50 IP 地址在 VLAN 上的两个虚拟机间 ping。
2. 在 **eth0** 上使用 **tcpdump** 来查看数据包是否到达 VLAN 标签。

### 其他资源

- [高级 Overcloud 自定义指南中的环境文件](#)。
- [使用 director 安装和管理 Red Hat OpenStack Platform 指南中的创建 overcloud 中的环境文件](#)。
- [命令行界面参考中设置的端口](#)

## 11.3. 查看中继插件

在红帽 openStack 部署期间，会默认启用中继插件。您可以查看控制器节点上的配置：

- 在控制器节点上，确认 `/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf` 文件中启用了 trunk 插件：

```
service_plugins=router,qos,trunk
```

## 11.4. 创建中继连接

要为 VLAN 标记的流量实施中继，请创建一个父端口并将新端口附加到现有的 neutron 网络。当您附加新端口时，OpenStack 网络会添加您所创建的父端口的中继连接。接下来，创建子端口。这些子端口将 VLAN 连接到实例，允许连接到中继。在实例操作系统中，还必须创建一个子接口，用于标记与子端口关联的 VLAN 的流量。

1. 识别包含需要访问中继 VLAN 的实例的网络。在本例中，这是 *公共网络*：

```
openstack network list
+-----+-----+-----+
| ID                | Name  | Subnets                |
+-----+-----+-----+
| 82845092-4701-4004-add7-838837837621 | private | 434c7982-cd96-4c41-a8c9-
b93adbdc197 |
| 8d8bc6d6-5b28-4e00-b99e-157516ff0050 | public  | 3fd811b4-c104-44b5-8ff8-
7a86af5e332c |
+-----+-----+-----+
```

2. 创建父中继端口，并将它附加到实例连接的网络中。在本例中，在公共网络上创建一个名为 parent-trunk-port 的 neutron 端口。此中继是父端口，您可以使用它来创建子端口。

```

openstack port create --network public parent-trunk-port
+-----+
| Field      | Value                                     |
+-----+
| admin_state_up | UP                                       |
| allowed_address_pairs |                                         |
| binding_host_id |                                         |
| binding_profile |                                         |
| binding_vif_details |                                         |
| binding_vif_type | unbound                                 |
| binding_vnic_type | normal                                 |
| created_at    | 2016-10-20T02:02:33Z                   |
| description   |                                         |
| device_id     |                                         |
| device_owner  |                                         |
| extra_dhcp_opts |                                         |
| fixed_ips     | ip_address='172.24.4.230', subnet_id='dc608964-9af3-4fed-9f06-6d3844fb9b9b' |
| headers       |                                         |
| id            | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39   |
| mac_address   | fa:16:3e:33:c4:75                       |
| name          | parent-trunk-port                       |
| network_id    | 871a6bd8-4193-45d7-a300-dcb2420e7cc3   |
| project_id    | 745d33000ac74d30a77539f8920555e7      |
| project_id    | 745d33000ac74d30a77539f8920555e7      |
| revision_number | 4                                       |
| security_groups | 59e2af18-93c6-4201-861b-19a8a8b79b23  |
| status        | DOWN                                    |
| updated_at    | 2016-10-20T02:02:33Z                   |
+-----+

```

3. 使用在第 2 步中创建的端口创建中继。在本例中，中继名为 parent-trunk。

```

openstack network trunk create --parent-port parent-trunk-port parent-trunk
+-----+
| Field      | Value                                     |
+-----+
| admin_state_up | UP                                       |
| created_at    | 2016-10-20T02:05:17Z                   |
| description   |                                         |
| id           | 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 |
| name         | parent-trunk                           |
| port_id      | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 |
| revision_number | 1                                       |
| status       | DOWN                                    |
| sub_ports    |                                         |
| tenant_id    | 745d33000ac74d30a77539f8920555e7      |
| updated_at    | 2016-10-20T02:05:17Z                   |
+-----+

```

4. 查看中继连接：

```
openstack network trunk list
```

```
+-----+-----+-----+-----+
| ID                | Name      | Parent Port          | Description |
+-----+-----+-----+-----+
| 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 | parent-trunk | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 |           |
+-----+-----+-----+-----+
```

#### 5. 查看中继连接的详情：

```
openstack network trunk show parent-trunk
```

```
+-----+-----+
| Field      | Value                |
+-----+-----+
| admin_state_up | UP                  |
| created_at   | 2016-10-20T02:05:17Z |
| description  |                      |
| id          | 0e4263e2-5761-4cf6-ab6d-b22884a0fa88 |
| name        | parent-trunk        |
| port_id     | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 |
| revision_number | 1                  |
| status      | DOWN                |
| sub_ports   |                      |
| tenant_id   | 745d33000ac74d30a77539f8920555e7 |
| updated_at  | 2016-10-20T02:05:17Z |
+-----+-----+
```

## 11.5. 在中继中添加子端口

### 1. 创建 neutron 端口。

这个端口是与中继的子端口连接。您还必须指定分配给父端口的 MAC 地址：

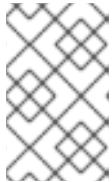
```
openstack port create --network private --mac-address fa:16:3e:33:c4:75 subport-trunk-port
```

```
+-----+-----+-----+-----+
| Field      | Value                |
+-----+-----+-----+-----+
| admin_state_up | UP                  |
| allowed_address_pairs |                    |
| binding_host_id |                    |
| binding_profile |                    |
| binding_vif_details |                    |
| binding_vif_type | unbound            |
| binding_vnic_type | normal             |
| created_at   | 2016-10-20T02:08:14Z |
| description  |                    |
| device_id    |                    |
| device_owner |                    |
| extra_dhcp_opts |                    |
| fixed_ips    | ip_address='10.0.0.11', subnet_id='1a299780-56df-4c0b-a4c0-c5a612cef2e8' |
| headers     |                    |
| id          | 479d742e-dd00-4c24-8dd6-b7297fab3ee9 |
| mac_address  | fa:16:3e:33:c4:75 |
| name        | subport-trunk-port |
+-----+-----+-----+-----+
```

```

| network_id      | 3fe6b758-8613-4b17-901e-9ba30a7c4b51 |
| project_id     | 745d33000ac74d30a77539f8920555e7 |
| project_id     | 745d33000ac74d30a77539f8920555e7 |
| revision_number | 4 |
| security_groups | 59e2af18-93c6-4201-861b-19a8a8b79b23 |
| status         | DOWN |
| updated_at     | 2016-10-20T02:08:15Z |
+-----+-----+-----+-----+

```



### 注意

如果您收到错误 **HttpException: Conflict**，请确认您在不同的网络上创建子端口到具有父中继端口的子端口。本例将公共网络用于父中继端口，子端口为 private。

2. 将端口与中继(父中继)关联，并指定 VLAN ID (55)：

```

openstack network trunk set --subport port=subport-trunk-port,segmentation-
type=vlan,segmentation-id=55 parent-trunk

```

## 11.6. 配置实例以使用中继

您必须将虚拟机实例操作系统配置为使用分配给子端口的 Red Hat OpenStack Platform (RHOSP) 网络服务(neutron)的 MAC 地址。您还可以在子端口创建步骤中将子端口配置为使用特定的 MAC 地址。

### 先决条件

- 如果您要对 Compute 节点进行实时迁移，请确保为您的 RHOSP 部署正确设置了 RHOSP 网络服务 RPC 响应超时。RPC 响应超时值可能会因站点而异，并依赖于系统速度。常规建议是为/100 中继端口至少设置 120 秒的值设为 120 秒。  
最佳实践是测量 RHOSP 部署的中继端口绑定进程时间，然后相应地设置 RHOSP 网络服务 RPC 响应超时。尝试使 RPC 响应超时值较低，但也为 RHOSP 网络服务提供足够的时间来接收 RPC 响应。更多信息请参阅 [第 11.7 节“配置网络服务 RPC 超时”](#)。

### 流程

1. 使用 network trunk 命令查看 **网络中继** 的配置。

#### 示例

```
$ openstack network trunk list
```

#### 输出示例

```

+-----+-----+-----+-----+
| ID          | Name          | Parent Port | Description |
+-----+-----+-----+-----+
| 0e4263e2-5761-4cf6- | parent-trunk | 20b6fdf8-0d43-475a- |
| ab6d-b22884a0fa88 |              | a0f1-ec8f757a4a39 |
+-----+-----+-----+-----+

```

#### 示例

```
$ openstack network trunk show parent-trunk
```

### 输出示例

```
+-----+-----+
| Field      | Value                                     |
+-----+-----+
| admin_state_up | UP                                       |
| created_at   | 2021-10-20T02:05:17Z                   |
| description  |                                           |
| id          | 0e4263e2-5761-4cf6-ab6d-b22884a0fa88   |
| name        | parent-trunk                           |
| port_id     | 20b6fdf8-0d43-475a-a0f1-ec8f757a4a39   |
| revision_number | 2                                       |
| status      | DOWN                                    |
| sub_ports   | port_id='479d742e-dd00-4c24-8dd6-b7297fab3ee9', segm |
|             | entation_id='55', segmentation_type='vlan' |
| tenant_id   | 745d33000ac74d30a77539f8920555e7     |
| updated_at  | 2021-08-20T02:10:06Z                   |
+-----+-----+
```

2. 创建使用父 **port-id** 作为其 vNIC 的实例。

### 示例

```
openstack server create --image cirros --flavor m1.tiny --security-group default --key-name sshaccess --nic port-id=20b6fdf8-0d43-475a-a0f1-ec8f757a4a39 testInstance
```

### 输出示例

```
+-----+-----+
| Property          | Value                                     |
+-----+-----+
| OS-DCF:diskConfig | MANUAL                                   |
| OS-EXT-AZ:availability_zone |                                           |
| OS-EXT-SRV-ATTR:host | -                                       |
| OS-EXT-SRV-ATTR:hostname | testinstance                           |
| OS-EXT-SRV-ATTR:hypervisor_hostname | -                                       |
| OS-EXT-SRV-ATTR:instance_name |                                           |
| OS-EXT-SRV-ATTR:kernel_id |                                           |
| OS-EXT-SRV-ATTR:launch_index | 0                                       |
| OS-EXT-SRV-ATTR:ramdisk_id |                                           |
| OS-EXT-SRV-ATTR:reservation_id | r-juqco0e1                             |
| OS-EXT-SRV-ATTR:root_device_name | -                                       |
| OS-EXT-SRV-ATTR:user_data | -                                       |
| OS-EXT-STS:power_state | 0                                       |
| OS-EXT-STS:task_state | scheduling                               |
| OS-EXT-STS:vm_state | building                                 |
| OS-SRV-USG:launched_at | -                                       |
| OS-SRV-USG:terminated_at | -                                       |
| accessIPv4          |                                           |
| accessIPv6          |                                           |
| adminPass           | uMyL8PnZRBwQ                           |
| config_drive        |                                           |
+-----+-----+
```



```

| created          | 2021-08-20T03:02:51Z          |
| description     | -                              |
| flavor          | m1.tiny (1)                   |
| hostId          |                                |
| host_status     |                                |
| id              | 88b7aede-1305-4d91-a180-67e7eac |
|                 | 8b70d                          |
| image           | cirros (568372f7-15df-4e61-a05f |
|                 | -10954f79a3c4)                |
| key_name        | sshaccess                      |
| locked          | False                          |
| metadata        | {}                              |
| name            | testInstance                   |
| os-extended-volumes:volumes_attached | []                              |
| progress        | 0                               |
| security_groups | default                         |
| status          | BUILD                           |
| tags            | []                              |
| tenant_id       | 745d33000ac74d30a77539f8920555e |
|                 | 7                               |
| updated         | 2021-08-20T03:02:51Z          |
| user_id         | 8c4aea738d774967b4ef388eb41fef5 |
|                 | e                               |
+-----+-----+

```

## 其他资源

- [配置网络服务 RPC 超时](#)

## 11.7. 配置网络服务 RPC 超时

在某些情况下，您必须修改 Red Hat OpenStack Platform (RHOSP) 网络服务(neutron) RPC 响应超时。例如，如果超时值过低，使用中继端口的 Compute 节点的实时迁移可能会失败。

RPC 响应超时值可能会因站点而异，并依赖于系统速度。常规建议是为/100 中继端口至少设置 120 秒的值为 120 秒。

如果您的站点使用中继端口，最佳实践是测量 RHOSP 部署的中继端口绑定进程时间，然后相应地设置 RHOSP 网络服务 RPC 响应超时。尝试使 RPC 响应超时值较低，但也为 RHOSP 网络服务提供足够的时间来接收 RPC 响应。

通过使用手动 hieradata 覆盖 `rpc_response_timeout`，您可以为 RHOSP 网络服务设置 RPC 响应超时值。

### 流程

1. 在 undercloud 主机上，以 stack 用户身份登录，创建自定义 YAML 环境文件。

### 示例

```
$ vi /home/stack/templates/my-modules-environment.yaml
```

## 提示

RHOSP 编排服务 (heat) 使用一组名为 *template* (模板) 的计划来安装和配置您的环境。您可以使用自定义环境文件来自定义 overcloud 的各个方面，这是为 heat 模板 提供自定义的特殊模板。

2. 在 **ExtraConfig** 下的 YAML 环境文件中，设置 **rpc\_response\_timeout** 的适当值（以秒为单位）。（默认值为 60 秒。）

## 示例

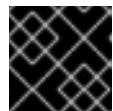
```
parameter_defaults:
  ExtraConfig:
    neutron::rpc_response_timeout: 120
```



## 注意

RHOSP 编排服务(heat)使用您在自定义环境文件中设置的值更新所有 RHOSP 节点，但这个值仅影响 RHOSP 网络组件。

3. 运行 **openstack overcloud deploy** 命令，并包含核心 heat 模板、环境文件和新的自定义环境文件。



## 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

## 示例

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-modules-environment.yaml
```

## 其他资源

- [高级 Overcloud 自定义指南中的环境文件](#)。
- [高级 Overcloud 自定义指南中的创建 overcloud 中包括环境文件](#)

## 11.8. 了解中继状态

- **ACTIVE** : 中继按预期工作，没有当前的请求。
- **下载** : 中继的虚拟和物理资源不同步。这可以在协商过程中是临时状态。
- **BUILD** : 已有一个请求，并且资源已调配。成功完成中继后，中继将返回到 **ACTIVE**。
- **DEGRADED** : 调配请求没有完成，因此中继仅被部分置备。建议您删除子端口并尝试。
- **ERROR** : 拉取请求失败。删除导致错误将中继返回到运行状况分级状态的资源。不要在 **ERROR** 状态下增加更多子端口，因为这可能导致更多问题。

## 第 12 章 配置 RBAC 策略

### 12.1. RBAC 策略概述

OpenStack 网络中基于角色的访问控制(RBAC)策略允许对共享的 *neutron* 网络进行精细控制。OpenStack 网络使用 RBAC 表来控制项目间共享 *neutron* 网络，允许管理员控制将实例附加到网络的权限。

因此，云管理员可以删除某些项目创建网络的能力，并可允许它们附加到与其项目对应的现有网络。

### 12.2. 创建 RBAC 策略

本例流程演示了如何使用基于角色的访问控制(RBAC)策略来授予项目对共享网络的访问权限。

1. 查看可用网络列表：

```
# openstack network 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. 查看项目列表：

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

3. 为 **web-servers** 网络创建一个 RBAC 条目，它授予对 *auditors* 项目的访问权限 (**4b0b98f8c6c040f38ba4f7146e8680f5**)：

```
# openstack network rbac create --type network --target-project
4b0b98f8c6c040f38ba4f7146e8680f5 --action access_as_shared web-servers
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_project | 4b0b98f8c6c040f38ba4f7146e8680f5 |
| project_id    | 98a2f53c20ce4d50a40dac4a38016c69 |
+-----+-----+
```

因此，*审核员* 项目中的用户可以将实例连接到 **web-servers** 网络。

## 12.3. 查看 RBAC 策略

1. 运行 **openstack network rbac list** 命令，以检索现有基于角色的访问控制(RBAC)策略的 ID：

```
# openstack network rbac list
+-----+-----+
| id                | object_type | object_id                |
+-----+-----+
| 314004d0-2261-4d5e-bda7-0181fcf40709 | network    | fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 |
| bbab1cf9-edc5-47f9-ae3-a413bd582c0a | network    | 9b2f4feb-fee8-43da-bb99-032e4aaf3f85 |
+-----+-----+
```

2. 运行 **openstack network rbac-show** 命令来查看特定 RBAC 条目的详情：

```
# openstack network 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_project | 4b0b98f8c6c040f38ba4f7146e8680f5 |
| project_id | 98a2f53c20ce4d50a40dac4a38016c69 |
+-----+-----+
```

## 12.4. 删除 RBAC 策略

1. 运行 **openstack network rbac list** 命令，以检索现有基于角色的访问控制(RBAC)策略的 ID：

```
# openstack network rbac list
+-----+-----+
| id                | object_type | object_id                |
+-----+-----+
| 314004d0-2261-4d5e-bda7-0181fcf40709 | network    | fa9bb72f-b81a-4572-9c7f-7237e5fcabd3 |
| bbab1cf9-edc5-47f9-ae3-a413bd582c0a | network    | 9b2f4feb-fee8-43da-bb99-032e4aaf3f85 |
+-----+-----+
```

2. 运行 **openstack network rbac delete** 命令，使用您要删除的 RBAC 的 ID 来删除 RBAC：

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

## 12.5. 为外部网络授予 RBAC 策略访问权限

您可以使用 `--action access_as_external` 参数为外部网络（附加了网关接口的网络）授予基于角色的访问控制(RBAC)策略访问权限。

完成以下示例流程中的步骤，为 `web-servers` 网络创建一个 RBAC，并授予对工程项目 (`c717f263785d4679b16a122516247deb`) 的访问权限：

- 使用 `--action access_as_external` 选项创建一个新的 RBAC 策略：

```
# openstack network rbac create --type network --target-project
c717f263785d4679b16a122516247deb --action access_as_external web-servers
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_project | c717f263785d4679b16a122516247deb |
| project_id | c717f263785d4679b16a122516247deb |
+-----+-----+
```

因此，工程团队项目中的用户可以查看网络或将实例连接到其中：

```
$ openstack network 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)

### 13.1. 了解分布式虚拟路由(DVR)

当您部署 Red Hat OpenStack Platform 时，您可以在集中式路由模型或 DVR 之间进行选择。

每个模型都有优点和缺点。使用本文档仔细规划集中式路由或 DVR 是否更适合您的需要。

新的默认 RHOSP 部署使用 DVR 和具有 Open Virtual Network 机制驱动程序(ML2/OVN)的 Modular Layer 2 插件。

在 ML2/OVS 部署中默认禁用 DVR。

#### 13.1.1. 第 3 层路由概述

Red Hat OpenStack Platform Networking 服务(neutron)为项目网络提供路由服务。如果没有路由器，项目网络中的虚拟机实例可以通过共享 L2 广播域与其他实例通信。创建路由器并将其分配到项目网络可让该网络中的实例与其他项目网络或上游通信（如果为路由器定义了外部网关）。

#### 13.1.2. 路由流

Red Hat OpenStack Platform (RHOSP)中的路由服务可分为三个主要流：

- **east-West 路由** - 同一项目中不同网络之间的流量路由。这个流量不会离开 RHOSP 部署。此定义适用于 IPv4 和 IPv6 子网。
- **带有浮动 IP 的北方路由** - 浮动 IP 寻址是一个一对一的网络地址转换(NAT)，可以修改，以及虚拟机实例之间的浮点。虽然浮动 IP 建模为浮动 IP 和网络服务(neutron)端口之间的一对一关联，但它们通过与执行 NAT 转换的网络服务路由器关联来实现。浮动 IP 本身从提供外部连接的 uplink 网络获取。因此，实例可以与外部资源（如互联网上的端点）通信，或者以其他方式通信。浮动 IP 是一个 IPv4 概念，不适用于 IPv6。假设项目使用的 IPv6 寻址使用全局多播地址(GUAs)，且项目之间没有重叠，因此可以在没有 NAT 的情况下路由。
- **没有浮动 IP 的 North-South 路由**（也称为 SNAT） - 网络服务为没有分配浮动 IP 的实例提供默认端口地址转换(PAT)服务。通过此服务，实例可以通过路由器与外部端点通信，但不能以其他方式通信。例如，一个实例可以浏览互联网上的网站，但外部的 Web 浏览器无法浏览托管在实例的网站。SNAT 仅适用于 IPv4 流量。此外，分配的 GUAs 前缀的网络服务网络不需要网络服务路由器外部网关端口上的 NAT 来访问外部世界。

#### 13.1.3. 集中式路由

最初，网络服务(neutron)设计为具有由 Neutron L3 代理管理的项目虚拟路由器的集中式路由模型，它们都由 Neutron L3 代理部署在专用节点或节点上（称为网络节点或 Controller 节点）。这意味着，每次需要路由功能(east/west、浮动 IP 或 SNAT)时，流量都会遍历拓扑中的专用节点。这引入了多个挑战，并导致非优化流量流。例如：

- **通过 Controller 节点的实例流间的网络流量** - 当两个实例需要通过 L3 相互通信时，流量必须经过 Controller 节点。即使实例调度到同一 Compute 节点上，流量仍然必须离开 Compute 节点，流向 Controller，并返回 Compute 节点。这对性能造成负面影响。
- **具有浮动 IP 接收和发送数据包的实例** - 外部网络接口仅在 Controller 节点上可用，因此流量是否源自于实例，还是从外部网络发送到实例，它必须通过 Controller 节点流。因此，在大型环境中，Controller 节点可能会大量流量负载。这会影响性能和可扩展性，还需要仔细规划以满足外部网络网关接口中的足够带宽。这同样适用于 SNAT 流量的要求。

为更好地扩展 L3 代理，网络服务可以使用 L3 HA 功能，该功能可在多个节点之间分发虚拟路由器。如果 Controller 节点丢失，HA 路由器将故障转移到另一节点上的待机，且在 HA 路由器故障转移完成前数据包丢失。

## 13.2. DVR 概述

分布式虚拟路由(DVR)提供替代路由设计。DVR 通过部署 L3 代理并在每个 Compute 节点上调度路由器，来隔离 Controller 节点的故障域并优化网络流量。DVR 具有以下特征：

- east-West 流量以分布式方式直接在 Compute 节点上路由。
- 具有浮动 IP 的北向流量在 Compute 节点上分发和路由。这需要外部网络连接到每个 Compute 节点。
- 没有浮动 IP 的 North-South 流量不会被分发，仍然需要专用 Controller 节点。
- Controller 节点上的 L3 代理使用 **dvr\_snat** 模式，以便节点只提供 SNAT 流量。
- neutron 元数据代理在所有 Compute 节点上分发并部署。元数据代理服务托管在所有分布式路由器上。

## 13.3. DVR 已知问题和注意事项

- 对 DVR 的支持仅限于 ML2 核心插件，以及 Open vSwitch (OVS)机制驱动程序或 ML2/OVN 机制驱动程序。不支持其他后端。
- 在 ML2/OVS DVR 部署中，Red Hat OpenStack Platform 负载均衡服务(octavia)的网络流量通过 Controller 和网络节点，而不是计算节点。
- 使用 ML2/OVS 机制驱动程序网络后端和 DVR 时，可以创建 VIP。但是，使用 **allowed\_address\_pairs** 分配给绑定端口的 IP 地址应与虚拟端口 IP 地址(/32)匹配。如果您将 CIDR 格式 IP 地址用于绑定的端口 **allowed\_address\_pairs**，则后端中没有配置端口转发，并且 CIDR 中任何 IP 的流量都无法访问绑定 IP 端口。
- SNAT（源网络地址转换）流量不会被分发，即使启用了 DVR。SNAT 可以正常工作，但所有入口/出口流量都必须遍历集中 Controller 节点。
- 在 ML2/OVS 部署中，即使启用了 DVR，IPv6 流量也不会分发。所有入口/出口流量都通过集中式 Controller 节点。如果您在 ML2/OVS 中广泛使用 IPv6 路由，请不要使用 DVR。请注意，在 ML2/OVN 部署中，始终分配所有 east/west 流量，并在配置 DVR 时分发 north/south 流量。
- 在 ML2/OVS 部署中，与 L3 HA 结合使用 DVR。如果您在 Red Hat OpenStack Platform 16.2 director 中使用 DVR，则禁用 L3 HA。这意味着，路由器仍然调度到网络节点上（并在 L3 代理之间共享）上，但如果一个代理失败，则此代理托管的所有路由器也会失败。这只会影响 SNAT 流量。在这种情况下，建议使用 **allow\_automatic\_l3agent\_failover** 功能，以便在一个网络节点失败时，路由器会被重新调度到不同的节点。
- 对于 ML2/OVS 环境，DHCP 服务器没有分布，并部署到 Controller 节点上。ML2/OVS neutron DHCP 代理（用于管理 DHCP 服务器）部署在 Controller 节点上的高可用性配置中，无论路由设计如何（集中式或 DVR）。
- Compute 节点需要附加到外部网桥的外部网络上的接口。它们使用这个接口附加到外部路由器网关的 VLAN 或扁平网络，以托管浮动 IP，并为使用浮动 IP 的虚拟机执行 SNAT。

- 在 ML2/OVS 部署中，每个 Compute 节点都需要一个额外的 IP 地址。这是因为外部网关端口和浮动 IP 网络命名空间的实现。
- VLAN、GRE 和 VXLAN 都支持项目数据分离。使用 GRE 或 VXLAN 时，您必须启用 L2 Population 功能。Red Hat OpenStack Platform director 在安装过程中强制实施 L2 Population。

### 13.4. 支持的路由架构

Red Hat OpenStack Platform (RHOSP)支持 RHOSP 版本中的集中式、高可用性(HA)路由和分布式虚拟路由(DVR)：

- RHOSP 集中式 HA 路由支持从 RHOSP 8 开始。
- RHOSP 分布式路由支持从 RHOSP 12 开始。

### 13.5. 使用 ML2 OVS 部署 DVR

要在 ML2/OVS 部署中部署和管理分布式虚拟路由(DVR)，您可以在 heat 模板和环境文件中配置设置。

您可以使用 heat 模板设置来置备主机网络：

- 为 Compute 和 Controller 节点上的外部网络流量配置连接到物理网络的接口。
- 在 Compute 和 Controller 节点上创建一个网桥，具有外部网络流量的接口。

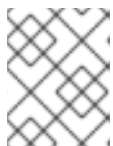
您还将网络服务(neutron)配置为与置备的网络环境匹配，并允许流量使用网桥。

默认设置仅作为指南提供。它们不应该在生产环境中工作，或测试可能需要自定义网络隔离、专用 NIC 或其他变量因素。在设置环境中，您需要正确配置 L2 代理使用的网桥映射类型参数，并为其他代理（如 L3 代理）使用外部的网桥。

以下示例演示了如何使用典型的默认值配置概念验证环境。

#### 流程

1. 验证 **OS::TripleO::Compute::Net::SoftwareConfig** 的值与 *overcloud-resource-registry.yaml* 文件或部署命令中包含的环境文件中的 **OS::TripleO::Controller::Net::SoftwareConfig** 值匹配。这个值命名一个文件，如 *net\_config\_bridge.yaml*。命名文件配置外部网络 Compute 节点 L2 代理的 Neutron 网桥映射。网桥路由在 DVR 部署中托管的 Compute 节点上的浮动 IP 地址的流量。通常，您可以在部署 overcloud 时使用的网络环境文件中找到此文件名值，如 *environments/net-multiple-nics.yaml*。



#### 注意

如果自定义 Compute 节点的网络配置，可能需要将适当的配置添加到自定义文件中。

2. 验证 Compute 节点是否具有外部网桥。
  - a. 生成 **openstack-tripleo-heat-templates** 目录的本地副本。
  - b. **\$ cd <local\_copy\_of\_templates\_directory>**
  - c. 运行 **process-templates** 脚本，将模板呈现到临时输出目录中：



```
$ ./tools/process-templates.py -r <roles_data.yaml> \
-n <network_data.yaml> -o <temporary_output_directory>
```

- d. 检查 `<temporary_output_directory>/network/config` 中的角色文件。
3. 如果需要，自定义 Compute 模板，使其包含与 Controller 节点匹配的外部网桥，并在环境文件中命名 `OS::TripleO::Compute::Net::Net::Net::Net::SoftwareConfig` 中的自定义文件路径。
4. 在部署 overcloud 时，在部署命令中包含 `environments/services/neutron-ovs-dvr.yaml` 文件：

```
$ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovs-dvr.yaml
```

5. 验证 L3 HA 是否已禁用。



### 注意

L3 代理的外部网桥配置已在 Red Hat OpenStack Platform 13 中弃用，并在 Red Hat OpenStack Platform 15 中删除。

## 13.6. 将集中式路由器迁移到分布式路由

本节介绍升级到使用 L3 HA 集中式路由的 Red Hat OpenStack Platform 部署的分布式路由。

### 流程

1. 升级部署，并验证它是否正常工作。
2. 运行 `director stack update` 来配置 DVR。
3. 确认路由功能通过现有的路由器正常工作。
4. 您无法将 L3 HA 路由器转换为直接 分发。相反，对于每个路由器，禁用 L3 HA 选项，然后启用分布式选项：
  - a. 禁用路由器：

#### 示例

```
$ openstack router set --disable router1
```

- b. 清除高可用性：

#### 示例

```
$ openstack router set --no-ha router1
```

- c. 将路由器配置为使用 DVR：

#### 示例

```
$ openstack router set --distributed router1
```

- d. 启用路由器：

### 示例

```
$ openstack router set --enable router1
```

- e. 确认分布式路由功能正确。

### 其他资源

- [使用 ML2 OVS 部署 DVR](#)

## 13.7. 部署禁用分布式虚拟路由(DVR)的 ML2/OVN OPENSTACK

新的 Red Hat OpenStack Platform (RHOSP)部署默认为 neutron Modular Layer 2 插件，并带有 Open Virtual Network 机制驱动程序(ML2/OVN)和 DVR。

在 DVR 拓扑中，具有浮动 IP 地址的计算节点路由虚拟机实例和为路由器提供外部连接（南向流量的网络）之间的流量。实例（南向流量）之间的流量也被分发。

您可以选择禁用 DVR 部署。这会禁用 north-south DVR，需要 north-south 流量来遍历控制器或网络器节点。east-west 路由始终在 ML2/OVN 部署中分发，即使 DVR 被禁用。

### 先决条件

- RHOSP 16.2 发行版可用于自定义和部署。

### 流程

1. 创建自定义环境文件并添加以下配置：

```
parameter_defaults:  
  NeutronEnableDVR: false
```

2. 若要应用此配置，可部署 overcloud，将自定义环境文件添加到堆栈中以及其他环境文件。例如：

```
(undercloud) $ openstack overcloud deploy --templates \  
-e [your environment files]  
-e /home/stack/templates/<custom-environment-file>.yaml
```

### 13.7.1. 其他资源

- *Networking Guide* 中的 [Understanding distributed virtual routing \(DVR\)](#)。

## 第 14 章 使用 IPV6 的项目网络

### 14.1. IPV6 子网选项

当您在 Red Hat OpenStack Platform (RHOSP)项目网络中创建 IPv6 子网时，您可以指定地址模式和路由器广告模式来获取特定的结果，如下表所述。




#### 注意

RHOSP 不支持来自 ML2/OVN 部署中的外部实体的 IPv6 前缀长度。您必须从外部委托路由器获取 Global Unicast Address 前缀，并在创建 IPv6 子网时使用 **subnet-range** 参数进行设置。

例如：

```
openstack subnet create
--subnet-range 2002:c000:200::64
--no-dhcp
--gateway 2002:c000:2fe::
--dns-nameserver 2002:c000:2fe::
--network provider
provider-subnet-2002:c000:200::
```

RA 模式	地址模式	结果
ipv6_ra_mode=not set	ipv6-address-mode=slaac	<p>该实例使用无状态地址自动配置 (SLAAC)从外部路由器（不是由 OpenStack 网络管理）接收 IPv6 地址。</p>  <p><b>注意</b></p> <p>OpenStack 网络只支持使用 EUI-64 IPv6 地址分配。这允许简化的 IPv6 网络，作为基于 64 位以及 MAC 地址的自分配地址。您不能使用不同的子网掩码和 <i>address_assign_type</i> 创建子网。</p>
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateful	<p>实例使用 DHCPv6 有状态的 <b>OpenStack Networking (dnsmasq)</b>接收 IPv6 地址和可选信息。</p>

RA 模式	地址模式	结果
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateless	该实例使用 SLAAC 从外部路由器接收 IPv6 地址，以及使用 <b>DHCPv6 stateless</b> 收集来自 OpenStack Networking (dnsmasq) 的可选信息。
ipv6_ra_mode=slaac	ipv6-address-mode=not-set	该实例使用 SLAAC 接收来自 OpenStack Networking (radvd) 的一个 IPv6 地址。
ipv6_ra_mode=dhcpv6-stateful	ipv6-address-mode=not-set	该实例使用 DHCPv6 <b>有状态</b> 从外部 DHCPv6 服务器接收 IPv6 地址和可选信息。
ipv6_ra_mode=dhcpv6-stateless	ipv6-address-mode=not-set	该实例使用 SLAAC 从 OpenStack Networking (radvd) 接收 IPv6 地址，以及使用 <b>DHCPv6 stateless</b> 接收来自外部 DHCPv6 服务器的可选信息。
ipv6_ra_mode=slaac	ipv6-address-mode=slaac	实例使用 <b>SLAAC</b> 接收来自 OpenStack Networking (radvd) 的一个 IPv6 地址。
ipv6_ra_mode=dhcpv6-stateful	ipv6-address-mode=dhcpv6-stateful	实例使用 <b>DHCPv6 stateful</b> 接收来自 OpenStack Networking (dnsmasq) 的一个 IPv6 地址，使用 <b>DHCPv6 stateful</b> 接收来自 OpenStack Networking (dnsmasq) 的可选信息。
ipv6_ra_mode=dhcpv6-stateless	ipv6-address-mode=dhcpv6-stateless	实例使用 <b>SLAAC</b> 接收来自 OpenStack Networking (radvd) 的一个 IPv6 地址，使用 <b>DHCPv6 stateless</b> 接收来自 OpenStack Networking (dnsmasq) 的可选信息。

## 14.2. 使用 STATEFUL DHCPV6 创建 IPV6 子网

您可以在 Red Hat OpenStack (RHOSP) 项目网络中创建 IPv6 子网。

例如，您可以在名为 QA 的项目中名为 database-servers 的网络中，使用 Stateful DHCPv6 创建一个 IPv6 子网。

### 流程

1. 检索您要创建 IPv6 子网的项目的项目 ID。这些值在 OpenStack 部署之间是唯一的，因此您的值与本例中的值不同。

```
# openstack project list
+-----+-----+
| ID                | Name  |
+-----+-----+
| 25837c567ed5458fbb441d39862e1399 | QA   |
| f59f631a77264a8eb0defc898cb836af | admin |
| 4e2e1951e70643b5af7ed52f3ff36539 | demo  |
| 8561dff8310e4cd8be4b6fd03dc8acf5 | services |
+-----+-----+
```

2. 检索 OpenStack Networking (neutron)中存在的所有网络列表，并记下您要托管 IPv6 子网的网络名称：

```
# openstack network 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. 在 **openstack subnet create** 命令中包含项目 ID、网络名称和 ipv6 地址模式：

```
# openstack subnet create --ip-version 6 --ipv6-address-mode dhcpv6-stateful --project 25837c567ed5458fbb441d39862e1399 --network database-servers --subnet-range fdf8:f53b:82e4::53/125 subnet_name
```

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 |
+-----+-----+-----+-----+
```

1. 通过查看网络列表来验证此配置。请注意，`database-servers` 的条目现在反映了新创建的 IPv6 子网：

```
# openstack network list
+-----+-----+-----+-----+
| id                | name                | subnets                |
+-----+-----+-----+-----+
| 6aff6826-4278-4a35-b74d-b0ca0cbbba340 | database-servers | cdfc3398-997b-46eb-9db1-ebb88f7de05 |
| df8: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    |
+-----+-----+-----+-----+
|-----+
|-----+
```

## 结果

因此，在添加到 `database-servers` 子网时，QA 项目创建的实例可能会接收 DHCP IPv6 地址：

```
# openstack server list
+-----+-----+-----+-----+-----+-----+
| ID                | Name                | Status | Task State | Power State | Networks
|
+-----+-----+-----+-----+-----+-----+
| fad04b7a-75b5-4f96-aed9-b40654b56e03 | corp-vm-01 | ACTIVE | -          | Running    |
| database-servers=df8:f53b:82e4::52 |
+-----+-----+-----+-----+-----+-----+
|-----+
|-----+
```

## 其他资源

要找到路由器公告模式和地址模式组合，以便在 IPv6 子网中获得特定结果，请参阅[网络指南](#)中的 [IPv6 子网选项](#)。

## 第 15 章 管理项目配额

### 15.1. 配置项目配额

OpenStack Networking (neutron)支持使用配额来限制租户/项目创建的资源数量。

#### 流程

- 您可以在 `/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf` 文件中为各种网络组件设置项目配额。  
例如，要限制项目可以创建的路由器数量，请更改 `quota_router` 值：

```
quota_router = 10
```

在本例中，每个项目限制为最多 10 个路由器。

有关配额设置列表，请参阅立即遵循的部分。

### 15.2. L3 配额选项

以下是可用于第 3 层(L3)网络的配额选项：

- `quota_floatingip` - 项目可用的浮动 IP 数量。
- `quota_network` - 项目可用的网络数量。
- `quota_port` - 项目可用的端口数。
- `quota_router` - 项目可用的路由器数量。
- `quota_subnet` - 项目可用的子网数量。
- `quota_vip` - 项目可用的虚拟 IP 地址数量。

### 15.3. 防火墙配额选项

以下是用于管理项目的防火墙的配额选项：

- `quota_firewall` - 项目可用的防火墙数。
- `quota_firewall_policy` - 项目可用的防火墙策略数。
- `quota_firewall_rule` - 项目可用的防火墙规则数量。

### 15.4. 安全组配额选项

网络服务配额引擎管理安全组和安全组规则，在创建默认安全组（以及接受 IPv4 和 IPv6 的所有出口流量）前无法将所有配额设置为零。当您创建新项目时，网络服务不会在创建网络或端口之前创建新的安全组，或者直到您列出安全组或安全组规则。

以下是可用于管理项目可以创建的安全组数的配额选项：

- `quota_security_group` - 项目可用的安全组数量。

- `quota_security_group_rule` - 项目可用的安全组规则数量。

## 15.5. 管理配额选项

以下是管理员用于管理项目配额的附加选项：

- `default_quota` - 项目可用的默认资源数量。
- `quota_health_monitor` - 项目可用的运行状况监视器数量。  
健康监视器不会消耗资源，但配额选项可用，因为 OpenStack 网络将运行状况监视器视为资源消费者。
- `quota_member` - 项目可用的池成员数量。  
池成员不消耗资源，但 `quota` 选项可用，因为 OpenStack 网络会将池成员视为资源消费者。
- `quota_pool` - 项目可用的池数量。



## 第 16 章 部署路由的供应商网络

### 16.1. 路由供应商网络的优点

在 Red Hat OpenStack Platform (RHOSP) 中，操作员可以创建路由供应商网络。路由提供商网络通常在边缘部署中使用，它依赖于多个第 2 层网络段，而不是仅有一个网段的传统网络。

路由提供商网络为最终用户简化云，因为它们只看到一个网络。对于云操作员，路由供应商网络提供可扩展和容错能力。例如，如果发生重大错误，则只有一个片段会受到影响，而不是整个网络失败。

在路由供应商网络前，Operator 通常需要从以下构架中选择：

- 单个大型第 2 层网络
- 多、较小的第 2 层网络

在扩展和减少容错（超出故障域中）时，大型第 2 层网络变得复杂。

多个较小的第 2 层网络可以更好地扩展并缩小故障域，但可为最终用户带来复杂性。

从 RHOSP 16.2 及更高版本开始，您可以使用带有 Open Virtual Network 机制驱动程序(ML2/OVN)的 Modular Layer 2 插件部署路由供应商网络。（对 ML2/Open vSwitch (OVS) 和 SR-IOV 机制驱动程序的 路由提供商网络支持在 RHOSP 16.1.1. 中引入。）

#### 其他资源

- [第 16.2 节 “路由供应商网络的基本信息”](#)

### 16.2. 路由供应商网络的基本信息

路由提供商网络与其他类型的网络不同，因为网络子网和网段之间的一对一关联。在过去，Red Hat OpenStack (RHOSP) 网络服务不支持路由供应商网络，因为网络服务要求所有子网都必须属于同一网段或没有网段。

使用路由的提供商网络时，虚拟机 (VM) 实例的 IP 地址取决于特定计算节点上可用的网络段。网络服务端口只能与一个网络段关联。

与传统网络类似，第 2 层（切换）处理同一网络段和第 3 层 (routing) 处理段之间流量传输之间的流量。

网络服务不在网段之间提供第 3 层服务。相反，它依赖于物理网络基础架构来路由子网。因此，网络服务和物理网络基础架构必须包含路由提供商网络的配置，类似于传统提供商网络。

由于 Compute 服务 (nova) 调度程序不是网络段感知，所以当部署路由供应商网络时，您必须将每个叶或机架网段或 DCN 边缘站点映射到 Compute 服务 host-aggregate 或 Availability 区域。

如果您需要 DHCP-metadata 服务，您必须为每个边缘站点或网络段定义一个可用区，以确保部署了本地 DHCP 代理。

#### 其他资源

- [第 16.1 节 “路由供应商网络的优点”](#)

### 16.3. 路由供应商网络的限制

所有机制驱动程序不支持路由供应商网络，且计算服务调度程序和其他软件的限制，如以下列表中所述：

- 不支持使用中央 SNAT 或浮动 IP 的北路路由。
- 使用 SR-IOV 或 PCI 透传时，物理网络 (physnet) 名称在中央和远程站点或网段中必须相同。您无法重复使用片段 ID。
- Compute 服务(nova)调度程序不是网段感知的。（您必须将每个网段或边缘站点映射到计算主机聚合或可用区。）目前，只有两个虚拟机实例引导选项：
  - 使用 **port-id** 而不是 IP 地址引导，指定计算可用区（网段或边缘站点）。
  - 使用 **network-id** 启动，指定计算可用区（网段或边缘站点）。
- 只有在指定目标 Compute 可用区（网段或边缘站点）时，冷或实时迁移才能正常工作。

## 16.4. 准备路由的提供商网络

在 Red Hat OpenStack Platform (RHOSP)中创建路由供应商网络前，您必须执行一些任务。

### 流程

1. 在网络中，为每个网段使用唯一的物理网络名称。这可让在子网间重复使用相同的分段详情。例如，在特定提供商网络的所有片段中使用相同的 VLAN ID。
2. 在网段之间实施路由。  
段上的每个子网必须包含该特定子网上路由器接口的网关地址。

表 16.1. 路由片段示例

segment	版本	addresses	gateway
segment1	4	203.0.113.0/24	203.0.113.1
segment1	6	fd00:203:0:113::/64	fd00:203:0:113::1
segment2	4	198.51.100.0/24	198.51.100.1
segment2	6	fd00:198:51:100::/64	fd00:198:51:100::1

3. 将片段映射到 Compute 节点。  
路由供应商网络意味着 Compute 节点驻留在不同的片段上。确保路由提供商网络中的每个 Compute 主机都与其其中一个片段直接连接。

表 16.2. 片段到 Compute 节点映射示例

主机	rack	物理网络
compute0001	rack 1	段 1
compute0002	rack 1	段 1

主机	rack	物理网络
...	...	...
compute0101	rack 2	片段 2
compute0102	rack 2	片段 2
compute0102	rack 2	片段 2
...	...	...

4. 当您使用具有 Open vSwitch 机制驱动程序(ML2/OVS)的 Modular Layer 2 插件部署时，您必须为每个片段至少部署一个 DHCP 代理。  
与传统的提供者网络不同，DHCP 代理无法支持网络中的多个网段。在包含片段的 Compute 节点上部署 DHCP 代理，而不是在网络节点上，以减少节点数。

表 16.3. 每个片段映射的 DHCP 代理示例

主机	rack	物理网络
network0001	rack 1	段 1
network0002	rack 1	段 1
...	...	...

您可以使用自定义角色文件在 Compute 节点上部署 DHCP 代理和 RHOSP Networking 服务 (neutron)元数据代理。

下面是一个示例：

```
#####
####
# Role: ComputeSriov #
#####
####
- name: ComputeSriov
  description: |
    Compute SR-IOV Role
  CountDefault: 1
  networks:
    External:
      subnet: external_subnet
    InternalApi:
      subnet: internal_api_subnet
  Tenant:
    subnet: tenant_subnet
  Storage:
    subnet: storage_subnet
```

```

RoleParametersDefault:
  TunedProfileName: "cpu-partitioning"
update_serial: 25
ServicesDefault:
- OS::TripleO::Services::Aide
- OS::TripleO::Services::AuditD
- OS::TripleO::Services::BootParams
- OS::TripleO::Services::CACerts
...
- OS::TripleO::Services::NeutronDhcpAgent
- OS::TripleO::Services::NeutronMetadataAgent
...

```

在自定义环境文件中添加以下键值对：

```

parameter_defaults:
  ....
  NeutronEnableIsolatedMetadata: 'True'
  ....

```

## 其他资源

- [第 16.5 节 “创建路由的提供商网络”](#)
- [高级 Overcloud 自定义指南中的可组合服务和自定义角色](#)

## 16.5. 创建路由的提供商网络

路由供应商网络为最终用户简化 Red Hat OpenStack Platform (RHOSP) 云，因为它们只看到一个网络。对于云操作员，路由供应商网络提供可扩展和容错能力。

执行此流程时，您可以创建一个具有两个网络片段的路由供应商网络。每个片段包含一个 IPv4 子网和一个 IPv6 子网。

### 先决条件

- 完成 `xref:prepare-routed-prov-network_deploy-routed-prov-networks` 中的步骤。

### 流程

1. 创建包含默认片段 VLAN 提供商网络。  
在本例中，VLAN 提供商网络名为 **multisegment1**，并使用名为 **provider1** 的物理网络，以及 ID 为 **128** 的 VLAN：

### 示例

```

$ openstack network create --share --provider-physical-network provider1 \
  --provider-network-type vlan --provider-segment 128 multisegment1

```

### 输出示例

```

+-----+-----+
| Field          | Value                               |

```

```

+-----+
| admin_state_up      | UP          |
| id                  | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| ipv4_address_scope  | None        |
| ipv6_address_scope  | None        |
| l2_adjacency        | True        |
| mtu                  | 1500        |
| name                 | multisegment1 |
| port_security_enabled | True        |
| provider:network_type | vlan        |
| provider:physical_network | provider1   |
| provider:segmentation_id | 128        |
| revision_number      | 1           |
| router:external      | Internal    |
| shared               | True        |
| status                | ACTIVE      |
| subnets              |             |
| tags                  | []          |
+-----+

```

## 2. 将默认网络段重命名为 **segment1**。

### a. 获取片段 ID :

```
$ openstack network segment list --network multisegment1
```

#### 输出示例

```

+-----+
| ID                  | Name        | Network                               | Network Type |
Segment |
+-----+
| 43e16869-ad31-48e4-87ce-acf756709e18 | None        | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 | vlan          |
+-----+

```

### b. 使用片段 ID, 将网络段重命名为 **segment1** :

```
$ openstack network segment set --name segment1 43e16869-ad31-48e4-87ce-acf756709e18
```

## 3. 在提供商网络上创建第二个网段。

在本例中, 网络片段使用名为 **provider2** 的物理网络, 以及 ID 为 **129** 的 VLAN :

#### 示例

```
$ openstack network segment create --physical-network provider2 \
--network-type vlan --segment 129 --network multisegment1 segment2
```

#### 输出示例

```

+-----+
| Field      | Value                               |
+-----+
| description | None                                 |
| headers    |                                     |
| id         | 053b7925-9a89-4489-9992-e164c8cc8763 |
| name       | segment2                             |
| network_id | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| network_type | vlan                                  |
| physical_network | provider2                           |
| revision_number | 1                                    |
| segmentation_id | 129                                  |
| tags       | []                                    |
+-----+

```

4. 验证网络是否包含 **segment1** 和 **segment2** 段：

```
$ openstack network segment list --network multisegment1
```

#### 输出示例

```

+-----+
----+
| ID                               | Name   | Network                               | Network Type | Segment |
+-----+
----+
| 053b7925-9a89-4489-9992-e164c8cc8763 | segment2 | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 | vlan         | 129     |
| 43e16869-ad31-48e4-87ce-acf756709e18 | segment1 | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 | vlan         | 128     |
+-----+
----+

```

5. 在 **segment1** 片段上创建一个 IPv4 子网和一个 IPv6 子网。  
在本例中，IPv4 子网使用 **203.0.113.0/24**：

#### 示例

```
$ openstack subnet create \
  --network multisegment1 --network-segment segment1 \
  --ip-version 4 --subnet-range 203.0.113.0/24 \
  multisegment1-segment1-v4
```

#### 输出示例

```

+-----+
| Field      | Value                               |
+-----+
| allocation_pools | 203.0.113.2-203.0.113.254         |
| cidr         | 203.0.113.0/24                     |
| enable_dhcp  | True                                 |
| gateway_ip   | 203.0.113.1                         |
| id          | c428797a-6f8e-4cb1-b394-c404318a2762 |
+-----+

```

```

| ip_version      | 4 |
| name            | multisegment1-segment1-v4 |
| network_id      | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| revision_number | 1 |
| segment_id      | 43e16869-ad31-48e4-87ce-acf756709e18 |
| tags            | [] |
+-----+-----+

```

在本例中，IPv6 子网使用 **fd00:203:0:113::/64**：

### 示例

```

$ openstack subnet create \
  --network multisegment1 --network-segment segment1 \
  --ip-version 6 --subnet-range fd00:203:0:113::/64 \
  --ipv6-address-mode slaac multisegment1-segment1-v6

```

### 输出示例

```

+-----+-----+
| Field      | Value |
+-----+-----+
| allocation_pools | fd00:203:0:113::2-fd00:203:0:113:ffff:ffff:ffff:ffff |
| cidr        | fd00:203:0:113::/64 |
| enable_dhcp  | True |
| gateway_ip   | fd00:203:0:113::1 |
| id          | e41cb069-9902-4c01-9e1c-268c8252256a |
| ip_version   | 6 |
| ipv6_address_mode | slaac |
| ipv6_ra_mode | None |
| name         | multisegment1-segment1-v6 |
| network_id   | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| revision_number | 1 |
| segment_id   | 43e16869-ad31-48e4-87ce-acf756709e18 |
| tags         | [] |
+-----+-----+

```



### 注意

默认情况下，提供商网络上的 IPv6 子网依赖于物理网络基础架构进行无状态地址自动配置(SLAAC)和路由器广告。

- 在 **segment2** 段上创建一个 IPv4 子网和一个 IPv6 子网。  
在本例中，IPv4 子网使用 **198.51.100.0/24**：

### 示例

```

$ openstack subnet create \
  --network multisegment1 --network-segment segment2 \
  --ip-version 4 --subnet-range 198.51.100.0/24 \
  multisegment1-segment2-v4

```

### 输出示例

```

+-----+
| Field      | Value                               |
+-----+
| allocation_pools | 198.51.100.2-198.51.100.254      |
| cidr        | 198.51.100.0/24                   |
| enable_dhcp  | True                               |
| gateway_ip   | 198.51.100.1                       |
| id          | 242755c2-f5fd-4e7d-bd7a-342ca95e50b2 |
| ip_version   | 4                                   |
| name        | multisegment1-segment2-v4         |
| network_id   | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| revision_number | 1                                   |
| segment_id   | 053b7925-9a89-4489-9992-e164c8cc8763 |
| tags        | []                                  |
+-----+

```

在本例中，IPv6 子网使用 **fd00:198:51:100::/64** 。

### 示例

```

$ openstack subnet create \
  --network multisegment1 --network-segment segment2 \
  --ip-version 6 --subnet-range fd00:198:51:100::/64 \
  --ipv6-address-mode slaac multisegment1-segment2-v6

```

### 输出示例

```

+-----+
| Field      | Value                               |
+-----+
| allocation_pools | fd00:198:51:100::2-fd00:198:51:100:ffff:ffff:ffff:ffff |
| cidr        | fd00:198:51:100::/64               |
| enable_dhcp  | True                               |
| gateway_ip   | fd00:198:51:100::1                 |
| id          | b884c40e-9cfe-4d1b-a085-0a15488e9441 |
| ip_version   | 6                                   |
| ipv6_address_mode | slaac                               |
| ipv6_ra_mode  | None                                |
| name        | multisegment1-segment2-v6         |
| network_id   | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| revision_number | 1                                   |
| segment_id   | 053b7925-9a89-4489-9992-e164c8cc8763 |
| tags        | []                                  |
+-----+

```

### 验证

1. 验证每个 IPv4 子网是否至少与一个 DHCP 代理关联：

```

$ openstack network agent list --agent-type dhcp --network multisegment1

```

### 输出示例



```

+-----+-----+-----+-----+-----+-----+
| ID                | Agent Type | Host      | Availability Zone | Alive | State | Binary |
+-----+-----+-----+-----+-----+-----+
| c904ed10-922c-4c1a-84fd-d928abaf8f55 | DHCP agent | compute0001 | nova              | :-)  |      |        |
| UP | neutron-dhcp-agent |
| e0b22cc0-d2a6-4f1c-b17c-27558e20b454 | DHCP agent | compute0101 | nova              | :-)  |      |        |
| UP | neutron-dhcp-agent |
+-----+-----+-----+-----+-----+-----+

```

- 验证 Compute 服务放置 API 中为每个片段 IPv4 子网创建了清单。  
对所有片段 ID 运行这个命令：

```

$ SEGMENT_ID=053b7925-9a89-4489-9992-e164c8cc8763
$ openstack resource provider inventory list $SEGMENT_ID

```

### 输出示例

在这个示例输出中，仅显示其中一个片段：

```

+-----+-----+-----+-----+-----+-----+
| resource_class | allocation_ratio | max_unit | reserved | step_size | min_unit | total |
+-----+-----+-----+-----+-----+-----+
| IPV4_ADDRESS  | 1.0 | 1 | 2 | 1 | 1 | 30 |
+-----+-----+-----+-----+-----+-----+

```

- 验证是否为 Compute 服务中的每个片段创建主机聚合：

```

$ openstack aggregate list

```

### 输出示例

在这个示例中，仅显示其中一个片段：

```

+-----+-----+-----+-----+-----+
| Id | Name                | Availability Zone |
+-----+-----+-----+-----+-----+
| 10 | Neutron segment id 053b7925-9a89-4489-9992-e164c8cc8763 | None              |
+-----+-----+-----+-----+-----+

```

- 启动一个或多个实例。每个实例会根据特定计算节点上使用的片段获取 IP 地址。



## 注意

如果在端口创建请求中指定固定 IP，则该特定 IP 会立即分配给端口。但是，创建一个端口并将其传递给实例会导致与传统网络不同的行为。如果在端口创建请求上没有指定固定 IP，则网络服务会将 IP 地址分配给端口，直到特定的计算节点变为明显。例如，当运行这个命令时：

```
$ openstack port create --network multisegment1 port1
```

## 输出示例

```
+-----+
| Field          | Value                                     |
+-----+
| admin_state_up | UP                                       |
| binding_vnic_type | normal                                   |
| id              | 6181fb47-7a74-4add-9b6b-f9837c1c90c4 |
| ip_allocation   | deferred                                 |
| mac_address     | fa:16:3e:34:de:9b                       |
| name            | port1                                    |
| network_id      | 6ab19caa-dda9-4b3d-abc4-5b8f435b98d9 |
| port_security_enabled | True                                     |
| revision_number | 1                                         |
| security_groups | e4fcef0d-e2c5-40c3-a385-9c33ac9289c5 |
| status          | DOWN                                     |
| tags            | []                                        |
+-----+
```

## 其他资源

- [第 16.4 节 “准备路由的提供商网络”](#)
- [命令行界面参考中的 network create](#)
- [命令行界面参考中的 网络片段创建](#)
- [Command Line Interface Reference 中的 subnet create](#)
- [命令行界面参考中的 端口创建](#)

## 16.6. 将非路由网络迁移到路由的提供商网络

您可以通过将网络的子网与网络段的 ID 关联，将非路由网络迁移到路由提供商网络。

### 先决条件

- 要迁移的非路由网络必须仅包含一个网段且只有一个子网。



## 重要

在包含多个子网或网络片段的非路由提供商网络中，无法安全地迁移到路由的提供商网络。在非路由网络中，子网分配池的地址将分配给端口，而无需考虑端口绑定到的网络段。

## 流程

1. 对于正在迁移的网络，获取当前网络段的 ID。

### 示例

```
$ openstack network segment list --network my_network
```

### 输出示例

```
+-----+-----+-----+-----+
+
+ | ID | Name | Network | Network Type | Segment |
+-----+-----+-----+-----+-----+
+
+ | 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7 | None | 45e84575-2918-471c-95c0-018b961a2984 | flat | None |
+-----+-----+-----+-----+-----+
+
```

2. 对于正在迁移的网络，获取当前子网的 ID。

### 示例

```
$ openstack network segment list --network my_network
```

### 输出示例

```
+-----+-----+-----+-----+
+ | ID | Name | Network | Subnet |
+-----+-----+-----+-----+
+ | 71d931d2-0328-46ae-93bc-126caf794307 | my_subnet | 45e84575-2918-471c-95c0-018b961a2984 | 172.24.4.0/24 |
+-----+-----+-----+-----+
```

3. 验证子网的当前 **segment\_id** 的值为 **None**。

### 示例

```
$ openstack subnet show my_subnet --c segment_id
```

### 输出示例

```
+-----+-----+
+ | Field | Value |
+-----+-----+
+ | segment_id | None |
+-----+-----+
```

4. 将 subnet **segment\_id** 的值改为网络段 ID。  
下面是一个示例：

```
$ openstack subnet set --network-segment 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7  
my_subnet
```

## 验证

- 验证子网现在是否与所需的网络段关联。

## 示例

```
$ openstack subnet show my_subnet --c segment_id
```

## 输出示例

```
+-----+-----+  
| Field | Value |  
+-----+-----+  
| segment_id | 81e5453d-4c9f-43a5-8ddf-feaf3937e8c7 |  
+-----+-----+
```

## 其他资源

- [命令行界面参考中的 subnet show](#)
- [命令行界面参考中的子网集](#)

## 第 17 章 配置允许的地址对

### 17.1. 允许的地址对概述

在 Red Hat OpenStack Platform (RHOSP) 网络环境中，*允许的地址对*用来识别特定的 MAC 地址、IP 地址或两者，以允许网络流量通过端口，而不考虑子网。当您定义允许的地址对时，您可以使用 VRRP（虚拟路由器冗余协议）等协议，该协议在两个虚拟机实例之间浮点数，以启用快速数据平面故障转移。其 IP 地址是允许对另一端口地址对的端口，称为虚拟端口(vport)。

您可以使用 Red Hat OpenStack Platform 命令行客户端 **openstack port** 命令定义允许的地址对。

#### 重要

请注意，您不应该在允许的地址对中使用带有更广泛的 IP 地址范围的默认安全组。这样做可让单个端口为同一网络中的所有其他端口绕过安全组。

例如，这个命令会影响网络中的所有端口并绕过所有安全组：

```
# openstack port set --allowed-address mac-address=3e:37:09:4b,ip-address=0.0.0.0/0 9e67d44eab334f07bf82fa1b17d824b6
```

#### 注意

使用 ML2/OVN 机制驱动程序网络后端，可以创建 VIP。但是，使用 **allowed\_address\_pairs** 分配给绑定端口的 IP 地址应与虚拟端口 IP 地址(/32)匹配。

如果您将 CIDR 格式 IP 地址用于绑定的端口 **allowed\_address\_pairs**，则后端中没有配置端口转发，并且 CIDR 中任何 IP 的流量都无法访问绑定 IP 端口。

#### 其他资源

- [命令行界面参考中的端口命令](#)
- [第 17.2 节 “创建端口并允许一个地址对”](#)
- [第 17.3 节 “添加允许的地址对”](#)

### 17.2. 创建端口并允许一个地址对

使用允许的地址对创建端口可让网络流量通过端口流，而不考虑子网。

#### 重要

不要在允许的地址对中使用带有更广泛的 IP 地址范围的默认安全组。这样做可让单个端口为同一网络中的所有其他端口绕过安全组。

#### 流程

- 使用以下命令来创建端口并允许一个地址对：

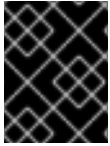
```
$ openstack port create --network <network> --allowed-address mac-address=<mac_address>,ip-address=<ip_cidr> <port_name>
```

## 其他资源

- [命令行界面参考中的端口命令](#)

## 17.3. 添加允许的地址对

您可以将允许的地址对添加到端口，以便网络流量能够通过端口流，而不考虑子网。



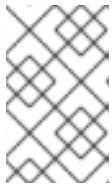
### 重要

不要在允许的地址对中使用带有更广泛的 IP 地址范围的默认安全组。这样做可让单个端口为同一网络中的所有其他端口绕过安全组。

## 流程

- 使用以下命令添加允许的地址对：

```
$ openstack port set --allowed-address mac-address=<mac_address>,ip-address=<ip_cidr>
<port>
```



### 注意

您无法设置与端口的 **mac\_address** 和 **ip\_address** 匹配的允许地址对。这是因为此类设置无效，因为与 **mac\_address** 和 **ip\_address** 匹配的流量已经允许通过端口传递。

## 其他资源

- [命令行界面参考中的端口命令](#)

## 第 18 章 日志记录安全组操作



### 注意

本发行版本中以 **技术预览** 的形式记录安全组操作，因此不受红帽完全支持。它应该只用于测试，不应在生产环境中部署。有关技术预览功能的更多信息，请参阅 [覆盖范围详细信息](#)。

请参阅以下安全组日志记录已知问题和临时解决方案：

[https://bugzilla.redhat.com/show\\_bug.cgi?id=2241184](https://bugzilla.redhat.com/show_bug.cgi?id=2241184) 和

[https://bugzilla.redhat.com/show\\_bug.cgi?id=2192918](https://bugzilla.redhat.com/show_bug.cgi?id=2192918)。

要监控虚拟机(VM)实例的流量流，您可以为安全组创建数据包日志。每个日志生成有关数据包流事件的数据流，并将其附加到启动虚拟机实例的 Compute 主机上的通用日志文件。

您可以将任何实例端口与一个或多个安全组关联，并为每个安全组定义一个或多个规则。例如，您可以创建一个规则来允许到安全组中任何虚拟机的入站 SSH 流量。您可以在同一安全组中创建另一条规则，以允许该组中的虚拟机启动和响应 ICMP (ping)消息。

然后，您可以创建日志来记录数据包流事件的组合。例如，以下命令会创建一个日志来捕获安全组 security-group1 中的所有 **ACCEPT** 事件。

```
$ openstack network log create my-log1 \
--resource-type security_group \
--resource security-group1 \
--event ACCEPT
```

您可以创建多个日志来捕获有关安全组和数据包流事件的特定组合的数据。

您可以配置以下参数：

### resource-type

您必须将此必需参数设置为 **security\_group**。

### 资源（安全组名称）

您可以选择使用 **target** 参数将日志限制到特定的安全组。例如：**--resource security-group1**。如果没有指定资源，日志将从项目中指定端口的所有安全组捕获事件。

### event（日志的事件类型）

您可以选择记录以下数据包流事件：

- **DROP**：为每个丢弃的传入或传出会话记录一个 **DROP** 日志条目。



### 注意

如果您在一个或多个安全组上日志丢弃流量，网络服务将丢弃所有安全组上的流量。

- **ACCEPT**：为安全组允许的每个新会话记录一个 **ACCEPT** 日志条目。
- **ALL**（拖放和接受）：记录所有 **DROP** 和 **ACCEPT** 事件。如果没有设置 **-event ACCEPT** 或 **-event DROP**，则网络服务默认为 **ALL**。



## 注意

网络服务将所有日志数据写入每个 Compute 节点上的同一文件中：  
`/var/log/containers/openvswitch/ovn-controller.log`。

## 18.1. 验证是否启用了安全组日志记录

要为网络数据包日志记录准备部署，请确保配置了日志记录服务插件和日志记录扩展。

### 流程

1. 提供可让您使用 RHOSP admin 角色访问 overcloud 的凭据文件。
2. 输入以下命令。

```
$ openstack extension list --max-width 80 | grep logging
```

如果正确配置了日志记录服务插件和扩展，输出包括：

```
| Logging API Extension | logging | Provides a logging API |
```

3. 如果 `openstack extension list` 输出不包括 Logging API 扩展：
  - a. 在环境文件中添加 **日志** 到 **NeutronPluginExtensions** 参数。

### 示例

```
parameter_defaults:
  NeutronPluginExtensions: "qos,port_security,log"
```

- b. 运行 **openstack overcloud deploy** 命令，并包含核心编配模板、环境文件和此环境文件。

## 18.2. 为安全组创建日志对象

使用资源类型 **security\_group** 创建日志对象。

### 先决条件

- 您已创建了安全组
- 您已为安全组创建了安全组规则
- 您已为安全组分配了端口

### 流程

1. 提供可让您使用 RHOSP admin 角色访问 overcloud 的凭据文件。
2. 使用 **openstack network log create** 命令和适当的参数集合来创建日志。

**示例 1：所有端口上安全组 sg1 中的日志 ACCEPT 事件**

```
$ openstack network log create my-log1 \
```



```
--resource-type security_group \  
--resource sg1 \  
-event ACCEPT
```

### 示例 2：所有端口上所有安全组中的日志 ACCEPT 事件

```
openstack network log create my-log3 \  
--resource-type security_group \  
-event ACCEPT
```

3. 验证日志是否已创建：

```
$ openstack network log list
```

## 18.3. 列出和查看安全组的日志对象

您可以列出和查看安全组日志对象。

### 流程

1. 提供可让您使用 RHOSP admin 角色访问 overcloud 的凭据文件。
2. 列出项目中的所有日志对象：

```
$ openstack network log list
```

3. 查看日志对象的详情：

```
$ openstack network log show <log_object_name>
```

将 <log\_object\_name> 替换为日志对象的名称。

## 18.4. 为安全组启用和禁用日志对象

当您创建日志对象时，它会被默认启用。您可以禁用或启用日志对象。

### 流程

1. 提供可让您使用 RHOSP admin 角色访问 overcloud 的凭据文件。
2. 要禁用日志对象，请输入以下命令：

```
$ openstack network log set --disable <log_object_name>
```

将 <log\_object\_name> 替换为日志对象的名称。

3. 要启用日志对象，请输入以下命令：

```
$ openstack network log set --enable <log_object_name>
```

将 <log\_object\_name> 替换为日志对象的名称。

## 18.5. 为安全组重命名日志对象

您可以更改日志对象的名称。

### 流程

1. 提供可让您使用 RHOSP admin 角色访问 overcloud 的凭据文件。
2. 要重命名日志对象，请输入以下命令：

```
$ openstack network log set --name <new_log_object_name> <object>
```

将 <new\_log\_object\_name> 替换为日志对象的新名称。将 <object> 替换为日志对象的旧名称或 ID。

## 18.6. 删除安全组的日志对象

您可以删除日志对象。

### 流程

1. 提供可让您使用 RHOSP admin 角色访问 overcloud 的凭据文件。
2. 要删除一个或多个日志对象，请输入以下命令：

```
$ openstack network log delete <log_object_name> [<log_object_name> ...]
```

将 <log\_object\_name> 替换为要删除的日志对象的名称。要删除多个日志对象，请输入日志对象名称列表，用空格分开。

## 18.7. 访问安全组日志内容

网络服务将 Compute 节点上的所有虚拟机实例的安全组日志聚合在 Compute 节点主机上的一个位置：`/var/log/containers/openvswitch/ovn-controller.log`。

日志文件包含其他日志对象。安全组日志条目包括字符串 `acl_log`。

## 18.8. 安全组日志内容示例

日志内容包括以下数据：

- 数据包流的时间戳。
- 流的状态：**ACCEPT** 或 **DROP**。
- 指明流的来源器。例如，哪个项目或日志资源会生成事件。
- 关联的实例接口(Neutron 端口 ID)的标识符。
- 第 2 层、3 和 4 信息，如 MAC、地址、端口和协议。

### 示例：从 ACCEPT 事件中记录数据

```
2022-11-30T03:29:12.868Z|00111|acl_log(ovn_pinctrl1)|INFO|name="neutron-bc53f8df-2318-4d08-
```

```
8e12-89e92b08deec", verdict=allow, severity=info, direction=from-lport:
udp,vlan_tci=0x0000,dl_src=fa:16:3e:70:c4:45,dl_dst=fa:16:3e:66:8b:18,nw_src=192.168.100.59,nw_ds
=192.168.100.1,nw_tos=0,nw_ecn=0,nw_ttl=64,tp_src=68,tp_dst=67
```

## 18.9. 为安全组日志记录调整速率和突发限制

为了避免通过传输日志记录数据造成 control plane，网络服务会设置每秒记录的最大数据包数量的限制。您可以使用 **NeutronOVNLoggingRateLimit** 参数更改此限制。

当日志记录数据包传输达到速率限制时，网络服务会排队要记录的数据包。您可以使用 **NeutronOVNLoggingBurstLimit** 参数更改排队数据包的最大数量。

默认值为 **NeutronOVNLoggingRateLimit:100** 数据包每秒，**NeutronOVNLoggingBurstLimit:25** 数据包在队列中。这些也是最低的必需值。限制不能以较低值正确操作。

日志记录率和突发限制不会限制数据流量的控制。它们仅限制日志记录数据的传输。

### 流程

1. 以 **stack** 用户身份登录 undercloud 主机。
2. 查找 undercloud 凭证文件：

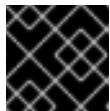
```
$ source ~/stackrc
```

3. 在自定义环境文件中设置参数。例如，**sg-logging.yaml**。

### 示例

```
parameter_defaults:
...
  NeutronOVNLoggingRateLimit=450
  NeutronOVNLoggingBurstLimit=50
```

4. 运行部署命令，并使用 **-r** 选项在部署命令中包含核心 Heat 模板、其他环境文件以及自定义角色数据文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

### 示例

```
$ openstack overcloud deploy --templates <core_heat_templates> \
-e <other_environment_files> \
-e /home/stack/templates/neutron-ovn-dvr-ha.yaml
```

## 第 19 章 常见的管理网络任务

有时，您可能需要在 Red Hat OpenStack Platform Networking 服务(neutron)上执行管理任务，如配置第 2 层 Population 驱动程序，或者通过内部 DNS 指定分配给端口的名称。

### 19.1. 配置 L2 填充驱动程序

L2 Population 驱动程序用于网络服务(neutron) ML2/OVS 环境，以启用广播、多播和单播流量，以便在大型覆盖网络上横向扩展。默认情况下，Open vSwitch GRE 和 VXLAN 将广播复制到每个代理，包括不托管目标网络的那些代理。这种设计需要接受大量网络和处理开销。L2 Population 驱动程序引入的替代设计为 ARP 解析和 MAC 学习流量实施部分网格；它还仅在托管网络的节点之间为特定网络创建隧道。此流量仅通过将流量封装为目标单播发送到必要的代理。

#### 先决条件

- 您必须具有 RHOSP 管理员特权。
- 网络服务必须使用 ML2/OVS 机制驱动程序。

#### 流程

1. 以 **stack** 用户身份登录 undercloud 主机。
2. 查找 undercloud 凭证文件：

```
$ source ~/stackrc
```

3. 创建自定义 YAML 环境文件。

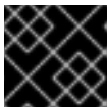
#### 示例

```
$ vi /home/stack/templates/my-environment.yaml
```

4. 您的环境文件必须包含 keywords **parameter\_defaults**。在这些关键字下，添加以下行：

```
parameter_defaults:
  NeutronMechanismDrivers: ['openvswitch', 'l2population']
  NeutronEnableL2Pop: 'True'
  NeutronEnableARPResponder: true
```

5. 运行部署命令，包括核心 heat 模板、环境文件和新的自定义环境文件。



#### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

#### 示例

```
$ openstack overcloud deploy --templates \
-e <your_environment_files> \
-e /home/stack/templates/my-environment.yaml
```

## 验证

1. 获取 OVS 代理的 ID。

```
$ openstack network agent list -c ID -c Binary
```

## 输出示例

```
+-----+-----+
| ID                | Binary                |
+-----+-----+
| 003a8750-a6f9-468b-9321-a6c03c77aec7 | neutron-openvswitch-agent |
| 02bbbb8c-4b6b-4ce7-8335-d1132df31437 | neutron-l3-agent          |
| 0950e233-60b2-48de-94f6-483fd0af16ea | neutron-openvswitch-agent |
| 115c2b73-47f5-4262-bc66-8538d175029f | neutron-openvswitch-agent |
| 2a9b2a15-e96d-468c-8dc9-18d7c2d3f4bb | neutron-metadata-agent   |
| 3e29d033-c80b-4253-aaa4-22520599d62e | neutron-dhcp-agent       |
| 3ede0b64-213d-4a0d-9ab3-04b5dfd16baa | neutron-dhcp-agent       |
| 462199be-0d0f-4bba-94da-603f1c9e0ec4 | neutron-sriov-nic-agent   |
| 54f7c535-78cc-464c-bdaa-6044608a08d7 | neutron-l3-agent         |
| 6657d8cf-566f-47f4-856c-75600bf04828 | neutron-metadata-agent   |
| 733c66f1-a032-4948-ba18-7d1188a58483 | neutron-l3-agent         |
| 7e0a0ce3-7ebb-4bb3-9b89-8cccf8cb716e | neutron-openvswitch-agent |
| dfc36468-3a21-4a2d-84c3-2bc40f224235 | neutron-metadata-agent   |
| eb7d7c10-69a2-421e-bd9e-aec3edfe1b7c | neutron-openvswitch-agent |
| ef5219b4-ee49-4635-ad04-048291209373 | neutron-sriov-nic-agent   |
| f36c7af0-e20c-400b-8a37-4ffc5d4da7bd | neutron-dhcp-agent       |
+-----+-----+
```

2. 使用其中一个 OVS 代理中的 ID，确认 OVS 代理上设置了 L2 Population 驱动程序。

## 示例

本例验证 **neutron-openvswitch-agent** 上 ID 为 **003a8750-a6f9-468b-9321-a6c03c77aec7** 的 L2 Population 驱动程序的配置：

```
$ openstack network agent show 003a8750-a6f9-468b-9321-a6c03c77aec7 -c configuration
-f json | grep l2_population
```

## 输出示例

```
"l2_population": true,
```

3. 确保为 OVS 代理启用了 ARP 响应器功能。

## 示例

```
$ openstack network agent show 003a8750-a6f9-468b-9321-a6c03c77aec7 -c configuration
-f json | grep arp_responder_enabled
```

## 输出示例

```
"arp_responder_enabled": true,
```

## 其他资源

- [OVN 支持的 DHCP 选项](#)
- [高级 Overcloud 自定义指南中的环境文件](https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/16.2/html/advanced_overcloud_customization/assembly_underst_heat-templates#con_environment-files_understanding-heat-templates)
- [高级 Overcloud 自定义指南中的创建 overcloud 中包括环境文件](#)

## 19.2. 调整 KEEPALIVED 以避免 VRRP 数据包丢失

如果单个主机上的高可用性(HA)路由器数量很高，当 HA 路由器失败时，虚拟路由器冗余协议(VRRP)消息可能会溢出 IRQ 队列。这个溢出停止 Open vSwitch (OVS)响应和转发这些 VRRP 消息。

为了避免 VRRP 数据包过载，您必须使用 Controller 角色的 **ExtraConfig** 部分中的 **ha\_vrrp\_advert\_int** 参数来增加 VRRP 广告间隔。

### 流程

1. 以 stack 用户身份登录 undercloud，再提供 **stackrc** 文件以启用 director 命令行工具。

#### 示例

```
$ source ~/stackrc
```

2. 创建自定义 YAML 环境文件。

#### 示例

```
$ vi /home/stack/templates/my-neutron-environment.yaml
```

#### 提示

Red Hat OpenStack Platform 编排服务(heat)使用一组名为 *template* 的计划来安装和配置您的环境。您可以使用自定义环境文件来自定义 overcloud 的各个方面，这是为 heat 模板 *提供自定义* 的特殊模板。

3. 在 YAML 环境文件中，使用 **ha\_vrrp\_advert\_int** 参数和特定于您的站点的值来增加 VRRP 公告间隔。（默认值为 **2** 秒。）

您还可以为 gratuitous ARP 信息设置值：

#### **ha\_vrrp\_garp\_master\_repeat**

过渡到 master 状态后一次发送的 gratuitous ARP 消息数量。（默认为 5 个消息。）

#### **ha\_vrrp\_garp\_master\_delay**

在 master 状态接收较低优先级 advert 后，第二个 gratuitous ARP 消息的延迟。（默认值为 5 秒。）

#### 示例

```
parameter_defaults:
  ControllerExtraConfig:
```

```
neutron::agents::l3::ha_vrrp_advert_int: 7
neutron::config::l3_agent_config:
  DEFAULT/ha_vrrp_garp_master_repeat:
    value: 5
  DEFAULT/ha_vrrp_garp_master_delay:
    value: 5
```

4. 运行 **openstack overcloud deploy** 命令，并包含核心 heat 模板、环境文件和新的自定义环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

### 示例

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-neutron-
environment.yaml
```

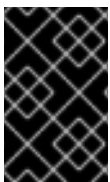
### 其他资源

- *RFC 4541* 中的 [2.1.2 Data Forwarding Rules, Subsection 2](#)
- *高级 Overcloud 自定义指南* 中的 [环境文件](#)。
- *高级 Overcloud 自定义指南* 中的 [创建 overcloud 中包括环境文件](#)

## 19.3. 指定 DNS 分配给端口的名称

当您为端口扩展(**dns\_domain\_ports**)启用 Red Hat OpenStack Platform (RHOSP)网络服务(neutron) DNS 域时，您可以指定由内部 DNS 分配给端口的名称。

您可以通过在 YAML 格式的环境文件中声明 RHOSP Orchestration (heat) NeutronPlugin **Extensions** 参数，为端口扩展启用 DNS 域。使用对应的参数 **NeutronDnsDomain**，您可以指定您的域名，这将覆盖默认值 **openstacklocal**。重新部署 overcloud 后，您可以使用 OpenStack 客户端端口命令、[端口设置](#) 或 [端口创建](#)，并使用 **--dns-name** 来分配端口名称。



### 重要

您必须为 DNS 启用端口扩展(**dns\_domain\_ports**)的 DNS 域，以便内部解析 RHOSP 环境中端口的名称。使用 **NeutronDnsDomain** 默认值 **openstacklocal** 意味着网络服务不会内部解析 DNS 的端口名称。

另外，当启用端口扩展的 DNS 域时，Compute 服务会在虚拟机实例引导过程中使用实例的 **hostname** 属性自动填充 **dns\_name** 属性。在引导过程结束时，dnsmasq 通过实例主机名识别分配的端口。

### 流程

1. 以 stack 用户身份登录 undercloud，再提供 **stackrc** 文件以启用 director 命令行工具。

## 示例

```
$ source ~/stackrc
```

2. 创建自定义 YAML 环境文件(**my-neutron-environment.yaml**)。



### 注意

括号内的值是此过程中的示例命令中使用的示例值。使用适合您的站点的值替换这些示例值。

## 示例

```
$ vi /home/stack/templates/my-neutron-environment.yaml
```

## 提示

undercloud 包括一组组成您的 overcloud 创建计划的编排服务模板。您可以使用环境文件自定义 overcloud 的各个方面，这些文件是 YAML 格式的文件，可覆盖核心编排服务模板集中的参数和资源。您可以根据需要纳入多个环境文件。

3. 在环境文件中，添加一个 **parameter\_defaults** 部分。在本节中，为端口扩展名 **dns\_domain\_ports** 添加 DNS 域。

## 示例

```
parameter_defaults:
  NeutronPluginExtensions: "qos,port_security,dns_domain_ports"
```



### 注意

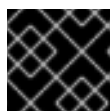
如果设置了 **dns\_domain\_ports**，请确保部署也没有使用 **dns\_domain**，DNS 集成扩展。这些扩展不兼容，并且无法同时定义这两个扩展。

4. 另外，在 **parameter\_defaults** 部分中，使用 **NeutronDnsDomain** 参数添加您的域名 (**example.com**)。

## 示例

```
parameter_defaults:
  NeutronPluginExtensions: "qos,port_security,dns_domain_ports"
  NeutronDnsDomain: "example.com"
```

5. 运行 **openstack overcloud deploy** 命令，并包含核心编配模板、环境文件和新环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

## 示例



```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-neutron-
environment.yaml
```

## 验证

1. 登录到 overcloud，并在网络 (**public**) 上创建一个新端口 (**new\_port**)。为端口分配 DNS 名称 (**my\_port**)。

### 示例

```
$ source ~/overcloudrc
$ openstack port create --network public --dns-name my_port new_port
```

2. 显示端口的详细信息(**new\_port**)。

### 示例

```
$ openstack port show -c dns_assignment -c dns_domain -c dns_name -c name new_port
```

### 输出

```
+-----+-----+
| Field          | Value                                     |
+-----+-----+
| dns_assignment | fqdn='my_port.example.com',             |
|                 | hostname='my_port',                     |
|                 | ip_address='10.65.176.113'              |
| dns_domain     | example.com                             |
| dns_name       | my_port                                  |
| name           | new_port                                  |
+-----+-----+
```

在 **dns\_assignment** 下，端口的完全限定域名(**fqdn**)值包含 DNS 名称(**my\_port**)的连接以及您之前使用 **NeutronDnsDomain** 设置的域名(**example.com**)。

3. 使用您刚才创建的端口(**new\_port**)创建新虚拟机实例(**my\_vm**)。

### 示例

```
$ openstack server create --image rhel --flavor m1.small --port new_port my_vm
```

4. 显示端口的详细信息(**new\_port**)。

### 示例

```
$ openstack port show -c dns_assignment -c dns_domain -c dns_name -c name new_port
```

### 输出

```
+-----+-----+
```

Field	Value
dns_assignment	fqdn='my_vm.example.com', hostname='my_vm', ip_address='10.65.176.113'
dns_domain	example.com
dns_name	my_vm
name	new_port

请注意，计算服务将 **dns\_name** 属性从其原始值(**my\_port**)更改为与端口关联的实例的名称(**my\_vm**)。

## 其他资源

- [高级 Overcloud 自定义指南中的环境文件](#)。
- [高级 Overcloud 自定义指南中的创建 overcloud 中包括环境文件](#)
- [命令行界面参考中的端口](#)
- [Command Line Interface Reference 中的 server create](#)

## 19.4. 为端口分配 DHCP 属性

您可以使用 Red Hat Openstack Platform (RHOSP)网络服务(neutron)扩展来添加网络功能。您可以使用额外的 DHCP 选项扩展(**extra\_dhcp\_opt**)来配置带有 DHCP 属性的 DHCP 客户端的端口。例如，您可以在 DHCP 客户端端口中添加 PXE 引导选项，如 **tftp-server**、**server-ip-address** 或 **bootfile-name**。

**extra\_dhcp\_opt** 属性的值是 DHCP 选项对象的数组，每个对象都包含一个 **opt\_name** 和 **opt\_value**。IPv4 是默认版本，但您可以通过包含第三个选项 **ip-version=6** 来将其更改为 IPv6。

当虚拟机实例启动时，RHOSP 网络服务使用 DHCP 协议为实例提供端口信息。如果您在已连接到正在运行的实例的端口中添加 DHCP 信息，则实例仅在实例重启时使用新的 DHCP 端口信息。

一些常见的 DHCP 端口属性有：**bootfile-name**、**dns-server**、**domain-name**、**mtu**、**server-ip-address** 和 **tftp-server**。有关 **opt\_name** 的可接受值的完整集合，请参阅 DHCP 规格。

### 先决条件

- 您必须具有 RHOSP 管理员特权。

### 流程

1. 以 **stack** 用户身份登录 undercloud 主机。
2. 查找 undercloud 凭证文件：

```
$ source ~/stackrc
```

3. 创建自定义 YAML 环境文件。

### 示例

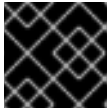
```
$ vi /home/stack/templates/my-environment.yaml
```

- 您的环境文件必须包含 keywords **parameter\_defaults**。在这些关键字下，添加额外的 DHCP 选项扩展 **extra\_dhcp\_opt**。

### 示例

```
parameter_defaults:
  NeutronPluginExtensions: "qos,port_security,extra_dhcp_opt"
```

- 运行部署命令，包括核心 heat 模板、环境文件和新的自定义环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

### 示例

```
$ openstack overcloud deploy --templates \
-e <your_environment_files> \
-e /home/stack/templates/my-environment.yaml
```

## 验证

- 提供您的凭据文件。

### 示例

```
$ source ~/overcloudrc
```

- 在网络(**public**)上创建一个新端口(**new\_port**)。从 DHCP 规格中为新端口分配有效属性。

### 示例

```
$ openstack port create --extra-dhcp-option \
name=domain-name,value=test.domain --extra-dhcp-option \
name=ntp-server,value=192.0.2.123 --network public new_port
```

- 显示端口的详细信息(**new\_port**)。

### 示例

```
$ openstack port show new_port -c extra_dhcp_opts
```

### 输出示例

```
+-----+-----+
| Field      | Value                                |
+-----+-----+
```

```
| extra_dhcp_opts | ip_version='4', opt_name='domain-name', opt_value='test.domain' |
|                 | ip_version='4', opt_name='ntp-server', opt_value='192.0.2.123'   |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

## 其他资源

- [OVN 支持的 DHCP 选项](#)
- [动态主机配置协议\(DHCP\)和 Bootstrap 协议\(BOOTP\)参数](#)
- [高级 Openstack 自定义指南中的环境文件](https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/16.2/html/advanced_overcloud_customization/assembly_underst_heat-templates#con_environment-files_understanding-heat-templates)
- [高级 Openstack 自定义指南中的创建 overcloud 中包括环境文件](#)
- [命令行界面参考中的端口创建](#)
- [命令行界面参考中的端口显示](#)

## 19.5. 载入内核模块

Red Hat OpenStack Platform (RHOSP)中的一些功能需要载入某些内核模块。例如，OVS 防火墙驱动程序要求您加载 `nf_conntrack_proto_gre` 内核模块，以支持两个虚拟机实例之间的 GRE 隧道。

通过使用特殊的编排服务(heat)参数 `ExtraKernelModules`，您可以确保 heat 存储有关 GRE 隧道等功能所需的内核模块的配置信息。之后，在正常的模块管理过程中，这些所需的内核模块会被加载。

### 流程

1. 在 undercloud 主机上，以 stack 用户身份登录，创建自定义 YAML 环境文件。

#### 示例

```
$ vi /home/stack/templates/my-modules-environment.yaml
```

#### 提示

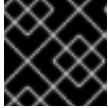
Heat 使用一组名为 `template` 的计划来安装和配置您的环境。您可以使用自定义环境文件来自定义 overcloud 的各个方面，这是为 heat 模板 *提供自定义* 的特殊模板。

2. 在 `parameter_defaults` 下的 YAML 环境文件中，将 `ExtraKernelModules` 设置为您要载入的模块的名称。

#### 示例

```
ComputeParameters:
  ExtraKernelModules:
    nf_conntrack_proto_gre: {}
ControllerParameters:
  ExtraKernelModules:
    nf_conntrack_proto_gre: {}
```

- 运行 **openstack overcloud deploy** 命令，并包含核心 heat 模板、环境文件和新的自定义环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

### 示例

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-modules-
environment.yaml
```

### 验证

- 如果 heat 正确载入该模块，您应该在 Compute 节点上运行 **lsmod** 命令时看到输出：

### 示例

```
sudo lsmod | grep nf_conntrack_proto_gre
```

### 其他资源

- [高级 Overcloud 自定义指南中的环境文件。](#)
- [高级 Overcloud 自定义指南中的创建 overcloud 中包括环境文件](#)

## 19.6. 配置共享安全组

当您希望一个或多个 Red Hat OpenStack Platform (RHOSP)项目可以共享数据时，您可以使用 RHOSP Networking 服务(neutron) RBAC 策略功能共享安全组。您可以使用 OpenStack 客户端创建安全组和网络安全服务基于角色的访问控制(RBAC)策略。

您可以在实例创建过程中直接将安全组应用到实例，或应用到正在运行的实例上的端口。



### 注意

您不能在实例创建过程中将基于角色的访问控制(RBAC)共享安全组直接应用到实例。要将 RBAC 共享安全组应用到实例，必须首先创建端口，将共享安全组应用到该端口，然后将该端口分配给实例。请参阅 [将安全组添加到端口](#)。

### 先决条件

- 您至少有两个要共享的 RHOSP 项目。
- 在其中一个项目中 (*当前项目*)，您创建了要与其他项目 (*目标项目*) 共享的安全组。在本例中，创建了 **ping\_ssh** 安全组：

### 示例

```
$ openstack security group create ping_ssh
```

## 流程

1. 为包含安全组的当前项目登录 overcloud。
2. 获取目标项目的名称或 ID。

```
$ openstack project list
```

3. 获取您要在 RHOSP 项目之间共享的安全组的名称或 ID。

```
$ openstack security group list
```

4. 使用上一步中的标识符，使用 **openstack network rbac create** 命令创建 RBAC 策略。在本例中，目标项目的 ID 是 **32016615de5d43bb88de99e7f2e26a1e**。安全组的 ID 是 **5ba835b7-22b0-4be6-bdbe-e0722d1b5f24** :

## 示例

```
$ openstack network rbac create --target-project \
32016615de5d43bb88de99e7f2e26a1e --action access_as_shared \
--type security_group 5ba835b7-22b0-4be6-bdbe-e0722d1b5f24
```

### **--target-project**

指定需要访问安全组的项目。

## 提示

您可以使用 **--target-all-projects** 参数而不是 **--target-project <target-project>** 以在 *所有* 项目中共享数据。默认情况下，只有 admin 用户具有此特权。

### **--action access\_as\_shared**

指定允许的项目做什么。

### **--type**

表示目标对象是一个安全组。

### **5ba835b7-22b0-4be6-bdbe-e0722d1b5f24**

是被授予访问权限的特定安全组的 ID。

目标项目在运行 OpenStack Client **安全组命令**时可以访问安全组，同时能够绑定到其端口。没有其他用户（除管理员和拥有者之外）无法访问安全组。

## 提示

要删除目标项目的访问，请使用 **openstack network rbac delete** 命令删除允许它的 RBAC 策略。

## 其他资源

- [创建和管理实例指南中的创建安全组](#)
- [命令行界面参考中的安全组创建](#)
- [命令行界面参考中的 network rbac create](#)

## 第 20 章 配置第 3 层高可用性(HA)

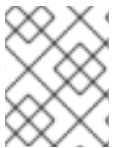
### 20.1. 没有高可用性(HA)的 RHOSP 网络服务

没有高可用性(HA)功能的 Red Hat OpenStack Platform (RHOSP)网络服务部署会受到物理节点故障的影响。

在典型的部署中，项目会创建虚拟路由器，这些路由器计划在物理网络服务层 3 (L3)代理节点上运行。当您丢失 L3 代理节点且依赖的虚拟机随后丢失到外部网络的连接时，这就会出现这个问题。任何浮动 IP 地址也不可用。此外，路由器主机的任何网络间会丢失连接。

### 20.2. 第 3 层高可用性概述(HA)

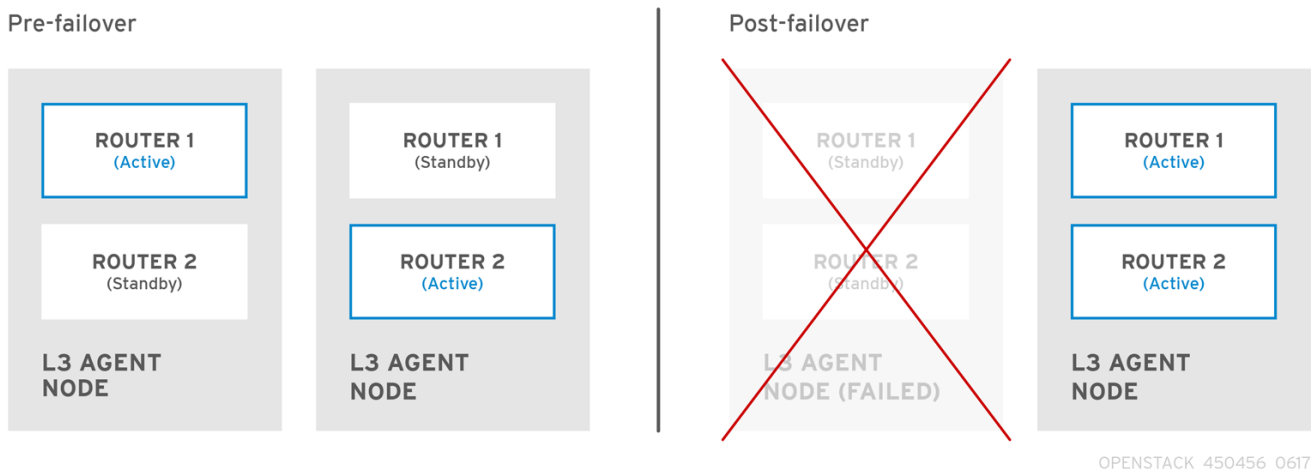
这个主动/被动高可用性(HA)配置使用行业标准 VRRP（如 RFC 3768 中定义的）来保护项目路由器和浮动 IP 地址。虚拟路由器随机调度到多个 Red Hat OpenStack Platform (RHOSP)网络服务节点上，一个指定为 *活跃* 路由器，剩余则以 *备用* 角色的形式调度。



#### 注意

要部署第 3 层(L3) HA，您必须在冗余网络服务节点上维护类似的配置，包括浮动 IP 范围和对外部网络的访问。

在下图中，活跃的 **Router1** 和 **Router2** 路由器在一个独立的物理 L3 网络服务代理节点上运行。L3 HA 在对应的节点上调度了备份虚拟路由器，在物理节点失败时准备好恢复服务。当 L3 代理节点失败时，L3 HA 会将受影响的虚拟路由器和浮动 IP 地址重新调度到工作节点：



在故障转移事件中，通过浮动 IP 实例 TCP 会话保持不受影响，并在不中断的情况下迁移到新的 L3 节点。只有 SNAT 流量会受到故障转移事件的影响。

在主动/主动 HA 模式中时，L3 代理会被进一步保护。

#### 其他资源

- [虚拟路由器冗余协议\(VRRP\)](#)

### 20.3. 第 3 层高可用性(HA)故障转移条件

Red Hat OpenStack Platform (RHOSP)网络服务的第 3 层(L3)高可用性(HA)在以下事件中自动重新调度受保护的资源：

- 因为硬件故障，网络服务 L3 代理节点关闭或丢失电源。
- L3 代理节点与物理网络隔离并丢失连接。



### 注意

手动停止 L3 代理服务不会导致故障转移事件。

## 20.4. 第 3 层高可用性(HA)的项目注意事项

Red Hat OpenStack Platform (RHOSP)网络服务层 3 (L3)高可用性(HA)配置发生在后端，对项目不可见。项目可以持续创建和管理其虚拟路由器，但设计 L3 HA 实现时了解有一些限制：

- L3 HA 支持每个项目最多 255 个虚拟路由器。
- 内部 VRRP 消息会传输在单独的内部网络中，并自动为每个项目创建。这个过程对用户透明发生。
- 在 ML2/OVS 上实施高可用性(HA)路由器时，每个 L3 代理为每个路由器生成 **haproxy** 和 **neutron-keepalived-state-change-monitor** 进程。每个进程占用大约 20MB 内存。默认情况下，每个 HA 路由器驻留在三个 L3 代理上，并消耗每个节点上的资源。因此，在调整 RHOSP 网络时，请确保分配了足够的内存来支持您计划实现的 HA 路由器数量。

## 20.5. RHOSP 网络服务的高可用性(HA)更改

Red Hat OpenStack Platform (RHOSP)网络服务(neutron) API 已更新，以便管理员在创建路由器时设置 **--ha=True/False** 标志，这会覆盖 `/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf` 中的 `I3_ha` 的默认配置。

- 对 **neutron-server** 的高可用性(HA)更改：
  - 第 3 层(L3) HA 会随机分配活动角色，无论网络服务使用的调度程序是什么（无论是随机还是至少路由器）。
  - 数据库架构已被修改，以处理虚拟 IP 地址(VIP)到虚拟路由器的分配。
  - 创建传输网络来直接 L3 HA 流量。
- HA 对**网络服务 L3 代理**的更改：
  - 添加了一个新的 keepalived 管理器，提供负载均衡和 HA 功能。
  - IP 地址转换为 VIP。

## 20.6. 在 RHOSP 网络服务节点上启用第 3 层高可用性(HA)

在安装过程中，当您至少有两个 RHOSP Controller 且不使用分布式虚拟路由(DVR)时，Red Hat OpenStack Platform (RHOSP) director 会默认为虚拟路由器启用高可用性(HA)。使用 RHOSP 编排服务(heat)参数 **max\_l3\_agents\_per\_router**，您可以设置在其上调度 HA 路由器的最大 RHOSP 网络服务层 3 (L3)代理数。

### 先决条件



- 您的 RHOSP 部署不使用 DVR。
- 您至少部署两个 RHOSP Controller。

## 流程

1. 以 stack 用户身份登录 undercloud，再提供 **stackrc** 文件以启用 director 命令行工具。

### 示例

```
$ source ~/stackrc
```

2. 创建自定义 YAML 环境文件。

### 示例

```
$ vi /home/stack/templates/my-neutron-environment.yaml
```

### 提示

编排服务 (heat) 使用一组名为 *template* (模板) 的计划来安装和配置您的环境。您可以使用自定义环境文件来自定义 overcloud 的各个方面，这是为 heat 模板 *提供自定义* 的特殊模板。

3. 在 YAML 环境文件中将 **NeutronL3HA** 参数设置为 **true**。这样可确保即使 director 默认未设置 HA，也会启用 HA。

```
parameter_defaults:
  NeutronL3HA: 'true'
```

4. 设置在其上调度 HA 路由器的 L3 代理的最大数量。  
将 **max\_l3\_agents\_per\_router** 参数设置为部署中网络节点的最小和最大值。（零值表示路由器调度到每个代理上。）

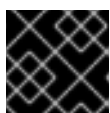
### 示例

```
parameter_defaults:
  NeutronL3HA: 'true'
  ControllerExtraConfig:
    neutron::server::max_l3_agents_per_router: 2
```

在本例中，如果您部署四个网络服务节点，则只有两个 L3 代理会保护每个 HA 虚拟路由器：一个活跃，一个备用服务。

如果将 **max\_l3\_agents\_per\_router** 的值设置为大于可用网络节点的数量，您可以通过添加新的 L3 代理来扩展备用路由器的数量。对于您部署的每个新的 L3 代理节点，网络服务会调度虚拟路由器的额外备用版本，直到达到 **max\_l3\_agents\_per\_router** 限制。

5. 运行 **openstack overcloud deploy** 命令，并包含核心 heat 模板、环境文件和新的自定义环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

## 示例

```
$ openstack overcloud deploy --templates \
-e [your-environment-files] \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/my-neutron-
environment.yaml
```



## 注意

当 **NeutronL3HA** 设为 **true** 时，创建的所有虚拟路由器默认为 HA 路由器。在创建路由器时，您可以通过在 **openstack router create** 命令中包含 **--no-ha** 选项来覆盖 HA 选项：

```
# openstack router create --no-ha
```

## 其他资源

- [高级 Overcloud 自定义指南中的环境文件](#)。
- [高级 Overcloud 自定义指南中的创建 overcloud 中包括环境文件](#)

## 20.7. 查看高可用性(HA) RHOSP 网络服务节点配置

### 流程

- 在虚拟路由器命名空间中运行 **ip address** 命令，以返回高可用性(HA)设备，以 *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
```

启用第 3 层 HA 后，虚拟路由器和浮动 IP 地址会对单个节点故障进行保护。

## 第 21 章 替换 NETWORKER 节点

在某些情况下，在高可用性集群中有一个 Networker 配置集的 Red Hat OpenStack Platform (RHOSP) 节点可能会失败。（如需更多信息，请参阅 *Director 安装和使用指南* 中的[将节点标记为配置集](#)。）在这些情况下，您必须从集群中删除节点，并将其替换为运行网络服务(neutron)代理的新的 Networker 节点。

本节中的主题是：

- [第 21.1 节 “准备替换网络节点”](#)
- [第 21.2 节 “替换网络器节点”](#)
- [第 21.3 节 “重新调度节点并清理网络服务”](#)

### 21.1. 准备替换网络节点

在 Red Hat OpenStack Platform (RHOSP) overcloud 上替换 Networker 节点需要执行几个准备步骤。完成所有必要的准备步骤可帮助您避免在网络器节点替换过程中出现复杂情况。

#### 先决条件

- 您的 RHOSP 部署具有高可用性，有三个或更多网络器节点。

#### 流程

1. 以 stack 用户身份登录 undercloud。
2. 查找 undercloud 凭证文件：

```
$ source ~/stackrc
```

3. 检查 undercloud 上 overcloud 栈的当前状态：

```
$ openstack stack list --nested
```

overcloud 堆栈及其后续子堆栈的状态应为 **CREATE\_COMPLETE** 或 **UPDATE\_COMPLETE**。

4. 运行 Relax-and-Recover 工具，确保您有一个 undercloud 节点的最新备份镜像。  
如需更多信息，请参阅 [备份和恢复 undercloud](#) 和 [control plane 节点](#) 指南。
5. 以 root 用户身份登录 Controller 节点。
6. 在容器中打开交互式 bash shell 并检查 Galera 集群的状态：

```
# pcs status
```

确保 Controller 节点处于 **Master** 模式。

#### 输出示例

```
* Container bundle set: galera-bundle [cluster.common.tag/rhosp16-openstack-
mariadb:pcmklatest]:
  * galera-bundle-0 (ocf::heartbeat:galera):      Master controller-0
```

```
* galera-bundle-1 (ocf::heartbeat:galera): Master controller-1
* galera-bundle-2 (ocf::heartbeat:galera): Master controller-2
```

7. 登录到 RHOSP director 节点并检查 nova-compute 服务：

```
$ sudo systemctl status tripleo_nova_compute
$ openstack baremetal node list
```

输出应该显示所有非维护模式节点为 up。

8. 确保所有 undercloud 服务都在运行：

```
$ sudo systemctl -t service
```

## 21.2. 替换网络器节点

在某些情况下，在高可用性集群中有一个 Networker 配置集的 Red Hat OpenStack Platform (RHOSP) 节点可能会失败。替换 Networker 节点需要运行 **openstack overcloud deploy** 命令，以使用新节点更新 overcloud。

### 先决条件

- 您的 RHOSP 部署具有高可用性，有三个或更多网络器节点。
- 要添加的节点必须能够通过网络连接到集群中的其他节点。
- 您已执行 [第 21.1 节 “准备替换网络节点”](#) 中介绍的步骤

### 流程

1. 以 **stack** 用户身份登录 undercloud。
2. 查找 undercloud 凭证文件：

### 示例

```
$ source ~/stackrc
```

3. 找出要删除的节点的索引：

```
$ openstack baremetal node list -c UUID -c Name -c "Instance UUID"
```

### 输出示例

```
+-----+-----+-----+
| UUID           | Name | Instance UUID          |
+-----+-----+-----+
| 36404147-7c8a-41e6-8c72-6af1e339da2a | None | 7bee57cf-4a58-4eaf-b851-f3203f6e5e05 |
| 91eb9ac5-7d52-453c-a017-0f2fb289c3cd | None | None                               |
| 75b25e9a-948d-424a-9b3b-0f2fb289c3cd | None | None                               |
| 038727da-6a5c-425f-bd45-16aa2bc4ba91 | None | 763bfec2-9354-466a-ae65-1fdf45d35c61 |
```

```
| dc2292e6-4056-46e0-8848-165d06fcc948 | None | 2017b481-706f-44e1-852a-
57fb03ecef11 |
| c7eadcea-e377-4392-9fc3-716f1bd57527 | None | 5f73c7d7-4826-49a5-b6be-
0a95c6bdd2f8 |
| da3a8d19-8a59-4e9d-923a-29254d688f6d | None | cfefaf60-8311-4bc3-9416-46852e2cb83f
|
| 807cb6ce-6b94-4cd1-9969-d390650854c7 | None | c07c13e6-a845-4791-9628-
c8514585fe27 |
| 0c245daa-7817-4ae9-a883-fed2e9c68d6c | None | 844c9a88-713a-4ff1-8737-
30858d724593 |
| e6499ef7-3db2-4ab4-bfa7-feb44c6591c6 | None | aef7c27a-f0b4-4814-b0ff-d3f792321212 |
| 7545385c-bc49-4eb9-b13c-201368ce1c62 | None | c2e40164-c659-4849-a28f-
a7b270ed2970 |
+-----+-----+-----+-----+-----+-----+
```

4. 使用 **baremetal node maintenance set** 命令将节点设置为维护模式。

#### 示例

```
$ openstack baremetal node maintenance set e6499ef7-3db2-4ab4-bfa7-ef59539bf972
```

5. 创建一个 JSON 文件，将新节点添加到包含 RHOSP director 的节点池中。

#### 示例

```
{
  "nodes":[
    {
      "mac":[
        "dd:dd:dd:dd:dd:dd"
      ],
      "cpu":"4",
      "memory":"6144",
      "disk":"40",
      "arch":"x86_64",
      "pm_type":"ipmi",
      "pm_user":"admin",
      "pm_password":"p@55w0rd!",
      "pm_addr":"192.168.24.207"
    }
  ]
}
```

有关更多信息，请参阅 *Director 安装和使用* 指南中的 [向 overcloud 添加节点](#)。

6. 运行 **openstack overcloud node import** 命令以注册新节点。

#### 示例

```
$ openstack overcloud node import newnode.json
```

7. 注册新节点后，使用以下命令启动内省过程：

```
$ openstack baremetal node manage <node>
$ openstack overcloud node introspect <node> --provide
```

- 使用 **openstack baremetal node set** 命令，使用 Networker 配置集标记新节点。

### 示例

```
$ openstack baremetal node set --property \
  capabilities='profile:networker,boot_option:local' \
  91eb9ac5-7d52-453c-a017-c0e3d823efd0
```

- 创建一个 **~/templates/remove-networker.yaml** 环境文件，该文件定义您要删除的节点的索引：

### 示例

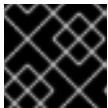
```
parameters:
  NetworkerRemovalPolicies:
    [{'resource_list': ['1']}]
```

- 创建 **~/templates/node-count-networker.yaml** 环境文件，并在文件中设置 Networker 节点总数。

### 示例

```
parameter_defaults:
  OvercloudNetworkerFlavor: networker
  NetworkerCount: 3
```

- 运行 **openstack overcloud deploy** 命令，并包含您修改的核心 heat 模板、环境文件和环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

```
$ openstack overcloud deploy --templates \
  -e <your_environment_files> \
  -e /home/stack/templates/node-count-networker.yaml \
  -e /home/stack/templates/remove-networker.yaml
```

RHOSP director 删除旧的 Networker 节点，创建一个新节点并更新 overcloud 堆栈。

## 验证

- 检查 overcloud 堆栈的状态：

```
$ openstack stack list --nested
```

- 验证新的 Networker 节点是否已列出，并且删除旧节点是否已移除。

```
$ openstack server list -c ID -c Name -c Status
```

## 输出示例

```
+-----+-----+-----+
| ID                | Name                | Status |
+-----+-----+-----+
| 861408be-4027-4f53-87a6-cd3cf206ba7a | overcloud-compute-0 | ACTIVE |
| 0966e9ae-f553-447a-9929-c4232432f718 | overcloud-compute-1 | ACTIVE |
| 9c08fa65-b38c-4b2e-bd47-33870bff06c7 | overcloud-compute-2 | ACTIVE |
| a7f0f5e1-e7ce-4513-ad2b-81146bc8c5af | overcloud-controller-0 | ACTIVE |
| cfefaf60-8311-4bc3-9416-6a824a40a9ae | overcloud-controller-1 | ACTIVE |
| 97a055d4-aefd-481c-82b7-4a5f384036d2 | overcloud-controller-2 | ACTIVE |
| 844c9a88-713a-4ff1-8737-6410bf551d4f | overcloud-networker-0 | ACTIVE |
| c2e40164-c659-4849-a28f-507eb7edb79f | overcloud-networker-2 | ACTIVE |
| 425a0828-b42f-43b0-940c-7fb02522753a | overcloud-networker-3 | ACTIVE |
+-----+-----+-----+
```

## 其他资源

- [Director 安装和使用指南中的将节点添加到 overcloud](#)
- [Director 安装和使用指南中的为 overcloud 注册节点。](#)
- [命令行接口参考中的裸机节点管理](#)
- [命令行界面参考中的 overcloud 节点内省](#)
- [高级 Overcloud 自定义指南中的环境文件。](#)
- [高级 Overcloud 自定义指南中的创建 overcloud 中包括环境文件](#)

## 21.3. 重新调度节点并清理网络服务

作为替换 Red Hat OpenStack Platform (RHOSP) Networker 节点的一部分，从数据库中删除移除节点上的所有网络服务代理。这样做可确保网络服务不会将代理识别为服务不足("dead")。对于 ML2/OVS 用户，从删除的节点中删除代理可让 DHCP 资源自动重新调度到其他 Networker 节点。

### 先决条件

- 您的 RHOSP 部署具有高可用性，有三个或更多网络器节点。

### 流程

1. 以 stack 用户身份登录 undercloud。
2. 查找 overcloud 凭证文件：

### 示例

```
$ source ~/overcloudrc
```

3. 验证 RHOSP 网络服务进程是否存在，并且标记为 **overcloud-networker-1** 的 out-of-service (**xxx**)。

```
$ openstack network agent list -c ID -c Binary -c Host -c Alive | grep overcloud-networker-1
```

## ML2/OVN 的输出示例

```
+-----+-----+-----+
| ID           | Host           | Alive | Binary           |
+-----+-----+-----+
| 26316f47-4a30-4baf-ba00-d33c9a9e0844 | overcloud-networker-1 | xxx | ovn-controller |
+-----+-----+-----+
```

## ML2/OVS 的输出示例

```
+-----+-----+-----+
| ID           | Host           | Alive | Binary           |
+-----+-----+-----+
| 8377-66d75323e466c-b838-1149e10441ee | overcloud-networker-1 | xxx | neutron-
metadata-agent |
| b55d-797668c336707-a2cf-cba875eeda21 | overcloud-networker-1 | xxx | neutron-l3-agent
|
| 9dcb-00a9e32ecde42-9458-01cfa9742862 | overcloud-networker-1 | xxx | neutron-ovs-
agent |
| be83-e4d9329846540-9ae6-1540947b2ffd | overcloud-networker-1 | xxx | neutron-dhcp-
agent |
+-----+-----+-----+
```

- 捕获为 **overcloud-networker-1** 注册的代理的 UUID。

```
$ AGENT_UUIDS=$(openstack network agent list -c ID -c Host -c Alive -c Binary -f value |
grep overcloud-networker-1 | cut -d\  -f1)
```

- 从数据库中删除任何剩余的 **overcloud-networker-1** 代理。

```
$ for agent in $AGENT_UUIDS; do neutron agent-delete $agent ; done
```

## 输出示例

```
Deleted agent(s): 26316f47-4a30-4baf-ba00-d33c9a9e0844
```

## 其他资源

- 命令行界面参考中的 [network agent list](#)



## 第 22 章 使用可用区使网络资源高度可用

从 16.2 版本开始，Red Hat OpenStack Platform (RHOSP) 支持 RHOSP Networking 服务(neutron)可用区(AZ)。

AZ 可让您使 RHOSP 网络资源高度可用。您可以对附加到不同 AZ 上的不同电源源的网络节点进行分组，然后将关键服务调度到单独的 AZ 上。

网络服务 AZ 通常与计算服务(nova) AZ 一起使用，以确保客户使用与运行工作负载的物理站点本地的特定虚拟网络。有关分布式 Compute [节点架构的更多信息](#)，请参[阅分布式计算节点和存储部署指南](#)。

### 22.1. 关于网络服务可用区

提供 Red Hat OpenStack Platform (RHOSP) 网络服务(neutron)可用区(AZ)功能所需的扩展是 **availability\_zone**、**router\_availability\_zone**，以及 **network\_availability\_zone**。带有 Open vSwitch (ML2/OVS) 机制驱动程序的 Modular Layer 2 插件支持所有这些扩展。



#### 注意

带有 Open Virtual Network (ML2/OVN) 机制驱动程序的 Modular Layer 2 插件只支持路由器可用区。ML2/OVN 有一个分布式 DHCP 服务器，因此不需要支持网络 AZ。

在创建网络资源时，您可以使用 OpenStack 客户端命令行选项 **--availability-zone-hint** 指定 AZ。您指定的 AZ 添加至 AZ 提示列表中。但是，在调度资源前，AZ 属性实际不会被设置。分配给您的网络资源的实际 AZ 可能与您通过 hint 选项指定的 AZ 不同。这个不匹配的原因是，区失败，或者指定的区域没有剩余容量。

如果用户在创建网络资源时无法指定 AZ，您可以为默认 AZ 配置网络服务。除了设置默认 AZ 外，您还可以配置特定的驱动程序来调度 AZ 上的网络和路由器。

#### 其他资源

- [使用 ML2/OVS 配置网络服务可用区](#)
- [使用 ML2/OVN 配置网络服务可用区](#)
- [手动将可用区分配给网络和路由器](#)

### 22.2. 为 ML2/OVS 配置网络服务可用区

当用户创建网络和路由器时，您可以设置由 Red Hat OpenStack Platform (RHOSP) 网络服务(neutron)自动分配的一个或多个默认可用区(AZ)。此外，您还可以设置网络服务用来将这些资源调度到相应的 AZ 的网络和路由器驱动程序。

本主题中包含的信息用于运行 RHOSP 网络服务的部署，这些服务使用带有 Open vSwitch 机制驱动程序 (ML2/OVS) 的模块第 2 个插件。

#### 先决条件

- 部署的 RHOSP 16.2 或更高版本。
- 运行使用 ML2/OVS 机制驱动程序的 RHOSP 网络服务。

- 在分布式计算节点(DCN)环境中使用网络服务 AZ 时，您必须将网络服务 AZ 名称与计算服务 (nova) AZ 名称匹配。  
如需更多信息，请参阅 [分布式计算节点和存储部署指南](#)。

## 流程

- 以 **stack** 用户身份登录 undercloud，再提供 **stackrc** 文件以启用 director 命令行工具。

### 示例

```
$ source ~/stackrc
```

- 创建自定义 YAML 环境文件。

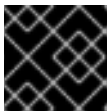
### 示例

```
$ vi /home/stack/templates/my-neutron-environment.yaml
```

### 提示

Red Hat OpenStack Platform 编排服务(heat)使用一组名为 *template* 的计划来安装和配置您的环境。您可以使用自定义环境文件来自定义 overcloud 的各个方面，这是为 heat 模板 *提供自定义* 的特殊模板。

- 在 YAML 环境文件的 **parameter\_defaults** 下，输入 **NeutronDefaultAvailabilityZones** 参数和一个或多个 AZ。如果用户在创建网络或路由器时使用 **--availability-zone-hint** 选项指定 AZ，则网络服务会分配这些 AZ。



### 重要

在 DCN 环境中，您必须将网络服务 AZ 名称与计算服务 AZ 名称匹配。

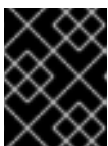
### 示例

```
parameter_defaults:
  NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
```

- 通过分别为参数输入 **NeutronDhcpAgentAvailabilityZone** 和 **NeutronL3AgentAvailabilityZone** 的值，确定 DHCP 和 L3 代理的 AZ。

### 示例

```
parameter_defaults:
  NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
  NeutronL3AgentAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
  NeutronDhcpAgentAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
```



### 重要

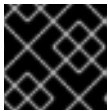
在 DCN 环境中，为 **NeutronDhcpAgentAvailabilityZone** 定义单个 AZ，以便在与特定边缘站点相关的 AZ 中调度端口。

- 默认情况下，网络和路由器调度程序分别是 **AZAwareWeightScheduler** 和 **AZLeastRoutersScheduler**。如果要更改其中一个或两者都更改，请分别使用 **NeutronNetworkSchedulerDriver** 和 **NeutronRouterSchedulerDriver** 参数输入新的调度程序。

### 示例

```
parameter_defaults:
  NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
  NeutronL3AgentAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
  NeutronDhcpAgentAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
  NeutronNetworkSchedulerDriver:
  'neutron.scheduler.dhcp_agent_scheduler.AZAwareWeightScheduler'
  NeutronRouterSchedulerDriver:
  'neutron.scheduler.l3_agent_scheduler.AZLeastRoutersScheduler'
```

- 运行 **openstack overcloud deploy** 命令，并包含核心 heat 模板、环境文件和新的自定义环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

### 示例

```
$ openstack overcloud deploy --templates \
-e <your-environment-files> \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
my-neutron-environment.yaml
```

### 验证

- 运行可用区 **list** 命令确认可用区 已正确定义。

### 示例

```
$ openstack availability zone list
```

### 输出示例

```
+-----+-----+
| Zone Name   | Zone Status |
+-----+-----+
| az-central  | available   |
| az-datacenter1 | available   |
| az-datacenter2 | available   |
+-----+-----+
```

### 其他资源

- [关于网络服务可用区](#)

- [使用 ML2/OVN 配置网络服务可用区](#)
- [手动将可用区分配给网络和路由器](#)

## 22.3. 使用 ML2/OVN 配置网络服务可用区

您可以在用户创建路由器时设置由 Red Hat OpenStack Platform (RHOSP)网络服务(neutron)自动分配的一个或多个默认可用区(AZ)。此外，您还可以设置网络服务用来为相应的 AZ 调度这些资源的路由器驱动程序。

本主题中包含的信息用于运行 RHOSP 网络服务的部署，这些服务使用具有 Open Virtual Network (ML2/OVN)机制驱动程序的 Modular Layer 2 插件。



### 注意

ML2/OVN 机制驱动程序只支持路由器可用区。ML2/OVN 有一个分布式 DHCP 服务器，因此不需要支持网络 AZ。

### 先决条件

- 部署的 RHOSP 16.2 或更高版本。
- 运行使用 ML2/OVN 机制驱动程序的 RHOSP 网络服务。
- 在分布式计算节点(DCN)环境中使用网络服务 AZ 时，您必须将网络服务 AZ 名称与计算服务(nova) AZ 名称匹配。  
如需更多信息，请参阅 [分布式计算节点和存储部署指南](#)。



### 重要

通过设置 `OVNCMOptions: 'enable-chassis-as-gw'`，并通过为 `OVNAvailabilityZone` 参数提供一个或多个 AZ 值来确保所有路由器网关端口驻留在 OpenStack Controller 节点上。执行此操作可防止路由器将所有机箱调度到路由器网关端口的潜在主机。

### 流程

1. 以 stack 用户身份登录 undercloud，再提供 `stackrc` 文件以启用 director 命令行工具。

#### 示例

```
$ source ~/stackrc
```

2. 创建自定义 YAML 环境文件。

#### 示例

```
$ vi /home/stack/templates/my-neutron-environment.yaml
```

## 提示

Red Hat OpenStack Platform 编排服务(heat)使用一组名为 *template* 的计划来安装和配置您的环境。您可以使用自定义环境文件来自定义 overcloud 的各个方面，这是为 heat 模板 提供自定义的特殊模板。

3. 在 YAML 环境文件的 **parameter\_defaults** 下，输入 **NeutronDefaultAvailabilityZones** 参数和一个或多个 AZ。



### 重要

在 DCN 环境中，您必须将网络服务 AZ 名称与计算服务 AZ 名称匹配。

如果用户在创建网络或路由器时使用 **--availability-zone-hint** 选项指定 AZ，则网络服务会分配这些 AZ。

## 示例

```
parameter_defaults:
  NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
```

4. 通过输入参数 **OVNAvailabilityZone** 的值，确定网关节点(Controller 和网络节点)的 AZ。



### 重要

**OVNAvailabilityZone** 参数替换 **OVNCMSOptions** 参数中使用 AZ 值。如果使用 **OVNAvailabilityZone** 参数，请确保 **OVNCMSOptions** 参数中没有 AZ 值。

## 示例

在本例中，为 **az-central** AZ 预定义了角色 **Controllers**，为 **datacenter1** 和 **datacenter2** AZ 预定义了角色 **Networkers**：

```
parameter_defaults:
  NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
  ControllerCentralParameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
  NetworkerDatacenter1Parameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-datacenter1'
  NetworkerDatacenter2Parameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-datacenter2'
```



### 重要

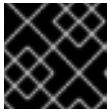
在 DCN 环境中，为 **ControllerCentralParameter** 定义单个 AZ，以便在与特定边缘站点相关的 AZ 中调度端口。

5. 默认情况下，路由器调度程序为 **AZLeastRoutersScheduler**。如果要更改此值，请使用 **NeutronRouterSchedulerDriver** 参数输入新的调度程序。

## 示例

```
parameter_defaults:
  NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
  ControllerCentralParameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-central,az-datacenter2,az-datacenter1'
  NetworkerDCN1Parameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-datacenter1'
  NetworkerDCN2Parameters:
    OVNCMSOptions: 'enable-chassis-as-gw'
    OVNAvailabilityZone: 'az-datacenter2'
  NeutronRouterSchedulerDriver:
    'neutron.scheduler.l3_agent_scheduler.AZLeastRoutersScheduler'
```

- 运行 **openstack overcloud deploy** 命令，并包含核心 heat 模板、环境文件和新的自定义环境文件。



### 重要

环境文件的顺序非常重要，因为后续环境文件中定义的参数和资源具有优先权。

## 示例

```
$ openstack overcloud deploy --templates \
-e <your-environment-files> \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
my-neutron-environment.yaml
```

## 验证

- 运行可用区 **list** 命令确认可用区 已正确定义。

## 示例

```
$ openstack availability zone list
```

## 输出示例

```
+-----+-----+
| Zone Name   | Zone Status |
+-----+-----+
| az-central  | available   |
| az-datacenter1 | available   |
| az-datacenter2 | available   |
+-----+-----+
```

## 其他资源

- [关于网络服务可用区](#)

- 使用 ML2/OVS 配置网络服务可用区
- 手动将可用区分配给网络和路由器

## 22.4. 手动将可用区分配给网络和路由器

在创建 RHOSP 网络或路由器时，您可以手动分配 Red Hat OpenStack Platform (RHOSP) 网络服务 (neutron) 可用区 (AZ)。AZ 可让您使 RHOSP 网络资源高度可用。您可以对附加到不同 AZ 上的不同电源的网络节点进行分组，然后调度运行关键服务的节点，使其位于单独的 AZ 上。



### 注意

如果您在创建网络或路由器时无法分配 AZ，RHOSP 网络服务会自动分配给为 RHOSP 编排服务 (heat) 参数指定的值。如果没有为 **NeutronDefaultAvailabilityZones** 定义值，则会在没有任何 AZ 属性的情况下调度资源。

对于使用具有 Open vSwitch (ML2/OVS) 机制驱动程序的 Modular Layer 2 插件的 RHOSP 网络服务代理，如果没有提供 AZ 提示，且没有为 **NeutronDefaultAvailabilityZones** 指定的值，则使用计算服务 (nova) AZ 值来调度代理。

### 先决条件

- 部署的 RHOSP 16.2 或更高版本。
- 运行使用 ML2/OVS 或 ML2/OVN (Open Virtual Network) 机制驱动程序的 RHOSP 网络服务。

### 流程

- 使用 OpenStack 客户端在 overcloud 上创建网络时，请使用 **--availability-zone-hint** 选项。



### 注意

ML2/OVN 机制驱动程序只支持路由器可用区。ML2/OVN 有一个分布式 DHCP 服务器，因此不需要支持网络 AZ。

在以下示例中，会创建一个网络 (**net1**)，并分配给 AZ **zone-1** 或 **zone-2**：

### 网络示例

```
$ openstack network create --availability-zone-hint zone-1 \
--availability-zone-hint zone-2 net1
```

### 输出示例

```
+-----+
| Field          | Value          |
+-----+
| admin_state_up | UP             |
| availability_zone_hints | zone-1        |
|                 | zone-2        |
| availability_zones |               |
| created_at     | 2021-07-31T22:14:12Z |
| description    |               |
```

```

| headers          |          |
| id               | ad88e059-e7fa-4cf7-8857-6731a2a3a554 |
| ipv4_address_scope | None    |
| ipv6_address_scope | None    |
| mtu              | 1450    |
| name             | net1    |
| port_security_enabled | True   |
| project_id       | cfd1889ac7d64ad891d4f20aef9f8d7c |
| provider:network_type | vxlan  |
| provider:physical_network | None  |
| provider:segmentation_id | 77   |
| revision_number  | 3       |
| router:external  | Internal |
| shared           | False   |
| status           | ACTIVE  |
| subnets         |         |
| tags             | []      |
| updated_at       | 2021-07-31T22:14:13Z |
+-----+-----+

```

- 使用 OpenStack 客户端在 overcloud 上创建路由器时，请使用 `--ha` 和 `--availability-zone-hint` 选项。

在以下示例中，创建一个路由器(`router1`)，并分配给 AZ `zone-1` 或 `zone-2`：

### 路由器示例

```

$ openstack router create --ha --availability-zone-hint zone-1 \
--availability-zone-hint zone-2 router1

```

### 输出示例

```

+-----+-----+
| Field          | Value          |
+-----+-----+
| admin_state_up | UP             |
| availability_zone_hints | zone-1      |
|                 | zone-2        |
| availability_zones |              |
| created_at      | 2021-07-31T22:16:54Z |
| description     |               |
| distributed     | False         |
| external_gateway_info | null        |
| flavor_id       | None          |
| ha              | False         |
| headers         |               |
| id              | ced10262-6cfe-47c1-8847-cd64276a868c |
| name            | router1       |
| project_id      | cfd1889ac7d64ad891d4f20aef9f8d7c |
| revision_number | 3             |
| routes         |               |
| status          | ACTIVE        |
| tags            | []            |
| updated_at      | 2021-07-31T22:16:56Z |
+-----+-----+

```



请注意，在创建网络资源时不会分配实际的 AZ。RHOSP 网络服务在调度资源时分配 AZ。

## 验证

- 输入适当的 OpenStack client **show** 命令，以确认在哪个区域中托管资源。

## 示例

```
$ openstack network show net1
```

## 输出示例

```
+-----+-----+
| Field          | Value                               |
+-----+-----+
| admin_state_up | UP                                   |
| availability_zone_hints | zone-1                               |
|                 | zone-2                               |
| availability_zones | zone-1                               |
|                 | zone-2                               |
| created_at      | 2021-07-31T22:14:12Z               |
| description     |                                       |
| headers         |                                       |
| id              | ad88e059-e7fa-4cf7-8857-6731a2a3a554 |
| ipv4_address_scope | None                                 |
| ipv6_address_scope | None                                 |
| mtu             | 1450                                 |
| name            | net1                                 |
| port_security_enabled | True                                 |
| project_id      | cfd1889ac7d64ad891d4f20aef9f8d7c   |
| provider:network_type | vxlan                               |
| provider:physical_network | None                                 |
| provider:segmentation_id | 77                                  |
| revision_number | 3                                   |
| router:external | Internal                             |
| shared          | False                               |
| status          | ACTIVE                              |
| subnets        |                                       |
| tags            | []                                   |
| updated_at      | 2021-07-31T22:14:13Z               |
+-----+-----+
```

## 其他资源

- [关于网络服务可用区](#)
- [使用 ML2/OVS 配置网络服务可用区](#)
- [使用 ML2/OVN 配置网络服务可用区](#)

## 第 23 章 使用标签识别虚拟设备

### 23.1. 标记虚拟设备

在 Red Hat OpenStack Platform 中，如果您启动具有多个网络接口或块设备的虚拟机实例，您可以使用设备标记将每个设备的预期角色与实例操作系统通信。标签在实例引导时分配给设备，可通过元数据 API 和配置驱动器（如果启用）提供给实例操作系统。

#### 流程

- 要标记虚拟设备，请在创建实例时使用标签参数 **--block-device** 和 **--nic**。

#### 示例

```
$ nova boot test-vm --flavor m1.tiny --image cirros \
--nic net-id=55411ca3-83dd-4036-9158-bf4a6b8fb5ce,tag=nfv1 \
--block-device id=b8c9bef7-aa1d-4bf4-a14d-17674b370e13,bus=virtio,tag=database-server
NFVappServer
```

生成的标签添加到现有实例元数据中，并通过元数据 API 和配置驱动器获取。

在这个示例中，以下 `devices` 部分填充元数据：

`meta_data.json` 文件的内容示例：

```
{
  "devices": [
    {
      "type": "nic",
      "bus": "pci",
      "address": "0030:00:02.0",
      "mac": "aa:00:00:00:01",
      "tags": ["nfv1"]
    },
    {
      "type": "disk",
      "bus": "pci",
      "address": "0030:00:07.0",
      "serial": "disk-vol-227",
      "tags": ["database-server"]
    }
  ]
}
```

设备标签元数据可使用 metadata API 中的 **GET /openstack/latest/meta\_data.json**。

如果启用了配置驱动器，并在实例操作系统的 `/configdrive` 下挂载，则元数据也存在于 `/configdrive/openstack/latest/meta_data.json` 中。