Red Hat OpenStack Platform 17.1

配置 Red Hat OpenStack Platform 网络

管理 OpenStack 网络服务 (neutron)

Last Updated: 2024-05-25

Red Hat OpenStack Platform 17.1 配置 Red Hat OpenStack Platform 网络

管理 OpenStack 网络服务 (neutron)

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

用于常见 OpenStack 网络任务的说明书。

目录

前言

注意

您无法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，您必须首先创建端口，将共享安全组应用到该端口，然后将该端口分配给实例。请参阅向端口添加安全组。

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 网络服务管理进出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，并管理交换机、路由器、端口和防火墙。

1.1. 管理 RHOSP 网络

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

- **提供与项目中的虚拟机实例的连接。**
  项目网络的主要目的是，让常规（非特权）项目在不涉及管理员的情况下管理网络。这些网络完全是虚拟的，需要虚拟路由器与其它项目网络和互联网等外部网络交互。项目网络通常向虚拟机（虚拟机）实例提供 DHCP 和元数据服务。RHOSP 支持以下项目网络类型：扁平（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。**
  提供商网络提供像项目网络一样的网络连接。但是，只有管理（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接口。RHOSP 支持以下提供商网络类型： flat 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）与使用 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，如果发生主要错误，则只有一个网段会受到影响，而不是整个网络失败。

  如需更多信息，请参阅部署路由供应商网络。

- **使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)来保持网络资源高度可用。Operator 对附加到不同 AZ 上不同电源源的网络节点进行分组。接下来，操作员将重要的服务（如 DHCP、L3、FW 等）调度到单独的 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)的替代路由设计基于 VRRP，可在每个 Compute 节点上部署 L3 代理和调度路由器。

  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容器，该规则控制入口（绑定到实例）和出口（从实例出站）网络流量。安全组使用默认拒绝策略，仅包含允许特定流量的规则。每个端口可以以增加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（如 iptables）的配置。

默认情况下，安全组是有状态的。在 ML2/OVN 部署中，您还可以创建无状态安全组。无状态安全组可以提供显著的性能优势。与有状态安全组不同，无状态安全组不会自动允许返回流量，因此您必须创建免费安全组规则，以允许返回相关的流量。

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（无论子网是什么）。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。

  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。**
通过使用 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横向扩展。

  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)策略将速率限制应用到出口和入口流量，为实例提供不同的服务级别。您可以将 QoS 策略应用到单个端口。您还可以将 QoS 策略应用到项目网络，其中未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。**
通过网络服务配额选项，您可以设置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。

- **优化网络功能虚拟化(NFV)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(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 上的虚拟机实例使用，并被其他实例和设备忽略。VLAN 中继通过将 VLAN 合并到单一中继端口来支持 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，并允许它们连接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 。

- **控制对实例的网络访问。**
您可以使用安全组来控制对实例的网络和协议访问。安全组是 IP 过滤规则的集合，例如，允许用户对实例执行 ICMP ping，并运行 SSH 来连接实例。安全组规则应用到项目中的所有实例。

  如需更多信息，请参阅配置安全组。

- **将流量流事件记录到实例进出。**
您可以为安全组创建数据包日志，以监控进出虚拟机 (VM) 实例的流量。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。

如需更多信息，请参阅日志记录安全组操作。

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，以及支持第 2 层网络和网关到外部网络之间的路由的第 3 层路由器构造的扩展。RHOSP 网络包括增加的插件列表，支持与各种商业和开源网络技术（包括路由器、交换机、虚拟交换机和软件定义网络(SDN)控制器）互操作性。

- 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
  接受并路由 RHOSP 服务之间的 RPC 请求，以完成 API 操作。

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实现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 框架区分可以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。
每种可用的网络类型由 ML2 类型驱动程序管理，它们维护任何特定于类型的网络状态。验证提供商网络的类型特定信息，类型驱动程序负责项目网络中空闲片段的分配。类型驱动程序的示例为 GENEVE、GRE 和 VXLAN 等。

机制驱动程序

定义访问特定类型的 RHOSP 网络的机制。
机制驱动程序获取类型驱动程序创建的信息，并将其应用到已启用的网络机制。机制驱动程序的示例是 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和类型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 将端口绑定到连接。根据机制驱动程序，ML2 支持以下网络段类型：

- Flat
- VLAN
- GENEVE 隧道

- VXLAN 和 GRE 隧道

Flat

所有虚拟机(VM)实例驻留在同一网络中，也可以与主机共享。没有 VLAN 标记或其他网络分离。

VLAN

使用 RHOSP 网络用户可以使用与物理网络中存在的 VLAN ID (802.1Q 标记)对应的 VLAN 创建多个供应商或项目网络。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、负载平衡器和其他网络基础架构通信。
您可以使用 VLAN 为在同一交换机上运行的计算机分段网络流量。这意味着，您可以通过将端口配置为不同网络 members，来以逻辑方式划分您的交换机，这些端口基本上是 mini-LAN，您可以出于安全原因使用它们来分隔流量。

例如，如果您的交换机总共有 24 个端口，您可以将端口 1-6 分配给 VLAN200，端口 7-18 分配给 VLAN201。因此，连接到 VLAN200 的计算机与 VLAN201 上的计算机完全独立；它们无法直接通信，如果它们希望通过路由器，则流量必须通过路由器，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变化能力和需求。它为隧道提供了一个框架，而不是对整个系统进行规范。Geneve 定义在封装期间添加的元数据的内容，并尝试适应各种虚拟化场景。它使用 UDP 作为其传输协议，并且使用可扩展的选项标头动态大小。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之间的私有通信。需要 RHOSP 网络路由器，以便流量遍历 GRE 或 VXLAN 项目网络的外部。还需要路由器直接连接到外部网络（包括互联网）；路由器提供使用浮动 IP 地址从外部网络直接连接实例的功能。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. 模块层 2 (ML2) 机制驱动程序

模块层 2 (ML2) 插件作为通用代码库的机制实施。这种方法可以重复使用代码，并消除了代码维护和测试方面的复杂性。

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

```
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，因为它前面是 **NeutronMechanismDrivers** 的值列表中的 **baremetal**。

红帽选择 ML2/OVN 作为从 RHOSP 15 开始的所有新部署的默认机制驱动程序，因为它现在为大多数客户提供了与 ML2/OVS 机制驱动程序直接的好处。当我们继续增强和改进 ML2/OVN 功能集时，每个版本都会有多大优势。

通过 RHOSP 17 发行版本，支持已弃用的 ML2/OVS 机制驱动程序。在此期间，ML2/OVS 驱动程序保持维护模式，接收程序错误修复和正常支持，大多数新的功能开发都会在 ML2/OVN 机制驱动程序中发生。

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，请开始评估迁移到机制驱动程序的计划。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。

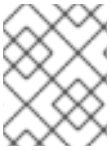
红帽需要在尝试从 ML2/OVS 迁移到 ML2/OVN 前提交主动支持问题单。红帽不支持在没有主动支持问题单的情况下进行迁移。请参见 如何在 Red Hat OpenStack Platform 上为计划的活动创建一个主动问题单？

其他资源

- *Component, Plug-In, and Driver Support in Red Hat OpenStack Platform* 中的 Neutron
- *自定义 Red Hat OpenStack Platform 部署* 指南中的环境文件 https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy the-overcloud-with-the-orchestration-service#assembly_understanding-heat-templates
- 在自定义 *Red Hat OpenStack Platform 部署* 指南中的 overcloud 创建中包括环境文件

1.6. OPEN VSWITCH

Open vSwitch (OVS) 是一种软件定义网络 (SDN) 虚拟交换机，类似于 Linux 软件网桥。OVS 向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 VXLAN 和 GRE 的隧道。



注意

为了降低 OVS 中网络循环的风险，只有一个接口或一个绑定只能是给定网桥的成员。如果需要多个绑定或接口，可以配置多个网桥。

其他资源

- *自定义 Red Hat OpenStack Platform 部署指南中的网络接口绑定*。https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy networks-for-the-rhosp-environment#assembly_network-interface-bonding

1.7. OPEN VIRTUAL NETWORK (OVN)

开放虚拟网络(OVN)是支持虚拟机和容器环境中的逻辑网络抽象的系统。OVN 称为 Open vSwitch 的开源虚拟网络，OVN 会补充 OVS 的现有功能，为逻辑网络抽象（如逻辑 L2 和 L3 覆盖、安全组和服务）添加原生支持，如 DHCP。

物理网络由物理线、交换机和路由器组成。虚拟网络将物理网络扩展到 hypervisor 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网络，它通过隧道或其他封装从物理网络中推断出来。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 重叠，而不会导致冲突。可以安排逻辑网络拓扑，而不考虑其上运行的物理网络的拓扑结构。因此，作为逻辑网络一部分的虚拟机可以在不中断网络的情况下从一个物理机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，这可能可以接受或是理想的选择，但在很多情况下，虚拟机和容器确实需要与物理网络连接。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 逻辑网络基础架构连接到物理网络元素，将 OVN 整合到物理网络元素中。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

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

网关

通过转发隧道和物理网络基础架构之间的数据包，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 Modular Layer 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
Open Virtual Network (OVN)	是	否	是	是 [1]	是
Open vSwitch (OVS)	是	是	是	是	否

[1] ML2/OVN VXLAN 支持为每个网络限制为 4096 个网络和 4096 端口。另外，依赖入口端口的 ACL 不适用于 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。扩展可以实现两种目的：它们允许在不需要版本更改的情况下引入 API 中的新功能，以及允许引入特定于供应商的小体功能。应用可以通过在 **/extensions** URI 上执行 GET 以编程方式列出可用的扩展。请注意，这是版本化的请求，即，一个 API 版本中提供的扩展可能在另一个版本中不可用。

ML2 插件还支持扩展驱动程序，允许其他可插拔驱动程序为网络对象扩展 ML2 插件中实施的核心资源。扩展驱动程序示例包括对 QoS、端口安全性等的支持。

第 2 章 使用 ML2/OVN

Red Hat OpenStack Platform (RHOSP) 网络由网络服务(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。

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

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，以支持网络 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，并监听 TCP 端口 **6641**。

OVN 北向服务 (ovn-northd)

此服务将 OVN NB 数据库中的逻辑网络配置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)

此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，并监听 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 控制器来控制和监控网络流量。它在定义了 **OS::Tripleo::Services::OVNController** 的所有 Compute 和 gateway 节点上运行。

OVN 元数据代理 (ovn-metadata-agent)

此代理创建用于管理 OVS 接口的 **haproxy** 实例、网络命名空间和 HAProxy 进程，用于代理元数据 API 请求。代理在定义了 **OS::TripleO::Services::OVNMetadataAgent** 的所有 Compute 和 gateway 节点上运行。

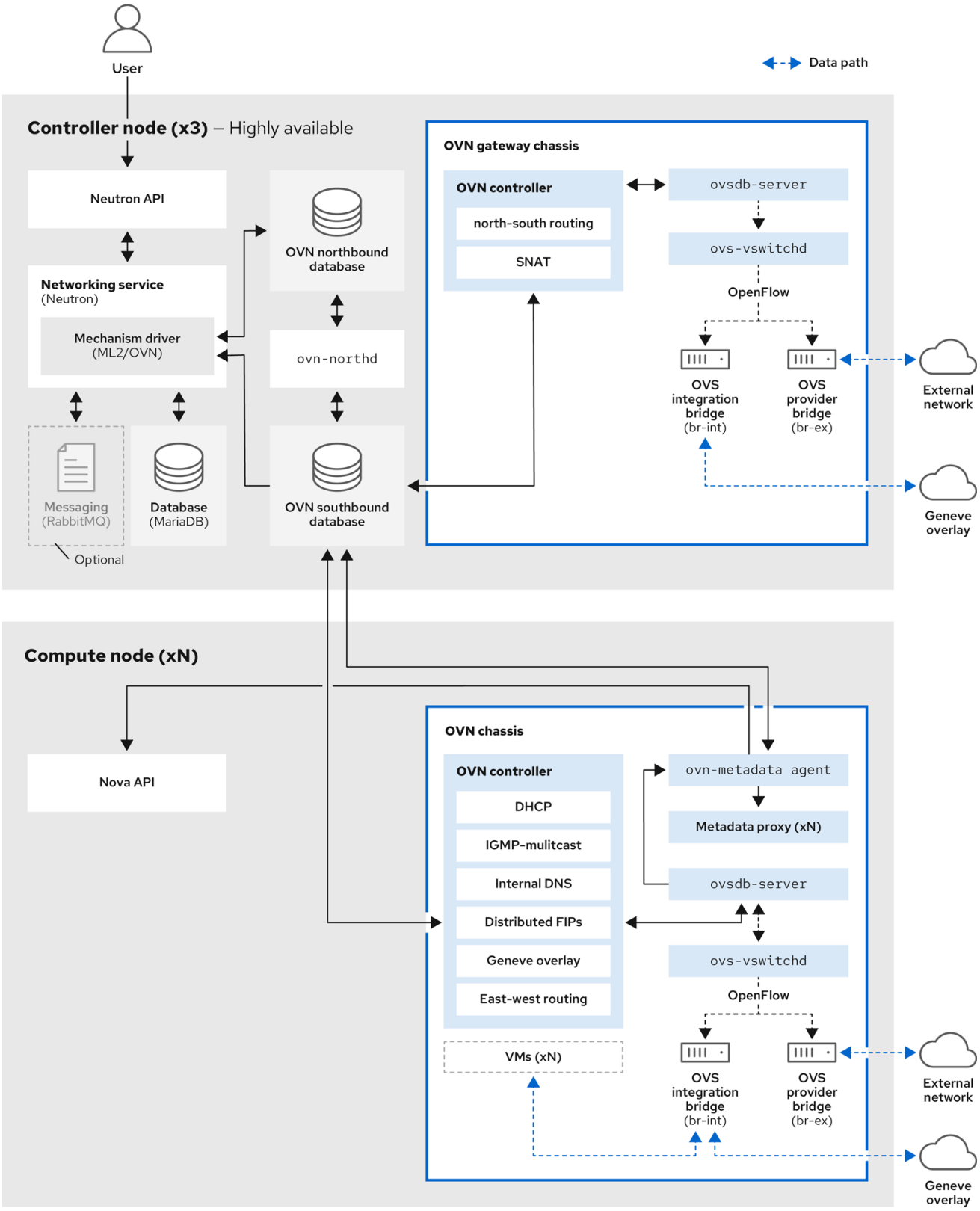
OVS 数据库服务器 (OVSDB)

托管 OVN 北向和南向数据库。另外，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**。

图 2.1. RHOSP 环境中的 OVN 架构



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确定问题。

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、逻辑路由器等形式显示。每个 RHOSP Networking 服务 (neutron) 对象都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来配置 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-vswitchd** 进行通信并安装 OpenFlow 流， **ovn-controller** 使用在 **ovn-controller** 启动时使用的 UNIX socket 路径（如 **unix:/var/run/openvswitch/db.sock**），连接到活跃的 **ovsdb-server** 服务器（托管 **conf.db**）之一。

ovn-controller 服务需要 **Open_vSwitch** 表格的 **external_ids** 列中的某些键值对； **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/ovn-metadata-container-puppet.yaml** 文件中配置，并通过 **OS::TripleO::Services::OVNMetadataAgent** 包含在默认 Compute 角色中。因此，带有默认参数的 OVN 元数据代理作为 OVN 部署的一部分部署。

OpenStack 客户机实例访问位于本地链接 IP 地址：169.254.169.254 的联网元数据服务。**neutron-ovn-metadata-agent** 可以访问存在计算元数据 API 的主机网络。每个 HAProxy 都位于一个不能到达适当主机网络的网络命名空间中。HAProxy 将所需的标头添加到元数据 API 请求，然后通过 UNIX 域套接字将请求转发到 **neutron-ovn-metadata-agent**。

OVN 网络服务为每个虚拟网络创建一个唯一的网络命名空间，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、计算角色和不同的存储角色类型中的节点。这些默认角色各自包含一组在核心 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 ovn-dbs 在 Controller 节点上运行。由于该服务是可组合的，您可以将其分配给另一个角色，如 Networker 角色。通过选择将 ML2/OVN 服务分配给另一个角色，您可以减少 Controller 节点上的负载，并通过隔离 Networker 节点上的网络服务来实施高可用性策略。

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 的 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。OVN 会自动将路由器端口调度到所有可用网关节点，这些网关可以充当指定的外部网络上的 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 OpenFlow 规则通过捆绑的 active_passive 输出进行管理。**ovn-controller** 处理地址解析协议 (ARP) 响应器和路由器启用和禁用。**ovn-controller** 定期发送 FIP 和路由器外部地址的 gratuitous ARP。

注意

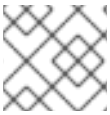
L3HA 使用 OVN 将路由器回原始网关节点，以避免任何节点成为瓶颈。

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。

每个网关节点在部署中的星号拓扑中监控所有其他网关节点。网关节点也监控计算节点，以便网关启用和禁用数据包和 ARP 响应和公告的路由。

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 DNAT)。Compute 节点不需要监控其他计算节点。



注意

未检测到外部网络故障，因为 ML2-OVS 配置会出现。

OVN 的 L3 HA 支持以下故障模式：

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

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. 主动-主动集群数据库服务模型

Red Hat OpenStack Platform (RHOSP) ML2/OVN 部署使用集群数据库服务模型，该服务应用 Raft 共识算法来增强 OVS 数据库协议流量的性能，并提供更快、更可靠的故障转移处理。从 RHOSP 17.0 开始，集群数据库服务模型替换了基于 pacemaker 的 active/backup 模型。

集群数据库在不同的主机上运行至少三个数据库服务器。服务器使用 Raft 共识算法在集群中同步写入并共享网络流量。集群选择一台服务器作为领导。集群中的所有服务器都可以处理数据库读取操作，这可以降低 control plane 的潜在瓶颈。写入操作由集群领导处理。

如果服务器失败，则会选择新的集群领导机，并在剩余的操作服务器之间重新分发流量。与基于 pacemaker 的模型相比，集群数据库服务模型可以更有效地处理故障转移。这可减少相关的停机时间和复杂情况，这些停机时间在较长的故障转移时间内可能发生。

领导选举过程需要大部分，因此容错容量受集群中最多奇数的限制。例如，如果一个服务器出现故障，三服务器集群将继续操作。五服务器集群最多容许两个故障。将服务器数量增加到偶数不会增加容错能力。例如，一个四服务器集群不会被比一个三服务器集群容忍更多的故障。

大多数 RHOSP 部署都使用三个服务器。

大于五个服务器的集群也可以正常工作，每个添加的服务器都允许集群容忍额外的故障，但写入性能会降低。

有关监控数据库服务器状态的信息，请参阅监控 OVN 数据库状态。

2.8. 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。您可以选择使用支持的自定义角色，如以下示例中描述的角色。

Networker

在专用的 networker 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。

带有 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

以下限制适用于在此发行版本中将 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一网关节点上，因为所有端口只有一个 HA Chasis Group。
- VLAN 租户网络上的 VF （直接）端口的北/南路由无法用于 SR-IOV，因为外部端口不与逻辑路由器网关端口在一起。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。
  如需更多信息，请参阅自定义 *Red Hat OpenStack Platform 部署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。如果它符合您的需要，请在 deploy 命令中直接使用它。或者，您可以生成自己的自定义角色文件，它们组合了其他自定义角色文件。

Deployment	角色	角色文件
Networker 角色	Networker	**Networker.yaml**
带有 SR-IOV 的 Networker 角色	NetworkerSriov	**NetworkerSriov.yaml**
与 SR-IOV 共存控制和网络程序	ControllerSriov	**ControllerSriov.yaml**

3. （可选）生成一个新的自定义角色数据文件，它将之前列出的自定义角色文件之一与其他自定义角色文件合并。
按照 *自定义 Red Hat OpenStack Platform 部署* 指南中的创建 roles_data 文件中 的说明进行操作。根据您的部署，包含适当的源角色文件。

4. （可选）要识别角色的特定节点，您可以创建特定的硬件类别，并将该类别分配到特定的节点。然后，使用环境文件来定义角色的类别，并指定节点数。
如需更多信息，请参阅自定义 *Red Hat OpenStack Platform 部署* 指南中的 创建新角色 中的示例。

5. 根据您的部署创建环境文件。

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，包含以下设置。

Deployment	设置

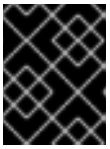
Networker 角色	```
	ControllerParameters:
	  OVNCMSOptions: ""
	ControllerSriovParameters:
	  OVNCMSOptions: ""
	NetworkerParameters:
	  OVNCMSOptions: "enable-chassis-as-gw"
	NetworkerSriovParameters:
	  OVNCMSOptions: ""
	``

Deployment	设置

带有 SR-IOV 的 Networker 角色	OS::TripleO::Services::NeutronDhcpAgent: OS::Heat::None
	
	ControllerParameters:
	  OVNCMSOptions: ""
	ControllerSriovParameters:
	  OVNCMSOptions: ""
	NetworkerParameters:
	  OVNCMSOptions: ""
	NetworkerSriovParameters:
	  OVNCMSOptions: "enable-chassis-as-gw"

与 SR-IOV 共存控制和网络程序	OS::TripleO::Services::NeutronDhcpAgent: OS::Heat::None
	
	ControllerParameters:
	  OVNCMSOptions: ""
	ControllerSriovParameters:
	  OVNCMSOptions: "enable-chassis-as-gw"
	NetworkerParameters:
	  OVNCMSOptions: ""
	NetworkerSriovParameters:
	  OVNCMSOptions: ""

7. 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

但是，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更为优先。

示例

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

验证步骤

1. 以 **tripleo-admin** 用户身份登录 Controller 或 Networker 节点：

示例

```
ssh tripleo-admin@controller-0
```

2. 确保 **ovn_metadata_agent** 正在运行。

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

输出示例

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

3. 确保具有 OVN 服务或专用 Networker 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
```

SR-IOV 部署的额外验证步骤

1. 以 **tripleo-admin** 用户身份登录 Compute 节点：

示例

```
ssh tripleo-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 ...
openstack-neutron-sriov-agent ...
kolla_start  23 hours ago  Up 21 hours ago  neutron_sriov_agent
```

3. 确保已成功检测到网络可用的 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

- *自定义 Red Hat OpenStack Platform 部署* 指南中的可组合服务和自定义角色
- *命令行界面参考*中的 overcloud 部署

2.9. 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 2.8 节 “使用 ML2/OVN 部署自定义角色”。

限制

以下限制适用于在此发行版本中将 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一网关节点上，因为所有端口只有一个 HA Chasis Group。
- VLAN 租户网络上的 VF （直接）端口的北/南路由无法用于 SR-IOV，因为外部端口不与逻辑路由器网关端口在一起。请参阅 https://bugs.launchpad.net/neutron/+bug/1875852。

其他资源

- *自定义 Red Hat OpenStack Platform 部署指南中的可组合服务和自定义角色*。

第 3 章 管理项目网络

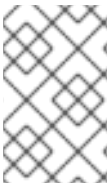
项目网络可帮助您隔离云计算的网络流量。创建项目网络的步骤包括规划和创建网络，以及添加子网和路由器。

3.1. VLAN 规划

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

例如，您的管理或 API 流量与服务 Web 流量的系统不相同，您的管理或 API 流量是理想的选择。VLAN 之间的流量穿过路由器，您可以在其中实施防火墙来管理流量流。

您必须将 VLAN 做为整个计划的一部分，包括部署中各种虚拟网络资源的流量隔离、高可用性和 IP 地址利用率。



注意

单个网络或网络节点的一个 OVS 代理中的最大 VLAN 数量为 4094。如果您需要多个 VLAN 数量上限，您可以创建多个提供商网络 (VXLAN 网络) 和多个网络节点，每个网络一个。每个节点最多可包含 4094 个专用网络。

3.2. 网络流量的类型

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，您可以对每种类型的网络拥有单独的 VLAN。只有 External 网络需要可以路由到外部物理网络。在本发行版本中，director 提供 DHCP 服务。

注意

对于每一个 OpenStack 部署，您不需要本节中的所有隔离 VLAN。例如，如果您的云用户没有根据需要创建临时虚拟网络，则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，并将实例配置为直接使用该提供商网络。

- **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，此网络还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，您必须为以下服务计划 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

- **存储** - 块存储、NFS、iSCSI 和其他存储服务。出于性能原因，将此网络隔离到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 后端通过存储管理网络连接的服务，因为它们不直接与 Ceph 交互，而是使用前端服务。请注意，RBD 驱动程序是例外；此流量直接连接到 Ceph。
- **项目网络** - Neutron 使用 VLAN 分隔（每个项目网络是一个网络 VLAN）或者使用 VXLAN 或 GRE 进行隧道提供自己的网络。网络流量在每个项目网络内被隔离。每个项目网络关联有一个 IP 子网，多个项目网络可能使用相同的地址。
- **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。您可以使用扁平网络或 VLAN 标签直接映射到数据中心中的现有物理网络。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

以下系统使用来自您分配的范围内的 IP 地址：

- **物理节点** - 每个物理 NIC 都需要一个 IP 地址。常见的做法是将物理 NIC 专用于特定功能。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)** - 计划为控制器节点共享的每个网络分配一个或多个 VIP。

3.4. 虚拟网络

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

- **项目网络** - 每个项目网络都需要一个子网，供它用于为实例分配 IP 地址。
- **虚拟路由器** - 插入子网的每个路由器接口都需要一个 IP 地址。如果要使用 DHCP，每个路由器接口都需要两个 IP 地址。
- **实例** - 每个实例都需要一个来自托管该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

要允许流量路由到新网络并从您的新网络中路由流量，您必须将其子网作为接口添加到现有的虚拟路由器中：

1. 在控制面板中，选择 **Project > Network > Routers**。
2. 在**路由器**列表中选择您的虚拟路由器名称，然后点 **添加接口**。
   在 Subnet 列表中，选择新子网的名称。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

本例显示了多个容纳多个子网的网络，每个子网被分配为 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 (包括浮动 IP)	10.1.2.10 - 10.1.3.222	469	255.255.254.0
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，需要确定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，并且最好在它们之间进行隔离措施，这将非常有用。例如，您可以指定一个子网中只有 webserver 流量，而数据库流量会遍历另一个流量。子网相互隔离，希望与其他子网通信的任何实例都必须具有路由器定向的流量。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。如果您要将网络与外部 VLAN 集成，请考虑将 VLAN ID 号附加到名称中。例如，**webservers_122**，如果在这个子网中托管 HTTP Web 服务器，您的 VLAN 标签为 **122**。或者，如果您打算将网络流量保持私有，而不将网络与外部网络集成，也可以使用 **internal-only**。
Admin State	控制网络是否立即可用。使用此字段创建 Down 状态的网络，其中逻辑上存在但不活跃。如果您不打算立即在生产环境中使用这个网络，这将非常有用。
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，并在 **Subnet** 选项卡中指定以下值：

字段	描述
子网名称	输入子网的描述性名称。
网络地址	以 CIDR 格式输入地址，其中包含一个值的 IP 地址范围和子网掩码。要确定地址，请计算子网掩码中屏蔽的位数，并将该值附加到 IP 地址范围。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 一起使用，请指定地址 192.168.122.0/24。
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。

字段	描述
网关 IP	默认网关的路由器接口的 IP 地址。此地址是路由外部位置的任何流量的下一跳，且必须在您在 Network Address 字段中指定的范围内。例如，如果您的 CIDR 网络地址为 192.168.122.0/24，那么您的默认网关可能是 192.168.122.1。
禁用网关	禁用转发和隔离子网。

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 自动向实例分配 IP 设置。
   - **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：
     - **不指定选项** - 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，请选择此选项。
     - **SLAAC （无状态地址自动配置）** - 实例根据从 OpenStack 网络路由器发送的路由器公告(RA)消息生成 IPv6 地址。使用此配置来创建 OpenStack 网络子网，并将 ra_mode 设置为 slaac，address_mode 设为 slaac。
     - **DHCPv6 stateful** - 实例从 OpenStack 网络 DHCPv6 服务接收 IPv6 地址以及附加选项（例如 DNS）。使用这个配置来创建将 ra_mode 设置为 dhcpv6-stateful 的子网，address_mode 设为 dhcpv6-stateful。
     - **DHCPv6 stateless** - 实例根据从 OpenStack 网络路由器发送的路由器公告(RA)消息生成 IPv6 地址。从 OpenStack 网络 DHCPv6 服务中分配附加选项（如 DNS）。使用这个配置来创建将 ra_mode 设置为 dhcpv6-stateless 的子网，address_mode 设为 dhcpv6-stateless。
   - **分配池** - 要分配的 IP 地址范围。例如，值 192.168.22.100,192.168.22.150 认为该范围内的所有地址都可用于分配。
   - **DNS 名称服务器** - 网络上可用的 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它们。例如，如果您的云托管 DNS，当云无法正常运行时，DNS 将不可用，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，后跟您要用于路由的下一跃点（例如 192.168.23.0/24、10.1.31.1）。如果您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** 来更改任何选项。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分配给子网，因此务必要考虑正确放置实例，以最适合其连接要求。

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，并且最好在它们之间进行隔离措施，这将非常有用。

例如，您可以指定一个子网中只有 webserver 流量，而数据库流量使用另外一个子网。

子网相互隔离，希望与其他子网通信的任何实例都必须具有路由器定向的流量。因此，您可以降低网络延迟和负载，方法是在同一子网中对需要大量流量的系统进行分组。

3.9. 创建子网

要创建子网，请按照以下步骤执行：

1. 在仪表板中，选择 **Project > Network > Networks**，然后在 **Networks** 视图中点您的网络名称。
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，其中包含一个值的 IP 地址范围和子网掩码。要确定 CIDR 地址，请计算子网掩码中屏蔽的位数，并将该值附加到 IP 地址范围。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 一起使用，请指定地址 192.168.122.0/24。
IP 版本	Internet 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。此地址是路由外部位置的任何流量的下一跳，且必须在您在 Network Address 字段中指定的范围内。例如，如果您的 CIDR 网络地址为 192.168.122.0/24，那么您的默认网关可能是 192.168.122.1。
禁用网关	禁用转发和隔离子网。

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 自动向实例分配 IP 设置。
   - **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：
     - **不指定选项** - 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，请选择此选项。

- **SLAAC（无状态地址自动配置）** - 实例根据从 OpenStack 网络路由器发送的路由器公告(RA)消息生成 IPv6 地址。使用此配置来创建 OpenStack 网络子网，并将 ra_mode 设置为 slaac，address_mode 设为 slaac。
- **DHCPv6 stateful** - 实例从 OpenStack 网络 DHCPv6 服务接收 IPv6 地址以及附加选项（例如 DNS）。使用这个配置来创建将 ra_mode 设置为 dhcpv6-stateful 的子网，address_mode 设为 dhcpv6-stateful。
- **DHCPv6 stateless** - 实例根据从 OpenStack 网络路由器发送的路由器公告(RA)消息生成 IPv6 地址。从 OpenStack 网络 DHCPv6 服务中分配附加选项（如 DNS）。使用这个配置来创建将 ra_mode 设置为 dhcpv6-stateless 的子网，address_mode 设为 dhcpv6-stateless。

- **分配池** - 要分配的 IP 地址范围。例如，值 192.168.22.100,192.168.22.150 认为该范围内的所有地址都可用于分配。
- **DNS 名称服务器** - 网络上可用的 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，后跟您要用于路由的下一跃点（例如 192.168.23.0/24、10.1.31.1）。如果您需要向实例分发静态路由，请提供这个值。

4. 点 **Create**。
   您可以查看 **Subnets** 列表中的子网。您还可以根据需要点 **Edit** 来更改任何选项。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。

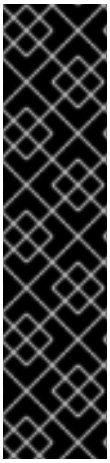
3.10. 添加路由器

OpenStack 网络使用基于 SDN 的虚拟路由器提供路由服务。路由器是您的实例与外部子网通信的要求，包括物理网络中的实例。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。

路由器的默认网关为路由器接收的任何流量定义下一跳。其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**，再点 **Create Router**。
2. 输入新路由器的描述性名称，再点**创建路由器**。
3. 在 **Routers** 列表中，点新路由器条目旁边的 **Set Gateway**。
4. 在 **External Network** 列表中，指定您要接收用于外部位置的流量的网络。
5. 点 **Set Gateway**。
   添加路由器后，您必须配置您创建的子网来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完成此操作。



重要

子网的默认路由不能被覆盖。当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出。如果删除了现有的路由器命名空间路由，要解决这个问题，请执行以下步骤：

1. 取消关联子网上的所有浮动 IP。
2. 将路由器与子网分离。
3. 重新将路由器附加到子网。
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. 在仪表板中，选择 **Project > Network > Routers**，然后点您要删除的路由器的名称。
2. 选择类型为 **Internal Interface** 的接口，然后点 **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，如果任何实例仍然配置为使用子网，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. 点您的网络名称。

3. 选择目标子网，再点**删除子网**。

3.14. 删除网络

在某些情况下可能需要删除之前创建的网络，如进行内务操作时或作为停用流程的一部分。您必须先删除或分离仍使用网络的任何接口，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. 进入到 **Project > Network > Routers**，点 **Routers** 列表中的虚拟路由器名称，并找到附加到您要删除的子网的接口。
您可以使用作为网关 IP 的 IP 地址将这个子网与其它子网区分开。您可以通过确保接口的网络 ID 与您在上一步中记录的 ID 匹配来进一步验证区别。

3. 点您要删除的接口的 **Delete Interface** 按钮。

4. 选择 **Project > Network > Networks**，然后点您的网络名称。

5. 点您要删除的子网的**删除子网**按钮。

   注意

   如果您仍然无法删除子网，请确保它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. 点 **Delete Networks**。

第 4 章 将虚拟机实例连接到物理网络

您可以使用扁平和 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) 具有两种服务类别，分布在多个节点类型上。

- **Neutron 服务器** - 此服务运行 OpenStack 网络 API 服务器，它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详细信息。
- **Neutron 代理** - 这些是执行 OpenStack 网络网络功能的服务：
  - **neutron-dhcp-agent** - 管理项目专用网络的 DHCP IP 寻址。
  - **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（也称为实例）的虚拟机监控程序。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 l2 代理运行的位置，如 **neutron-openvswitch-agent**。

其他资源

- 第 4.2 节 “OpenStack 网络服务放置”

4.2. OPENSTACK 网络服务放置

OpenStack 网络服务可以在同一物理服务器或单独的专用服务器上运行，它们根据其角色命名：

- *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种节点类型的环境。如果您的部署同时拥有同一物理节点上的 Controller 和网络节点角色，则必须从该服务器上的两个部分执行这些步骤。这也适用于高可用性(HA)环境，其中所有三个节点都可能运行使用 HA 的 Controller 节点和网络节点服务。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，并且计划将每个计算和网络节点连接到这些外部网络，这将非常有用。

先决条件

- 您有一个多个物理网络。
  这个示例分别使用名为 **physnet1** 和 **physnet2** 的物理网络。

- 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，创建一个自定义 YAML 环境文件。

   示例

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

   提示

   Red Hat OpenStack Platform Orchestration 服务(heat)使 用一组名为 *template* 的计划来安装和配置您的环境。您可以使用一个*自定义环境文件*来自定义 overcloud 的各个方面，它是为编配模板提供自定义的特殊模板类型。

2. 在 **parameter_defaults** 下的 YAML 环境文件中，使用 **NeutronBridgeMappings** 来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，使用附加接口来配置网桥。

   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
   ...
   ```

4. 运行 **openstack overcloud deploy** 命令，并包含模板和环境文件，包括修改的自定义 NIC 模板和新的环境文件。

重要

但是，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更为优先。

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. 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。将其配置为共享网络（使用 **--share**），以便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
```

2. 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
```

3. 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

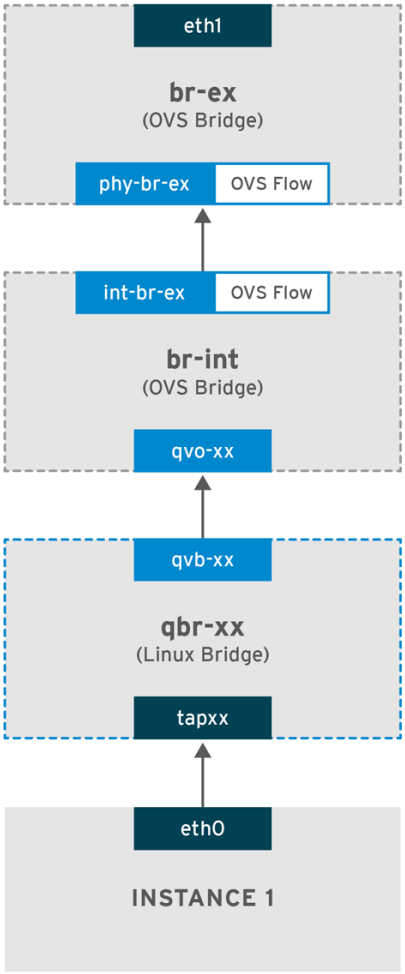
- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 环境文件
- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的创建 overcloud 中的环境文件。
- *命令行界面参考*中的 network create
- *命令行接口参考*中的 subnet create
- *命令行接口参考*中的 server create

4.4. 扁平提供商网络数据包流的工作方式是什么？

本节介绍了如何将流量流进到具有扁平供应商网络配置的实例。

扁平提供商网络中的传出流量流

下图描述了离开实例的流量的数据包流，直接到达外部网络。配置 **br-ex** 外部网桥后，将物理接口添加到网桥，并将实例生成到 Compute 节点后，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。

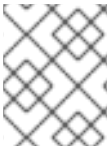
'qbr-xx'Linux 网桥配置示例：

```
# 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 "qvof63599ba-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 标签是 **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 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
```

当此数据包到达 **br-ex** 上的 **phy-br-ex** 时，**br-ex** 中的 OVS 流会剥离 VLAN 标签(5)，并将它转发到物理接口。

在以下示例中，输出中的 **phy-br-ex** 的端口号显示为 **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**)的数据包，其 VLAN 标签为 **5** (**dl_vlan=5**)。此外，br-ex 中的 OVS 流将剥离 VLAN 标签，并转发数据包到物理接口。

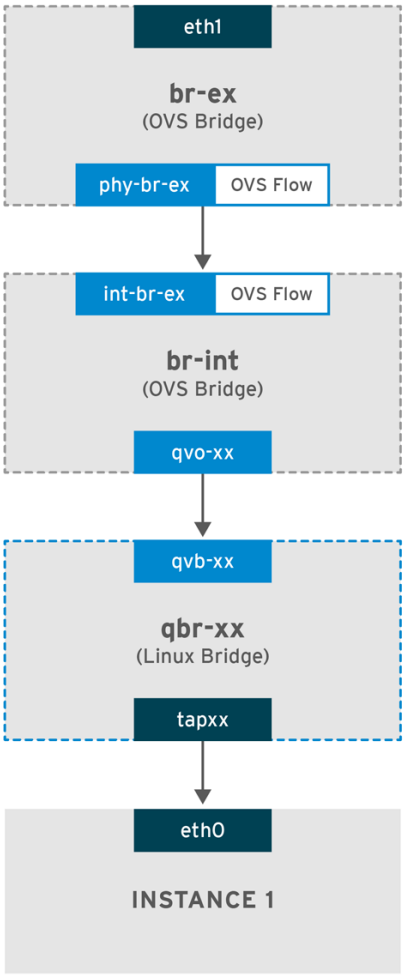
```
# ovs-ofctl dump-flows br-ex
```

```
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，则物理接口将标签添加到数据包。

扁平提供商网络中传入流量流

本节包含有关来自外部网络的传入流量流的信息，直到到达实例的接口。



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**。查看包含 **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
```

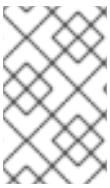
```
SET_NW_SRC SET_NW_DST SET_NW_TOS SET_TP_SRC SET_TP_DST ENQUEUE
 15(int-br-ex): addr:12:4e:44:a9:50:f4
     config:     0
     state:      0
     speed: 0 Mbps now, 0 Mbps max
```

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 流规则是添加内部 VLAN 标签 **5** 的数据包。查看 **actions=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)且无 VLAN 标签(vlan_tci=0x0000)的数据包：此规则将 VLAN 标签 5 添加到数据包(**actions=mod_vlan_vid:5,NORMAL**)，并将它转发到 **qvoxxx**。
3. 在剥离 VLAN 标签后，**qvoxxx** 接受数据包并将其转发到 **qvbxx**。
4. 然后，数据包到达实例。



注意

VLAN 标签 5 是一个 VLAN 示例，用于在具有扁平提供商网络的测试 Compute 节点上使用；这个值由 **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，且在两个单独的 Compute 节点上同一网络可能会有所不同。

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的工作方式"中提供的输出提供了足够的调试信息，以排除扁平提供商网络的问题。以下步骤包含有关故障排除流程的更多信息。

流程

1. 检查 **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 **flat** 类型：

示例

在本例中，会查询网络 **provider-flat** 的详细信息：

```
$ 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**，并检查流是否剥离传出数据包的内部 VLAN ID，并为传入的数据包添加 VLAN ID。当您生成实例到特定 Compute 节点上的此网络时，会首先添加此流。

   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，请检查 **bridge_mapping** 设置。

   b. 最后，检查 **ifcfg-br-ex** 和 **ifcfg-ethx** 配置。确保 **ethX** 作为端口添加到 **br-ex** 中，并且 **ifcfg-br-ex** 和 **ifcfg-ethx** 在 **ip** 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，并将它们发送到 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是什么？”
- 配置网桥映射

4.6. 配置 VLAN 提供商网络

当您将一个 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 提供商网络可以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**。
- 您的网络节点和 Compute 节点使用物理接口连接到物理网络。
  本例使用连接到物理网络 **physnet1** 的网络节点和 Compute 节点，使用物理接口 **eth1**。
- 这些接口连接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，创建一个自定义 YAML 环境文件。

   示例

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

   提示

   Red Hat OpenStack Platform Orchestration 服务(heat)使 用一组名为 *template* 的计划来安装和配置您的环境。您可以使用一个*自定义环境文件*来自定义 overcloud 的各个方面，它是为编配模板提供自定义的特殊模板类型。

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. 创建一个外部网络网桥 (*br-ex*)，并将端口 (*eth*1) 与它关联。
   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'
   ```

5. 运行 **openstack overcloud deploy** 命令，并包含核心模板和环境文件，包括这个新的环境文件。

重要

但是，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更为优先。

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. 创建外部网络作为类型 **vlan**，并将它们与配置的 **physical_network** 关联。
运行以下示例命令创建两个网络：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
```

2. 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 dashboard 来创建这些子网。确保您从网络管理员收到的外部子网详情已与每个 VLAN 正确关联。

在此示例中，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

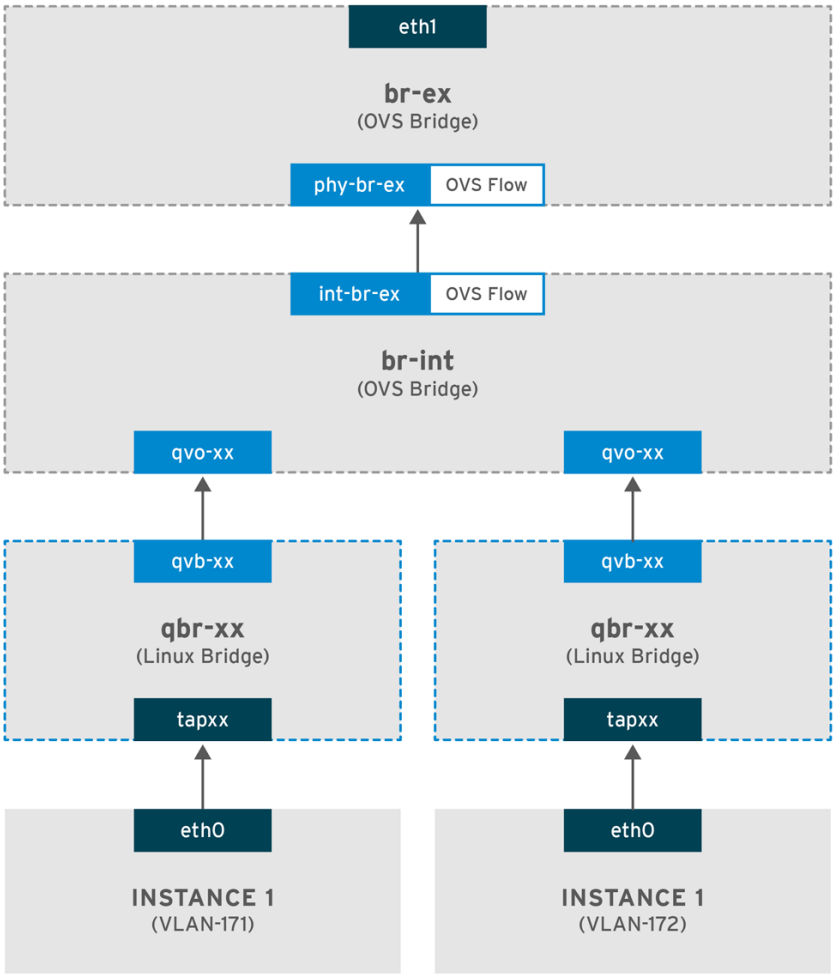
- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 自定义网络接口模板
- *自定义 Red Hat OpenStack Platform 部署* 指南中的环境文件 https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy the-overcloud-with-the-orchestration-service#con_environment-files_understanding-heat-templates
- 在自定义 *Red Hat OpenStack Platform 部署* 指南中的 overcloud 创建中包括环境文件
- *命令行界面参考*中的 network create
- *命令行接口参考*中的 subnet create

4.7. VLAN 提供商网络数据包流的工作方式？

本节介绍了流量如何流进具有 VLAN 供应商网络配置的实例。

VLAN 提供商网络中的传出流量流

下图显示了离开实例的流量的数据包流，并直接到达 VLAN 提供程序外部网络。本例使用连接到两个 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，并将实例生成到 Compute 节点后，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. *qbr-xx* 使用 veth 对 *qvbxx* <→ *qvoxxx* 连接到 *br-int*。

3. *qvbxx* 连接到 linux 网桥 *qbr-xx*，*qvoxx* 连接到 Open vSwitch 网桥 *br-int*。

Linux 网桥上的 *qbr-xx* 配置示例。

这个示例具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"
```

- **qvoxx** 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 上的补丁对等点配置示例：

```
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
```

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

以下命令的输出显示 *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，其 VLAN 标签为 2 (**dl_vlan=2**)。Open vSwitch 将 VLAN 标签替换为 171 (**actions=mod_vlan_vid:171,NORMAL**)，并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，其 VLAN 标签为 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.流使用集成网桥 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。

以下命令的输出显示了 *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
```

以下命令的输出显示了 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
```

传入的流示例

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*。
- 数据包与流的条件匹配(**in_port=18,dl_vlan=172**)。
- 流操作(**actions=mod_vlan_vid:3,NORMAL**)将 VLAN 标签 172 替换为内部 VLAN 标签 3，并将数据包转发到实例具有普通的第 2 层处理。

其他资源

- 第 4.4 节 “扁平提供商网络数据包流的工作方式是什么？”

4.8. 对 VLAN 提供商网络上的实例物理网络连接进行故障排除

在对 VLAN 提供商网络中的连接进行故障排除时，请参阅"VLAN 提供程序网络数据包流工作" 中描述的数据包流。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是否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

   在本例中，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。

   如果这没有在重启服务后创建，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，请将外部 VLAN ID 映射到内部 VLAN ID。
   当您首次生成实例到此网络时，neutron OVS 代理会添加此流。

   c. 如果在生成实例后没有创建此流，请确保网络已创建为 **vlan**，是 **外部**，并且 **physical_network** 名称是正确的。另外，再次检查 **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
   ```

   在本例中，**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

在本示例输出中，**eth1** 已添加为端口，并且内核配置为将所有数据包从接口移动到 OVS 网桥 **br-ex**。这由条目 **master ovs-system** 演示。

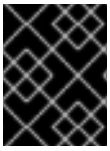
```
5: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq master ovs-system state UP qlen 1000
```

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. 为 ML2/OVS 部署中的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填充多播数据包，您必须启用多播 snooping。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，您可以通过在 YAML 格式的环境文件中声明 RHOSP Orchestration (heat) **NeutronEnableIgmpSnooping** 参数来实现此目的。



重要

在将其应用到生产环境之前，您应该全面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网络行为错误。

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，物理路由器必须发送提供商网络上的 IGMP 查询数据包，以便从多播组成员请求常规 IGMP 报告，以维护 OVS 中的侦听缓存（以及物理网络）。
- RHOSP 网络服务安全组规则必须就位，以允许到虚拟机实例（或禁用端口安全性）的入站 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，创建一个自定义 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，它是为 heat 模板提供自定义的特殊模板类型。

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

```
parameter_defaults:
  NeutronEnableIgmpSnooping: true
  ...
```

重要

确保在冒号 (:) 和 **true** 间添加一个空格字符。

3. 运行 **openstack overcloud deploy** 命令，并包含核心 heat 模板、环境文件以及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更为优先。

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

- 验证是否启用了多播 snooping。

示例

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

输出示例

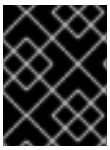
```
...
mcast_snooping_enable: true
...
other_config: {mac-table-size="50000", mcast-snooping-disable-flood-unregistered=True}
...
```

其他资源

- *Component, Plug-In, and Driver Support in Red Hat OpenStack Platform* 中的 Neutron
- *自定义 Red Hat OpenStack Platform 部署* 指南中的环境文件 https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy the-overcloud-with-the-orchestration-service#con_environment-files_understanding-heat-templates
- 在自定义 *Red Hat OpenStack Platform 部署* 指南中的 overcloud 创建中包括环境文件
- *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，以允许多播流量到达多播组中的虚拟机(VM)实例。要防止多播流量填充，启用 IGMP snooping。



重要

在将任何多播配置应用到生产环境之前，请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网络行为错误。

先决条件

- 使用 ML2/OVN 机制驱动程序的 OpenStack 部署。

流程

1. 配置安全性，以允许多播到适当的虚拟机实例。例如，创建一对安全组规则，以允许来自 IGMP querier 的 IGMP 流量进入和退出虚拟机实例，以及允许多播流量的第三个规则。

   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
   ```

   另一种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，一些操作员选择有选择地禁用网络上的端口安全性。如果您选择禁用端口安全性，请考虑和规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 **NeutronEnableIgmpSnooping: True**。例如，将以下行添加到 ovn-extras.yaml 中。

   示例

```
parameter_defaults:
      NeutronEnableIgmpSnooping: True
```

3. 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>** 替换为属于您现有部署的环境文件列表。

验证

1. 验证是否启用了多播 snooping。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"}
...
```

网络服务(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_snoop 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

- *Component, Plug-In, and Driver Support in Red Hat OpenStack Platform* 中的 Neutron
- *自定义 Red Hat OpenStack Platform 部署* 指南中的环境文件 https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy

the-overcloud-with-the-orchestration-service#con_environment-files_understanding-heat-templates

- 在自定义 *Red Hat OpenStack Platform 部署* 指南中的 overcloud 创建中包括环境文件

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外部网络，并且将外部路由器配置为其默认网关。没有使用 OpenStack Networking (neutron)路由器。这意味着 *neutron* 路由器无法用于将来自实例的元数据请求代理到 *nova-metadata* 服务器，这可能会导致运行 *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此问题。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 地址直接绑定在实例内部，并允许实例直接与外部网络通信。

第 5 章 管理浮动 IP 地址

除了具有私有的固定 IP 地址外，虚拟机实例还可以具有公共或浮动 IP 地址来与其他网络通信。本节中的信息描述了如何使用 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 地址将网络流量定向到 OpenStack 实例。首先，您必须定义一个可有效路由的外部 IP 地址池，然后您可以动态分配给实例。OpenStack 网络将所有目的地为该浮动 IP 的流量路由到您与浮动 IP 关联的实例。

注意

OpenStack 网络从 CIDR 格式的同一 IP 范围分配浮动 IP 地址到所有项目（租户）。因此，所有项目都可以使用每个浮动 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

- 在创建外部子网时，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
```

如果子网主机仅托管浮动 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。（请参见下面的链接。）

其他资源

- *命令行接口参考*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** 命令为实例分配浮动 IP 地址。

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 已与您的实例关联。

  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

- *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

从 **Admin** 视图在控制面板中创建网络时，管理员可使用高级网络选项。使用这些选项来指定项目，并定义要使用的网络类型。

流程

1. 在控制面板中，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. 检查 **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 ID 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** - 使用一个网络覆盖，跨多个节点进行实例间的私有通信。出口覆盖的流量必须被路由。
   - **VXLAN** - 与 GRE 类似，并使用网络覆盖来跨越多个节点进行实例之间的私有通信。出口覆盖的流量必须被路由。
4. 点**创建网络**。
   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 地址池动态分配浮动 IP 地址到虚拟机实例。

先决条件

- 可路由外部 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 已与您的实例关联。

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，请确保属性 **handle_internal_only_routers** 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。

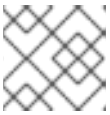
流程

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

但是，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更为优先。

```
$ 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

- *自定义 Red Hat OpenStack Platform 部署* 指南中的环境文件 https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy the-overcloud-with-the-orchestration-service#con_environment-files_understanding-heat-templates
- 在自定义 *Red Hat OpenStack Platform 部署* 指南中的 overcloud 创建中包括环境文件

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 &**lt;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 **&lt**;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和从实例连接。

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介。

因此，所有遍历 **eth0** 的流量都使用配置的 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**，并使用适合您的站点的网络值更新以下参数：
   - 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，并将它们提供给物理网络。

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

此流程使用网络拓扑功能。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. 在控制面板中，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，然后点**添加接口**。
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 指示流量按预期路由。
4. 点 **Add interface**。
   Network Topology 图表会自动更新，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将流量定向到子网，您可以删除接口。

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. 点托管您要删除的接口的路由器的名称。
3. 选择接口类型(**Internal Interface**)，然后点 **Delete Interfaces**。

第 6 章 监控和故障排除网络

在 Red Hat OpenStack Platform 中监控并排除网络连接的诊断流程与物理网络的诊断流程类似。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，而不是完全独立的环境。ML2/OVS 网络和默认的 ML2/OVN 网络故障排除之间存在一些区别。

6.1. 基本 PING 测试

ping 命令是用于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示器，但可能无法完全排除所有连接问题，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。

在从遇到网络问题的计算机运行时，ping 测试最有用，因此如果计算机似乎完全离线，则可能需要通过 VNC 管理控制台连接到命令行。

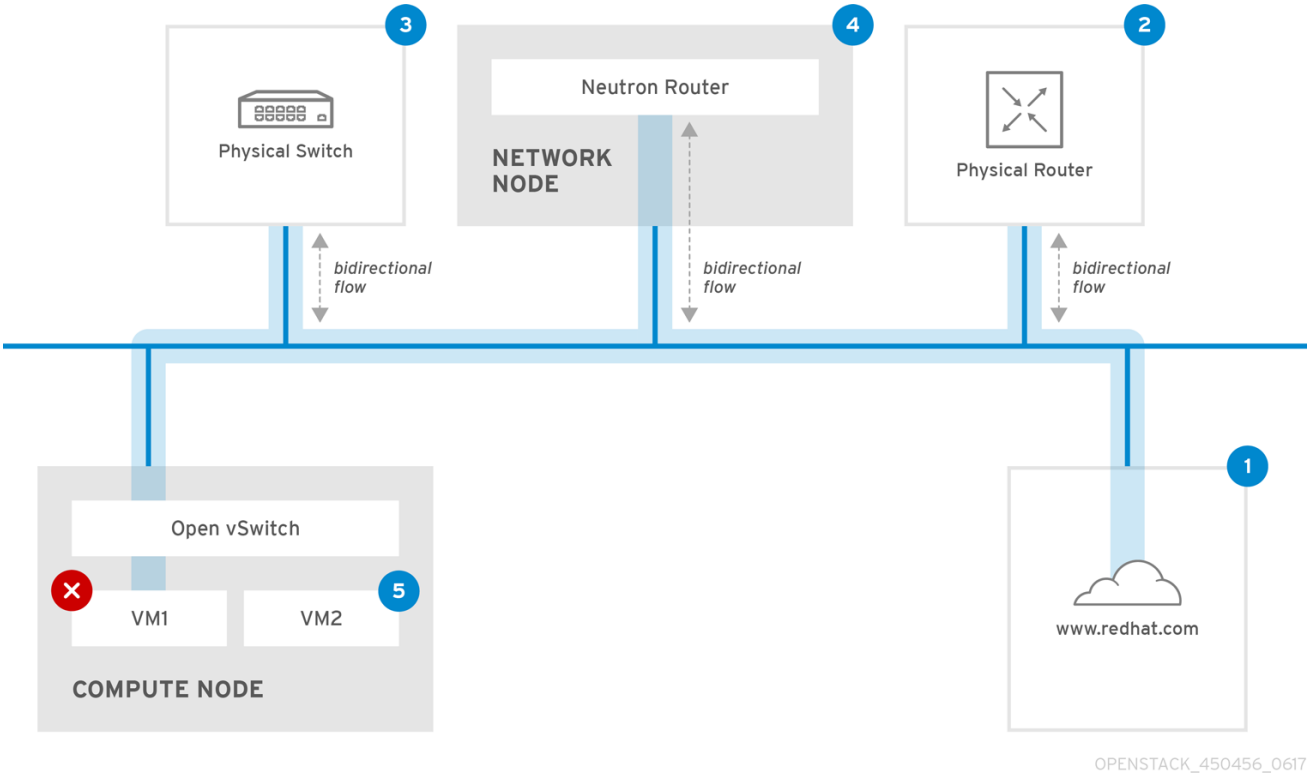
例如，以下 ping test 命令会验证网络基础架构的多个层，以便成功；名称解析、IP 路由和网络切换都必须正确正常工作：

```
$ 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，之后显示结果摘要。数据包丢失的百分比为零代表连接稳定且不超时。

```
--- 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
```

根据您测试的目标，ping 测试结果可能会非常显示。例如，在下图中，VM1 遇到某种形式的连接问题。可能的目的地以蓝色编号，显示成功或失败结果的结论：



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*：此测试表示机器和互联网之间的所有网络点都正确运行。这包括虚拟和物理网络基础架构。

   - *失败*：ping 测试到距离互联网位置的方法可能会失败。如果您的网络中的其他机器能够成功 ping 互联网，证明互联网连接可以正常工作，且此问题可能会在本地计算机的配置中。

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*：对物理路由器的 Ping 测试可以确定本地网络和底层交换机是否正常运行。这些数据包不会遍历路由器，因此它们不会证明默认网关上是否存在路由问题。

   - *失败*：这表示 VM1 和默认网关间的问题。路由器/交换机可能会停机，或者您可能使用错误的默认网关。将配置与您知道在正确运行的另一个服务器上的配置进行比较。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，并检查 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，这表示此段正常工作。

   - *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 节点上尝试 ping 同一子网上的虚拟机。

   - *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检查 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与路由器连接的所有端口的清单，并确定端口状态(**DOWN** 或 **ACTIVE**)。

流程

1. 要查看附加到名为 **r1** 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": "a592fdba-
babd-48e0-96e8-2dd9117614d3", "ip_address": "192.168.200.1"} |
| c45e998d-98a1-4b23-bb41-5d24797a12a4 |      | fa:16:3e:ee:6a:f7 | {"subnet_id": "43f8f625-
c773-4f18-a691-fd4ebfb3be54", "ip_address": "172.24.4.225"}  |
+--------------------------------------+------+-------------------+-------------------------------------------
-----------------------------------------+
```

2. 要查看每个端口的详细信息，请运行以下命令：包含您要查看的端口的端口 ID。结果包括端口状态，如下例所示，如 **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:vnic_type     | normal                                                                              |
| device_id             | 49c6ebdc-0e62-49ad-a9ca-58cea464472f                                                |
| device_owner          | network:router_interface                                                            |
| extra_dhcp_opts       |                                                                                     |
| fixed_ips             | {"subnet_id": "a592fdba-babd-48e0-96e8-2dd9117614d3", "ip_address":
"192.168.200.1"} |
| id                    | b58d26f0-cc03-43c1-ab23-ccdb1018252a                                                |
| mac_address           | fa:16:3e:94:a7:df                                                                   |
```

```
| name                  |                                                                                                  |
| network_id            | 63c24160-47ac-4140-903d-8f9a670b0ca4
|
| security_groups       |                                                                                                  |
| status                | ACTIVE                                                                                           |
| tenant_id             | d588d1112e0f496fb6cac22f9be45d49                                                                 |
+-----------------------+--------------------------------------------------------------------------------------------------
+
```

3. 为每个端口执行第 2 步以确定其状态。

6.3. 与 VLAN 提供商网络连接的故障排除

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。
   考虑这个示例，其中使用以下命令创建了一个网络：

```
# 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. 确认您具有关联的 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。（请参见第 1 步中的示例中的 **--provider:segmentation_id=120**。）

   b. 使用命令 **ovs-ofctl dump-flows <bridge-name>** 命令确认网桥接口上的 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故障排除，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 VLAN 范围等网络配置值。

流程

1. 使用 **openstack network agent list** 命令来验证 RHOSP 网络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. 检查 **/var/log/containers/neutron/openvswitch-agent.log**。查找确认创建进程使用 **ovs-ofctl** 命令配置 VLAN 中继。

3. 验证 **/etc/neutron/l3_agent.ini** 文件中的 **external_network_bridge**。如果 **external_network_bridge** 参数中有一个硬编码的值，则无法使用带有 L3-agent 的提供商网络，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 agent configuration file bridge mappings**，确认映射到 **phy-eno1** 的网桥存在，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 凭证登录 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 tripleo-admin@compute0.ctlplane
```

5. 运行 **ip netns list** 命令以查看 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** 命令来 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，以便项目可以配置网络，而无需相互干扰。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，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中，这可让您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
```

输出中包含一个与 **web-servers** 网络 ID 匹配的命名空间。

在此输出中，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。
在本例中，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. 在命名空间内执行高级 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，而不是虚拟机实例。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**。该容器在 Controller 节点和 Compute 节点上运行。为简化您对命令的访问，请创建并提供定义别名的脚本。

先决条件

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

流程

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

   示例

   ```
   $ ssh tripleo-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

   在本例中，**ovn-sbctl**、**ovn-nbctl** 和 **ovn-trace** 命令已添加到别名文件中：

   ```
   REMOTE_IP=$(sudo ovs-vsctl get open . external_ids:ovn-remote)
   NBDB=$(echo $REMOTE_IP | sed 's/6642/6641/g')
   SBDB=$REMOTE_IP
   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. Source 脚本文件。

   示例

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

2. 运行命令以确认您的脚本文件正常工作。

   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 使用逻辑流表，它们是具有优先级、匹配和操作的流表。这些逻辑流分布到每个 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 数据库命令创建一个别名文件。
  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 Controller 主机。

   示例

   ```
   $ ssh tripleo-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=(outport =
"_MC_flood"; output;)
  table=13(ls_in_l2_lkup      ), priority=50   , match=(eth.dst == 00:00:00:00:00:01), action=
(outport = "sw0-port1"; output;)
  table=13(ls_in_l2_lkup      ), priority=50   , match=(eth.dst == 00:00:00:00:00:02), action=
(outport = "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=(outport == "sw0-port1" && eth.dst ==
{00:00:00:00:00:01}), action=(output;)
  table=8 (ls_out_port_sec_l2 ), priority=50   , match=(outport == "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 端口是位于网络中某个位置的逻辑实体，而不是单个交换机上的物理端口。
- OVN 为管道中的每个表提供一个名称，编号除外。name 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将超出 OpenFlow 的逻辑流。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 跟踪。
ovn-trace 命令可以模拟数据包如何通过 OVN 逻辑流传输，或者帮助您确定丢弃数据包的原因。使用以下参数提供 **ovn-trace** 命令：

DATAPATH
: 启动模拟数据包的逻辑交换机或逻辑路由器。

MICROFLOW
: 模拟数据包，采用 **ovn-sb** 数据库使用的语法。

示例

这个示例在模拟数据包中显示 **--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

如需更多详情，这个同一模拟数据包的 **--summary** 输出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。

- **outport** 变量设为 **sw0-port2**，这表示此数据包的预期目的地为 **sw0-port2**。
- 数据包从 ingress 管道输出，并导向 **outport** 变量设置为 **sw0-port2** 的 **sw0** 的 egress 管道。
- 输出操作在出口管道中执行，它会将数据包输出到 **outport** 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") {
    outport = "sw0-port2";
    output;
    egress(dp="sw0", inport="sw0-port1", outport="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) 网络的逻辑交换机上的 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 Controller 主机。

   示例

   ```
   $ ssh tripleo-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，它类似于以下输出：

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. 监控 OVN 数据库状态

您可以使用 **ovs-appctl** 命令监控 OVN 数据库服务器之间的连接。

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。
   单一 Controller 主机上的服务器监控提供了验证基本集群健康状况和诊断许多类型问题所需的信息。对于非常全面的分析，请在所有 Controller 上执行这个步骤。

   示例

   ```
   $ ssh tripleo-admin@compute-0
   ```

2. 运行 **ovs-appctl** 命令。

   示例：北向数据库

   ```
   $ ovs-appctl -t /var/lib/openvswitch/ovn/ovnnb_db.ctl cluster/status OVN_Northbound
   ```

   示例：南向数据库

   ```
   ovs-appctl -t /var/lib/openvswitch/ovn/ovnsb_db.ctl cluster/status OVN_Southbound
   ```

3. 检查输出，它类似于以下输出：

   输出示例：南向数据库

   此示例输出是在服务器 1114 上生成的，此时是后续。

   ■

```
1114
Name: OVN_Southbound
Cluster ID: 017a (017add73-58f1-4fcd-ae35-bacc0f07ce57)
Server ID: 1114 (1114865d-4f42-443a-b758-d4431fc35748)
Address: tcp:[fd00:fd00:fd00:2000::4a]:6644
Status: cluster member
Role: follower
Term: 90
Leader: ca6e
Vote: ca6e

Last Election started 27881511 ms ago, reason: leadership_transfer
Last Election won: 27881503 ms ago
Election timer: 16000
Log: [51470, 51737]
Entries not yet committed: 0
Entries not yet applied: 0
Connections: ->ca6e ->0f90 <-ca6e <-0f90
Disconnections: 0
Servers:
    1114 (1114 at tcp:[fd00:fd00:fd00:2000::4a]:6644) (self)
    ca6e (ca6e at tcp:[fd00:fd00:fd00:2000::18f]:6644) last msg 5141 ms ago
    0f90 (0f90 at tcp:[fd00:fd00:fd00:2000::2e0]:6644) last msg 22106129 ms ago
```

诊断显示示例输出

右点箭头(→)代表从这个服务器到另一个 A left-pointing 箭头(netobserv)的出站连接代表从其他服务器到这个服务器的入站连接。

所有服务器都处于活动状态且已连接

connections: ->ca6e ->0f90 <-ca6e <-0f90
此三节点集群显示为健康。服务器 1114 具有与其他两台服务器的入站和出站连接，即 ca6e 和 0f90。

服务器与集群断开连接

connections: ->ca6e (->0f90)<-ca6e
服务器 0f90 的传入连接没有列出。传出连接的括号表示出站消息到 0f90 失败。在大多数情况下，连接到集群中的任何服务器会提供足够信息来确定集群是否存在问题。在所有服务器上运行诊断提供了更详细的信息，并可能会检测您无法从单一服务器检测的问题。

集群丢失仲裁

```
Role: candidate
...
Leader: unknown
```

此服务器是候选服务器，领导机未知。

ovsdb-server 在此节点上停机

```
2024-03-27T22:10:28Z|00001|unixctl|WARN|failed to connect to
/var/lib/openvswitch/ovn/ovnsb_db.ctl
ovs-appctl: cannot connect to "/var/lib/openvswitch/ovn/ovnsb_db.ctl" (Connection
```

```
refused)

<exits with non-zero status>
```

在这种情况下，您无法从单一服务器获取您需要的所有信息。例如，您无法确定其他服务器是否正在运行。如果服务器停机，请在另一台服务器上运行 ovs-appctl。

从每个后续者中的最后一个消息到领导的时间（只在领导中更新）

```
Servers:
    1114 (1114 at tcp:[fd00:fd00:fd00:2000::4a]:6644) next_index=51737
match_index=51736 last msg 224 ms ago
    ca6e (ca6e at tcp:[fd00:fd00:fd00:2000::18f]:6644) (self) next_index=51470
match_index=51736
    0f90 (0f90 at tcp:[fd00:fd00:fd00:2000::2e0]:6644) next_index=51737
match_index=51736 last msg 224 ms ago
```

登录集群领导主机并运行 ovs-appctl。请注意，可以随时选择新的领导。

其他资源

- **ovs-appctl --help** 命令

6.12. 验证 ML2/OVN 部署

在 Red Hat OpenStack Platform (RHOSP) 部署中验证 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 specfic 容器是否正在运行。

先决条件

- RHOSP 的新部署，ML2/OVN 作为网络服务 (neutron) 默认机制驱动程序。
- 为 OVN 数据库命令创建一个别名文件。
  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 Controller 主机。

示例

```
$ ssh tripleo-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，输出应列出 OVN 容器：

输出示例

```
container-puppet-ovn_controller
ovn_cluster_north_db_server
ovn_cluster_south_db_server
ovn_cluster_northd
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 容器的所需特权的用户身份登录计算主机。

示例

```
$ ssh tripleo-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，输出应列出 OVN 容器：

输出示例

```
container-puppet-ovn_controller
ovn_metadata_agent
ovn_controller
```

4. 检查日志文件是否有错误消息。

```
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. 查询北向和南向数据库，以检查响应。

```
# 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 Red Hat Solution 中描述的步骤，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
- *命令行接口参考*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

6.13. 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。在不需要额外的调试信息时，将日志记录重新设置为 info 模式以使用较少的磁盘空间。

先决条件

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

流程

1. 以具有访问 OVN 容器所需的权限的用户身份，登录 Controller 或 Compute 节点。

   示例

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

2. 设置 ML2/OVN 日志记录模式。

   Debug 日志模式

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

   Info 日志模式

   ```
   $ 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 debug 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 6.15 节 “ML2/OVN 日志文件”

6.14. 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 17.1 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，它会注册另一个包含另一个 encap 条目的机箱条目。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. 如果您还没有这么做，请为稍后您将在此流程中使用的必要 OVN 数据库命令创建别名。如需更多信息，请参阅为 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 Controller 主机。

示例

```
$ ssh tripleo-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，以确认删除了机箱：

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

6.15. 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/openvswitch...
Controller, Compute, Networking	OVS 北向数据库服务器	**.../ovn-controller.log**
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。某些交换机包括特定于供应商的配置。

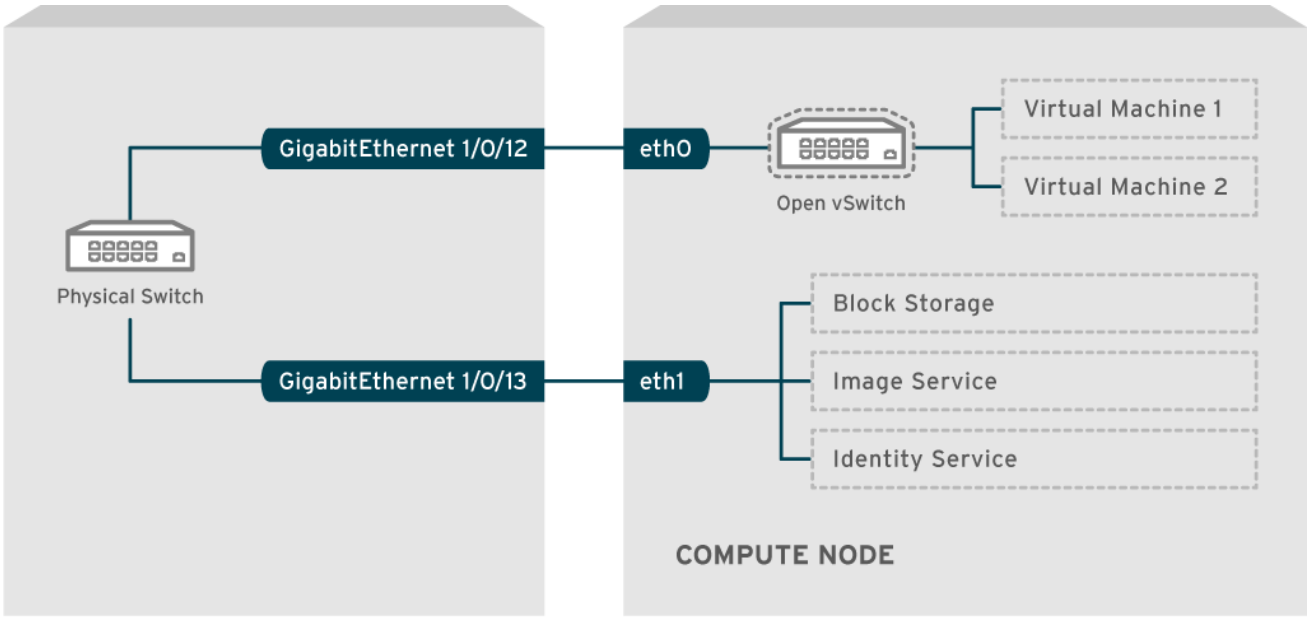
7.1. 规划您的物理网络环境

OpenStack 节点中的物理网络适配器执行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 的流量类型会影响您必须在物理交换机上配置端口。

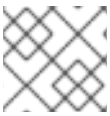
首先，您必须决定要处理哪些类型的物理 NIC oFn 您的 Compute 节点。然后，当 NIC 电缆到物理交换机端口时，您必须配置交换机端口以允许中继或常规流量。

例如，下图描述了带有两个 NIC (eth0 和 eth1) 的 Compute 节点。每个 NIC 都连接到物理交换机上的千兆以太网端口，eth0 承载实例流量，以及为 OpenStack 服务提供连接的 eth1 ：

图 7.1. 网络布局示例



OPENSTACK_377160_1115



注意

此图不包括容错所需的任何其他冗余 NIC。

其他资源

- *自定义 Red Hat OpenStack Platform 部署指南中的网络接口绑定*。https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy networks-for-the-rhosp-environment#assembly_network-interface-bonding

7.2. 配置 CISCO CATALYST 交换机

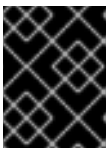
7.2.1. 关于中继端口

通过 OpenStack 网络，您可以将实例连接到物理网络上已存在的 VLAN。术语 *trunk* 用于描述允许多个 VLAN 遍历同一端口的端口。通过使用这些端口，VLAN 可以跨越多个交换机，包括虚拟交换机。例如，在物理网络中标记为 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 的流量传递到您的实例。
  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。您必须更改此示例中的值，使其与环境中的值匹配。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：

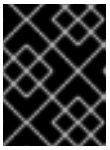
字段	描述
interface GigabitEthernet1/0/12	X 节点连接的交换机端口。确保将 **GigabitEthernet1/0/12** 值替换为您的环境的正确端口值。使用 show interface 命令查看端口列表。
描述 Trunk 到 Compute 节点	一个唯一的具有描述性的值，可用于识别此接口。
生成树端口快速中继	如果您的环境使用 STP，则将此值设置为指示此端口用于中继流量的 Port Fast。
switchport 中继封装点1q	启用 802.1q 中继标准（而不是 ISL）。此值因交换机支持的配置而异。
switchport 模式中继	将此端口配置为中继端口，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。
switchport 中继原生 vlan 2	设置原生 VLAN，以指示发送未标记（非 VLAN）流量的交换机。
switchport trunk allowed vlan 2,110,111	定义通过中继允许哪些 VLAN。

7.2.3. 关于访问端口

不是 Compute 节点上的所有 NIC 都传输实例流量，因此您不需要配置所有 NIC 以允许多个 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，通常需要配置比中继端口更简单。

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 交换机上）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。您必须更改此示例中的值，使其与环境中的值匹配。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
interface GigabitEthernet1/0/13	X 节点连接的交换机端口。确保将 **GigabitEthernet1/0/12** 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。
Compute 节点的描述访问端口	一个唯一的具有描述性的值，可用于识别此接口。
switchport 模式访问	将此端口配置为访问端口，而不是中继端口。
switchport 访问 vlan 200	配置端口，以允许 VLAN 200 上的流量。您必须使用此 VLAN 中的 IP 地址配置 Compute 节点。
acrossing-tree portfast	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，从而在初始连接期间允许更快的端口握手（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 捆绑在一起，以形成单个逻辑频道。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动态绑定。您必须在两个物理端配置 LACP：物理 NIC 和物理交换机端口上。

其他资源

- *Installing and managing Red Hat OpenStack Platform with director* 指南中的Network Interface Bonding。

7.2.6. 在物理 NIC 上配置 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

- *自定义 Red Hat OpenStack Platform 部署指南中的网络接口绑定*。https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy networks-for-the-rhosp-environment#assembly_network-interface-bonding

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. 在 Compute 节点上物理将 Compute 节点上的 NIC 连接到交换机（例如，端口 12 和 13）。
2. 创建 LACP 端口频道：

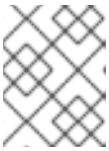
```
interface port-channel1
  switchport access vlan 100
  switchport mode access
  spanning-tree guard root
```

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
```

4. 检查您的新端口频道。生成的输出列出了新 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)
```

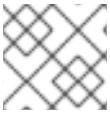


注意

请注意，需要将 running-config 复制到 startup-config 以使变化生效：**copy running-config startup-config**。

7.2.8. 关于 MTU 设置

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



注意

您必须更改流量预期通过的所有跃点（包括任何虚拟交换机）的端到端的 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。

1. 查看当前的 MTU 设置：

```
sw01# show system mtu

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

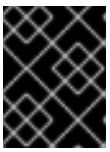
2. 在 3750 交换机上更改 MTU 设置，而不是针对各个接口更改。运行以下命令，将交换机配置为使用 9000 字节的巨型帧。如果您的交换机支持此功能，您可能希望为单个接口配置 MTU 设置。

```
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

请注意，需要将 running-config 复制到 startup-config 以保持变化：**copy running-config startup-config**。

3. 重新加载交换机以应用更改。



重要

重新加载交换机会导致对依赖于交换机的任何设备造成网络中断。因此，仅在计划的维护期间重新加载交换机。

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

4. 重新加载交换机后，确认新的巨型 MTU 大小。
具体输出可能会因交换机模型而异。例如，**系统 MTU** 可能适用于非Gigabit 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、端口详细信息和可用的 VLAN。与 Cisco Discovery Protocol (CDP) 类似，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

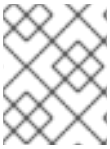
流程

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

请注意，需要将 running-config 复制到 startup-config 以保持变化：**copy running-config startup-config**。

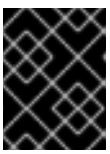
7.3. 配置 CISCO NEXUS 交换机

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上已存在的 VLAN。术语 *trunk* 用于描述允许多个 VLAN 遍历同一端口的端口。通过使用这些端口，VLAN 可以跨越多个交换机，包括虚拟交换机。例如，在物理网络中标记为 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 的流量传递到您的实例。
  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 **Ethernet1/12** 接口。



重要

这些值是示例。您必须更改此示例中的值，使其与环境中的值匹配。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 都传输实例流量，因此您不需要配置所有 NIC 以允许多个 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，通常需要配置比中继端口更简单。

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 交换机上）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 **Ethernet1/13** 接口。

重要

这些值是示例。您必须更改此示例中的值，使其与环境中的值匹配。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 捆绑在一起，以形成单个逻辑频道。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动态绑定。您必须在两个物理端配置 LACP：物理 NIC 和物理交换机端口上。

其他资源

- *Installing and managing Red Hat OpenStack Platform with director* 指南中的Network Interface Bonding。

7.3.6. 在物理 NIC 上配置 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

- *自定义 Red Hat OpenStack Platform 部署指南中的网络接口绑定*。https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy networks-for-the-rhosp-environment#assembly_network-interface-bonding

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

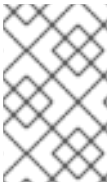
1. 物理地将 Compute 节点 NIC 连接到交换机（例如，端口 12 和 13）。
2. 确认 LACP 已启用：

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

3. 将端口 1/12 和 1/13 配置为访问端口，以及作为频道组的成员。
   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，因此您可能需要单独配置访问端口。这些接口通常包含 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** 和 **no lacp suspend-individual** 来调出端口并引导服务器。如需更多信息，请参阅 Cisco 交换机文档。

7.3.8. 关于 MTU 设置

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

注意

您必须更改流量预期通过的所有跃点（包括任何虚拟交换机）的端到端的 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 系列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

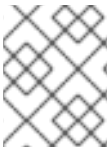
ironic-python-agent 服务侦听来自连接的交换机的 LLDP 数据包。收集的信息可以包含交换机名称、端口详细信息和可用的 VLAN。与 Cisco Discovery Protocol (CDP) 类似，LLDP 有助于在 director 内省过程中发现物理硬件。

7.3.11. 为 Cisco Nexus 7000 交换机配置 LLDP

流程

- 您可以为 Cisco Nexus 7000-series 交换机上的单个接口启用 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

请注意，需要将 running-config 复制到 startup-config 以保持变化：**copy running-config startup-config**。

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上已存在的 VLAN。术语 *trunk* 用于描述允许多个 VLAN 遍历同一端口的端口。通过使用这些端口，VLAN 可以跨越多个交换机，包括虚拟交换机。例如，在物理网络中标记为 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，以切换物理交换机上的端口 swp1 和 swp2。

重要

这些值是示例。您必须更改此示例中的值，使其与环境中的值匹配。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 的流量传递到您的实例。

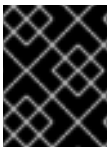
```
auto bridge
iface bridge
  bridge-vlan-aware yes
  bridge-ports glob swp1-2
  bridge-vids 100 200
```

7.4.3. 关于访问端口

不是 Compute 节点上的所有 NIC 都传输实例流量，因此您不需要配置所有 NIC 以允许多个 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，通常需要配置比中继端口更简单。

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 access/trunk 端口。



重要

这些值是示例。您必须更改此示例中的值，使其与环境中的值匹配。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 捆绑在一起，以形成单个逻辑频道。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动态绑定。您必须在两个物理端配置 LACP：物理 NIC 和物理交换机端口上。

其他资源

- *Installing and managing Red Hat OpenStack Platform with director* 指南中的Network Interface Bonding。

7.4.6. 关于 MTU 设置

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

注意

您必须更改流量预期通过的所有跃点（包括任何虚拟交换机）的端到端的 MTU 设置。

其他资源

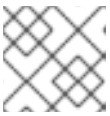
- 配置最大传输单元(MTU)设置

7.4.7. 为 Cumulus Linux 交换机配置 MTU 设置

流程

- 这个示例在 Cumulus Linux 交换机中启用巨型帧。

```
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、端口详细信息和可用的 VLAN。与 Cisco Discovery Protocol (CDP) 类似，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
```

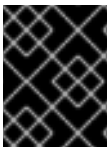
7.5. 配置 EXTREME EXOS 交换机

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上已存在的 VLAN。术语 *trunk* 用于描述允许多个 VLAN 遍历同一端口的端口。通过使用这些端口，VLAN 可以跨越多个交换机，包括虚拟交换机。例如，在物理网络中标记为 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 交换机上配置中继端口

如果使用 X-670 系列交换机，请参考以下示例，允许 VLAN 110 和 111 传递给您的实例。



重要

这些值是示例。您必须更改此示例中的值，使其与环境中的值匹配。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。

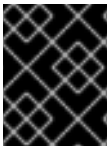
```
#create vlan DATA tag 110
#create vlan MNGT tag 111
#configure vlan DATA add ports 24 tagged
#configure vlan MNGT add ports 24 tagged
```

7.5.3. 关于访问端口

不是 Compute 节点上的所有 NIC 都传输实例流量，因此您不需要配置所有 NIC 以允许多个 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，通常需要配置比中继端口更简单。

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。您必须更改此示例中的值，使其与环境中的值匹配。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

- 在此配置示例中，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 捆绑在一起，以形成单个逻辑频道。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动态绑定。您必须在两个物理端配置 LACP：物理 NIC 和物理交换机端口上。

其他资源

- *Installing and managing Red Hat OpenStack Platform with director* 指南中的Network Interface Bonding。

7.5.6. 在物理 NIC 上配置 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

- *自定义 Red Hat OpenStack Platform 部署指南中的网络接口绑定*。https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy networks-for-the-rhosp-environment#assembly_network-interface-bonding

7.5.7. 在 Extreme Networks EXOS 交换机上配置 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

7.5.8. 关于 MTU 设置

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

注意

您必须更改流量预期通过的所有跃点（包括任何虚拟交换机）的端到端的 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 交换机上配置 MTU 设置

流程

- 运行本示例中的命令，在 Extreme Networks EXOS 交换机上启用巨型帧，并配置对转发 IP 数据包的支持(9000 字节)：

```
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、端口详细信息和可用的 VLAN。与 Cisco Discovery Protocol (CDP) 类似，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

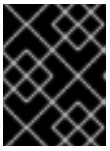
7.6.1. 关于中继端口

通过 OpenStack 网络，您可以将实例连接到物理网络上已存在的 VLAN。术语 *trunk* 用于描述允许多个 VLAN 遍历同一端口的端口。通过使用这些端口，VLAN 可以跨越多个交换机，包括虚拟交换机。例如，在物理网络中标记为 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 的流量传递到您的实例。
  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。您必须更改此示例中的值，使其与环境中的值匹配。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;
                }
            }
    }
```

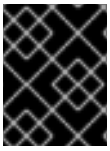
7.6.3. 关于访问端口

不是 Compute 节点上的所有 NIC 都传输实例流量，因此您不需要配置所有 NIC 以允许多个 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，通常需要配置比中继端口更简单。

7.6.4. 为 Juniper EX 系列交换机配置访问端口

此示例为 Juniper EX 系列开关，显示 **ge-1/0/13** 作为 **eth1** 的访问端口。

+



重要

这些值是示例。您必须更改此示例中的值，使其与环境中的值匹配。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 捆绑在一起，以形成单个逻辑频道。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动态绑定。您必须在两个物理端配置 LACP：物理 NIC 和物理交换机端口上。

其他资源

- *Installing and managing Red Hat OpenStack Platform with director* 指南中的Network Interface Bonding。

7.6.6. 在物理 NIC 上配置 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

- *自定义 Red Hat OpenStack Platform 部署指南中的网络接口绑定*。https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy networks-for-the-rhosp-environment#assembly_network-interface-bonding

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)以加入端口聚合 **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，若要从绑定进行 PXE 引导，您必须将其中一个绑定成员配置为 lacp 强制，以确保在内省和第一次引导过程中只有一个绑定成员上线。使用 lacp force-up 配置的绑定成员必须是在 *instackenv.json* (ironic 已知的 MAC 地址配置)中 MAC 地址的相同绑定成员。

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:

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

6. 检查您的新端口频道。生成的输出列出了带有成员端口 **ge-1/0/12** 和 **ge-1/0/13** 的新端口聚合 **ae1** ：

```
> 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

记住通过运行 **commit** 命令应用您的更改。

7.6.8. 关于 MTU 设置

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

注意

您必须更改流量预期通过的所有跃点（包括任何虚拟交换机）的端到端的 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 交换机中启用巨型帧。

注意

根据您使用 Juniper 还是 Cisco 设备，MTU 值的计算方式不同。例如，对于 Cisco，Juniper 上的 **9216** 将等于 **9202**。额外的字节用于 L2 标头，其中 Cisco 会自动将此设置添加到指定的 MTU 值中，但可用的 MTU 将小于使用 Juniper 时指定的 14 字节。为了支持 VLAN 上的 MTU **9000**，必须在 Juniper 上配置 **9014** 的 MTU。

流程

1. 对于 Juniper EX 系列交换机，为各个接口设置 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

记住通过运行 **commit** 命令保存您的更改。

2. 如果使用 LACP 聚合，则需要在其中设置 MTU 大小，而不是在成员 NIC 中设置 MTU。例如，此设置配置 ae1 聚合的 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、端口详细信息和可用的 VLAN。与 Cisco Discovery Protocol (CDP) 类似，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

您可以为所有接口或只为单个接口启用 LLDP。

流程

- 在 Juniper EX 4200 交换机上，使用以下太全局启用 LLDP：

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

- 使用以下内容为单一接口 **ge-1/0/14** 启用 LLDP：

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

注意

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

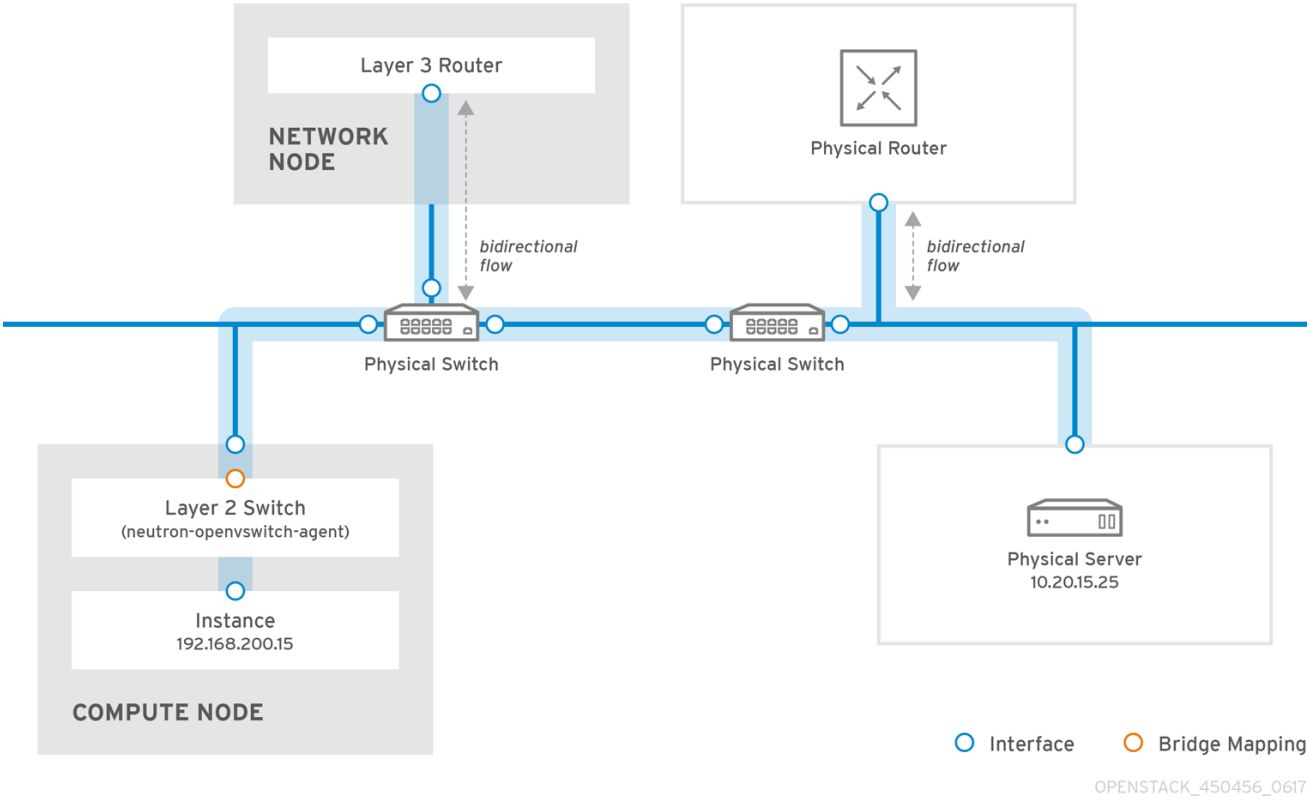
OpenStack 网络可以计算您可以安全应用到实例的最大可能最大传输单元(MTU)大小。MTU 值指定单个网络数据包可以传输的最大数据量；这个数字是变量，具体取决于应用程序最合适的大小。例如，NFS 共享可能需要不同的 MTU 大小与 IANA 应用程序的不同 MTU 大小。

注意

您可以使用 **openstack network show <network_name&** gt; 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，它存在于某些实施中。您可以向 DHCPv4 客户端公告给 DHCPv4 客户端（如果实例支持）以及通过路由器公告(RA)数据包的 IPv6 客户端。要发送路由器公告，网络必须附加到路由器。

您必须从端到端一致配置 MTU 设置。这意味着 MTU 设置必须在数据包通过时相同，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，下图中的圆圈表示必须为实例和物理服务器之间的流量调整 MTU 值的不同点。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* 传输到物理服务器 *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，最常见的随机数据包丢失会导致连接丢弃和减慢网络性能。此问题会有问题，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 中设置 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

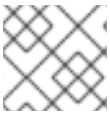
您可以查看计算的 MTU 值，这是实例可以使用的最大可能 MTU 值。使用此计算的 MTU 值来配置网络流量路径涉及的所有接口。

```
# 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 策略应用到单个端口，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继承策略。



注意

网络策略应用中排除了内部网络拥有的端口，如 DHCP 和内部路由器端口。

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，您只能在没有使用策略所分配端口的实例时应用修改。

9.1. QOS 规则

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

最小带宽 (minimum_bandwidth)

对某些类型的流量提供最低带宽限制。如果实施，将尽力为应用该规则的每个端口提供指定带宽不足。

带宽限制(bandwidth_limit)

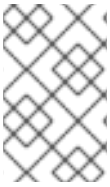
提供网络、端口、浮动 IP (FIP)和路由器网关 IP 的带宽限制。如果实施，超过指定率的流量都会被丢弃。

DSCP marking (dscp_marking)

使用不同服务代码点(DSCP)值标记网络流量。

QoS 策略可以在不同的上下文中实施，包括虚拟机实例放置、浮动 IP 分配和网关 IP 分配。

根据实施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（从实例下载）或两者。



注意

从 Red Hat OpenStack Platform (RHOSP) 17.1 开始，在 ML2/OVN 部署中，您可以为硬件卸载的端口启用最小带宽和带宽限制出口策略。您无法为硬件卸载的端口启用 ingress 策略。如需更多信息，请参阅 第 9.2 节 “为 QoS 策略配置网络服务”。

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]	出口和入口

DSCP 标记	仅限出口	N/A	仅限 Egress[7]

[1] OVS 出口带宽限制在TAP 接口中执行，并且是流量策略，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，在硬件卸载的端口中支持 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：扁平和 VLAN。

[5] 使用 **ip link** 命令应用网络接口中的 QoS 策略，在硬件卸载的端口中支持 OVS 出口最小带宽。

[6] 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	出口和入口	出口和入口	目前，不支持

表 9.3. 强制类型的驱动程序支持的流量方向（仅带宽限制）

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	出口和入口	出口和入口
网关 IP	出口和入口	Egress 和 ingress [1]

[1] RHOSP 17.1 中的技术预览.请参阅 BZ 2088291。

其他资源

- 创建并应用带宽限制 QoS 策略和规则
- 创建并应用保证最小带宽 QoS 策略和规则
- 为出口流量创建并应用 DSCP 标记 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)中的服务质量功能通过 **qos** 服务插件提供。使用 ML2/OVS 和 ML2/OVN 机制驱动程序时，**qos** 默认被加载。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，还必须在对应的代理上加载 **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：

- 对于所有类型的 QoS 策略：
  - 添加 **qos** 服务插件。
  - 为代理添加 **qos** 扩展（仅限OVS 和 SR-IOV）。
- 在 ML2/OVN 部署中，您可以为硬件卸载的端口启用最小带宽和带宽限制出口策略。您无法为硬件卸载的端口启用 ingress 策略。
- 仅使用最小带宽策略调度虚拟机实例的其他任务：
  - 如果与计算服务(nova)使用的名称不同，请指定管理程序名称。
  - 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） Mark **vnic_types** 不支持。
- 在将 ML/OVS 与隧道搭配使用的系统上标记策略的额外任务：
  - 将 **dscp_inherit** 设置为 **true**。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

1. 以 **stack** 用户身份登录 undercloud 主机。
2. 提供 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。如果您没有收到错误，则会加载插件，您不需要执行此主题中的步骤。

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关键字 **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，那么还必须使用 **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. 在 ML2/OVN 部署中，您可以为硬件卸载的端口启用出口最小和最大带宽策略。要做到这一点，将 **OvnHardwareOffloadedQos** 参数设置为 **true** ：

```
parameter_defaults:
  NeutronServicePlugins: "qos"
  OvnHardwareOffloadedQos: true
```

8. 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，并确保列表也包含 **qos** ：

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

b. 如果系统管理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** 指定备选管理程序名称：

- 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** ：

- ML2/OVS

```
parameter_defaults:
  ExtraConfig:

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

- 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，单位为 kbps ：

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

- 仅配置入口带宽，单位为 kbps ：

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

- 配置 egress 和 ingress 带宽，单位为 kbps ：

```
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它们且在放置服务中跟踪多个代理时，要把 **vnic_types** 标记为不被支持，也会在您的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

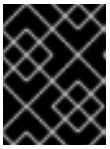
```
parameter_defaults:
  ...
  NeutronSriovVnicTypeBlacklist: direct
```

9. 如果要创建 DSCP 标记策略，并使用 ML2/OVS 和隧道协议(VXLAN 或 GRE)，然后在 **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。

10. 运行部署命令，并包含核心 heat 模板、其他环境文件以及新的自定义环境文件。



重要

但是，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更为优先。

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
```

如果加载了 **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
- *自定义 Red Hat OpenStack Platform 部署* 指南中的环境文件 https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy the-overcloud-with-the-orchestration-service#con_environment-files_understanding-heat-templates
- 在自定义 *Red Hat OpenStack Platform 部署* 指南中的 overcloud 创建中包括环境文件
- 第 9.3.1 节 “使用网络服务后端强制强制实施最小带宽”
- 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)，可在两个不同的上下文中强制实施保证的最小带宽 QoS 规则：网络服务后端强制和资源分配调度强制。

ML2/OVS 或 ML2/SR-IOV 的网络后端会尝试确保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实施、资源分配调度强制或两者都应用 QoS minumum 带宽规则。

下表标识了支持最小带宽 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

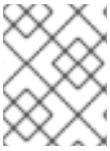
ML2 机制驱动程序	Agent	VNIC 类型
ML2/SR-IOV	sriovnicswitch	direct
ML2/OVS	openvswitch	normal

其他资源

- 第 9.3.1 节 “使用网络服务后端强制强制实施最小带宽”
- 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实施最小带宽

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为端口提供最小带宽。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 网络服务(neutron)必须加载 **qos** 服务插件。（这是默认值。）
- 不要在同一物理接口中使用和不使用带宽保证的端口混合，因为这可能导致拒绝必要的资源（分片）到端口，而无需保证。

提示

创建主机聚合到将带宽保证与这些端口分开的端口，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，您会收到 **ResourceNotFound** 错误，并且您必须加载 **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 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** 的策略创建具有最小带宽为 **40000000** kbps 的 ingress 和 egress 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** 策略应用到端口 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，并显示物理功能中持有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 addrgenmode 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:fffe prio 0 rate 4Gbit ceil 34359Mbit burst 9000b cburst 8589b
  class htb 1:1 parent 1:fffe prio 0 rate 72Kbit ceil 34359Mbit burst 9063b cburst 8589b
  class htb 1:fffe 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 rule create
- *命令行界面参考* 中设置的端口

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，以确保在其上生成 Red Hat OpenStack Platform (RHOSP)虚拟机实例的主机具有最小网络带宽。

先决条件

- RHOSP 网络服务(neutron)必须加载 **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。
- 只有当没有实例使用分配给策略的任何端口时，才能修改最小带宽 QoS 策略。如果绑定了端口，网络服务无法更新放置 API 使用信息。
- 放置服务必须支持 microversion 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，您会收到 **ResourceNotFound** 错误，并且您必须加载 **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 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** 的策略创建具有最小带宽为 **40000000** kbps 的 ingress 和 egress 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** 策略应用到端口 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. 提供 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，通过主机 **dell-r730-014** 上的接口 **enp6s0f1** 提供的带宽会被检查，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 rule create

- *命令行界面参考* 中设置的端口

9.4. 使用 QOS 策略限制网络流量

您可以创建一个 Red Hat OpenStack Platform (RHOSP)网络服务(neutron)服务质量(QoS)策略，该策略限制了 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。（默认情况下会加载插件。）

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，您会收到 **ResourceNotFound** 错误，并且您必须加载 **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 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

您可以在策略中添加多个规则，只要每个规则的类型或方向不同。例如，您可以指定两个带宽限制规则，一个规则带有 egress，另一个具有入口方向。

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
```

6. 您可以创建带有附加策略的端口，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

- 确认带限制策略已应用到端口。
  - 获取策略 ID。

    示例

    在本例中，会查询 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，会查询 **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                                              |
+-------------------------+-------------------------------------------------------------------+
```

其他资源

- *命令行界面参考中的* network qos rule create
- *命令行界面参考* 中设置的网络 qos 规则
- *命令行界面参考中的* network qos 规则删除
- *命令行界面参考中的 网络 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)网络中实施服务质量(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，您会收到 **ResourceNotFound** 错误，并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 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 DSCP 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 DSCP 值。

示例

在本例中，规则 **d7f976ec-7fab-4e60-af70-f59bf88198e6** 中的 DSCP 标记值被改为 22，在 **qos-web-servers** 策略中：

```
$ openstack network qos rule set --dscp-mark 22 qos-web-servers d7f976ec-7fab-4e60-af70-f59bf88198e6
```

7. 您可以删除 DSCP 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，DSCP 规则 **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

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 rule create
- *命令行界面参考* 中设置的网络 qos 规则
- *命令行界面参考中的* network qos 规则删除
- *命令行界面参考中的 网络 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实施最小带宽”
- 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)创建的桥接相关联。RHOSP 网络服务(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：

10.1. 网桥映射概述

在 Red Hat OpenStack Platform (RHOSP)网络服务(neutron)中，您可以使用网桥映射来允许供应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的机制驱动程序而异：

- ML2/OVS： **br-int** 和 **br-ex** 之间的跳接端口允许流量通过提供商网络的网桥，以及外向物理网络。
- ML2/OVN：只有虚拟机绑定到 hypervisor 且虚拟机需要端口时，网络服务仅在虚拟机监控程序上创建跳接端口。

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站，如提供商网络代表。

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，该 ID 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** 到达 **br-int** 时，数据包的外部 VLAN ID 被 **br-int** 中的内部 VLAN 标签替代，并且允许 **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中）。

10.3. 配置网桥映射

要修改 Red Hat OpenStack Platform (RHOSP)网络服务(neutron)用来与物理网络连接的供应商网络流量的网桥映射，您需要修改所需的 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 underclod 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 Compute 节点配置网桥映射。

流程

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

```
$ source ~/stackrc
```

3. 创建自定义 YAML 环境文件。

   示例

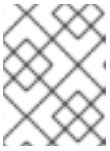
```
$ vi /home/stack/templates/my_bridge_mappings.yaml
```

4. 您的环境文件必须包含关键字 **parameter_defaults**。在 **parameter_defaults** 关键字后面，添加 **NeutronBridgeMappings** heat 参数，其值适合您的站点。

   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

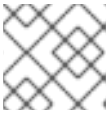
如果没有使用 **NeutronBridgeMappings** 参数，则默认将主机(br-ex)上的外部网桥映射到物理名称(datacentre)。

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 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**。

6. 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"
   ```

7. 运行部署命令，并包含核心 heat 模板、环境文件以及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
   ```

8. 执行以下步骤：

   a. 使用网络 VLAN 范围，创建代表对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，您可以使用物理名称。）

   b. 将外部网络与路由器接口连接到您的项目网络。

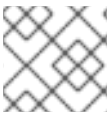
其他资源

- *使用 director 安装和管理 Red Hat OpenStack Platform 指南中的更新网络配置文件的格式*
- 在自定义 *Red Hat OpenStack Platform 部署* 指南中的 overcloud 创建中包括环境文件

10.4. 为 OVS 维护网桥映射

在移除任何 OVS 网桥映射后，您必须执行后续的清理，以确保网桥配置被任何关联的跳接端口条目清除。您可以使用以下方法执行此操作：

- 手动端口清理 - 需要小心删除多余的补丁端口。不需要中断网络连接。
- 自动化端口清理 - 执行自动清理，但需要停机，并且要求重新添加必要的网桥映射。当可以容忍网络连接中断时，在调度的维护窗口中选择这个选项。



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 跳接端口

在移除任何 OVS 网桥映射后，还必须删除关联的跳接端口。

先决条件

- 您清理的跳接端口必须是 Open Virtual Switch (OVS)端口。
- **不需要** 系统中断来执行手动补丁端口清理。
- 您可以通过它们的命名约定来识别要清理的补丁端口：
  - 在 **br-$external_bridge** 补丁端口中，命名为 **phy-<external bridge name>** （例如 phy-br-ex2）。
  - 在 **br-int** 中，跳接端口命名为 **int-<external bridge name>** （例如 **int-br-ex2**）。

流程

1. 使用 **ovs-vsctl** 删除与删除的网桥映射条目关联的 OVS 跳接端口：

```
# 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 跳接端口

在移除任何 OVS 网桥映射后，还必须删除关联的跳接端口。

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

- 您清理的跳接端口必须是 Open Virtual Switch (OVS)端口。
- 使用 **neutron-ovs-cleanup** 命令自动清理跳接端口会导致网络连接中断，并且仅在计划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 网桥拔出所有跳接端口（实例、qrouter 等）。

流程

1. 使用 **--ovs_all_ports** 标志运行 **neutron-ovs-cleanup** 命令。

重要

执行此步骤将导致所有网络中断。

```
# /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，您的网桥映射值将重新应用。

注意

重启后，OVS 代理不会干扰 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

- 在自定义 *Red Hat OpenStack Platform 部署* 指南中的 overcloud 创建中包括环境文件

第 11 章 VLAN 感知实例

11.1. VLAN 中继和 VLAN 透明网络

虚拟机实例可以在单个虚拟 NIC 上发送和接收 VLAN 标记的流量。这对希望 VLAN 标记流量的 NFV 应用(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 部署中的替代方案，您可以使用中继支持 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 上的实例使用，其他实例和设备会忽略。在 VLAN 透明网络中，VLAN 在实例中进行管理。您不需要在 OpenStack Networking Service (neutron)中设置 VLAN。

VLAN 中继通过将 VLAN 合并到单一中继端口来支持 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，而实例则会看到使用 VLAN ID 标记的流量。在网络数据包被注入到实例之前，它们会立即标记，且无需在整个网络标记。

下表比较了 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	VM 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。实例必须使用在中继中配置的 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。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。不要在扁平类型提供商网络部署中使用 VLAN 透明性。
- 确保外部交换机支持在两个 VLAN 上使用 ethertype 0x8100 的 802.1q VLAN 堆栈。OVN VLAN 透明性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>** 替换为属于您现有部署的环境文件列表。

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 设置为比 underlay 网络的 MTU 小 4 字节，以适应 VLAN 透明度所需的额外标记。例如，如果 underlay 网络 MTU 为 1500，则将接口 MTU 设置为 1496。

以下示例命令在 **eth0** 上添加了一个 MTU 为 1496 的 VLAN 接口。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 上的两个虚拟机之间 ping。
2. 在 **eth0** 上使用 **tcpdump** 以查看数据包是否到达 VLAN 标签是否完好。

其他资源

- *自定义 Red Hat OpenStack Platform 部署* 指南中的环境文件
  https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy the-overcloud-with-the-orchestration-service#con_environment-files_understanding-heat-templates
- 在自定义 *Red Hat OpenStack Platform 部署* 指南中的 overcloud 创建中包括环境文件
- *命令行界面参考* 中设置的端口

11.3. 检查中继插件

在红帽 openStack 部署期间，默认启用 trunk 插件。您可以查看控制器节点上的配置：

- 在控制器节点上，确认在 /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 网络会向您创建的父端口添加一个中继连接。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-
   b93adbdcb197 |
   ```

```
| 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。在本例中，*在公共网络上* 创建一个名为 parent-trunk-port 的 neutron 端口。这个中继是 *父* 端口，因为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一个中继。在本例中，中继命名为 **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。
   这个端口是到中继的子端口连接。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**，请确认您要在不同网络上创建子端口到具有父中继端口的子端口。这个示例将公共网络用于父中继端口，并将 private 用于子端口。

2. 将端口与中继关联(**parent-trunk**)，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. 将实例配置为使用中继

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过程中将子端口配置为使用特定的 MAC 地址。

先决条件

- 如果要执行 Compute 节点的实时迁移，请确保为 RHOSP 部署正确设置 RHOSP 网络服务 RPC 响应超时。RPC 响应超时值可能因站点而异，它取决于系统速度。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过程时间，然后正确设置 RHOSP 网络服务 RPC 响应超时。尝试保持 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 命令检查 **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. 创建一个实例，它使用父 **port-id** 作为其 vNIC。

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-juqco0el                     |
| 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，如果超时值太低，使用中继端口的 Compute 节点的实时迁移可能会失败。

RPC 响应超时值可能因站点而异，它取决于系统速度。常规建议是为/100 中继端口至少设置 120 秒的值设为 120 秒。

如果您的站点使用中继端口，则最佳实践是测量 RHOSP 部署的中继端口绑定过程时间，然后相应地设置 RHOSP 网络服务 RPC 响应超时。尝试保持 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，创建一个自定义 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 自定义 overcloud 的各个方面，它是为 heat 模板 *提供自定义* 的特殊模板类型。

2. 在 **ExtraConfig** 下的 YAML 环境文件中，为 **rpc_response_timeout** 设置适当的值（以秒为单位）。（默认值为 60 秒。）

示例

```
parameter_defaults:
  ExtraConfig:
    neutron::rpc_response_timeout: 120
```

注意

RHOSP Orchestration 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，但这个值只会影响 RHOSP Networking 组件。

3. 运行 **openstack overcloud deploy** 命令，并包含核心 heat 模板、环境文件以及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更为优先。

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

- *自定义 Red Hat OpenStack Platform 部署* 指南中的环境文件 https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_deploy the-overcloud-with-the-orchestration-service#con_environment-files_understanding-heat-templates
- 在自定义 *Red Hat OpenStack Platform 部署* 指南中的 overcloud 创建中包括环境文件

11.8. 了解中继状态

- **ACTIVE** ：中继按预期工作，且没有当前请求。
- **下载** ：中继的虚拟和物理资源不同步。这可以是协商过程中的临时状态。

- **BUILD** ：已有请求并调配资源。成功完成后，中继返回到 **ACTIVE**。
- **DEGRADED**: 置备请求没有完成，因此中继仅被部分置备。建议您删除子端口并重试。
- **ERROR** ：取消置备请求失败。删除导致错误将中继返回到健康状态的资源。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 表来控制在项目间共享 *neutron* 网络，允许管理员控制哪些项目被授予将实例附加到网络的权限。

因此，云管理员可以移除某些项目创建网络的能力，并可以允许它们连接到与其项目对应的预先存在的网络。

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)策略来授予共享网络的项目访问权限。

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，其授予对 *审核员* 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-aee3-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-aee3-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 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 RBAC，并授予工程项目 (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，工程项目中的用户可以查看网络或连接实例：

```
$ 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。

每个模型都有优缺点。使用此文档来仔细规划集中式路由或 DVR 是否适合您的需求。

新的默认 RHOSP 部署使用带有 Open Virtual Network 机制驱动程序(ML2/OVN)的 DVR 和 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，允许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)中的路由服务可归类为三个主要流：

- **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。这个定义适用于 IPv4 和 IPv6 子网。
- **使用浮动 IP 的 North-South 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值。虽然浮动 IP 作为浮动 IP 和网络服务(neutron)端口之间的一对一关联建模，它们则通过与执行 NAT 转换的网络服务路由器关联来实施。浮动 IP 本身取自 uplink 网络，该网络为路由器提供外部连接。因此，实例可以与外部资源（如互联网上的端点）或其他方法通信。浮动 IP 是 IPv4 概念，不适用于 IPv6。假设项目使用的 IPv6 寻址(GUAs)使用跨项目没有重叠的全局广播地址(GUAs)，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。

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，每次需要路由功能时(east/west、浮动 IP 或 SNAT)，流量都会遍历拓扑中的专用节点。这带来了多个挑战，并导致出现最佳流量流。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，流量仍必须离开 Compute 节点，通过控制器流，并且重新路由到 Compute 节点。这会对性能造成负面影响。
- 具有浮动 IP 的实例通过 Controller 节点接收和发送数据包 - 外部网络网关接口仅适用于 Controller 节点，因此流量是否源自实例，还是来自外部网络的实例，它必须通过 Controller 节点流过。因此，在大型环境中，Controller 节点会受大量流量负载的影响。这会影响性能和可扩展性，还需要仔细规划来适应外部网络网关接口中的足够带宽。相同的要求适用于 SNAT 流量。

为了更好地扩展 L3 代理，网络服务可以使用 L3 HA 功能，该功能将虚拟路由器分布到多个节点。如果 Controller 节点丢失，HA 路由器将在另一个节点上切换到备用节点，并在 HA 路由器故障转移完成后出现数据包丢失。

13.2. DVR 概述

分布式虚拟路由(DVR)提供了备选路由设计。DVR 通过在每个计算节点上部署 L3 代理和调度路由器来隔离 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 North-South 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，仍然需要一个专用的 Controller 节点。
- Controller 节点上的 L3 代理使用 **dvr_snat** 模式，以便节点仅服务 SNAT 流量。
- neutron 元数据代理在所有 Compute 节点上分发和部署。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 核心插件和 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)的网络流量会通过 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 的流量将无法访问绑定的 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式 Controller 节点。
- 在 ML2/OVS 部署中，IPv6 流量不会被分发，即使启用了 DVR。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，所有 east/west 流量始终分布，在 DVR 配置时，北北流量会被分发。
- 在 ML2/OVS 部署中，RHD 不支持与 L3 HA 结合使用。如果您将 DVR 与 Red Hat OpenStack Platform 17.1 director 搭配使用，则 L3 HA 被禁用。这意味着路由器仍然调度到网络节点上（以及 L3 代理之间的负载），但如果一个代理失败，则由此代理托管的所有路由器也失败。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 DCHP 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 节点需要外部网络上的接口附加到外部网桥。它们使用此接口连接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 网络命名空间的实施。
- 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 在安装过程中强制执行 L2 Population。

13.4. 支持的路由架构

Red Hat OpenStack Platform (RHOSP)支持列出的 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. 将集中式路由器迁移到分布式路由

本节包含有关使用 L3 HA 集中式路由的 Red Hat OpenStack Platform 部署升级到分布式路由的信息。

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. 确认路由通过现有的路由器正常工作。
4. 您无法将 L3 HA 路由器直接转换为 *分布式*。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

13.6. 使用禁用分布式虚拟路由(DVR)部署 ML2/OVN OPENSTACK

新的 Red Hat OpenStack Platform (RHOSP)部署默认为带有 Open Virtual Network 机制驱动程序(ML2/OVN)和 DVR 的 neutron Modular Layer 2 插件。

在 DVR 拓扑中，具有浮动 IP 地址的计算节点在虚拟机实例和网络之间路由流量，该网络为路由器提供外部连接（南北流量）。实例(east-west 流量)之间的流量也被分发。

您可以选择禁用 DVR 部署。这将禁用南北 DVR，需要南北流量来遍历控制器或网络程序节点。东西路由始终在 ML2/OVN 部署中分发，即使 DVR 被禁用。

先决条件

- RHOSP 17.1 发布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. 要应用此配置，请部署 overcloud，将自定义环境文件添加到堆栈和其他环境文件。例如：

```
(undercloud) $ openstack overcloud deploy --templates \
  -e [your environment files]
  -e /home/stack/templates/<custom-environment-file>.yaml
```

13.6.1. 其他资源

- *配置 Red Hat OpenStack Platform* 网络指南中的 了解分布式虚拟路由(DVR)。

第 14 章 使用 IPV6 的项目网络

14.1. IPV6 子网选项

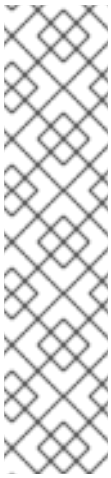
当您在 Red Hat OpenStack Platform (RHOSP)项目网络中创建 IPv6 子网时，您可以指定地址模式和路由器公告模式来获取特定结果，如下表所述。

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	该实例使用无状态地址自动配置(SLAAC)接收来自外部路由器（不由 OpenStack 网络管理）的 IPv6 地址。  **注意** OpenStack 网络只支持 SLAAC 的 EUI-64 IPv6 地址分配。这允许简化的 IPv6 网络，作为基于基础 64 位加 MAC 地址的主机自我分配地址。您不能使用不同的子网掩码和 SLAAC 的 *address_assign_type* 创建子网。
ipv6_ra_mode=not set	ipv6-address-mode=dhcpv6-stateful	实例使用 **DHCPv6 stateful** 接收来自 OpenStack Networking (dnsmasq)的 IPv6 地址和可选信息。

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 **DHCPv6 stateless**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	实例使用 DHCPv6 **有状态，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 **DHCPv6 stateless** 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 **SLAAC** 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 **DHCPv6 stateful** 接收来自 OpenStack Networking (**dnsmasq**) 的一个 IPv6 地址，使用 **DHCPv6 stateful** 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 **SLAAC** 接收来自 OpenStack Networking (**radvd**) 的一个 IPv6 地址，使用 **DHCPv6 stateless** 接收来自 OpenStack Networking (**dnsmasq**) 的可选信息。

14.2. 使用有状态 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，使用名为 database-servers 的网络中的 Stateful DHCPv6 创建 IPv6 子网。

流程

1. 检索您要创建 IPv6 子网的项目 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有所不同。

```
# 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，并记下要托管 IPv6 子网的网络的名称：

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

验证步骤

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

结果

因此，在添加到 database-servers 子网时，QA 项目创建的实例可以接收 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=fdf8:f53b:82e4::52 |
+--------------------------------------+------------+--------+------------+-------------+--------------------
----------------+
```

其他资源

要查找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，请参阅配置 Red Hat OpenStack Platform 网络 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，若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。

如需配额设置列表，请参阅立即跟随的部分。

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** - 项目可用的端口数量。
- **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

以下是可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量。
- **quota_firewall_policy** - 项目可用的防火墙策略数量。
- **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，且无法在创建默认安全组前将所有配额设置为零（以及接受 IPv4 和 IPv6 的所有出口流量的两个默认安全组规则）。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默认安全组，或直到列出安全组或安全组规则为止。

以下是用于管理项目可以创建的安全组数量的配额选项：

- **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的配额的附加选项：

- **default_quota*** - 项目可用的默认资源数。
- **quota_health_monitor*** - 项目可用的运行状况监视器数量。
  运行状况监视器不消耗资源，但配额选项可用，因为 OpenStack 网络会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，但配额选项可用，因为 OpenStack 网络会将池成员视为资源使用者。
- **quota_pool** - 项目可用的池数量。

第 16 章 部署路由供应商网络

16.1. 路由供应商网络的优点

在 Red Hat OpenStack Platform (RHOSP)中，管理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。对于管理员，路由供应商网络提供可扩展和容错能力。例如，如果发生主要错误，则只有一个网段会受到影响，而不是整个网络失败。

在路由供应商网络前，管理员通常必须从以下构架之一中进行选择：

- 单个大型第 2 层网络
- 多个、较小的第 2 层网络

在扩展和减少容错（递增故障域）时，单一的大型第 2 层网络变得复杂。

多个更小的第 2 层网络扩展更好和缩小故障域，但可能会给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。

与传统的网络类似，第 2 层（交换机）处理同一网络段和第 3 层(routing)间的端口间的流量传输。

网络服务不会在网段之间提供第 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，类似于传统的提供商网络。

您可以配置计算调度程序来过滤与路由网络段关联的 Compute 节点，以便调度程序仅将实例放在所需路由提供商网络段中的 Compute 节点上。

如果需要 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

Red Hat OpenStack Platform 中路由供应商网络已知的限制包括：

- 不支持使用中央 SNAT 或浮动 IP 的南北路由。
- 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。您不能重复使用片段 ID。

16.4. 准备路由的提供商网络

要在 Red Hat OpenStack Platform (RHOSP)中创建路由供应商网络，您必须首先收集创建它所需的网络信息。您必须将 overcloud 配置为创建一个自定义角色，为包含网络片段的 Compute 节点部署 RHOSP Networking 服务(neutron)元数据代理。对于使用 ML2/OVS 机制驱动程序的环境，除了元数据代理外，还必须在 Compute 节点上包含 **NeutronDhcpAgent** 服务。在运行计算调度程序服务的控制器上，您必须启用对路由供应商网络的调度支持。

先决条件

- 您必须是一个具有 **admin** 角色的 RHOSP 用户。

流程

1. 从您要在其上创建路由的网络的 **tripleo-heat-templates/network_data.yaml** 文件中收集 VLAN ID，并为您在路由供应商网络上创建的每个段分配唯一的物理网络名称。这允许在子网之间重复使用相同的分段详情。
   创建参考表来视觉化 VLAN ID、片段和物理网络名称之间的关系：

   表 16.1. 示例 - 路由供应商网络段定义

路由提供商网络	VLAN ID	segment	物理网络
multisegment1	128	segment1	provider1
multisegment1	129	segment2	provider2

2. 规划网段之间的路由。
   网段上的每个子网必须包含该特定子网中路由器接口的网关地址。您需要 IPv4 和 IPv6 格式的子网地址。

   表 16.2. 示例 - 路由提供商网络片段的路由计划

路由提供商网络	segment	子网地址	网关地址
multisegment1	segment1 (IPv4)	203.0.113.0/24	203.0.113.1
multisegment1	segment1 (IPv6)	fd00:203:0:113::/64	fd00:203:0:113::1
multisegment1	segment2 (IPv4)	198.51.100.0/24	198.51.100.1
multisegment1	segment2 (IPv6)	fd00:198:51:100::/64	fd00:198:51:100::1

3. 路由的提供商网络要求 Compute 节点位于不同的网段上。检查 **templates/overcloud-baremetal-deployed.yaml** 文件，以确保路由提供商网络中的每个 Compute 主机都可以与其其中一个片段直接连接。
有关更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 为 overcloud 置备裸机节点。

4. 确保 **NeutronMetadataAgent** 服务包含在包含该片段的 Compute 节点的 **templates/roles_data-custom.yaml** 中：

```
...
- name: Compute
  ...
  ServicesDefault:
    - OS::TripleO::Services::NeutronMetadataAgent
...
```

如需更多信息，请参阅自定义 *Red Hat OpenStack Platform 部署指南中的可组合服务和自定义角色*。

5. 在使用 ML2/OVS 机制驱动程序时，除了 **NeutronMetadataAgent** 服务外，还确保 **NeutronDhcpAgent** 服务包含在包含该片段的 Compute 节点的 **templates/roles_data-custom.yaml** 中：

```
...
- name: Compute
  ...
  ServicesDefault:
    - OS::TripleO::Services::NeutronDhcpAgent
    - OS::TripleO::Services::NeutronMetadataAgent
...
```

提示

与传统的提供商网络不同，DHCP 代理无法支持网络中的多个网段。在包含片段而不是网络节点上的 Compute 节点上部署 DHCP 代理，以减少节点数。

6. 创建路由供应商网络环境文件，如 **rpn_env.yaml**。

7. 配置 DHCP，以便在隔离的网络中启用元数据支持：

```
parameter_defaults:
  NeutronEnableIsolatedMetadata: true
```

8. 确保 **片段** 服务插件加载到网络服务中：

```
$ openstack extension list --network --max-width 80 | grep -E "Segment"
```

如果缺少 **片段** 插件，将其添加到 **NeutronServicePlugins** 参数中：

示例

```
parameter_defaults:
  NeutronEnableIsolatedMetadata: true
  NeutronServicePlugins: 'router,qos,segments,trunk,placement'
```

重要

当您向 **NeutronServicePlugins** 参数添加新值时，RHOSP director 会使用您要添加的值覆盖之前声明的值。因此，当您添加 **段** 时，还必须包含任何之前声明的网络服务插件。

9. 要在在主机上调度实例前，要使用放置服务验证网络，请在运行计算调度程序服务的控制器上启用路由提供商网络的调度支持。

   示例

```
parameter_defaults:
  NeutronEnableIsolatedMetadata: true
  NeutronServicePlugins: 'router,qos,segments,trunk,placement'
  NovaSchedulerQueryPlacementForRoutedNetworkAggregates: true
```

10. 使用其他环境文件将路由的供应商网络环境文件添加到堆栈中，并部署 overcloud：

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

后续步骤

- 创建路由提供商网络

其他资源

- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 为 overcloud 置备裸机节点。
- *自定义 Red Hat OpenStack Platform 部署* 指南中的可组合服务和自定义角色

16.5. 创建路由提供商网络

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。对于管理员，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 第 16.4 节 “准备路由的提供商网络” 中的步骤。
- 您必须是一个具有 **admin** 角色的 RHOSP 用户。

流程

1. 创建一个包含默认网段的 VLAN 提供商网络。
   在本例中，VLAN 提供商网络名为 **multisegment1**，它使用名为 **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         | 128     |
   +--------------------------------------+---------+--------------------------------------+--------------
   +---------+
   ```

   b. 使用网段 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片段。
   在本例中，网络段使用一个名为 **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** 片段：

```
$ 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** 网段上创建一个 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 子网依赖物理网络基础架构进行无状态地址自动配置(SLAAC)和路由器公告。

6. 在 **segment2** 网段上创建一个 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 子网都与至少一个 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 |
+--------------------------------------+------------+-------------+-------------------+-------+-------+------
--------------+
```

2. 验证已经为 Compute 服务放置 API 中每个网段 IPv4 子网创建清单。
为所有段 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 |
+----------------+------------------+----------+----------+-----------+----------+-------+
```

3. 验证是否为 Compute 服务中的每个片段创建了主机聚合：

```
$ openstack aggregate list
```

输出示例

在本例中，仅显示其中一个片段：

```
+----+-----------------------------------------------------------+------------------+
| Id | Name                                                      | Availability Zone |
+----+-----------------------------------------------------------+------------------+
| 10 | Neutron segment id 053b7925-9a89-4489-9992-e164c8cc8763 | None              |
+----+-----------------------------------------------------------+------------------+
```

4. 启动一个或多个实例。每个实例会根据它在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由用户指定了固定 IP，则该特定 IP 会立即分配给端口。但是，创建端口并将其传递给实例会产生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，网络服务会将 IP 地址延迟分配给端口，直到特定计算节点变得明显。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* network segment create
- *命令行接口参考*中的 subnet create
- *命令行界面参考*中的 port create

16.6. 将非路由网络迁移到路由提供商网络

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，子网分配池中的地址分配给端口，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. 将子网 **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
  ```

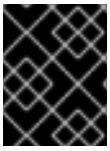
  输出示例

  ```
  +------------+--------------------------------------+
  | 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设置*

第 17 章 使用路由器类别创建自定义虚拟路由器



重要

本节的内容在此发行版本中 *作为技术预览提供*，因此不受红帽完全支持。它只应用于测试，不应部署在生产环境中。如需更多信息，请参阅 技术预览。

您可以使用路由器类别在 Red Hat OpenStack Platform (RHOSP) ML2/OVN 环境中部署自定义虚拟路由器。启用路由器类别功能后，您可以创建路由器类别，并使用它们创建自定义路由器。

在 RHOSP 部署中，您可以将基于路由器类别的虚拟自定义路由器与默认 OVN 类型的路由器合并。

此可选功能不会影响默认 OVN 虚拟路由器的操作。如果您启用了路由器类别功能，则默认 OVN 路由器将被视为 default 类别，不会影响其配置或操作。

17.1. 启用路由器类型并为自定义路由器创建服务提供商

在使用可选路由器类别功能前，RHOSP 管理员必须通过将 **ovn-routers** 服务替换为 **ovn-router-flavors** 服务来启用该功能，并为每个自定义路由器创建一个服务提供商。

您必须在 Networking 服务(neutron)目录中的模块中部署您的服务供应商代码。红帽建议 **neutron.services.ovn_l3.service_providers.user_defined** 模块。

您可以在 **neutron.services.ovn_l3.service_providers.user_defined** 模块中找到名为 **UserDefined** 的示例服务提供商。

注意

以下流程涉及直接编辑 Controller 节点上的 **.conf** 文件。红帽正在开发 heat 模板方法和 OpenStack 命令，以取代此直接编辑方法。

先决条件

- 您已为部署创建了路由器类别服务供应商。
- 您可以访问 RHOSP Controller 节点并更新配置文件的权限。

流程

1. 在每个 Controller 节点上，对 **/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf** 进行以下更改，然后重新启动网络服务(Neutron)。
   a. 在 **service_plugins** 列表中，将 **ovn-routers** 更改为 **ovn-router-flavors**。
   b. 创建 **service_providers** 部分，并为计划使用的每个路由器类别添加一个服务供应商变量。

   示例

   本例添加了两个服务提供程序： **user_defined_1** 和 **user_defined_2**。

   ```
   [DEFAULT]
   service_plugins = qos,ovn-router-flavors,trunk,segments,port_forwarding,log
   ```

```
...

[service_providers]
service_provider =
L3_ROUTER_NAT:user_defined_1:neutron.services.ovn_l3.service_providers.user_defined.UserDefined_1
service_provider =
L3_ROUTER_NAT:user_defined_2:neutron.services.ovn_l3.service_providers.user_defined.UserDefined_2
```

路由器类别服务提供商定义具有以下元素：

服务供应商常数

L3_ROUTER_NAT

Name

服务提供商的名称，它是两个冒号字符之间的描述性字符串。例如，**:user_defined_1:** 和 **:user_defined_2:**。名称在环境中必须是唯一的。

路径

红帽建议使用此路径： **neutron.services.ovn_l3.service_providers.user_defined**

类

服务提供商的 python 类名称。每个提供程序都有自己的类。例如，**UserDefined_1** 和 **UserDefined_2**。

验证

- 验证您的用户定义的服务供应商是否已加载：

```
$ openstack network service provider list
```

如果流程成功，新服务会出现在列表中。

```
+-----------------------+-------+---------+
| Service Type  | Name          | Default |
+---------------+---------------+---------+
| L3_ROUTER_NAT | user_defined_1  | False   |
| L3_ROUTER_NAT | user_defined_1  | False   |
| L3_ROUTER_NAT | ovn           | True    |
+---------------+---------------+---------+
```

17.2. 创建路由器类型

您可以创建路由器类型，用于在 Red Hat OpenStack Platform (RHOSP) ML2/OVN 部署中创建自定义虚拟路由器。

先决条件

- 您已启用了路由器类型功能。

流程

1. 为路由器类别创建服务配置文件：

```
$ openstack network flavor profile create            \
  --description "User defined router flavor profile" \
  --enable  \
  --driver   \
  neutron.services.ovn_l3.service_providers.user_defined.Userdefined_1
```

--driver 参数值中的路径是您在前面的流程中创建的一个。

输出示例

```
+-------------+-----------------------------------------------------------------+
| Field       | Value                                                           |
+-------------+-----------------------------------------------------------------+
| description | User defined router flavor profile                              |
| driver      | neutron.services.ovn_l3.service_providers.user_defined.Userdefined_1 |
| enabled     | True                                                            |
| id          | a717c92c-63f7-47e8-9efb-6ad0d61c4875                            |
| meta_info   |                                                                 |
| project_id  | None                                                            |
+-------------+-----------------------------------------------------------------+
```

2. 创建路由器类型：

```
$ openstack network flavor create \
   --service-type L3_ROUTER_NAT     \
   --description "User defined flavor for routers in the L3 OVN plugin" \
   user-defined_router-flavor
```

输出示例

```
+--------------------+-----------------------------------------------------+
| Field              | Value                                               |
+--------------------+-----------------------------------------------------+
| description        | User defined flavor for routers in the L3 OVN plugin |
| enabled            | True                                                |
| id                 | e47c1c5c-629b-4c48-b49a-78abe6ac7696                |
| name               | user-defined-router-flavor                          |
| service_profile_ids | []                                                 |
| service_type       | L3_ROUTER_NAT                                       |
+--------------------+-----------------------------------------------------+
```

3. 将服务配置集添加到 router 类别中，如下例所示：

```
$ openstack network flavor add profile user-defined-router-flavor     \
  a717c92c-63f7-47e8-9efb-6ad0d61c4875
```

17.3. 使用路由器类型创建自定义虚拟路由器

启用路由器类别功能并创建路由器类别后，您可以创建具有路由器类别的自定义路由器。

先决条件

- 您已启用了路由器类型功能。

- 您创建了一个或多个路由器类型。

流程

1. 获取路由器类别 ID：

```
$ openstack network flavor list -c ID -c Name
```

输出示例

```
+--------------------------------------+--------------------------+
| ID                                   | Name                     |
+--------------------------------------+--------------------------+
| 4b37f895-e78e-49df-a96b-1916550f9116 | user-defined-router-flavor |
+--------------------------------------+--------------------------+
```

2. 如以下示例所示，创建自定义路由器：

```
$ openstack router create  \
  --flavor-id 4b37f895-e78e-49df-a96b-1916550f9116  \
  user-defined-flavor-router
```

如果不使用 **--flavor** 参数，**openstack router create** 命令会创建一个默认的 OVN 路由器。

3. 列出部署的路由器以验证路由器创建：

```
$ openstack router list
```

输出示例

```
+--------------------------------------+--------------------------+--------------+
| ID                                   | Name                     | Status | State
|Project                               |
+--------------------------------------+--------------------------+--------------+
| 21889ed3-b8df-4b0e-9a64-92ba9fab655d | ovn-flavor-router        | ACTIVE | UP
| b807321af03f44dc808ff06bbc845804     |
| 9f5fec56-1829-4bad-abe5-7b4221649c8e | user-defined-flavor-router| ACTIVE | UP
| b807321af03f44dc808ff06bbc845804     |
| e9f25566-ff73-4a76-aeb4-969c819f9c47 | router1                  | ACTIVE | UP
| 1bf97e3957654c0182a48727d619e00f     |
+--------------------------------------+--------------------------+--------------+
```

第 18 章 配置允许的地址对

18.1. 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)。

重要

在 RHOSP 联网环境中，在创建虚拟机实例时，请不要将实例绑定到虚拟端口 (vport)。反之，使用其 IP 地址不是另一个端口允许的地址对成员的端口。

将 vport 绑定到实例可防止实例生成并生成类似如下的错误消息：

```
WARNING nova.virt.libvirt.driver [req-XXXX - - - default default] [instance:
XXXXXXXXX] Timeout waiting for [('network-vif-plugged', 'XXXXXXXXXX')] for
instance with vm_state building and task_state spawning.: eventlet.timeout.Timeout:
300 seconds
```

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，您不应该将默认安全组与允许的地址对中的更广泛的 IP 地址范围一起使用。这样，可以允许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。

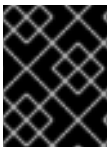
如果您将 CIDR 格式 IP 地址用于绑定端口 **allowed_address_pairs**，则后端中没有配置端口转发，并且 CIDR 中任何 IP 的流量将无法访问绑定的 IP 端口。

其他资源

- *命令行界面参考中的 port 命令*
- 第 18.2 节 “创建端口并允许一个地址对”
- 第 18.3 节 “添加允许的地址对”

18.2. 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过该端口，而不考虑子网。



重要

不要在允许的地址对中使用具有更广泛的 IP 地址范围的默认安全组。这样，可以允许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 port 命令*

18.3. 添加允许的地址对

您可以将允许的地址对添加到端口，以便网络流量通过端口流，而不考虑子网。

重要

流程

-

```
$ openstack port set --allowed-address mac-address=<mac_address>,ip-address=<ip_cidr>
<port>
```

注意

其他资源

-

第 19 章

安全组规则应用到项目中的所有实例。

注意

-
-
-
-

注意

您无法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，您必须首先创建端口，将共享安全组应用到该端口，然后将该端口分配给实例。请参阅向端口添加安全组。

19.1.

流程

1.
```
$ openstack security group list
$ openstack security group rule list <sec_group>
```
   -

2. .

```
$ openstack security group create [--stateless] mySecGroup
```

   - **注意**

3.
```
$ openstack security group rule create --protocol <protocol> \
[--dst-port <port-range>] \
[--remote-ip <ip-address> | --remote-group <group>] \
[--ingress | --egress] mySecGroup
```
   -
   -
   -
   - **注意**

4. $ openstack security group rule create --protocol tcp \
 --dst-port 22 mySecGroup

19.2.

流程

1. $ openstack security group list
2.
3. $ openstack security group rule create --protocol <protocol> \
[--dst-port <port-range>] \
[--remote-ip <ip-address> | --remote-group <group>] \
[--ingress | --egress] <group_name>
   -
   -
   -
   -
   -
4. $ openstack security group rule create --protocol tcp \
 --dst-port 22 mySecGroup

19.3.

流程

1. $ openstack security group list
2. $ openstack security group show <sec-group>
3. $ openstack security group rule delete <rule> [<rule> ...]

19.4.

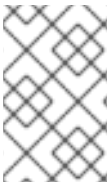
流程

1. $ openstack security group list
2. $ openstack port list

3. $ openstack port show <port-uuid> -c security_group_ids

4. $ openstack security group delete <group> [<group> ...]

19.5.



注意

您无法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，您必须首先创建端口，将共享安全组应用到该端口，然后将该端口分配给实例。请参阅向端口添加安全组。

先决条件

-
- 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。

  示例

  $ openstack security group create ping_ssh

流程

1.

2. $ openstack project list

3. $ openstack security group list

4. **示例**

   $ openstack network rbac create --target-project \
   32016615de5d43bb88de99e7f2e26a1e --action access_as_shared \
   --type security_group 5ba835b7-22b0-4be6-bdbe-e0722d1b5f24

   --target-project

   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。

   --action access_as_shared

   --type

   5ba835b7-22b0-4be6-bdbe-e0722d1b5f24

提示

其他资源

-
-
-

第 20 章

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。

```
$ openstack network log create my-log1 \
--resource-type security_group \
--resource security-group1 \
--event ACCEPT
```

resource-type

- **注意**
-
-

注意

20.1.

流程

1.
2. 输入以下命令。

```
$ openstack extension list --max-width 80 | grep logging
```

```
| Logging API Extension   | logging       | Provides a logging API   |
```

3. a. **示例**

```
parameter_defaults:
  NeutronPluginExtensions: "qos,port_security,log"
```

b.

其他资源

-

20.2.

先决条件

-

-
-

流程

1.
2.
```
$ openstack network log create my-log1 \
--resource-type security_group \
--resource sg1 \
–event ACCEPT
```
```
openstack network log create my-log3 \
--resource-type security_group \
–event ACCEPT
```
3.
```
$ openstack network log list
```

20.3.

流程

1.
2.
```
$ openstack network log list
```
3.
```
$ openstack network log show <log_object_name>
```

20.4.

流程

1.
2.
```
$ openstack network log set --disable <log_object_name>
```
3.
```
$ openstack network log set --enable <log_object_name>
```

20.5.

流程

1.
2.
```
$ openstack network log set --name <new_log_object_name> <object>
```

20.6.

流程

1.

2. ```
   $ openstack network log delete <log_object_name> [<log_object_name> ...]
   ```

20.7.

20.8.

-
-
-
-
-

```
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
```

20.9.

流程

1. **以 stack 用户身份登录 undercloud 主机。**

2. ```
   $ source ~/stackrc
   ```

3. **示例**

   ```
   parameter_defaults:
   ...
     NeutronOVNLoggingRateLimit=450
     NeutronOVNLoggingBurstLimit=50
   ```

4. 

   重要

   但是，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更为优先。

   示例

```
$ openstack overcloud deploy --templates <core_heat_templates> \
-e <other_environment_files> \
-e /home/stack/templates/neutron-ovn-dvr-ha.yaml
```

第 21 章

21.1.

先决条件

-
-

流程

1. **以 stack 用户身份登录 undercloud 主机。**
2. ```
   $ source ~/stackrc
   ```
3. **示例**

   ```
   $ vi /home/stack/templates/my-environment.yaml
   ```
4. ```
   parameter_defaults:
     NeutronMechanismDrivers: ['openvswitch', 'l2population']
     NeutronEnableL2Pop: 'True'
     NeutronEnableARPResponder: true
   ```
5. 

   重要

   但是，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更为优先。

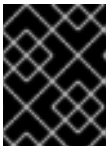
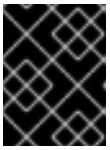
   示例

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

验证

1. ```
   $ 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. **示例**

```
$ openstack network agent show 003a8750-a6f9-468b-9321-a6c03c77aec7 -c configuration
-f json | grep l2_population
```

输出示例

```
    "l2_population": true,
```

3. **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

-
-
-

21.2.

流程

1. **示例**

```
$ source ~/stackrc
```

2. **示例**

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

提示

Red Hat OpenStack Platform Orchestration 服务(heat)使 用一组名为 *template* 的计划来安装和配置您的环境。

3. **ha_vrrp_garp_master_repeat**

   ha_vrrp_garp_master_delay

   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、环境文件以及新的自定义环境文件。

重要

但是，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更为优先。

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
-
-

21.3. 指定 DNS 分配给端口的名称

重要

流程

1. **示例**

```
$ source ~/stackrc
```

2. **注意**

示例

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

提示

您可以根据需要纳入多个环境文件。

3. **示例**

```
parameter_defaults:
  NeutronPluginExtensions: "qos,port_security,dns_domain_ports"
```

注意

4. **示例**

```
parameter_defaults:
   NeutronPluginExtensions: "qos,port_security,dns_domain_ports"
   NeutronDnsDomain: "example.com"
```

5. 

重要

但是，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更为优先。

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)。**

示例

```
$ source ~/overcloudrc
$ openstack port create --network public --dns-name my_port new_port
```

2. **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                                    |
+-------------------------+---------------------------------------------+
```

3. **示例**

```
$ openstack server create --image rhel --flavor m1.small --port new_port my_vm
```

4. **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                                    |
+-------------------------+---------------------------------------------+
```

其他资源

-
-
-
- ***命令行接口参考*中的 server create**

21.4.

先决条件

-

流程

1. **以 stack 用户身份登录 undercloud 主机。**
2. ```
   $ source ~/stackrc
   ```
3. **示例**

   ```
   $ vi /home/stack/templates/my-environment.yaml
   ```
4. **示例**

   ```
   parameter_defaults:
     NeutronPluginExtensions: "qos,port_security,extra_dhcp_opt"
   ```
5. 

   重要

   但是，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更为优先。

   示例

   ```
   $ openstack overcloud deploy --templates \
   -e <your_environment_files> \
   -e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml \
   -e /home/stack/templates/my-environment.yaml
   ```

验证

1. **示例**

   ```
   $ source ~/overcloudrc
   ```
2. **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
   ```
3. **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

-
-
-
-
- *命令行界面参考*中的 port create
-

21.5.

先决条件

- 访问 stack 用户的 undercloud 主机和凭据。

流程

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

```
$ source ~/stackrc
```

3. **要启用 port_numa_affinity_policy 扩展，打开定义 NeutronPluginExtensions 参数的环境文件，并将 port_numa_affinity_policy 添加到列表中：**

```
parameter_defaults:
  NeutronPluginExtensions: "qos,port_numa_affinity_policy"
```

4. **使用其他环境文件将您修改的环境文件添加到堆栈中，并重新部署 overcloud：**

重要

但是，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更为优先。

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

验证

1. **提供您的凭据文件。**

示例

```
$ source ~/overcloudrc
```

2. **创建新端口。**
   在创建端口时，请使用以下选项之一指定要应用到端口的 NUMA 关联性策略：

   - **--NUMA-policy-required - 调度此端口所需的 NUMA 关联性策略。**
   - **--NUMA-policy-preferred - 首选 NUMA 关联性策略来调度此端口。**
   - **--NUMA-policy-legacy - 使用传统模式调度此端口的 NUMA 关联性策略。**

     示例

     ```
     $ openstack port create --network public \
       --numa-policy-legacy  myNUMAAffinityPort
     ```

3. **显示端口的详细信息。**

   示例

   ```
   $ openstack port show myNUMAAffinityPort -c numa_affinity_policy
   ```

   输出示例

   加载扩展后，Value 列应当读取、传统的、preferred 或 required。如果扩展无法加载，则 值 会读取 None ：

   ```
   +----------------------+--------+
   | Field                | Value  |
   +----------------------+--------+
   | numa_affinity_policy | legacy |
   +----------------------+--------+
   ```

其他资源

- *自定义 Red Hat OpenStack Platform 部署*指南中的环境文件
  https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_de the-overcloud-with-the-orchestration-service#con_environment-files_understanding-heat-templates
- 在自定义 *Red Hat OpenStack Platform 部署*指南中的 overcloud 创建中包括环境文件
- 创建和管理实例指南中的创建带有端口上的 NUMA *关联性* 的实例

21.6. 载入内核模块

Red Hat OpenStack Platform (RHOSP)中的一些功能需要加载某些内核模块。例如，OVS 防火墙驱动程序要求您加载 nf_conntrack_proto_gre 内核模块来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，在常规模块管理过程中会加载这些所需的内核模块。

流程

1. **在 undercloud 主机上，以 stack 用户身份登录，创建一个自定义 YAML 环境文件。**

   示例

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

   提示

   Heat 使用一组称为 *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，它是为 heat 模板 *提供自定义* 的特殊模板类型。

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: {}
   ```

3. **运行 openstack overcloud deploy 命令，并包含核心 heat 模板、环境文件以及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更为优先。

   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 正确载入了该模块，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

- ***自定义 Red Hat OpenStack Platform 部署*指南中的环境文件**
  https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_de

the-overcloud-with-the-orchestration-service#con_environment-files_understanding-heat-templates

- 在自定义 *Red Hat OpenStack Platform 部署*指南中的 overcloud 创建中包括环境文件

21.7. 将查询限制为元数据服务

为了保护 RHOSP 环境免受网络威胁，如拒绝服务(DoS)攻击，网络服务(neutron)可让管理员限制虚拟机实例可以查询 Compute 元数据服务的速率。管理员通过将值分配给 neutron.conf 配置文件的 metadata_rate_limiting 部分中的一组参数来实现此目的。网络服务使用这些参数来配置 HAProxy 服务器来执行速率限制。HAProxy 服务器在 OVS 后端中的 L3 路由器和 DHCP 代理内运行，并在 OVN 后端中的元数据服务内运行。

先决条件

- 您可以访问 RHOSP Compute 节点以及更新配置文件的权限。
- 您的 RHOSP 环境使用 IPv4 网络。目前，网络服务不支持 IPv6 网络的元数据速率限制。
- 此流程要求您重启 OVN 元数据服务或 OVS 元数据代理。将这个活动调度到维护窗口，以最大程度降低任何潜在中断的操作影响。

流程

1. 在每个 Compute 节点上，在 /var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf 的 metadata_rate_limiting 部分中，为以下参数设置值：

   rate_limit_enabled

   可让您限制元数据请求的速度。默认值为 false。将值设为 true 以启用元数据速率限制。

   ip_versions

   IP 版本 4 用于您要控制查询率的元数据 IP 地址。RHOSP 尚不支持 IPv6 网络的元数据速率限制。

   base_window_duration

   查询请求限制的时间跨度（以秒为单位）。默认值为 10 秒。

   base_query_rate_limit

   在 base_window_duration 期间允许的最大请求数。默认值为 10 个请求。

   burst_window_duration

   允许请求高于 base_window_duration 的时间范围（以秒为单位）。默认值为 10 秒。

   burst_query_rate_limit

   burst_window_duration 期间允许的最大请求数。默认值为 10 个请求。

   #### 示例

   在本例中，网络服务配置 为基础 时间和速率，允许实例在 60 秒期间查询 IPv4 元数据服务 IP 地址 6 次。网络服务也配置为 突发 时间和速率，允许每个 10 秒内更高的 2 查询速率：

   ```
   [metadata_rate_limiting]
   rate_limit_enabled = True
   ip_versions = 4
   base_window_duration = 60
   ```

```
base_query_rate_limit = 6
burst_window_duration = 10
burst_query_rate_limit = 2
```

2. **重启元数据服务。**
   根据部署使用的网络服务机制，执行以下操作之一：

   ML2/OVN

   在 Compute 节点上，重启 tripleo_ovn_metadata_agent.service。

   ML2/OVS

   在 Compute 节点上，重启 tripleo_neutron_metadata_agent.service。

第 22 章 配置第 3 层高可用性(HA)

22.1. RHOSP 网络服务没有高可用性(HA)

没有高可用性(HA)功能的 Red Hat OpenStack Platform (RHOSP)网络服务部署容易受到物理节点故障的影响。

在典型的部署中，项目创建虚拟路由器，这些路由器被调度到物理网络服务层 3 (L3)代理节点上运行。当您丢失 L3 代理节点，而依赖的虚拟机随后丢失与外部网络的连接时，就会出现问题。任何浮动 IP 地址都不可用。此外，路由器主机的任何网络之间也会丢失连接。

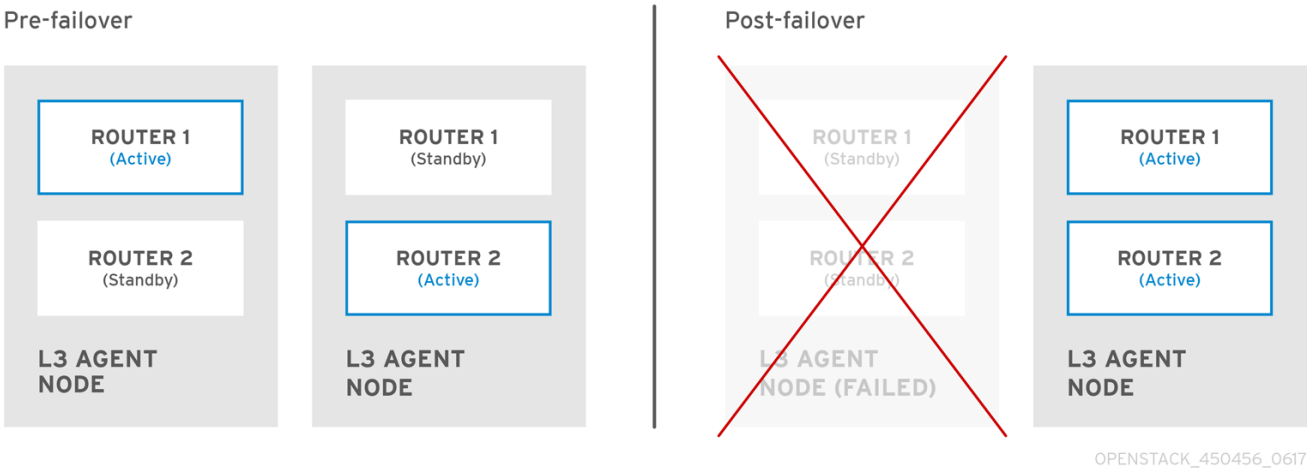
22.2. 第 3 层高可用性(HA)概述

这种主动/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。虚拟路由器在多个 Red Hat OpenStack Platform (RHOSP)网络服务节点上随机调度，一个被指定为 *活跃* 路由器，其余则以 *待机* 角色服务。

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 范围和外部网络访问。

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 已在对应的节点上调度了备份虚拟路由器，准备好在物理节点出现故障时恢复服务。当 L3 代理节点出现故障时，L3 HA 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：



在故障转移事件期间，通过浮动 IP 的实例 TCP 会话保持不受影响，并且迁移到新的 L3 节点而不中断。只有 SNAT 流量会受到故障转移事件的影响。

当处于主动/主动 HA 模式时，L3 代理进一步受到保护。

其他资源

- **虚拟路由器冗余协议(VRRP)**

22.3. 第 3 层高可用性(HA)故障转移状况

第 3 层(L3) Red Hat OpenStack Platform (RHOSP)网络服务的高可用性(HA)会在以下事件中自动重新调度受保护的资源：

- 由于硬件故障，网络服务 L3 代理节点关闭或断电。
- 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。

22.4. 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)配置发生在后端中，对项目不可见。项目可以继续创建和管理其虚拟路由器，但设计 L3 HA 实施时需要考虑一些限制：

- L3 HA 支持每个项目最多 255 个虚拟路由器。
- 内部 VRRP 消息在单独的内部网络中传输，为各个项目自动创建。此过程对用户透明地发生。
- 在 ML2/OVS 中实施高可用性(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。每个进程消耗大约 20MB 内存。默认情况下，每个 HA 路由器驻留在三个 L3 代理上，并在每个节点上消耗资源。因此，在调整 RHOSP 网络大小时，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。

22.5. 对 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 **l3_ha** 的默认配置。

- 对 neutron-server 的高可用性(HA)更改：
  - 第 3 层(L3) HA 会随机分配活跃角色，无论网络服务使用的调度程序（无论是随机或最小路由器）。
  - 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，提供负载平衡和 HA 功能。
  - IP 地址转换为 VIP。

22.6. 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 且没有使用分布式虚拟路由(DVR)时，Red Hat OpenStack Platform (RHOSP) director 会为虚拟路由器启用高可用性(HA)。使用 RHOSP Orchestration 服务(heat)参数 **max_l3_agents_per_router**，您可以设置调度 HA 路由器的 RHOSP 网络服务层 3 (L3)代理的最大数量。

先决条件

- 您的 RHOSP 部署不使用 DVR。
- 您至少部署了两个 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 自定义 overcloud 的各个方面，它是为 heat 模板 *提供自定义* 的特殊模板类型。

3. 在 YAML 环境文件中将 **NeutronL3HA** 参数设置为 **true**。这样可确保 HA 被启用，即使 director 默认没有设置它。

   ```
   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值和总数。（零值表示路由器被调度到每个代理上。）

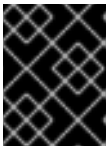
   示例

   ```
   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 代理保护每个 HA 虚拟路由器：一个活动，另一个备用。

   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 代理来扩展待机路由器的数量。对于您部署的每个新的 L3 代理节点，网络服务会调度额外的虚拟路由器备用版本，直到达到 **max_l3_agents_per_router** 限制为止。

5. 运行 **openstack overcloud deploy** 命令，并包含核心 heat 模板、环境文件以及新的自定义环境文件。



重要

但是，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更为优先。

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都默认为 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

- *自定义 Red Hat OpenStack Platform 部署*指南中的环境文件 https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/customizing_your_red_hat_openstack_platform_de the-overcloud-with-the-orchestration-service#con_environment-files_understanding-heat-templates
- 在自定义 *Red Hat OpenStack Platform 部署*指南中的 overcloud 创建中包括环境文件

22.7. 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，以返回结果中的高可用性(HA)设备，以前缀为 *ha-*。**

```
# 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 23 章 使用可用性区域使网络资源高度可用

从版本 16.2 开始，Red Hat OpenStack Platform (RHOSP)支持 RHOSP Networking 服务(neutron)可用区(AZ)。

AZs 可让您使 RHOSP 网络资源高度可用。您可以对附加到不同 AZ 上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。

网络服务 AZ 与计算服务(nova) AZ 结合使用，以确保客户使用本地到工作负载运行的物理站点的特定虚拟网络。有关分布式计算节点架构的更多信息，*请参阅部署分布式计算节点架构指南*。

23.1. 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** 和 **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，因此支持网络 AZ 是不必要的。

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 添加到 AZ hint 列表中。但是，在调度资源前，不会实际设置 AZ 属性。分配给网络资源的实际 AZ 可能与您在 hint 选项指定的 AZ 不同。此不匹配的原因可能是区失败，或者指定了区没有剩余容量。

如果用户无法在创建网络资源时指定 AZ，则可以为默认 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在 AZ 上调度网络和路由器。

其他资源

- 使用 ML2/OVS 配置网络服务可用域
- 使用 ML2/OVN 配置网络服务可用域
- 手动将可用区分配给网络和路由器

23.2. 为 ML2/OVS 配置网络服务可用区

当用户创建网络和路由器时，您可以设置一个或多个由 Red Hat OpenStack Platform (RHOSP)网络服务(neutron)自动分配的默认可用区(AZ)。另外，您还可以设置网络服务用来为对应 AZ 调度这些资源的网络和路由器驱动程序。

本主题中包含的信息适用于运行 RHOSP 网络服务的部署，该服务使用带有 Open vSwitch 机制驱动程序(ML2/OVS)的 Module Layer 2 插件。

先决条件

- 已部署 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，*请参阅部署分布式计算节点架构指南*。

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 Orchestration 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，它是为 heat 模板 *提供自定义* 的特殊模板类型。

3. 在 YAML 环境文件的 parameter_defaults 下，输入 NeutronDefaultAvailabilityZones 参数，以及一个或多个 AZ。如果用户无法在创建网络或路由器时通过 --availability-zone-hint 选项指定带有 --availability-zone-hint 选项的 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 Compute 服务 AZ 名称匹配。

   示例

   ```
   parameter_defaults:
     NeutronDefaultAvailabilityZones: 'az-central,az-datacenter2,az-datacenter1'
   ```

4. 通过输入参数 NeutronDhcpAgentAvailabilityZone 和 NeutronL3AgentAvailabilityZone，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。

5. **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这两者，请分别输入 NeutronNetworkSchedulerDriver 和 NeutronRouterSchedulerDriver 参数的新调度程序。**

   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'
```

6. **运行 openstack overcloud deploy 命令，并包含核心 heat 模板、环境文件以及新的自定义环境文件。**

重要

但是，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更为优先。

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 availability zone list 命令来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 配置网络服务可用域

- 手动将可用区分配给网络和路由器

23.3. 使用 ML2/OVN 配置网络服务可用域

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)。另外，您还可以设置网络服务用来为对应 AZ 调度这些资源的路由器驱动程序。

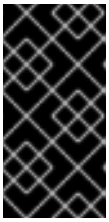
本主题中包含的信息是用于运行带有 Open Virtual Network (ML2/OVN)机制驱动程序的 Modular Layer 2 插件的部署。

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，因此支持网络 AZ 是不必要的。

先决条件

- 已部署 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，*请参阅部署分布式计算节点架构指南*。



重要

通过设置 **OVNCMSOptions: '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 Orchestration 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，它是为 heat 模板*提供自定义*的特殊模板类型。

3. 在 YAML 环境文件的 **parameter_defaults** 下，输入 **NeutronDefaultAvailabilityZones** 参数，以及一个或多个 AZ。

重要

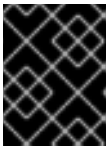
在 DCN 环境中，您必须将网络服务 AZ 名称与 Compute 服务 AZ 名称匹配。

如果用户无法在创建网络或路由器时通过 --availability-zone-hint 选项指定带有 **--availability-zone-hint** 选项的 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** 的值来确定网关节点的 AZ (Controller 和 Network 节点)。



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。

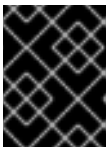
5. 默认情况下，路由器调度程序是 **AZLeastRoutersScheduler**。如果要更改此功能，请输入带有 **NeutronRouterSchedulerDriver** 参数的新调度程序。

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'
```

6. 运行 **openstack overcloud deploy** 命令，并包含核心 heat 模板、环境文件以及新的自定义环境文件。



重要

但是，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更为优先。

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 **availability zone list** 命令来确认可用区已正确定义。

  示例

  ```
  $ openstack availability zone list
  ```

  输出示例

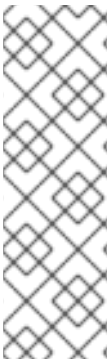
  ```
  +---------------+-------------+
  | Zone Name     | Zone Status |
  +---------------+-------------+
  | az-central    | available   |
  | az-datacenter1 | available   |
  | az-datacenter2 | available   |
  +---------------+-------------+
  ```

其他资源

- 关于网络服务可用区
- 使用 ML2/OVS 配置网络服务可用域
- 手动将可用区分配给网络和路由器

23.4. 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)。AZs 可让您使 RHOSP 网络资源高度可用。您可以对附加到不同 AZ 上不同电源源的网络节点进行分组，然后将运行关键服务的节点调度到单独的 AZ。



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分配给 RHOSP 编排服务(heat)参数的值的资源。如果没有为 **NeutronDefaultAvailabilityZones** 定义值，则在没有 AZ 属性的情况下调度资源。

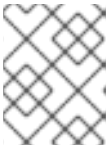
对于使用带有 Open vSwitch (ML2/OVS)机制驱动程序的 Modular Layer 2 插件的 RHOSP 网络服务代理，如果没有提供 AZ hint，且没有为 **NeutronDefaultAvailabilityZones** 指定的值，则使用 Compute 服务(nova) AZ 值来调度代理。

先决条件

- 已部署 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

- 在使用 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，因此支持网络 AZ 是不必要的。

在以下示例中，创建了一个网络(**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，会创建一个路由器(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 AZ。RHOSP 网络服务在调度资源时分配 AZ。

验证

- **输入适当的 OpenStack 客户端 show 命令，以确认资源在哪个区域中托管。**

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 配置网络服务可用域
- 使用 ML2/OVN 配置网络服务可用域