Red Hat OpenStack Platform 17.1

配置网络功能虚拟化

在 Red Hat Openstack Platform 中计划和配置网络功能虚拟化(NFV)

Last Updated: 2024-05-30

Red Hat OpenStack Platform 17.1 配置网络功能虚拟化

在 Red Hat Openstack Platform 中计划和配置网络功能虚拟化(NFV)

OpenStack Team
rhos-docs@redhat.com

法律通告

摘要

本指南包含重要的规划信息，并描述了红帽 OpenStack 平台部署中单一根输入/输出虚拟化 (SR-IOV) 和数据平面开发套件 (DPDK) 的配置流程。

目录

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 了解红帽网络功能虚拟化(NFV)

网络功能虚拟化(NFV) 是一种基于软件的解决方案，可帮助通信服务提供商(CSP)超越传统专有硬件，以实现更高的效率和灵活性，同时降低操作成本。

NFV 环境通过利用在标准硬件设备（如交换机、路由器和存储）上运行的标准虚拟化技术（如交换机、路由器和存储来虚拟化网络功能(VNF)）来提供 IT 和网络聚合。管理和编配逻辑部署并维持这些服务。NFV 还包括系统管理、自动化和生命周期管理，从而减少了手动工作。

1.1. NFV 的优点

实施网络功能虚拟化(NFV)的主要优点如下：

- 通过允许您快速部署和扩展新的网络服务来满足瞬息万变的需求，加速产品投放市场的时间。
- 通过使服务开发人员能够自我管理其资源和使用生产中使用的相同平台，支持创新。
- 在不牺牲安全性或性能的情况下，在数小时或数分钟内满足客户的需求，而不是几周或数天。
- 降低资本支出，因为它使用商用硬件，而非昂贵的定制设备。
- 使用简化的操作和自动化来优化日常任务，以提高员工生产力并降低操作成本。

1.2. NFV 部署支持的配置

您可以使用 Red Hat OpenStack Platform director 工具包来隔离特定的网络类型，如外部、项目、内部 API 等。您可以在单个网络接口上部署网络，或通过多个主机网络接口分发。使用 Open vSwitch，您可以通过为单个网桥分配多个接口来创建绑定。使用模板文件在 Red Hat OpenStack Platform 安装中配置网络隔离。如果没有提供模板文件，则服务网络会在 provisioning 网络上部署。

模板配置文件有两种类型：

- **network-environment.yaml**
  此文件包含 overcloud 节点的网络详细信息，如子网和 IP 地址范围。此文件还包含不同的设置，用于覆盖不同场景的默认参数值。
- 主机网络模板，如 compute.yaml 和 controller.yaml
  这些模板定义 overcloud 节点的网络接口配置。网络详情的值由 **network-environment.yaml** 文件提供。

这些 heat 模板文件位于 undercloud 节点上的 **/usr/share/openstack-tripleo-heat-templates/** 中。有关 NFV 的这些 heat 模板文件 示例，请参阅 DPDK SR-IOV YAML 文件示例。

硬件要求和软件要求部分提供了关于如何使用 Red Hat OpenStack Platform director 为 NFV 规划和配置 heat 模板文件的更多详细信息。

您可以编辑 YAML 文件来配置 NFV。有关 YAML 文件格式简介，请参阅 Nutshell 中的 YAML。

数据平面开发套件(DPDK)和单根 I/O 虚拟化(SR-IOV)

Red Hat OpenStack Platform (RHOSP)支持 NFV 部署，其中包含自动化 OVS-DPDK 和 SR-IOV 配置。

重要

红帽不支持将 OVS-DPDK 用于非 NFV 工作负载。如果您需要用于非 NFV 工作负载的 OVS-DPDK 功能，请联系您的大客户经理(TAM)或打开客户服务请求案例，以讨论支持例外和其他选项。要创建一个客户服务请求问题单，请访问创建一个问题单并选择 **Account > Customer Service Request**。

超融合基础架构(HCI)

您可以将 Compute 子系统与 Red Hat Ceph Storage 节点并置。这种超融合模式提供更低的条目成本、较小的初始部署空间、最大化容量利用率和 NFV 用例中的高效管理。有关 HCI 的更多信息，*请参阅部署超融合基础架构*。

可组合角色

您可以使用可组合角色创建自定义部署。可组合角色允许您为每个角色添加或删除服务。有关可组合角色的更多信息，请参阅自定义 *Red Hat OpenStack Platform 部署中的* 可组合服务和自定义角色。

使用 LACP 的 Open vSwitch (OVS)

从 OVS 2.9 开始，完全支持带有 OVS 的 LACP。对于 Openstack control plane 流量，我们不推荐这样做，因为 OVS 或 Openstack 网络中断可能会影响管理。如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 中的 Open vSwitch (OVS)绑定选项。

OVS 硬件卸载

Red Hat OpenStack Platform 支持部署 OVS 硬件卸载。有关使用硬件卸载部署 OVS 的详情，请参考配置 OVS 硬件卸载。

Open Virtual Network (OVN)

RHOSP 16.1.4 中提供以下 NFV OVN 配置：

- 使用 OVS-DPDK 和 SR-IOV 部署 OVN
- 使用 OVS TC 流卸载部署 OVN

1.3. NFV 数据平面连接

随着 NFV 的引入，更多网络供应商开始将其传统设备实施为 VNF。虽然大多数网络供应商正在考虑虚拟机，其中一些也将基于容器的方法作为设计选择进行调查。由于两个主要原因，基于 OpenStack 的解决方案应具有丰富且灵活：

- **应用就绪** - 网络供应商目前正在将其设备转换为 VNF。市场中的不同 VNF 具有不同的成熟度级别；这种就绪性的常见障碍包括启用 RESTful 接口，将数据模型演变为无状态，并提供自动化管理操作。OpenStack 应为所有提供一个通用的平台。
- **广泛的用例** - NFV 包括各种提供不同用例的应用程序。例如，Virtual Customer Premise Equipment (vCPE)旨在提供一些网络功能，如路由、防火墙、虚拟专用网络(VPN)，以及客户内部的网络地址转换(NAT)。虚拟 Evolved Packet Core (vEPC)是一种云架构，为 Long-Term Evolution (LTE)网络的核心组件提供经济的平台，允许动态置备网关和移动端点，以保持从智能手机和其他设备的数据流量增加。
  这些用例是使用不同的网络应用程序和协议实现的，并且需要基础架构中不同的连接、隔离和性能特性。在 control plane 接口和协议和实际转发平面之间是分开的。OpenStack 必须足够灵活，以提供不同的数据路径连接选项。

在原则上，提供数据平面连接虚拟机的两种常见方法：

- **直接硬件访问** 会绕过 linux 内核，并使用 PCI Passthrough 或单个根 I/O 虚拟化(SR-IOV)等技术为虚拟功能(VF)和物理功能(PF)通过提供安全直接内存访问(DMA)。

- **使用虚拟交换机(vswitch)**，作为虚拟机监控程序的软件服务实施。虚拟机使用虚拟接口(vNIC)连接到 vSwitch，vSwitch 能够在虚拟机之间转发流量，以及虚拟机和物理网络之间的流量。

一些快速数据路径选项如下：

- **单根 I/O 虚拟化(SR-IOV)** 是一种标准，使单个 PCI 硬件设备显示为多个虚拟 PCI 设备。它的工作原理是引入物理功能(PF)，它们是代表物理硬件端口的全功能 PCIe 功能，以及分配给虚拟机的轻量级功能(VF)。对于虚拟机，VF 类似于与硬件直接通信的常规 NIC。NIC 支持多个 VF。

- **Open vSwitch (OVS)** 是一个开源软件交换机，设计为在虚拟化服务器环境中用作虚拟交换机。OVS 支持常规 L2-L3 交换机的功能，并提供对 OpenFlow 等 SDN 协议的支持，以创建用户定义的覆盖网络（如 VXLAN）。OVS 使用 Linux 内核网络来利用物理 NIC 在虚拟机和主机之间切换数据包。OVS 现在支持具有内置防火墙功能的连接跟踪(Conntrack)，以避免使用 iptables/ebtables 的 Linux 网桥的开销。Red Hat OpenStack Platform 环境的 Open vSwitch 提供默认的 OpenStack Networking (neutron)与 OVS 集成。

- **数据平面开发套件(DPDK)** 由一组库和轮询模式驱动程序(PMD)组成，用于快速处理数据包。它设计为在用户空间中运行，使应用程序能够直接从或到 NIC 执行自己的数据包处理。DPDK 可减少延迟并允许处理更多数据包。DPDK Poll Mode Drivers (PMD)在忙碌循环中运行，持续扫描客户机中主机和 vNIC 端口的 NIC 端口，以获得数据包。

- **DPDK 加速 Open vSwitch (OVS-DPDK)** 是与 DPDK 捆绑的 Open vSwitch，用于使用 Linux 内核绕过高性能用户空间解决方案，并将内存访问(DMA)直接传递给物理 NIC。其理念是将标准 OVS 内核数据路径替换为基于 DPDK 的数据路径，并在使用 DPDK 内部进行数据包转发的主机上创建用户空间 vSwitch。这种架构的优点是它对用户来说是透明的。它公开的接口（如 OpenFlow、OVSDB、命令行）保持不变。

1.4. ETSI NFV 架构

欧洲电信标准研究所(ETSI)是一个独立的标准化组，其开发了欧洲信息和通信技术(ICT)标准标准。

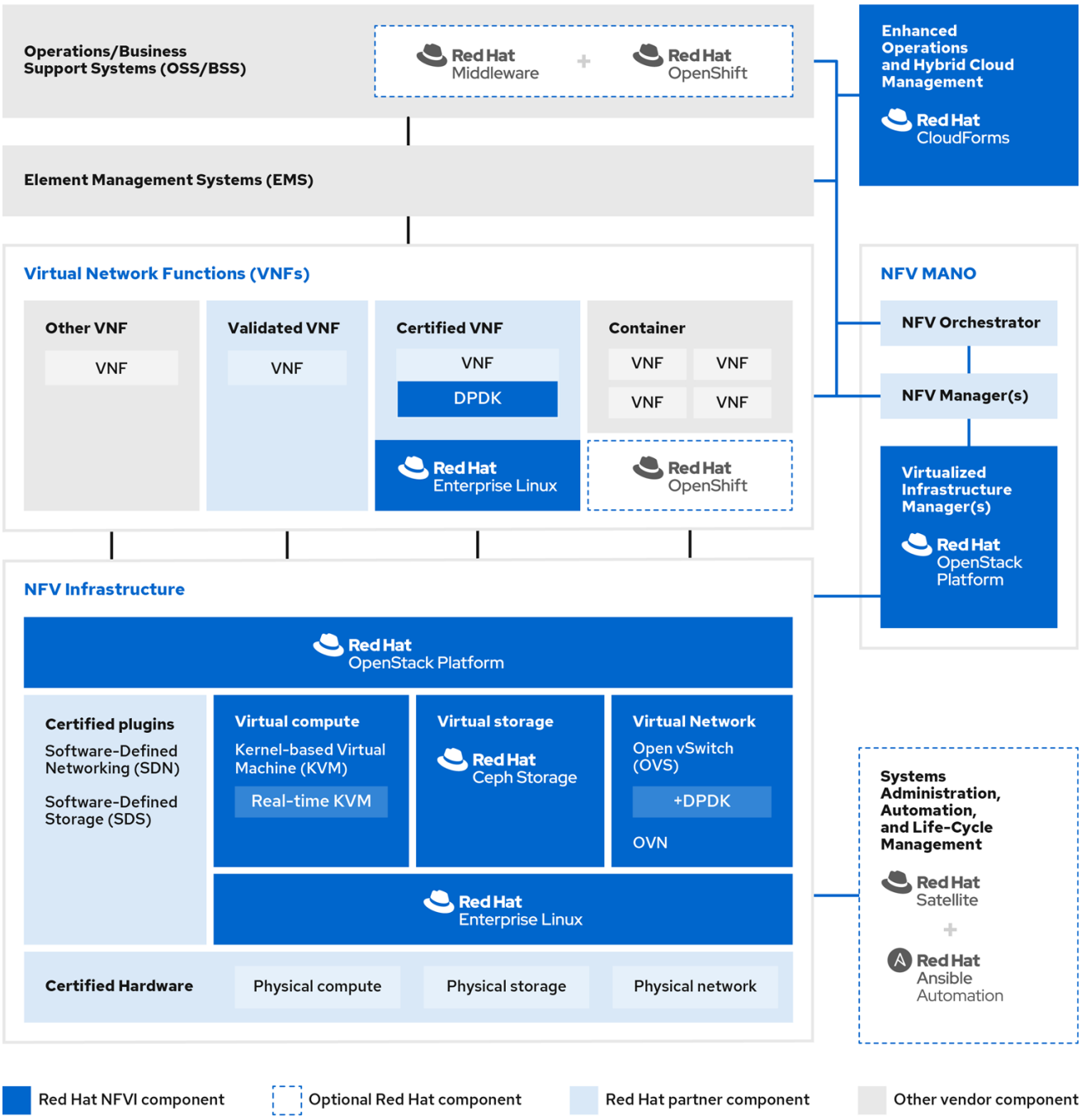
网络功能虚拟化(NFV)侧重于解决使用专有硬件设备中涉及的问题。借助 NFV，安装网络特定设备的要求会降低，具体取决于用例要求和经济优势。网络功能虚拟化(ETSI ISG NFV)的 ETSI 行业规范组设定了确保支持虚拟化功能所需的要求、参考架构和基础架构规格。

红帽提供基于开源的云优化解决方案，以帮助通信服务提供商(CSP)实现 IT 和网络融合。红帽向 Red Hat OpenStack 添加了带有数据平面开发套件(OVS-DPDK)的单根 I/O 虚拟化(SR-IOV)和 Open vSwitch 等 NFV 功能。

1.5. NFV ETSI 架构和组件

通常，网络功能虚拟化(NFV)平台具有以下组件：

图 1.1. NFV ETSI 架构和组件



- **虚拟化网络功能(VNF)** - 路由器、防火墙、负载均衡器、宽带网关、移动数据包处理器、服务节点、信号、位置服务和其他网络功能的软件实现。

- **NFV 基础架构(NFVi)** - 组成基础架构的物理资源（计算、存储、网络）和虚拟化层。网络包括用于在虚拟机和主机之间转发数据包的数据路径。这样，您可以安装 VNF，而不必关注底层硬件的详细信息。NFVI 构成 NFV 堆栈的基础。NFVI 支持多租户，并由虚拟基础架构管理器(VIM)管理。增强的平台感知(EPA)通过向 VNF 公开低级别 CPU 和 NIC 加速组件来提高虚拟机数据包转发性能（吞吐量、延迟、jitter）。

- **NFV 管理和编排(MANO)** - 管理和编排层侧重于 VNF 的整个生命周期中所需的所有服务管理任务。MANO 的主要目标是允许将操作器提供的网络功能与物理基础架构分离的服务定义、自动化、错误关联、监控和生命周期管理。这种分离需要额外的管理层，由虚拟网络功能管理器(VNFM)提供。VNFM 通过直接与它们交互或通过 VNF 供应商提供的元素管理系统(EMS)来管理

虚拟机和 VNF 的生命周期。MANO 定义的其他重要组件是 Orchestrator，也称为 NFVO。带有各种数据库和系统的 NFVO 接口，包括操作/业务支持系统(OSS/BSS)和底部的 VNFM。如果 NFVO 希望为客户创建新服务，它将要求 VNFM 触发 VNF 的实例化，这可能会导致多个虚拟机。

- **运营和业务支持系统(OSS/BSS)** 提供基本的业务功能应用程序，例如运营支持和计费。OSS/BSS 需要适应 NFV，并集成传统系统和新的 MANO 组件。BSS 系统根据服务订阅设定策略，并管理报告和计费。

- **系统管理、自动化和生命周期管理 - 管理系统管理、基础架构组件自动化和** NFVi 平台的生命周期。

1.6. RED HAT NFV 组件

红帽 NFV 解决方案包括一系列产品，可在 ETSI 模型中充当 NFV 框架的不同组件。来自红帽产品组合的以下产品集成到 NFV 解决方案中：

- Red Hat OpenStack Platform - 支持 IT 和 NFV 工作负载。增强平台感知(EPA)功能通过支持 SR-IOV 和 OVS-DPDK 的 CPU 固定、巨页、非统一内存访问(NUMA)关联性和网络适应器(NIC)提供确定的性能改进。

- Red Hat Enterprise Linux 和 Red Hat Enterprise Linux Atomic Host - 创建虚拟机和容器作为 VNF。

- Red Hat Ceph Storage - 为服务提供商工作负载的所有需求提供统一弹性和高性能存储层。

- 红帽 JBoss 中间件和 OpenShift Enterprise （可选）提供 OSS/BSS 组件现代化的能力。

- 红帽 CloudForms - 提供 VNF 管理器，并在统一显示中显示来自多种来源的数据，如 VIM 和 NFVi。

- Red Hat Satellite 和 Ansible - （可选）提供增强的系统管理、自动化和生命周期管理。

1.7. NFV 安装摘要

Red Hat OpenStack Platform director 安装和管理完整的 OpenStack 环境。director 基于上游 OpenStack TripleO 项目，该项目是"OpenStack-On-OpenStack"的缩写。此项目利用 OpenStack 组件安装完全可正常运行的 OpenStack 环境；这包括一个名为 undercloud 的最小 OpenStack 节点。undercloud 调配和控制 overcloud （一系列裸机系统用作生产 OpenStack 节点）。director 提供了一个简单的方法，用于安装精益且可靠的完整 Red Hat OpenStack Platform 环境。

有关安装 undercloud 和 overcloud 的更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform*。

要安装 NFV 功能，请完成以下步骤：

- 在 **network-environment.yaml** 文件中包含 SR-IOV 和 PCI Passthrough 参数，更新 **post-install.yaml** 文件以进行 CPU 调优，修改 **compute.yaml** 文件，并运行 **overcloud_deploy.sh** 脚本来部署 overcloud。

- 通过直接从 NIC 轮询数据，安装 DPDK 库和驱动程序以便快速处理数据包。在 **network-environment.yaml** 文件中包括 DPDK 参数，为 CPU 调整更新 **post-install.yaml** 文件，更新 **compute.yaml** 文件以使用 DPDK 端口设置网桥，更新 **controller.yaml** 文件来设置配置了 VLAN 的网桥和接口，并运行 **overcloud_deploy.sh** 脚本来部署 overcloud。

第 2 章 NFV 性能注意事项

要使网络功能虚拟化(NFV)解决方案非常有用，其虚拟化功能必须满足或超过物理实施的性能。红帽的虚拟化技术基于高性能基于内核的虚拟机(KVM)管理程序，在 OpenStack 和云部署中很常见。

Red Hat OpenStack Platform director 配置 Compute 节点以强制实施资源分区并微调，以实现客户机虚拟网络功能(VNF)的线性能。NFV 用例中的主要性能因素包括吞吐量、延迟和 jitter。

您可以使用数据平面开发套件(DPDK)加速虚拟机在物理 NIC 和虚拟机之间启用高性能数据包切换。OVS 2.10 嵌入了对 DPDK 17 的支持，并包括对 vhost-user 多队列的支持，从而允许扩展性能。OVS-DPDK 为客户机 VNF 提供行式性能。

单根 I/O 虚拟化(SR-IOV)网络提供增强的性能，包括提高特定网络和虚拟机的吞吐量。

性能调优的其他重要功能包括巨页、NUMA 对齐、主机隔离和 CPU 固定。VNF 类别需要大页面和仿真程序线程隔离，才能提高性能。主机隔离和 CPU 固定提高了 NFV 性能，并防止伪装的数据包丢失。

2.1. CPU 和 NUMA 节点

在以前的版本中，x86 系统上的所有内存都会被系统中所有 CPU 相等访问。这会导致内存访问时间相同，无论系统中的 CPU 正在执行该操作，并被称为 Uniform Memory Access (UMA)。

在非统一内存访问(NUMA)中，系统内存被分成名为 nodes （分配给特定 CPU 或套接字）的区域。对 CPU 本地访问的内存比连接到该系统上的远程 CPU 的内存要快。通常，NUMA 系统上的每个套接字都有一个本地内存节点，其内容可以比节点本地到另一个 CPU 的内存或所有 CPU 共享的总线上的内存快。

同样，物理 NIC 放置在 Compute 节点硬件上的 PCI 插槽中。这些插槽连接到与特定 NUMA 节点关联的特定 CPU 套接字。为获得最佳性能，请将您的数据路径 NIC 连接到 CPU 配置(SR-IOV 或 OVS-DPDK)中的同一 NUMA 节点。

NUMA 丢失的性能影响显著，通常从 10% 的性能命中或更高性能开始。每个 CPU 套接字可以有多个 CPU 内核，这些内核被视为用于虚拟化目的的独立 CPU。

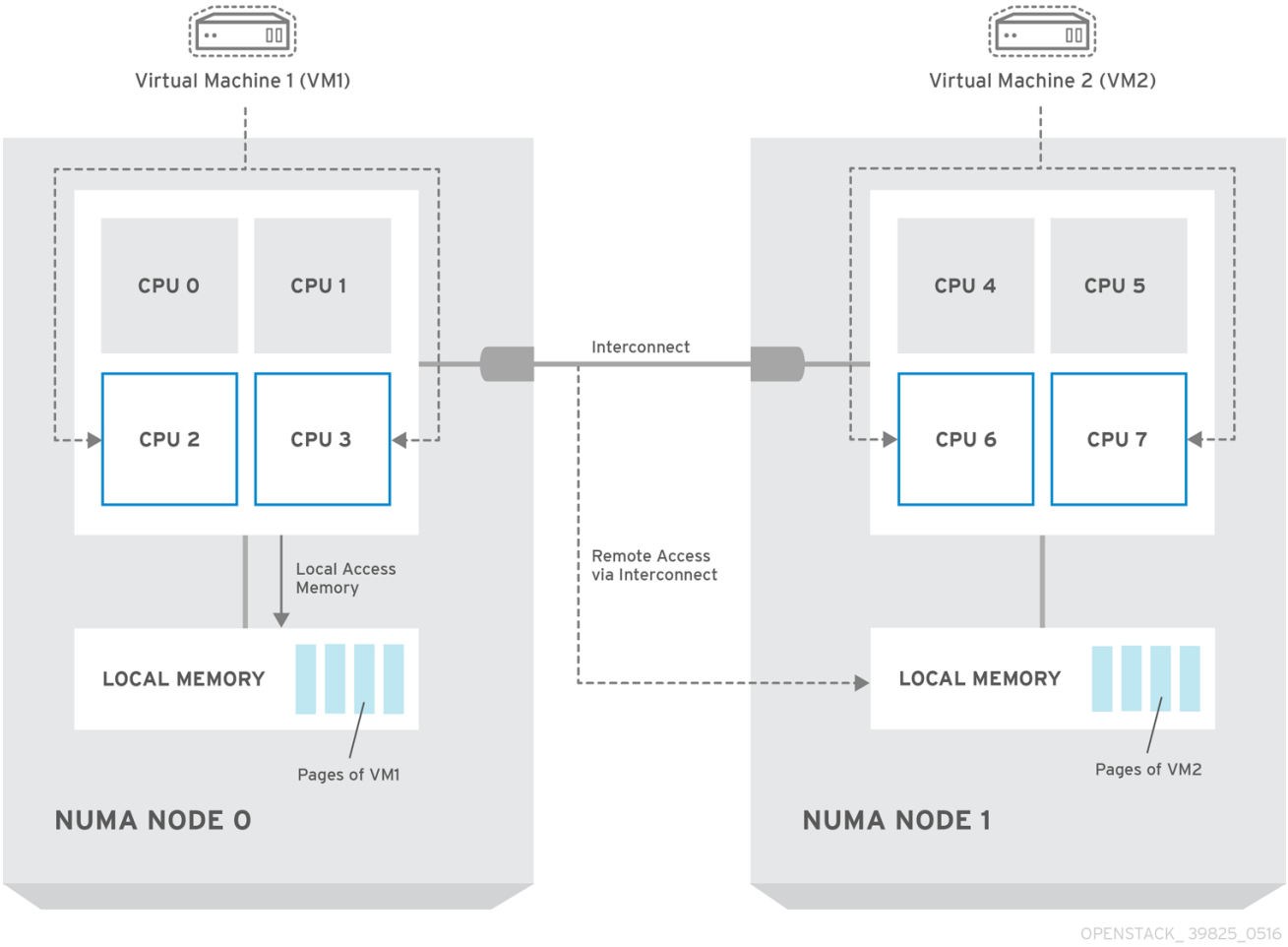
提示

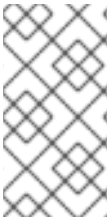
有关 NUMA 的更多信息，请参阅 NUMA 是什么以及如何在 Linux 上使用？

2.1.1. NUMA 节点示例

下图显示了双节点 NUMA 系统的示例，以及 CPU 内核和内存页面可用的方式：

图 2.1. 示例：双节点 NUMA 系统





注意

只有在来自 NUMA 节点 0 中的 VM1 中存在一个 NUMA 节点 1 中的 CPU 内核时，才可通过 Interconnect 访问远程内存。在这种情况下，NUMA 节点 1 的内存充当 VM1 第三个 CPU 内核的本地（例如，如果 VM1 在上图中使用 CPU 4 分配），但同时充当同一虚拟机其他 CPU 内核的远程内存。

2.1.2. NUMA 感知实例

您可以将 OpenStack 环境配置为使用 NUMA 架构的系统上的 NUMA 拓扑感知。在虚拟机(VM)中运行客户机操作系统时，涉及两个 NUMA 拓扑：

- 主机物理硬件的 NUMA 拓扑
- 向客户端操作系统公开的虚拟硬件的 NUMA 拓扑

您可以通过将虚拟硬件与物理硬件 NUMA 拓扑保持一致来优化客户端操作系统的性能。

2.2. CPU 固定

CPU 固定是在给定主机上在特定物理 CPU 上运行特定虚拟机的虚拟 CPU 的功能。vCPU 固定为裸机系统上任务固定提供了类似的优势。由于虚拟机作为用户空间任务在主机操作系统上运行，因此固定可提高缓存效率。

有关如何配置 CPU 固定的详情，请参考 配置 Compute *服务中的*

link:https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/configuring_the_compute_service_for_instance_creation/assembl cpus-on-compute-nodes#assembly_configuring-cpu-pinning-on-compute-nodes_cpu-pinning [在 Compute 节点上配置 CPU 固定]。

2.3. 巨页

物理内存被分成连续区域，称为页面。为提高效率，系统通过访问整个页面而不是单个字节内存来检索内存。要执行此转换，系统会在 Translation Lookaside Buffers (TLB)中查找包含最新或常用页面的物理到虚拟地址映射。当系统无法在 TLB 中找到映射时，处理器必须遍历所有页表以确定地址映射。优化 TLB 以最大程度降低这些 TLB 丢失期间发生的性能损失。

x86 系统中的典型页面大小为 4KB，其它更大的页面大小可用。更大的页面大小意味着总页面减少，因此增加了其虚拟到存储在 TLB 中的物理地址转换的系统内存量。因此，这可以减少 TLB 未命中，这会提高性能。对于较大的页面大小，内存利用率增加，因为进程必须在页面中分配，但不一定需要所有内存。因此，在提供更快的访问时间与较大的页面之间，选择页大小会很折现，并确保使用较小的页面的最大内存使用率。

2.4. 端口安全性

端口安全性是一种反欺骗措施，阻止任何与原始网络端口的源 IP 和源 MAC 地址不匹配的出口流量。您不能使用安全组规则查看或修改此行为。

默认情况下，在 OpenStack 中新创建的 Neutron 网络上会将 **port_security_enabled** 参数设置为 **enabled**。新创建的端口从创建它们的网络中复制 **port_security_enabled** 参数的值。

对于某些 NFV 用例，如构建防火墙或路由器，您必须禁用端口安全性。

要在单一端口上禁用端口安全性，请运行以下命令：

```
openstack port set --disable-port-security <port-id>
```

要防止端口安全性在网络上任何新创建的端口上启用，请运行以下命令：

```
openstack network set --disable-port-security <network-id>
```

第 3 章 NFV 的硬件要求

本节介绍 NFV 的硬件要求。

红帽认证用于 Red Hat OpenStack Platform 的硬件。如需更多信息，请参阅 认证硬件。

3.1. 为 NFV 测试的 NIC

有关 NFV 测试的 NIC 列表，请参阅红帽知识库解决方案 网络适配器 Fast Datapath 功能支持列表。

将默认驱动程序用于支持的 NIC，除非您在 NVIDIA (Mellanox)网络接口上配置 OVS-DPDK。对于 NVIDIA 网络接口，您必须在 j2 网络配置模板中设置对应的内核驱动程序。

示例

在本例中，对 Mellanox ConnectX-5 网络接口设置了 **mlx5_core** 驱动程序：

```
members
 - type: ovs_dpdk_port
   name: dpdk0
   driver: mlx5_core
   members:
   - type: interface
     name: enp3s0f0
```

3.2. 对硬件卸载的故障排除

在 Red Hat OpenStack Platform (RHOSP) 17.1 部署中，带有 **switchdev**-enabled 端口和 Mellanox ConnectX5 NIC 的虚拟机可能无法卸载流。要排除并配置卸载流，请禁用 **ESWITCH_IPV4_TTL_MODIFY_ENABLE** Mellanox 固件参数。有关 RHOSP 17.1 中的 OVS Hardware Offload 的更多信息，请参阅 OpenStack Platform 16.2 中带有 Mellanox NIC 的红帽知识库解决方案 OVS Hardware Offload。

流程

1. 在带有您要配置的 Mellanox NIC 的 RHOSP 部署中登录到 Compute 节点。
2. 使用 **mstflint** 工具查询 **ESWITCH_IPV4_TTL_MODIFY_ENABLE** Mellanox 固件参数。

```
[root@compute-1 ~]# yum install -y mstflint
[root@compute-1 ~]# mstconfig -d <PF PCI BDF> q
ESWITCH_IPV4_TTL_MODIFY_ENABLE
```

3. 如果启用了 **ESWITCH_IPV4_TTL_MODIFY_ENABLE** 参数，并将其设置为 **1**，然后将值设为 **0** 以禁用它。

```
[root@compute-1 ~]# mstconfig -d <PF PCI BDF> s
ESWITCH_IPV4_TTL_MODIFY_ENABLE=0`
```

4. 重新引导节点。

3.3. 发现 NUMA 节点拓扑

在规划部署时，您必须了解 Compute 节点的 NUMA 拓扑，以对 CPU 和内存资源进行分区，以获得最佳性能。要确定 NUMA 信息，请执行以下任务之一：

- 启用硬件内省以从裸机节点检索此信息。
- 登录每个裸机节点，以手动收集信息。

注意

您必须安装并配置 undercloud，然后才能通过硬件内省检索 NUMA 信息。有关 undercloud 配置的更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform*。

3.4. 获取硬件内省详细信息

裸机服务 hardware-inspection-extras 功能默认启用，您可以使用它来检索 overcloud 配置的硬件详情。有关 **undercloud.conf** 文件中的 **inspection_extras** 参数的更多信息，请参阅 Director 配置参数。

例如，**numa_topology** 收集程序就是硬件检查额外功能的一部分，包括每个 NUMA 节点的以下信息：

- RAM（单位为 KB）
- 物理 CPU 内核数和同级线程数
- 和 NUMA 节点关联的 NIC

流程

- 要获得以上列出的信息，请使用裸机节点的 UUID 替换 <UUID> 来完成以下命令：

```
# openstack baremetal introspection data save <UUID> | jq .numa_topology
```

以下示例显示获取的裸机节点 NUMA 信息：

```
{
  "cpus": [
    {
      "cpu": 1,
      "thread_siblings": [
        1,
        17
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        10,
        26
      ],
      "numa_node": 1
    },
    {
      "cpu": 0,
      "thread_siblings": [
```

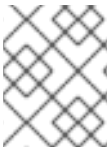
```
    0,
    16
   ],
   "numa_node": 0
  },
  {
   "cpu": 5,
   "thread_siblings": [
    13,
    29
   ],
   "numa_node": 1
  },
  {
   "cpu": 7,
   "thread_siblings": [
    15,
    31
   ],
   "numa_node": 1
  },
  {
   "cpu": 7,
   "thread_siblings": [
    7,
    23
   ],
   "numa_node": 0
  },
  {
   "cpu": 1,
   "thread_siblings": [
    9,
    25
   ],
   "numa_node": 1
  },
  {
   "cpu": 6,
   "thread_siblings": [
    6,
    22
   ],
   "numa_node": 0
  },
  {
   "cpu": 3,
   "thread_siblings": [
    11,
    27
   ],
   "numa_node": 1
  },
  {
   "cpu": 5,
   "thread_siblings": [
```

```
        5,
        21
      ],
      "numa_node": 0
    },
    {
      "cpu": 4,
      "thread_siblings": [
        12,
        28
      ],
      "numa_node": 1
    },
    {
      "cpu": 4,
      "thread_siblings": [
        4,
        20
      ],
      "numa_node": 0
    },
    {
      "cpu": 0,
      "thread_siblings": [
        8,
        24
      ],
      "numa_node": 1
    },
    {
      "cpu": 6,
      "thread_siblings": [
        14,
        30
      ],
      "numa_node": 1
    },
    {
      "cpu": 3,
      "thread_siblings": [
        3,
        19
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        2,
        18
      ],
      "numa_node": 0
    }
  ],
  "ram": [
    {
```

```
      "size_kb": 66980172,
      "numa_node": 0
     },
     {
      "size_kb": 67108864,
      "numa_node": 1
     }
    ],
    "nics": [
     {
      "name": "ens3f1",
      "numa_node": 1
     },
     {
      "name": "ens3f0",
      "numa_node": 1
     },
     {
      "name": "ens2f0",
      "numa_node": 0
     },
     {
      "name": "ens2f1",
      "numa_node": 0
     },
     {
      "name": "ens1f1",
      "numa_node": 0
     },
     {
      "name": "ens1f0",
      "numa_node": 0
     },
     {
      "name": "eno4",
      "numa_node": 0
     },
     {
      "name": "eno1",
      "numa_node": 0
     },
     {
      "name": "eno3",
      "numa_node": 0
     },
     {
      "name": "eno2",
      "numa_node": 0
     }
    ]
}
```

3.5. NFV BIOS 设置

下表描述了 NFV 所需的 BIOS 设置：



注意

您必须在 BIOS 中启用 SR-IOV 全局和 NIC 设置，或使用 SR-IOV Compute 节点的 Red Hat OpenStack Platform (RHOSP)部署将失败。

表 3.1. BIOS 设置

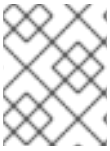
参数	设置
C3 Power State	已禁用。
C6 Power State	已禁用。
MLC Streamer	enabled。
MLC Spatial Prefetcher	enabled。
DCU Data Prefetcher	enabled。
DCA	enabled。
CPU Power 和性能	性能。
Memory RAS 和 Performance Config → NUMA Optimized	enabled。
turbo Boost	在需要确定性能的 NFV 部署中禁用。 在所有其他场景中启用。
VT-d	如果需要 VFIO 功能，则启用了 Intel 卡。
NUMA 内存交错	已禁用。

在使用 **intel_idle** 驱动程序的处理器中，Red Hat Enterprise Linux 可以忽略 BIOS 设置并重新启用处理器 C-state。

您可以通过在内核引导命令行中指定键值对 **intel_idle.max_cstate=0** 来禁用 **intel_idle**，并使用 **acpi_idle** 驱动程序进行替代。

通过检查 **current_driver** 的内容，确认处理器正在使用 **acpi_idle** 驱动程序：

```
# cat /sys/devices/system/cpu/cpuidle/current_driver
acpi_idle
```



注意

在更改驱动程序后，您会遇到一些延迟，因为它需要时间才能启动 Tuned 守护进程。但是，在 Tuned 加载后，处理器不使用更深的 C-state。

第 4 章 NFV 的软件要求

本节介绍支持的配置和驱动程序，以及 NFV 所需的订阅详细信息。

4.1. 注册并启用软件仓库

要安装 Red Hat OpenStack Platform，您必须使用 Red Hat Subscription Manager 注册 Red Hat OpenStack Platform director，并订阅所需的频道。有关注册和更新 undercloud 的更多信息， *请参阅使用 director 安装和管理 Red Hat OpenStack Platform 中的注册 undercloud 和 附加订阅*。

流程

1. 使用 Content Delivery Network 注册您的系统，在提示时输入您的客户门户网站用户名和密码。

```
[stack@director ~]$ sudo subscription-manager register
```

2. 确定 Red Hat OpenStack Platform director 的权利池 ID，如以下命令和输出中的 {Pool ID}：

```
[stack@director ~]$ sudo subscription-manager list --available --all --matches="Red Hat OpenStack"
Subscription Name:   Name of SKU
Provides:            Red Hat Single Sign-On
                     Red Hat Enterprise Linux Workstation
                     Red Hat CloudForms
                     Red Hat OpenStack
                     Red Hat Software Collections (for RHEL Workstation)
SKU:                 SKU-Number
Contract:            Contract-Number
Pool ID:             {Pool-ID}-123456
Provides Management: Yes
Available:           1
Suggested:           1
Service Level:       Support-level
Service Type:        Service-Type
Subscription Type:   Sub-type
Ends:                End-date
System Type:         Physical
```

3. 在以下命令中包括 **Pool ID** 值，以附加 Red Hat OpenStack Platform 17.1 权利。

```
[stack@director ~]$ sudo subscription-manager attach --pool={Pool-ID}-123456
```

4. 禁用默认存储库。

```
subscription-manager repos --disable=*
```

5. 为带有 NFV 的 Red Hat OpenStack Platform 启用所需的存储库。

```
$ sudo subscription-manager repos --enable=rhel-9-for-x86_64-baseos-eus-rpms \
--enable=rhel-9-for-x86_64-appstream-eus-rpms \
--enable=rhel-9-for-x86_64-highavailability-eus-rpms \
--enable=ansible-2.9-for-rhel-9-x86_64-rpms \
```

```
--enable=openstack-17.1-for-rhel-9-x86_64-rpms \
--enable=rhel-9-for-x86_64-nfv-rpms \
--enable=fast-datapath-for-rhel-9-x86_64-rpms
```

6. 更新您的系统，以便您获得最新的基本系统软件包。

```
[stack@director ~]$ sudo dnf update -y
[stack@director ~]$ sudo reboot
```

4.2. NFV 部署支持的配置

Red Hat OpenStack Platform (RHOSP)使用 director 支持以下 NFV 部署：

- 单根 I/O 虚拟化(SR-IOV)
  如需更多信息，请参阅配置 SR-IOV。
- Open vSwitch 硬件卸载
  如需更多信息，请参阅配置 OVS 硬件卸载。
- Open vSwitch 带有数据平面开发套件(OVS-DPDK)
  如需更多信息，请参阅配置 OVS-DPDK 部署。

另外，您可以使用以下任一功能部署 RHOSP：

- 实施可组合服务和自定义角色。
  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*。
- 在同一主机上并置计算和 Ceph 存储服务。
  如需更多信息，*请参阅部署超融合基础架构*。
- 配置 Red Hat Enterprise Linux Real Time KVM (RT-KVM)。
  如需更多信息，请参阅为 NFV 工作负载启用 RT-KVM。

4.3. NFV 支持的驱动程序

有关支持的驱动程序的完整列表，请参阅 Red Hat OpenStack Platform 中的组件、插件和驱动程序支持。

有关使用 NFV 的 Red Hat OpenStack Platform 部署测试的 NIC 列表，请参阅为 NFV 测试的 NIC。

4.4. 与第三方软件兼容

有关与 Red Hat OpenStack Platform 进行测试、支持和认证以与 Red Hat OpenStack Platform 兼容的第三方软件的完整列表，请参阅与 Red Hat OpenStack Platform 兼容的第三方软件。您可以根据产品版本和软件类别过滤列表。

有关使用 Red Hat Enterprise Linux 测试、支持和认证的产品和服务的完整列表，请参阅 与 Red Hat Enterprise Linux 兼容的第三方软件。您可以根据产品版本和软件类别过滤列表。

第 5 章 NFV 的网络注意事项

undercloud 主机至少需要以下网络：

- 置备网络 - 提供 DHCP 和 PXE 引导功能，以帮助发现 overcloud 中使用的裸机系统。
- 外部网络 - 与所有节点的远程连接的独立网络。连接到此网络的接口需要一个可路由的 IP 地址（静态定义），或者从外部 DHCP 服务动态生成。

最小 overcloud 网络配置包括以下 NIC 配置：

- 单 NIC 配置 - 一个 NIC 在原生 VLAN 中用于 Provisioning 网络，并用于 tagged VLAN（使用子网处理不同的 overcloud 网络类型）。
- 双 NIC 配置 - 一个 NIC 用于 provisioning 网络，另一个用于外部网络的 NIC。
- 双 NIC 配置 - 一个 NIC 在原生 VLAN 中用于 provisioning 网络，另一个 NIC 用于标记的 VLAN，用于不同 overcloud 网络类型的子网。
- 多 NIC 配置 - 每个 NIC 都使用一个子网来分别处理 overcloud 中不同的网络类型。

如需有关网络要求的更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 中的 准备 undercloud 网络。

第 6 章 规划 SR-IOV 部署

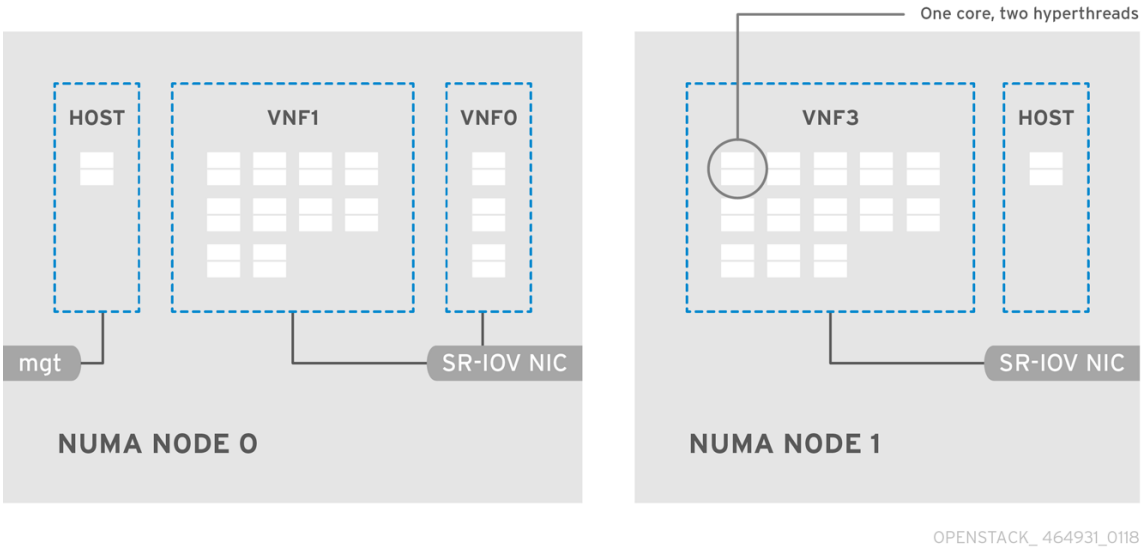
通过根据您的 Compute 节点硬件设置单个参数，为 NFV 优化单根 I/O 虚拟化(SR-IOV)部署。

要评估您对 SR-IOV 参数的硬件影响，请参阅发现 NUMA 节点拓扑。

6.1. SR-IOV 部署的硬件分区

要使用 SR-IOV 实现高性能，对主机和客户机之间的资源进行分区。

图 6.1. NUMA 节点拓扑

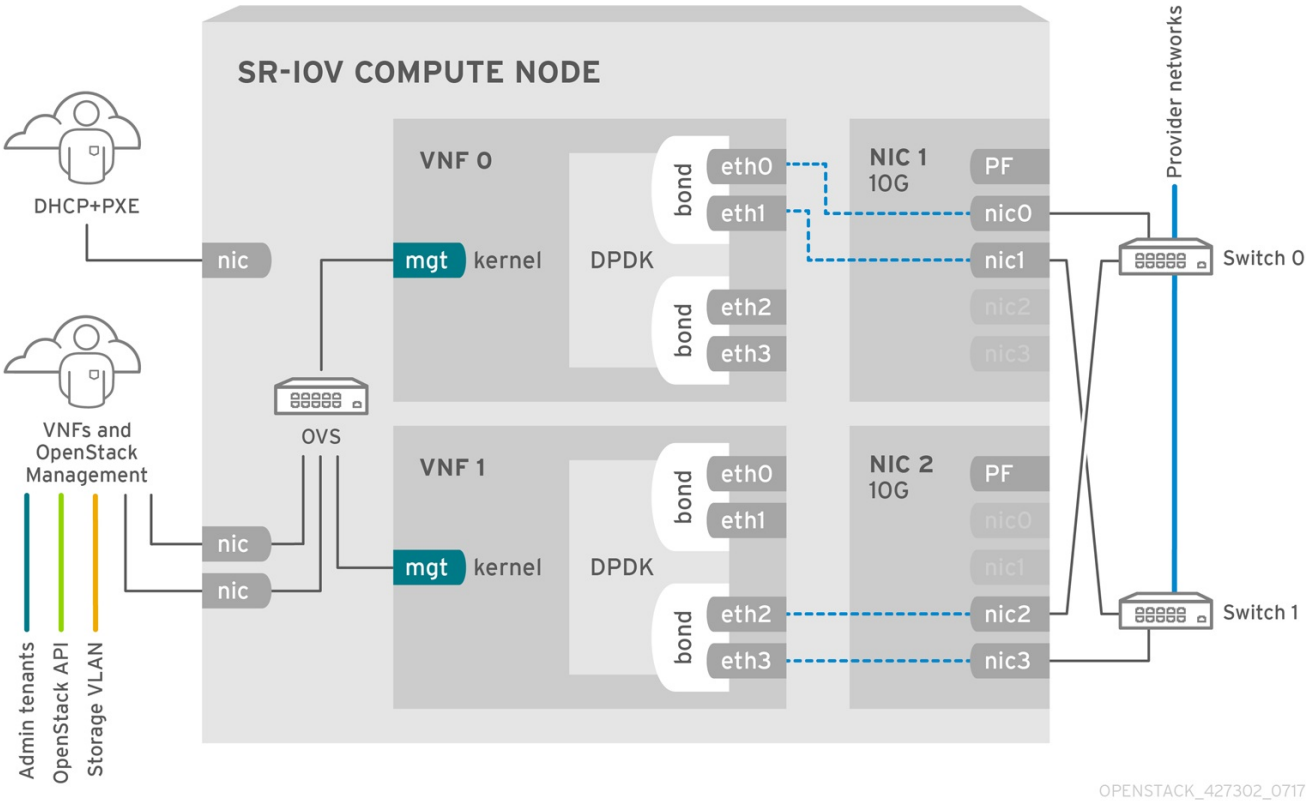


典型的拓扑包括双套接字 Compute 节点上每个 NUMA 节点 14 个内核。支持超线程(HT)和非HT 内核。每个内核有两个同级线程。一个内核专用于每个 NUMA 节点上的主机。虚拟网络功能(VNF)处理 SR-IOV 接口绑定。所有中断请求(IRQ)在主机内核中路由。VNF 核心专用于 VNF。它们提供与其他 VNF 隔离和与主机隔离。每个 VNF 都必须使用单一 NUMA 节点上的资源。VNF 使用的 SR-IOV NIC 还必须与同一 NUMA 节点关联。这个拓扑没有虚拟化开销。主机、OpenStack 网络(neutron)和计算(nova)配置参数在单个文件中公开，以简化、一致性，并避免严重隔离，从而导致抢占和数据包丢失。主机和虚拟机隔离取决于 **tuned** 配置集，它根据隔离的 CPU 列表定义引导参数和任何 Red Hat OpenStack Platform 修改。

6.2. NFV SR-IOV 部署拓扑

下图各自有两个 VNF，它们分别带有由 **mgt** 和 data plane 接口表示的管理接口。管理界面管理 **ssh** 访问，以此类推。数据平面接口将 VNF 绑定到 DPDK，以确保高可用性，因为 VNF 使用 DPDK 库绑定了数据平面接口。该镜像也有两个提供商网络用于冗余。Compute 节点绑定有两个常规 NIC，并在 VNF 管理与 Red Hat OpenStack Platform API 管理之间共享。

图 6.2. NFV SR-IOV 拓扑

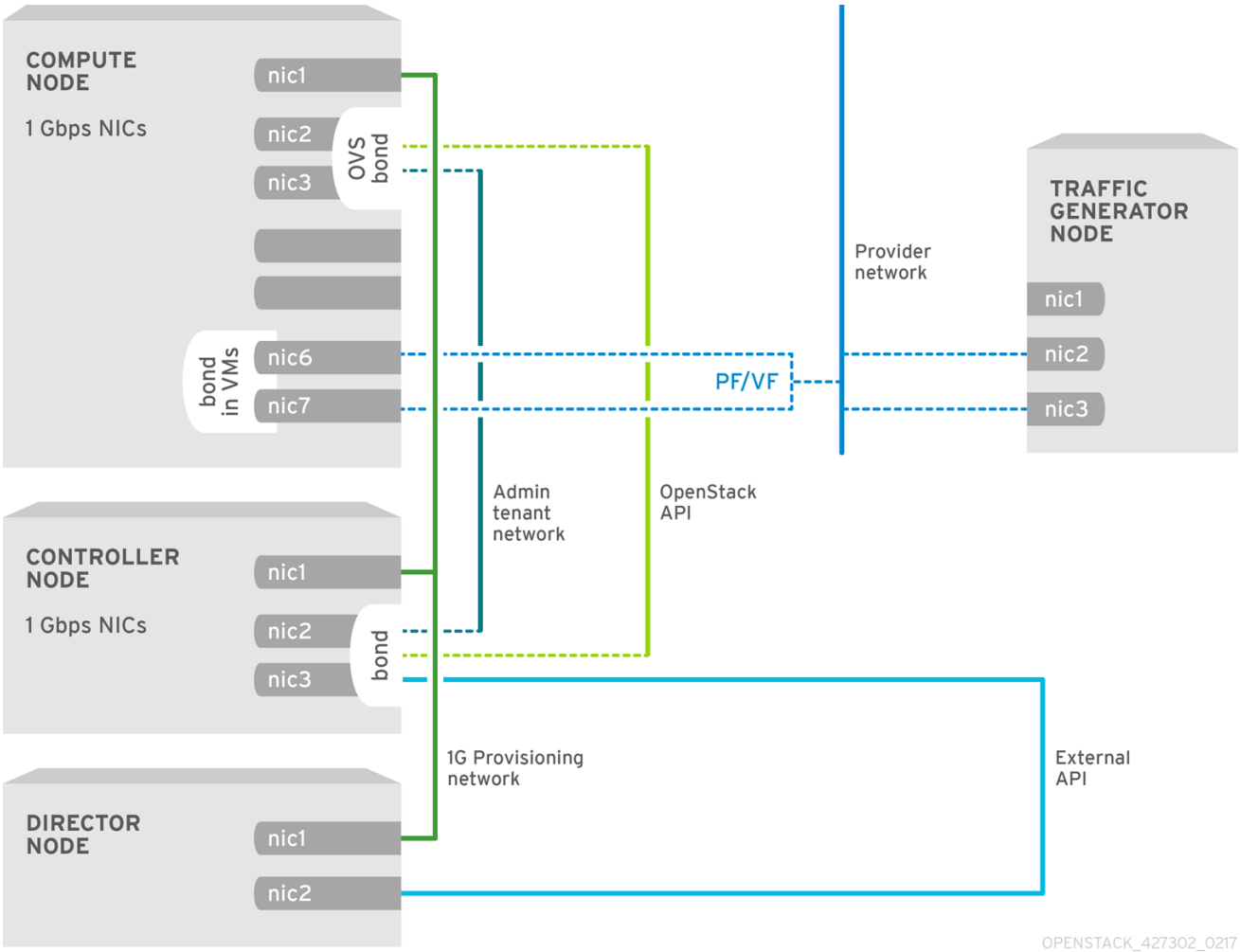


镜像显示一个 VNF，它在应用级别使用 DPDK，并可访问 SR-IOV 虚拟功能(VF)和物理功能(PF)，具体取决于结构的配置。DPDK 提高了性能，而 VF/PF DPDK 绑定则支持故障转移和高可用性。VNF 供应商必须确保 DPDK 轮询模式驱动程序(PMD)支持作为 VF/PF 公开的 SR-IOV 卡。管理网络使用 OVS，因此 VNF 看到使用标准 virtIO 驱动程序的 mgmt 网络设备。您可以使用该设备初始连接到 VNF，并确保 DPDK 应用绑定了两个 VF/PF。

6.3. 没有 HCI 的 NFV SR-IOV 拓扑

在下图中观察 SR-IOV 的拓扑，没有超融合基础架构(HCI)。它由带有 1 Gbps NIC 和 director 节点的计算和控制器节点组成。

图 6.3. 没有 HCI 的 NFV SR-IOV 拓扑



第 7 章 配置 SR-IOV 部署

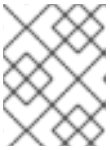
在 Red Hat OpenStack Platform NFV 部署中，当您通过虚拟资源配置从实例到共享 PCIe 资源时，您可以使用单根 I/O 虚拟化(SR-IOV)提高性能。

重要

本节包括必须为拓扑和用例修改的示例。如需更多信息，请参阅 NFV 的硬件要求。

先决条件

- RHOSP undercloud。
  在部署 overcloud 之前，您必须安装和配置 undercloud。如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform*。

  注意

  RHOSP director 通过您在模板和自定义环境文件中指定的键值对修改 SR-IOV 配置文件。您不能直接修改 SR-IOV 文件。

- 访问 **stack** 用户的 undercloud 主机和凭据。
- 访问包含 NIC 的主机。
- 确保保持 NIC 固件更新。
  yum 或 **dnf** 更新可能无法完成固件更新。如需更多信息，请参阅您的厂商文档。

流程

使用 Red Hat OpenStack Platform (RHOSP) director 在 SR-IOV 环境中安装和配置 RHOSP。高级步骤有：

1. 按照 *使用 director* 配置 overcloud 网络中的说明，创建网络配置文件 **network_data.yaml** 以为您的 overcloud 配置 物理网络。
2. 生成角色和镜像文件。
3. 为 SR-IOV 配置 PCI 透传设备 。
4. 添加特定于角色的参数和配置覆盖。
5. 创建裸机节点定义文件。
6. 为 SR-IOV 创建 NIC 配置模板 。
7. （可选） 分区 NIC。
8. 置备 overcloud 网络和 VIP。
   如需更多信息，请参阅：
   - *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 配置和置备 overcloud 网络定义。
   - *使用 director 安装和管理 Red Hat OpenStack Platform 指南中的为 overcloud 配置和置备网络 VIP*。

9. 置备 overcloud 裸机节点。
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。

10. 部署 SR-IOV overcloud。

其他资源

- 第 7.7 节 “NIC 分区配置示例”
- 第 7.9 节 “在 SR-IOV 或 OVS TC-flower 硬件卸载环境中创建主机聚合”
- 第 7.10 节 “在 SR-IOV 或 OVS TC-flower 硬件卸载环境中创建实例”

7.1. 为 SR-IOV 生成角色和镜像文件

Red Hat OpenStack Platform (RHOSP) director 使用角色为节点分配服务。在 SR-IOV 环境中部署 RHOSP 时，**ComputeSriov** 是 RHOSP 安装提供的默认角色，它除默认的计算服务外，还包括 **NeutronSriovAgent** 服务。

undercloud 安装需要一个环境文件来确定从何处获取容器镜像以及如何存储它们。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 生成一个名为 **roles_data_compute_sriov.yaml** 的新角色数据文件，其中包含 **Controller** 和 **ComputeSriov** 角色：

```
$  openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_compute_sriov.yaml \
 Controller ComputeSriov
```

注意

如果您在 RHOSP 环境、OVS-DPDK、SR-IOV 和 OVS 硬件卸载中使用多种技术，则仅生成一个角色数据文件来包含所有角色：

```
$ openstack overcloud roles generate -o /home/stack/templates/\
roles_data.yaml Controller ComputeOvsDpdk ComputeOvsDpdkSriov \
Compute:ComputeOvsHwOffload
```

4. 要生成镜像文件，请运行 **openstack tripleo container image prepare** 命令。需要以下输入：
    - 您在之前步骤中生成的角色数据文件，如 **roles_data_compute_sriov.yaml**。

- 适合您的网络服务机制驱动程序的 SR-IOV 环境文件：
  - ML2/OVN 环境
    /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml
  - ML2/OVS 环境
    /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-sriov.yaml

    示例

    在本例中，为 ML2/OVN 环境生成 **overcloud_images.yaml** 文件：

    ```
    $ sudo openstack tripleo container image prepare \
      --roles-file ~/templates/roles_data_compute_sriov.yaml \
      -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml \
      -e ~/containers-prepare-parameter.yaml \
      --output-env-file=/home/stack/templates/overcloud_images.yaml
    ```

5. 注意角色数据文件的路径和文件名以及您创建的镜像文件。您稍后会在部署 overcloud 时使用这些文件。

后续步骤

- 继续 第 7.2 节 “为 SR-IOV 配置 PCI 透传设备”。

其他资源

- 如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform 中的可组合服务和自定义角色*。
- *使用 director 安装和管理 Red Hat OpenStack Platform 中的准备容器镜像*。https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/installing_and_managing_red_hat_openstack_platform_for-director-installation#proc_preparing-container-images_preparing-for-director-installation

7.2. 为 SR-IOV 配置 PCI 透传设备

为 SR-IOV 环境部署 Red Hat OpenStack Platform 时，您必须在自定义环境文件中为 SR-IOV 计算节点配置 PCI 透传设备。

先决条件

- 访问包含 PCI 卡的一个或多个物理服务器。
- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

1. 在具有 PCI 卡的物理服务器上使用以下命令之一：
   - 如果部署了 overcloud：

```
$ lspci -nn -s  <pci_device_address>
```

输出示例

```
3b:00.0 Ethernet controller [0200]: Intel Corporation Ethernet
Controller X710 for 10GbE SFP+ [<vendor_id>: <product_id>] (rev 02)
```

- 如果您的 overcloud 尚未部署：

```
$ openstack baremetal introspection data save <baremetal_node_name> | jq
'.inventory.interfaces[] | .name, .vendor, .product'
```

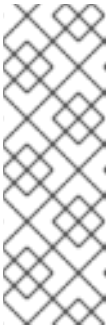
2. 为 SR-IOV 计算节点上 PCI 透传设备保留供应商和产品 ID。后续步骤中您将需要这些 ID。
3. 以 **stack** 用户的身份登录 undercloud。
4. Source **stackrc** 文件：

```
$ source ~/stackrc
```

5. 创建自定义环境 YAML 文件，如 **sriov-overrides.yaml**。通过在文件中添加以下内容，为 SR-IOV 计算节点配置 PCI 透传设备：

```
parameter_defaults:
  ComputeSriovParameters:
    ...
    NovaPCIPassthrough:
      - vendor_id: "<vendor_id>"
        product_id: "<product_id>"
        address: <NIC_address>
        physical_network: "<physical_network>"
    ...
```

- 将 **<vendor_id** > 替换为 PCI 设备的厂商 ID。
- 将 **<product_id** > 替换为 PCI 设备的产品 ID。
- 将 **<NIC_address** > 替换为 PCI 设备的地址。
- 使用 PCI 设备所在的物理网络的名称替换 **<physical_network>**。



注意

在配置 PCI 透传时不要使用 **devname** 参数，因为 NIC 的设备名称可以更改。要在 PF 上创建网络服务(neutron)端口，在 **NovaPCIPassthrough** 中指定 **vendor_id**、**product_id** 和 PCI 设备地址，并使用 **--vnic-type direct-physical** 选项创建端口。要在虚拟功能(VF)上创建网络服务端口，请在 **NovaPCIPassthrough** 中指定 **vendor_id** 和 **product_id**，并使用 **--vnic-type 直接** 选项创建端口。**vendor_id** 和 **product_id** 参数的值可能因物理功能(PF)和 VF 上下文而异。

6. 另外，在自定义环境文件中，确保 **PciPassthroughFilter** 和 **AggregateInstanceExtraSpecsFilter** 位于 **NovaSchedulerEnabledFilters** 参数的过滤器列表中，计算服务(nova)用来过滤节点：

```
parameter_defaults:
  ComputeSriovParameters:
    ...
    NovaPCIPassthrough:
      - vendor_id: "<vendor_id>"
        product_id: "<product_id>"
        address: <NIC_address>
        physical_network: "<physical_network>"
    ...
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - AggregateInstanceExtraSpecsFilter
```

7. 请注意您创建的自定义环境文件的路径和文件名。您稍后会在部署 overcloud 时使用此文件。

后续步骤

- 继续 第 7.3 节 "添加特定于角色的参数和配置覆盖"。

其他资源

- 为实例创建配置 *Compute 服务中的配置* ***NovaPCIPassthrough*** 的指南

7.3. 添加特定于角色的参数和配置覆盖

您可以为 SR-IOV Compute 节点添加特定于角色的参数，并在 Red Hat OpenStack Platform (RHOSP) director 在部署 SR-IOV 环境时使用的自定义环境 YAML 文件中覆盖默认配置值。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 打开您在 第 7.2 节 "为 SR-IOV 配置 PCI 透传设备" 中创建的自定义环境 YAML 文件，或创建新环境。
4. 将 SR-IOV Compute 节点的特定于角色的参数添加到自定义环境文件。

示例

```
  ComputeSriovParameters:
    IsolCpusList: 9-63,73-127
    KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=100 amd_iommu=on
iommu=pt numa_balancing=disable processor.max_cstate=0 isolcpus=9-63,73-127
    NovaReservedHostMemory: 4096
    NovaComputeCpuSharedSet: 0-8,64-72
    NovaComputeCpuDedicatedSet: 9-63,73-127
```

5. 查看 RHOSP director 用来配置 SR-IOV 的配置默认值。这些默认值在文件中提供，它们会根据您的机制驱动程序而有所不同：

   - ML2/OVN
     /usr/share/openstack-tripleo-heat-templates/environment/services/neutron-ovn-sriov.yaml

   - ML2/OVS
     /usr/share/openstack-tripleo-heat-templates/environment/services/neutron-sriov.yaml

6. 如果您需要覆盖任何配置默认值，请将覆盖添加到自定义环境文件中。
   例如，此自定义环境文件是您可以添加 Nova PCI 白名单值或设置网络类型的位置。

   示例

   在本例中，Networking 服务(neutron)网络类型被设置为 VLAN，并为租户添加范围：

```
parameter_defaults:
  NeutronNetworkType: 'vlan'
  NeutronNetworkVLANRanges:
    - tenant:22:22
    - tenant:25:25
  NeutronTunnelTypes: ''
```

7. 如果您创建了新的自定义环境文件，请记下其路径和文件名。您稍后会在部署 overcloud 时使用此文件。

后续步骤

- 继续 第 7.4 节 “为 SR-IOV 创建裸机节点定义文件”

其他资源

- *自定义 Red Hat OpenStack Platform 部署* 指南中的 支持的自定义角色

7.4. 为 SR-IOV 创建裸机节点定义文件

使用 Red Hat OpenStack Platform (RHOSP) director 和定义文件为 SR-IOV 环境置备裸机节点。在裸机节点定义文件中，定义您要部署并分配 overcloud 角色的裸机节点的数量和属性。另外，也定义节点的网络布局。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 按照 *使用 director 的 Red Hat OpenStack Platform 置备裸机节点指南中所述，创建裸机节点定义文件，如 **overcloud** -baremetal-deploy.yaml*。
4. 在裸机节点定义文件中，向 Ansible playbook 添加声明 **cli-overcloud-node-kernelargs.yaml**。playbook 包含置备裸机节点时要使用的内核参数。

```
- name: ComputeSriov
...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
...
```

5. 如果要在运行 playbook 时设置任何额外的 Ansible 变量，请使用 **extra_vars** 属性来设置它们。

注意

您添加到 **extra_vars** 的变量应该是之前添加到 第 7.3 节 "添加特定于角色的参数和配置覆盖" 中的自定义环境文件中的 SR-IOV Compute 节点的角色特定参数。

示例

```
- name: ComputeSriov
...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
      extra_vars:
        kernel_args: 'default_hugepagesz=1GB hugepagesz=1G hugepages=100
amd_iommu=on iommu=pt isolcpus=9-63,73-127'
        tuned_isolated_cores: '9-63,73-127'
        tuned_profile: 'cpu-partitioning'
        reboot_wait_timeout: 1800
```

6. 注意您创建的裸机节点定义文件的路径和文件名。稍后，在配置 NIC 时，并在置备节点时使用 **此文件作为 overcloud 节点置备** 命令的输入文件。

后续步骤

- 继续 第 7.5 节 "为 SR-IOV 创建 NIC 配置模板"。

其他资源

- *使用 director 安装和管理 Red Hat OpenStack Platform* 中的可组合服务和自定义角色
- 为 NFV 测试的 NIC
- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 裸机节点置备属性

7.5. 为 SR-IOV 创建 NIC 配置模板

通过修改 Red Hat OpenStack Platform (RHOSP)附带的示例 Jinja2 模板的副本来定义您的 NIC 配置模板。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 复制示例网络配置模板。
从 **/usr/share/ansible/roles/tripleo_network_config/templates/** 目录中的示例复制 NIC 配置 Jinja2 模板。选择最符合 NIC 要求的值。根据需要进行修改。
4. 在 NIC 配置模板中，如 **single_nic_vlans.j2**，添加您的 PF 和 VF 接口。要创建 SR-IOV VF，请将接口配置为独立 NIC。

示例

```
...
- type: sriov_pf
  name: enp196s0f0np0
  mtu: 9000
  numvfs: 16
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true
  promisc: false
...
```

注意

numvfs 参数替换网络配置模板中的 **NeutronSriovNumVFs** 参数。红帽不支持在部署后修改 **NeutronSriovNumVFs** 参数或 **numvfs** 参数。如果您在部署后修改任何参数，修改可能会导致该 PF 上具有 SR-IOV 端口的运行实例中断。在这种情况下，您必须硬重启这些实例以使 SR-IOV PCI 设备再次可用。

5. 将自定义网络配置模板添加到您在 第 7.4 节 “为 SR-IOV 创建裸机节点定义文件” 中创建的裸机节点定义文件。

示例

```
- name: ComputeSriov
  count: 2
  hostname_format: compute-%index%
```

```
  defaults:
    networks:
      - network: internal_api
        subnet: internal_api_subnet
      - network: tenant
        subnet: tenant_subnet
      - network: storage
        subnet: storage_subnet
    network_config:
      template: /home/stack/templates/single_nic_vlans.j2
...
```

6. 请注意您创建的 NIC 配置模板的路径和文件名。如果要对 NIC 进行分区，您可以稍后使用此文件。

后续步骤

1. 如果要对 NIC 进行分区，请继续 第 7.6 节 “配置 NIC 分区”。
2. 否则，请执行以下步骤：
   a. *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 配置和置备 overcloud 网络定义
   b. *使用 director 安装和管理 Red Hat OpenStack Platform 指南* 中的为 overcloud 配置和置备网络 VIP。
   c. *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 为 overcloud 置备裸机节点。
   d. 第 7.8 节 “部署 SR-IOV overcloud”

7.6. 配置 NIC 分区

您可以通过为 Red Hat OpenStack Platform (RHOSP)管理网络和供应商网络配置单一根 I/O 虚拟化(SR-IOV)虚拟功能(VF)来减少每个主机所需的 NIC 数量。当您将单个、高速 NIC 分区为多个 VF 时，您可以将 NIC 用于控制和数据平面流量。此功能已在 Intel Fortville NIC 和 Mellanox CX-5 NIC 上进行验证。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。
- 确保 NIC、其应用、VF 客户机和 OVS 驻留在相同的 NUMA Compute 节点上。
  这有助于防止性能下降跨 NUMA 操作。
- 确保保持 NIC 固件更新。
  yum 或 **dnf** 更新可能无法完成固件更新。如需更多信息，请参阅您的厂商文档。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 打开您之前在 第 7.5 节 “为 SR-IOV 创建 NIC 配置模板” 中创建的 NIC 配置模板，如 **single_nic_vlans.j2**。

   提示

   完成本节中的步骤时，您可以参阅 第 7.7 节 “NIC 分区配置示例”。

4. 为接口类型 **sriov_pf** 添加一个条目，以配置主机可以使用的物理功能：

```
- type: sriov_pf
  name: <interface_name>
  use_dhcp: false
  numvfs: <number_of_vfs>
  promisc: <true/false>
```

   - 将 **<interface_name** > 替换为接口名称。
   - 将 **<number_of_vfs** > 替换为 VF 的数量。
   - 可选：将 **<true/false>** 替换为 **true** 来设置 promiscuous 模式，设置为 **false** 来禁用 promiscuous 模式。默认值为 **true**。

   注意

   numvfs 参数替换网络配置模板中的 **NeutronSriovNumVFs** 参数。红帽不支持在部署后修改 **NeutronSriovNumVFs** 参数或 **numvfs** 参数。如果您在部署后修改了任一参数，则可能会导致在该物理功能(PF)上运行具有 SR-IOV 端口的实例中断。在这种情况下，您必须硬重启这些实例以使 SR-IOV PCI 设备再次可用。

5. 为接口类型 **sriov_vf** 添加一个条目，以配置主机可以使用的虚拟功能：

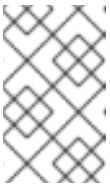
```
- type: <bond_type>
  name: internal_bond
  bonding_options: mode=<bonding_option>
  use_dhcp: false
  members:
  - type: sriov_vf
      device: <pf_device_name>
      vfid: <vf_id>
  - type: sriov_vf
      device:  <pf_device_name>
      vfid: <vf_id>

- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  spoofcheck: false
  device: internal_bond
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
  routes:
    list_concat_unique:
    - get_param: InternalApiInterfaceRoutes
```

- 将 **<bond_type>** 替换为所需的绑定类型，如 **linux_bond**。您可以为其他绑定在绑定上应用 VLAN 标签，如 **ovs_bond**。
- 将 **<bonding_option>** 替换为以下支持的绑定模式之一：
  - **active-backup**
  - **balance-slb**

注意

不支持 LACP 绑定。

- 在 **members** 部分中，指定 **sriov_vf** 作为要绑定的接口类型。



注意

如果您使用 OVS 网桥作为接口类型，则只能在 **sriov_pf** 设备的 **sriov_vf** 上配置一个 OVS 网桥。单个 **sriov_pf** 设备中的多个 OVS 网桥可能会导致 VF 间的数据包重复，并降低性能。

- 将 **<pf_device_name>** 替换为 PF 设备的名称。
- 如果使用 **linux_bond**，则必须分配 VLAN 标签。如果设置了 VLAN 标签，请确保为与单个 **sriov_pf** 设备关联的每个 VF 设置唯一标签。同一 VLAN 上不能有两个来自同一 PF 的 VF。
- 将 **<vf_id>** 替换为 VF 的 ID。适用的 VF ID 范围从零开始，以 VF 减去的最大数量结束。
- 禁用欺骗检查。
- 在 **sriov_vf** 上为 VF 上的 **linux_bond** 应用 VLAN 标签。

6. 要为实例保留 VF，请在环境文件中包含 **NovaPCIPassthrough** 参数。

示例

```
NovaPCIPassthrough:
 - address: "0000:19:0e.3"
   trusted: "true"
   physical_network: "sriov1"
 - address: "0000:19:0e.0"
   trusted: "true"
   physical_network: "sriov2"
```

RHOSP director 识别主机 VF，并生成可用于实例的 VF 的 PCI 地址。

7. 在需要 NIC 分区的所有节点上启用 **IOMMU**。

示例

例如，如果您想要 Compute 节点的 NIC 分区，请使用该角色的 **KernelArgs** 参数启用 IOMMU：

```
parameter_defaults:
  ComputeParameters:
    KernelArgs: "intel_iommu=on iommu=pt"
```

注意

当您首先将 **KernelArgs** 参数添加到角色的配置中时，overcloud 节点会自动重启。如果需要，您可以禁用自动重新引导节点，并在每个 overcloud 部署后手动重启节点。

8. 确保将这个 NIC 配置模板（如 **single_nic_vlans.j2**）添加到您在 第 7.4 节 "为 SR-IOV 创建裸机节点定义文件" 中创建的裸机节点定义文件中。

后续步骤

1. *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 配置和置备 overcloud 网络定义
2. *使用 director 安装和管理 Red Hat OpenStack Platform 指南* 中的为 overcloud 配置和置备网络 VIP。
3. *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. 第 7.8 节 "部署 SR-IOV overcloud"

其他资源

- 第 7.7 节 "NIC 分区配置示例"

7.7. NIC 分区配置示例

如果要在 Red Hat OpenStack Platform SR-IOV 环境中对 NIC 进行分区时，请参阅这些示例配置。

VF 上的 Linux 绑定

以下示例通过 VF 配置 Linux 绑定，禁用 **spoofcheck**，并将 VLAN 标签应用到 **sriov_vf** ：

```
- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  members:
    - type: sriov_vf
      device: eno2
      vfid: 1
      vlan_id:
        get_param: InternalApiNetworkVlanID
      spoofcheck: false
    - type: sriov_vf
      device: eno3
      vfid: 1
      vlan_id:
        get_param: InternalApiNetworkVlanID
      spoofcheck: false
  addresses:
    - ip_netmask:
      get_param: InternalApiIpSubnet
  routes:
    list_concat_unique:
    - get_param: InternalApiInterfaceRoutes
```

VF 上的 OVS 网桥

以下示例在 VF 上配置 OVS 网桥：

```
- type: ovs_bridge
  name: br-bond
  use_dhcp: true
  members:
    - type: vlan
      vlan_id:
      get_param: TenantNetworkVlanID
  addresses:
  - ip_netmask:
    get_param: TenantIpSubnet
  routes:
    list_concat_unique:
      - get_param: ControlPlaneStaticRoutes
  - type: ovs_bond
    name: bond_vf
    ovs_options: "bond_mode=active-backup"
    members:
      - type: sriov_vf
        device: p2p1
        vfid: 2
      - type: sriov_vf
        device: p2p2
        vfid: 2
```

VF 上的 OVS 用户网桥

以下示例在 VF 上配置 OVS 用户网桥，并将 VLAN 标签应用到 **ovs_user_bridge** ：

```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  mtu: 9000
  ovs_extra:
    - str_replace:
        template: set port br-link0 tag=_VLAN_TAG_
        params:
          _VLAN_TAG_:
            get_param: TenantNetworkVlanID
  addresses:
    - ip_netmask:
    list_concat_unique:
      - get_param: TenantInterfaceRoutes
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      ovs_extra:
        - set port dpdkbond0 bond_mode=balance-slb
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          members:
```

```
      - type: sriov_vf
        device: eno2
        vfid: 3
    - type: ovs_dpdk_port
      name: dpdk1
      members:
        - type: sriov_vf
          device: eno3
          vfid: 3
```

其他资源

- 第 7.6 节 “配置 NIC 分区”

7.8. 部署 SR-IOV OVERCLOUD

在 SR-IOV 环境中配置 Red Hat OpenStack Platform (RHOSP) overcloud 的最后一步是运行 **openstack overcloud deploy** 命令。命令的输入包括您构建的所有 overcloud 模板和环境文件。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。
- 您已执行了本节前面流程中列出的所有步骤，并编译了所有各种 heat 模板和环境文件，以用作 **overcloud deploy** 命令的输入。

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. 输入 **openstack overcloud deploy** 命令。
   以特定顺序列出 **openstack overcloud deploy** 命令的输入非常重要。常规规则是首先指定默认的 heat 模板文件，后跟包含自定义配置的自定义环境文件和自定义模板，如覆盖默认属性。

   按照以下顺序将输入添加到 **openstack overcloud deploy** 命令中：

   a. 包含 overcloud 上 SR-IOV 网络规格的自定义网络定义文件，如 **network-data.yaml**。
   如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 网络定义文件配置选项。

   b. 包含 RHOSP director 用来部署 OVS 硬件卸载环境的 **Controller** 和 **ComputeOvsHwOffload** 角色的角色文件。
   示例： **roles_data_compute_sriov.yaml**

   如需更多信息，请参阅 第 7.1 节 “为 SR-IOV 生成角色和镜像文件”。

   c. 来自置备 overcloud 网络的输出文件。
   示例： **overcloud-networks-deployed.yaml**

如需更多信息，请参阅 *使用 director 安装和管理 Red Hat OpenStack Platform 指南* 中的配置和管理 overcloud 网络定义。

d. 来自置备 overcloud VIP 的输出文件。
示例： **overcloud-vip-deployed.yaml**

如需更多信息，请参阅 *安装和管理 Red Hat OpenStack Platform* 指南中的 为 overcloud 配置和置备网络 VIP。

e. 置备裸机节点的输出文件。
示例： **overcloud-baremetal-deployed.yaml**

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。

f. director 用来确定获取容器镜像的位置以及如何存储它们的镜像文件。
示例： **overcloud_images.yaml**

如需更多信息，请参阅 第 7.1 节 “为 SR-IOV 生成角色和镜像文件”。

g. 您的环境使用的网络服务(neutron)机制驱动程序和路由器方案的环境文件：

- ML2/OVN
  - 分布式虚拟路由(DVR)： **neutron-ovn-dvr-ha.yaml**
  - 集中式虚拟路由： **neutron-ovn-ha.yaml**
- ML2/OVS
  - 分布式虚拟路由(DVR)： **neutron-ovs-dvr.yaml**
  - 集中式虚拟路由： **neutron-ovs.yaml**

h. SR-IOV 的环境文件，具体取决于您的机制驱动程序：

- ML2/OVN
  - **neutron-ovn-sriov.yaml**
- ML2/OVS
  - **neutron-sriov.yaml**

注意

如果您也有一个 OVS-DPDK 环境，并希望在同一节点上找到 OVS-DPDK 和 SR-IOV 实例，请在部署脚本中包含以下环境文件：

- ML2/OVN
  neutron-ovn-dpdk.yaml
- ML2/OVS
  neutron-ovs-dpdk.yaml

i. 一个或多个包含您的配置的自定义环境文件：

- SR-IOV 节点的 PCI 透传设备。

- SR-IOV 节点的特定于角色的参数
- 覆盖 SR-IOV 环境的默认配置值。
  示例： **sriov-overrides.yaml**

  如需更多信息，请参阅：

- 第 7.2 节 "为 SR-IOV 配置 PCI 透传设备" .
- 第 7.3 节 "添加特定于角色的参数和配置覆盖".

  示例

  示例 **openstack overcloud deploy** 命令摘录演示了使用 DVR 的 SR-IOV ML2/OVN 环境正确排序命令输入：

```
$ openstack overcloud deploy \
--log-file overcloud_deployment.log \
--templates /usr/share/openstack-tripleo-heat-templates/ \
--stack overcloud \
-n /home/stack/templates/network_data.yaml \
-r /home/stack/templates/roles_data_compute_sriov.yaml \
-e /home/stack/templates/overcloud-networks-deployed.yaml \
-e /home/stack/templates/overcloud-vip-deployed.yaml \
-e /home/stack/templates/overcloud-baremetal-deployed.yaml \
-e /home/stack/templates/overcloud-images.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-dvr-ha.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-sriov.yaml \
-e /home/stack/templates/sriov-overrides.yaml
```

4. 运行 **openstack overcloud deploy** 命令。
   完成 overcloud 创建后，RHOSP director 会提供帮助您访问 overcloud 的详细信息。

验证

1. *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的执行验证 overcloud 部署 中的步骤。
2. 要验证您的 NIC 是否已正确分区，请执行以下操作：
   a. 以 **tripleo-admin** 用户身份登录 overcloud Compute 节点，并检查 VF 的数量：

      示例

      在本例中，**p4p1** 和 **p4p2** 的 VF 数量都是 **10** ：

```
$ sudo cat /sys/class/net/p4p1/device/sriov_numvfs

10

$ sudo cat /sys/class/net/p4p2/device/sriov_numvfs

10
```

b. 显示 OVS 连接：

```
$ sudo ovs-vsctl show
```

输出示例

您应该看到类似如下的输出：

```
b6567fa8-c9ec-4247-9a08-cbf34f04c85f
    Manager "ptcp:6640:127.0.0.1"
        is_connected: true
    Bridge br-sriov2
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        datapath_type: netdev
        Port phy-br-sriov2
            Interface phy-br-sriov2
                type: patch
                options: {peer=int-br-sriov2}
        Port br-sriov2
            Interface br-sriov2
                type: internal
    Bridge br-sriov1
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        datapath_type: netdev
        Port phy-br-sriov1
            Interface phy-br-sriov1
                type: patch
                options: {peer=int-br-sriov1}
        Port br-sriov1
            Interface br-sriov1
                type: internal
    Bridge br-ex
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        datapath_type: netdev
        Port br-ex
            Interface br-ex
                type: internal
        Port phy-br-ex
            Interface phy-br-ex
                type: patch
                options: {peer=int-br-ex}
    Bridge br-tenant
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        datapath_type: netdev
        Port br-tenant
            tag: 305
            Interface br-tenant
                type: internal
```

```
        Port phy-br-tenant
            Interface phy-br-tenant
                type: patch
                options: {peer=int-br-tenant}
        Port dpdkbond0
            Interface dpdk0
                type: dpdk
                options: {dpdk-devargs="0000:18:0e.0"}
            Interface dpdk1
                type: dpdk
                options: {dpdk-devargs="0000:18:0a.0"}
    Bridge br-tun
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        datapath_type: netdev
        Port vxlan-98140025
            Interface vxlan-98140025
                type: vxlan
                options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.37"}
        Port br-tun
            Interface br-tun
                type: internal
        Port patch-int
            Interface patch-int
                type: patch
                options: {peer=patch-tun}
        Port vxlan-98140015
            Interface vxlan-98140015
                type: vxlan
                options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.21"}
        Port vxlan-9814009f
            Interface vxlan-9814009f
                type: vxlan
                options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.159"}
        Port vxlan-981400cc
            Interface vxlan-981400cc
                type: vxlan
                options: {df_default="true", egress_pkt_mark="0", in_key=flow,
local_ip="152.20.0.229", out_key=flow, remote_ip="152.20.0.204"}
    Bridge br-int
        Controller "tcp:127.0.0.1:6633"
            is_connected: true
        fail_mode: secure
        datapath_type: netdev
        Port int-br-tenant
            Interface int-br-tenant
                type: patch
                options: {peer=phy-br-tenant}
        Port int-br-ex
            Interface int-br-ex
                type: patch
                options: {peer=phy-br-ex}
```

```
        Port int-br-sriov1
            Interface int-br-sriov1
                type: patch
                options: {peer=phy-br-sriov1}
        Port patch-tun
            Interface patch-tun
                type: patch
                options: {peer=patch-int}
        Port br-int
            Interface br-int
                type: internal
        Port int-br-sriov2
            Interface int-br-sriov2
                type: patch
                options: {peer=phy-br-sriov2}
        Port vhu4142a221-93
            tag: 1
            Interface vhu4142a221-93
                type: dpdkvhostuserclient
                options: {vhost-server-path="/var/lib/vhost_sockets/vhu4142a221-93"}
    ovs_version: "2.13.2"
```

c. 以 **tripleo-admin** 用户身份登录 SR-IOV Compute 节点并检查 Linux 绑定：

```
$ cat /proc/net/bonding/<bond_name>
```

输出示例

您应该看到类似如下的输出：

```
Ethernet Channel Bonding Driver: v3.7.1 (April 27, 2011)

Bonding Mode: fault-tolerance (active-backup)
Primary Slave: None
Currently Active Slave: eno3v1
MII Status: up
MII Polling Interval (ms): 0
Up Delay (ms): 0
Down Delay (ms): 0
Peer Notification Delay (ms): 0

Slave Interface: eno3v1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 4e:77:94:bd:38:d2
Slave queue ID: 0

Slave Interface: eno4v1
MII Status: up
Speed: 10000 Mbps
Duplex: full
Link Failure Count: 0
Permanent HW addr: 4a:74:52:a7:aa:7c
Slave queue ID: 0
```

3. 列出 OVS 绑定：

```
$ sudo ovs-appctl bond/show
```

输出示例

您应该看到类似如下的输出：

```
---- dpdkbond0 ----
bond_mode: balance-slb
bond may use recirculation: no, Recirc-ID : -1
bond-hash-basis: 0
updelay: 0 ms
downdelay: 0 ms
next rebalance: 9491 ms
lacp_status: off
lacp_fallback_ab: false
active slave mac: ce:ee:c7:58:8e:b2(dpdk1)

slave dpdk0: enabled
  may_enable: true

slave dpdk1: enabled
  active slave
  may_enable: true
```

4. 如果您使用 **NovaPCIPassthrough** 将 VF 传递给实例，请通过部署 SR-IOV 实例进行测试。

其他资源

- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的创建 overcloud
- *命令行界面参考*中的 overcloud 部署
- 第 7.10 节 "在 SR-IOV 或 OVS TC-flower 硬件卸载环境中创建实例"

7.9. 在 SR-IOV 或 OVS TC-FLOWER 硬件卸载环境中创建主机聚合

为了在 Red Hat OpenStack Platform (RHOSP) SR-IOV 或 OVS TC-flower 硬件卸载环境中提高性能，请部署具有 CPU 固定和巨页的客户机。您可以通过匹配聚合元数据和类别元数据，在主机子集中调度高性能实例。

先决条件

- 为 SR-IOV 或 OVS 硬件卸载环境配置的 RHOSP overcloud。
- 必须为 **AggregateInstanceExtraSpecsFilter** 配置您的 RHOSP overcloud。
  如需更多信息，请参阅 第 7.2 节 "为 SR-IOV 配置 PCI 透传设备"。

流程

1. 创建聚合组，并添加相关主机。
   定义与定义的类别元数据匹配的元数据，如 **sriov=true**。

```
$ openstack aggregate create sriov_group
$ openstack aggregate add host sriov_group compute-sriov-0.localdomain
$ openstack aggregate set --property sriov=true sriov_group
```

2. 创建类别。

```
$ openstack flavor create <flavor> --ram <size_mb> --disk <size_gb> \
--vcpus <number>
```

3. 设置其他类别属性。
   请注意，定义的元数据 **sriov=true** 与 SR-IOV 聚合中定义的元数据匹配。

```
$ openstack flavor set --property sriov=true \
--property hw:cpu_policy=dedicated \
--property hw:mem_page_size=1GB <flavor>
```

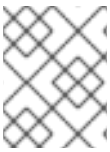
其他资源

- *命令行界面参考中的 聚合*
- *命令行界面参考中的 flavor*

7.10. 在 SR-IOV 或 OVS TC-FLOWER 硬件卸载环境中创建实例

您可以使用多个命令在 Red Hat OpenStack Platform (RHOSP) SR-IOV 或 OVS TC-flower 硬件卸载环境中创建实例。

使用主机聚合来分隔高性能计算主机。如需更多信息，请参阅 第 7.9 节 “在 SR-IOV 或 OVS TC-flower 硬件卸载环境中创建主机聚合”。



注意

固定 CPU 实例可以位于与未固定实例相同的 Compute 节点上。如需更多信息，请参阅配置 Compute *服务以进行实例创建指南中的在 Compute* 节点上配置 CPU 固定。

先决条件

- 为 SR-IOV 或 OVS 硬件卸载环境配置的 RHOSP overcloud。

流程

1. 创建类别。

```
$ openstack flavor create <flavor_name> --ram <size_mb> \
--disk <size_gb> --vcpus <number>
```

提示

您可以通过将额外的 spec **hw:pci_numa_affinity_policy** 添加到类别，为 PCI 透传设备和 SR-IOV 接口指定 NUMA 关联性策略。有关更多信息，请参阅配置 *计算服务以进行实例创建中的类别元数据* 。

2. 创建网络和子网：

```
$ openstack network create <network_name> \
--provider-physical-network tenant \
--provider-network-type vlan --provider-segment <vlan_id>

$ openstack subnet create <name> --network <network_name> \
--subnet-range <ip_address_cidr> --dhcp
```

3. 创建虚拟功能(VF)端口或物理功能(PF)端口：

   - VF 端口：

     ```
     $ openstack port create --network <network_name> \
     --vnic-type direct <port_name>
     ```

   - 专用于单个实例的 PF 端口：
     此 PF 端口是一个网络服务(neutron)端口，但不由网络服务控制，因此不能作为网络适配器可见，因为它是传递给实例的 PCI 设备。

     ```
     $ openstack port create --network <network_name> \
     --vnic-type direct-physical <port_name>
     ```

4. 创建一个实例。

```
$ openstack server create --flavor <flavor> --image <image_name> \
--nic port-id=<id> <instance_name>
```

其他资源

- *命令行界面参考中的 flavor create*
- *命令行界面参考*中的 network create
- *命令行接口参考*中的 subnet create
- *命令行界面参考*中的 port create
- *命令行接口参考*中的 server create

第 8 章 配置 OVS TC-FLOWER 硬件卸载

在 Red Hat OpenStack Platform (RHOSP)网络功能虚拟化(NFV)部署中，您可以使用 Open vSwitch (OVS) TC-flower 硬件卸载获得更高的性能。硬件卸载将网络任务从 CPU 划分为网络接口控制器(NIC)上的专用处理器。这些专用硬件资源提供了额外的计算能力，使 CPU 能够执行更多宝贵的计算任务。

为 OVS 硬件卸载配置 RHOSP 与为 SR-IOV 配置 RHOSP 类似。

重要

本节包括必须为拓扑和功能要求修改的示例。如需更多信息，请参阅 NFV 的硬件要求。

先决条件

- RHOSP undercloud。
  在部署 overcloud 之前，您必须安装和配置 undercloud。如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform*。

  注意

  RHOSP director 通过您在 director 模板和自定义环境文件中指定的键值对修改 OVS 硬件卸载配置文件。您不能直接修改 OVS 硬件卸载配置文件。

- 访问 **stack** 用户的 undercloud 主机和凭据。
- 确保 NIC、其应用、VF 客户机和 OVS 驻留在相同的 NUMA Compute 节点上。
  这有助于防止性能下降跨 NUMA 操作。
- 在包含 NIC 的主机上访问 sudo。
- 确保保持 NIC 固件更新。
  yum 或 **dnf** 更新可能无法完成固件更新。如需更多信息，请参阅您的厂商文档。
- 在 **switchdev** 端口上启用安全组和端口安全性，以用于连接跟踪(conntrack)模块，将 OpenFlow 流卸载到硬件。

流程

使用 RHOSP director 在 OVS 硬件卸载环境中安装和配置 RHOSP。高级步骤有：

1. 按照 *使用 director* 配置 overcloud 网络中的说明，创建网络配置文件 **network_data.yaml** 以为您的 overcloud 配置 物理网络。
2. 生成角色和镜像文件。
3. 为 OVS 硬件卸载配置 PCI 透传设备。
4. 添加特定于角色的参数和其他配置覆盖。
5. 创建裸机节点定义文件。
6. 为 OVS 硬件卸载创建 NIC 配置模板。
7. 置备 overcloud 网络和 VIP。
   如需更多信息，请参阅：

- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 配置和置备 overcloud 网络定义。
- *使用 director 安装和管理 Red Hat OpenStack Platform 指南中的为 overcloud 配置和置备网络 VIP*。

8. 置备 overcloud 裸机节点。
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。

9. 部署 OVS 硬件卸载 overcloud。

其他资源

- 第 8.7 节 "在 SR-IOV 或 OVS TC-flower 硬件卸载环境中创建主机聚合"
- 第 8.8 节 "在 SR-IOV 或 OVS TC-flower 硬件卸载环境中创建实例"
- 第 8.9 节 "OVS TC-flower 硬件卸载故障排除"
- 第 8.10 节 "调试 TC-flower 硬件卸载流"

8.1. 为 OVS TC-FLOWER 硬件卸载生成角色和镜像文件

Red Hat OpenStack Platform (RHOSP) director 使用角色为节点分配服务。在 OVS TC-flower 硬件卸载环境中配置 RHOSP 时，您可以创建一个基于默认角色 **Compute** （由 RHOSP 安装提供的）新角色。

undercloud 安装需要一个环境文件来确定从何处获取容器镜像以及如何存储它们。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 为 OVS 硬件卸载生成 overcloud 角色，它基于 **Compute** 角色：

示例

在本例中，根据 Compute 角色创建角色 ComputeOvsHwOffload。命令生成的角色文件名为 **roles_data_compute_ovshwol.yaml** ：

```
$ openstack overcloud roles generate -o \
roles_data_compute_ovshwol.yaml Controller Compute:ComputeOvsHwOffload
```

注意

如果您的 RHOSP 环境包含 OVS-DPDK、SR-IOV 和 OVS TC-flower 硬件卸载技术，则只生成一个角色数据文件，如 **roles_data.yaml** 以包含所有角色：

```
$ openstack overcloud roles generate -o /home/stack/templates/\
roles_data.yaml Controller ComputeOvsDpdk ComputeOvsDpdkSriov \
Compute:ComputeOvsHwOffload
```

4. （可选）更改 **ComputeOvsHwOffload** 角色的 **HostnameFormatDefault: '%stackname%-%index%'** 名称。

5. 要生成镜像文件，请运行 **openstack tripleo container image prepare** 命令。需要以下输入：

   - 您在之前步骤中生成的角色数据文件，如 **roles_data_compute_ovshwol.yaml**。
   - 适合您的网络服务机制驱动程序的 SR-IOV 环境文件：
     - ML2/OVN 环境 **/usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml**
     - ML2/OVS 环境 **/usr/share/openstack-tripleo-heat-templates/environments/services/neutron-sriov.yaml**

     示例

     在本例中，为 ML2/OVN 环境生成 **overcloud_images.yaml** 文件：

     ```
     $ sudo openstack tripleo container image prepare \
       --roles-file ~/templates/roles_data_compute_ovshwol.yaml \
       -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-sriov.yaml \
       -e ~/containers-prepare-parameter.yaml \
       --output-env-file=/home/stack/templates/overcloud_images.yaml
     ```

6. 注意角色数据文件的路径和文件名以及您创建的镜像文件。您稍后会在部署 overcloud 时使用这些文件。

后续步骤

- 继续 第 8.2 节 “为 OVS TC-flower 硬件卸载配置 PCI 透传设备” 。

其他资源

- 如需更多信息， *请参阅使用 director 安装和管理 Red Hat OpenStack Platform 中的可组合服务和自定义角色*。
- *使用 director 安装和管理 Red Hat OpenStack Platform 中的准备容器镜像*。https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/installing_and_managing_red_hat_openstack_platform_for-director-installation#proc_preparing-container-images_preparing-for-director-installation

8.2. 为 OVS TC-FLOWER 硬件卸载配置 PCI 透传设备

为 OVS TC-flower 硬件卸载环境部署 Red Hat OpenStack Platform 时，您必须在自定义环境文件中为计算节点配置 PCI 透传设备。

先决条件

- 访问包含 PCI 卡的一个或多个物理服务器。
- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

1. 在包含 PCI 卡的物理服务器上使用以下命令之一：
   - 如果部署了 overcloud：

     ```
     $ lspci -nn -s  <pci_device_address>
     ```

     输出示例

     ```
     3b:00.0 Ethernet controller [0200]: Intel Corporation Ethernet
     Controller X710 for 10GbE SFP+ [<vendor_id>: <product_id>] (rev 02)
     ```

   - 如果您的 overcloud 尚未部署：

     ```
     $ openstack baremetal introspection data save <baremetal_node_name> | jq
     '.inventory.interfaces[] | .name, .vendor, .product'
     ```

2. 注意 ComputeOvsHwOffload 节点上的 PCI 透传设备的厂商和产品 ID。后续步骤中您将需要这些 ID。
3. 以 **stack** 用户的身份登录 undercloud。
4. Source **stackrc** 文件：

   ```
   $ source ~/stackrc
   ```

5. 创建自定义环境 YAML 文件，例如 **ovshwol-overrides.yaml**。通过在文件中添加以下内容，为计算节点配置 PCI 透传设备：

   ```
   parameter_defaults:
     NeutronOVSFirewallDriver: iptables_hybrid
     ComputeOvsHwOffloadParameters:
       IsolCpusList: 2-9,21-29,11-19,31-39
       KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=128 intel_iommu=on
   iommu=pt"
       OvsHwOffload: true
       TunedProfileName: "cpu-partitioning"
       NeutronBridgeMappings:
         - tenant:br-tenant
       NovaPCIPassthrough:
         - vendor_id: <vendor-id>
           product_id: <product-id>
   ```

```
    address: <address>
    physical_network: "tenant"
  - vendor_id: <vendor-id>
    product_id: <product-id>
    address: <address>
    physical_network: "null"
NovaReservedHostMemory: 4096
NovaComputeCpuDedicatedSet: 1-9,21-29,11-19,31-39
...
```

注意

如果您使用 Mellanox 智能 NIC，请在 **ComputeOvsHwOffloadParameters** 参数下添加 **DerivePciWhitelistEnabled: true**。在使用 OVS 硬件卸载时，计算服务(nova)调度程序与生成实例的 SR-IOV 透传类似。

- 将 **<vendor_id** > 替换为 PCI 设备的厂商 ID。
- 将 **<product_id** > 替换为 PCI 设备的产品 ID。
- 将 **<NIC_address** > 替换为 PCI 设备的地址。
- 使用 PCI 设备所在的物理网络的名称替换 **<physical_network>**。
- 对于 VLAN，在部署后将 **physical_network** 参数设置为您在 neutron 中创建的网络的名称。这个值还应位于 **NeutronBridgeMappings**。
- 对于 VXLAN，将 **physical_network** 参数设置为 **null**。

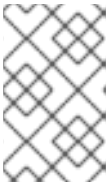
注意

在配置 PCI 透传时不要使用 **devname** 参数，因为 NIC 的设备名称可以更改。要在 PF 上创建网络服务(neutron)端口，在 **NovaPCIPassthrough** 中指定 **vendor_id**、**product_id** 和 PCI 设备地址，并使用 **--vnic-type direct-physical** 选项创建端口。要在虚拟功能(VF)上创建网络服务端口，请在 **NovaPCIPassthrough** 中指定 **vendor_id** 和 **product_id**，并使用 **--vnic-type 直接** 选项创建端口。**vendor_id** 和 **product_id** 参数的值可能因物理功能(PF)和 VF 上下文而异。

6. 在自定义环境文件中，确保 **PciPassthroughFilter** 和 **NUMATopologyFilter** 位于 **NovaSchedulerEnabledFilters** 参数的过滤器列表中。Compute 服务(nova)使用此参数来过滤节点：

```
parameter_defaults:
  ...
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
```

```
    - PciPassthroughFilter
    - NUMATopologyFilter
    - AggregateInstanceExtraSpecsFilter
```



注意

可选：有关如何对带有 Mellanox ConnectX5 NIC 的 RHOSP 17.1 中的 OVS Hardware Offload 问题进行故障排除和配置 OVS Hardware Offload 的详情，请参阅 故障排除 Hardware Offload。

7. 请注意您创建的自定义环境文件的路径和文件名。您稍后会在部署 overcloud 时使用此文件。

后续步骤

- 继续 第 8.3 节 "为 OVS TC-flower 硬件卸载添加特定于角色的参数和配置覆盖"。

其他资源

- 为实例创建配置 *Compute 服务中的配置* ***NovaPCIPassthrough*** 的指南

8.3. 为 OVS TC-FLOWER 硬件卸载添加特定于角色的参数和配置覆盖

您可以为 ComputeOvsHwOffload 节点添加特定于角色的参数，并在 Red Hat OpenStack Platform (RHOSP) director 在部署 OVS TC-flower 硬件卸载环境时覆盖自定义环境 YAML 文件中的默认配置值。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 打开您在 第 8.2 节 "为 OVS TC-flower 硬件卸载配置 PCI 透传设备" 中创建的自定义环境 YAML 文件，或创建新环境。
4. 将 ComputeOvsHwOffload 节点的特定于角色的参数添加到自定义环境文件。

   示例

```
  ComputeOvsHwOffloadParameters:
    IsolCpusList: 9-63,73-127
    KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=100 amd_iommu=on
iommu=pt numa_balancing=disable processor.max_cstate=0 isolcpus=9-63,73-127
    NovaReservedHostMemory: 4096
    NovaComputeCpuSharedSet: 0-8,64-72
    NovaComputeCpuDedicatedSet: 9-63,73-127
    TunedProfileName: "cpu-partitioning"
```

5. 在特定于角色的参数下添加 **OvsHwOffload** 参数，值设为 **true**。

```
  ComputeOvsHwOffloadParameters:
    IsolCpusList: 9-63,73-127
    KernelArgs: default_hugepagesz=1GB hugepagesz=1G hugepages=100 amd_iommu=on
iommu=pt numa_balancing=disable processor.max_cstate=0 isolcpus=9-63,73-127
    NovaReservedHostMemory: 4096
    NovaComputeCpuSharedSet: 0-8,64-72
    NovaComputeCpuDedicatedSet: 9-63,73-127
    TunedProfileName: "cpu-partitioning"
    OvsHwOffload: true
  ...
```

6. 检查 RHOSP director 用来配置 OVS 硬件卸载的配置默认值。这些默认值在文件中提供，它们会根据您的机制驱动程序而有所不同：

   - ML2/OVN
     /usr/share/openstack-tripleo-heat-templates/environment/services/neutron-ovn-sriov.yaml

   - ML2/OVS
     /usr/share/openstack-tripleo-heat-templates/environment/services/neutron-sriov.yaml

7. 如果您需要覆盖任何配置默认值，请将覆盖添加到自定义环境文件中。
   例如，此自定义环境文件是您可以添加 Nova PCI 白名单值或设置网络类型的位置。

   示例

   在本例中，Networking 服务(neutron)网络类型被设置为 VLAN，并为租户添加范围：

```
parameter_defaults:
  NeutronNetworkType: vlan
  NeutronNetworkVLANRanges:
    - tenant:22:22
    - tenant:25:25
  NeutronTunnelTypes: ''
```

8. 如果您创建了新的自定义环境文件，请记下其路径和文件名。您稍后会在部署 overcloud 时使用此文件。

后续步骤

- 继续 第 8.4 节 “为 OVS TC-flower 硬件卸载创建裸机节点定义文件”

其他资源

- *自定义 Red Hat OpenStack Platform 部署* 指南中的 支持的自定义角色

8.4. 为 OVS TC-FLOWER 硬件卸载创建裸机节点定义文件

使用 Red Hat OpenStack Platform (RHOSP) director 和定义文件为您的 OVS TC-flower 硬件卸载环境置备裸机节点。在裸机节点定义文件中，定义您要部署并分配 overcloud 角色的裸机节点的数量和属性。另外，也定义节点的网络布局。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 按照 *使用 director 的 Red Hat OpenStack Platform 置备裸机节点指南中所述，创建裸机节点定义文件，如* ***overcloud*** *-baremetal-deploy.yaml*。
4. 在裸机节点定义文件中，向 Ansible playbook 添加声明 **cli-overcloud-node-kernelargs.yaml**。playbook 包含置备裸机节点时要使用的内核参数。

```
- name: ComputeOvsHwOffload
...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
...
```

5. 如果要在运行 playbook 时设置任何额外的 Ansible 变量，请使用 **extra_vars** 属性来设置它们。

注意

您添加到 **extra_vars** 的变量应该是与之前添加到 第 8.3 节 "为 OVS TC-flower 硬件卸载添加特定于角色的参数和配置覆盖" 中的自定义环境文件中的 ComputeOvsHwOffload 节点的特定于角色的参数。

示例

```
- name: ComputeOvsHwOffload
...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
      extra_vars:
        kernel_args: 'default_hugepagesz=1GB hugepagesz=1G hugepages=100
amd_iommu=on iommu=pt isolcpus=9-63,73-127'
        tuned_isolated_cores: '9-63,73-127'
        tuned_profile: 'cpu-partitioning'
        reboot_wait_timeout: 1800
```

6. 注意您创建的裸机节点定义文件的路径和文件名。稍后，在配置 NIC 时，并在置备节点时使用 **此文件作为 overcloud 节点置备** 命令的输入文件。

后续步骤

- 继续 第 8.5 节 "为 OVS TC-flower 硬件卸载创建 NIC 配置模板" 。

其他资源

- *使用 director 安装和管理 Red Hat OpenStack Platform* 中的可组合服务和自定义角色
- 为 NFV 测试的 NIC
- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 裸机节点置备属性

8.5. 为 OVS TC-FLOWER 硬件卸载创建 NIC 配置模板

通过修改 Red Hat OpenStack Platform (RHOSP)附带的 Jinja2 模板，为 OVS TC-flower 硬件卸载环境定义您的 NIC 配置模板。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。
- 确保 NIC、其应用、VF 客户机和 OVS 驻留在相同的 NUMA Compute 节点上。这有助于防止性能下降跨 NUMA 操作。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 复制示例网络配置模板。从 **/usr/share/ansible/roles/tripleo_network_config/templates/** 目录中的示例复制 NIC 配置 Jinja2 模板。选择最符合 NIC 要求的值。根据需要进行修改。
4. 在 NIC 配置模板中，如 **single_nic_vlans.j2**，添加您的 PF 和 VF 接口。要创建 VF，请将接口配置为独立 NIC。

示例

```
...
- type: sriov_pf
  name: enp196s0f0np0
  mtu: 9000
  numvfs: 16
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true
  promisc: false
  link_mode: switchdev
...
```

注意

numvfs 参数替换网络配置模板中的 **NeutronSriovNumVFs** 参数。红帽不支持在部署后修改 **NeutronSriovNumVFs** 参数或 **numvfs** 参数。如果您在部署后修改任何参数，修改可能会导致该 PF 上具有 SR-IOV 端口的运行实例中断。在这种情况下，您必须硬重启这些实例以使 SR-IOV PCI 设备再次可用。

5. 将自定义网络配置模板添加到您在 第 8.4 节 "为 OVS TC-flower 硬件卸载创建裸机节点定义文件" 中创建的裸机节点定义文件。

   示例

```
- name: ComputeOvsHwOffload
  count: 2
  hostname_format: compute-%index%
  defaults:
    networks:
      - network: internal_api
        subnet: internal_api_subnet
      - network: tenant
        subnet: tenant_subnet
      - network: storage
        subnet: storage_subnet
    network_config:
      template: /home/stack/templates/single_nic_vlans.j2
...
```

6. 在 **compute-sriov.yaml** 配置文件中配置一个或多个用于硬件卸载的网络接口：

```
- type: ovs_bridge
  name: br-tenant
  mtu: 9000
  members:
  - type: sriov_pf
    name: p7p1
    numvfs: 5
    mtu: 9000
    primary: true
    promisc: true
    use_dhcp: false
    link_mode: switchdev
```

注意

- 在配置 OVS 硬件卸载时，不要使用 **NeutronSriovNumVFs** 参数。虚拟功能的数量使用 **os-net-config** 使用的网络配置文件中的 **numvfs** 参数指定。红帽不支持在更新或重新部署过程中修改 **numvfs** 设置。
- 不要将 Mellanox 网络接口配置为 nic-config 接口类型 **ovs-vlan**，因为这会阻止 VXLAN 等隧道端点因为驱动程序限制而传递流量。

7. 请注意您创建的 NIC 配置模板的路径和文件名。如果要对 NIC 进行分区，您可以稍后使用此文件。

后续步骤

1. 调配 overcloud 网络。
   如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform 指南中的 配置和管理 overcloud 网络定义*。

2. 置备 overcloud VIP。

如需更多信息，请参阅 *安装和管理 Red Hat OpenStack Platform* 指南中的 为 overcloud 配置和置备网络 VIP。

3. 置备裸机节点。
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. 部署 overcloud。
如需更多信息，请参阅 第 8.6 节 “部署 OVS TC-flower 硬件卸载 overcloud” 。

8.6. 部署 OVS TC-FLOWER 硬件卸载 OVERCLOUD

在 OVS TC-flower 硬件卸载环境中部署 Red Hat OpenStack Platform (RHOSP) overcloud 的最后一步是运行 **openstack overcloud deploy** 命令。命令的输入包括您构建的所有 overcloud 模板和环境文件。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。
- 在包含 NIC 的主机上访问 sudo。
- 您已执行了本节前面流程中列出的所有步骤，并编译了所有各种 heat 模板和环境文件，以用作 **overcloud deploy** 命令的输入。

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. 输入 **openstack overcloud deploy** 命令。
以特定顺序列出 **openstack overcloud deploy** 命令的输入非常重要。常规规则是首先指定默认的 heat 模板文件，后跟包含自定义配置的自定义环境文件和自定义模板，如覆盖默认属性。

   按照以下顺序将输入添加到 **openstack overcloud deploy** 命令中：

   a. 包含 overcloud 上 SR-IOV 网络规格的自定义网络定义文件，如 **network-data.yaml**。
   如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 网络定义文件配置选项。

   b. 包含 RHOSP director 用来部署 OVS 硬件卸载环境的 **Controller** 和 **ComputeOvsHwOffload** 角色的角色文件。
   示例： **roles_data_compute_ovshwol.yaml**

   如需更多信息，请参阅 第 8.1 节 “为 OVS TC-flower 硬件卸载生成角色和镜像文件” 。

   c. 来自置备 overcloud 网络的输出文件。
   示例： **overcloud-networks-deployed.yaml**

   如需更多信息，请参阅 *使用 director 安装和管理 Red Hat OpenStack Platform 指南* 中的配置和管理 overcloud 网络定义。

   d. 来自置备 overcloud VIP 的输出文件。

示例： **overcloud-vip-deployed.yaml**

如需更多信息，请参阅 *安装和管理 Red Hat OpenStack Platform* 指南中的 为 overcloud 配置和置备网络 VIP。

e. 置备裸机节点的输出文件。
示例： **overcloud-baremetal-deployed.yaml**

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。

f. director 用来确定获取容器镜像的位置以及如何存储它们的镜像文件。
示例： **overcloud_images.yaml**

如需更多信息，请参阅 第 8.1 节 “为 OVS TC-flower 硬件卸载生成角色和镜像文件” 。

g. 您的环境使用的网络服务(neutron)机制驱动程序和路由器方案的环境文件：

- ML2/OVN
  - 分布式虚拟路由(DVR)： **neutron-ovn-dvr-ha.yaml**
  - 集中式虚拟路由： **neutron-ovn-ha.yaml**
- ML2/OVS
  - 分布式虚拟路由(DVR)： **neutron-ovs-dvr.yaml**
  - 集中式虚拟路由： **neutron-ovs.yaml**

h. SR-IOV 的环境文件，具体取决于您的机制驱动程序：

- ML2/OVN
  - **neutron-ovn-sriov.yaml**
- ML2/OVS
  - **neutron-sriov.yaml**

注意

如果您也有一个 OVS-DPDK 环境，并希望在同一节点上找到 OVS-DPDK 和 SR-IOV 实例，请在部署脚本中包含以下环境文件：

- ML2/OVN
  neutron-ovn-dpdk.yaml
- ML2/OVS
  neutron-ovs-dpdk.yaml

i. 一个或多个包含您的配置的自定义环境文件：

- ComputeOvsHwOffload 节点的 PCI 透传设备。
- ComputeOvsHwOffload 节点的特定于角色的参数
- 覆盖 OVS 硬件卸载环境的默认配置值。

示例： **ovshwol-overrides.yaml**

如需更多信息，请参阅：

- 第 8.2 节 “为 OVS TC-flower 硬件卸载配置 PCI 透传设备”.
- 第 8.3 节 “为 OVS TC-flower 硬件卸载添加特定于角色的参数和配置覆盖”.

示例

示例 **openstack overcloud deploy** 命令摘录演示了使用 DVR 的 SR-IOV ML2/OVN 环境正确排序命令输入：

```
$ openstack overcloud deploy \
--log-file overcloud_deployment.log \
--templates /usr/share/openstack-tripleo-heat-templates/ \
--stack overcloud \
-n /home/stack/templates/network_data.yaml \
-r /home/stack/templates/roles_data_compute_ovshwol.yaml \
-e /home/stack/templates/overcloud-networks-deployed.yaml \
-e /home/stack/templates/overcloud-vip-deployed.yaml \
-e /home/stack/templates/overcloud-baremetal-deployed.yaml \
-e /home/stack/templates/overcloud-images.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-dvr-ha.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-sriov.yaml \
-e /home/stack/templates/ovshwol-overrides.yaml
```

4. 运行 **openstack overcloud deploy** 命令。
完成 overcloud 创建后，RHOSP director 会提供帮助您访问 overcloud 的详细信息。

验证

- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的执行验证 overcloud 部署 中的步骤。

后续步骤

1. 确保 NIC 的 e-switch 模式设置为 **switchdev**。
switchdev 模式建立映射到 VF 的 NIC 上的端口。

重要

您必须在 **switchdev** 端口上启用安全组和端口安全性，以便连接跟踪(conntrack)模块将 OpenFlow 流卸载到硬件。

a. 运行以下命令检查 NIC：

示例

在本例中，会查询 NIC **pci/0000:03:00.0** ：

```
$ sudo devlink dev eswitch show pci/0000:03:00.0
```

输出示例

您应该看到类似如下的输出：

```
pci/0000:03:00.0: mode switchdev inline-mode none encap enable
```

b. 要将 NIC 设置为 **switchdev** 模式，请运行以下命令：

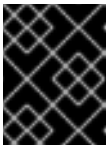
示例

在本例中，NIC **pci/0000:03:00.0** 的 e-switch 模式被设置为 **switchdev** ：

```
$ sudo devlink dev eswitch set pci/0000:03:00.0 mode switchdev
```

2. 要从 **switchdev**-enabled NIC 分配端口，请执行以下操作：

a. 以具有 **admin** 角色的 RHOSP 用户身份登录，并使用 **binding-profile** 值 **的功能** 创建 neutron 端口，并禁用端口安全性：



重要

您必须在 **switchdev** 端口上启用安全组和端口安全性，以便连接跟踪(conntrack)模块将 OpenFlow 流卸载到硬件。

```
$ openstack port create --network private --vnic-type=direct --binding-profile
'{"capabilities": ["switchdev"]}' direct_port1 --disable-port-security
```

b. 在创建实例时传递此端口信息。
您可以将代表器端口与实例 VF 接口关联，并将代表器端口连接到 OVS 网桥 **br-int**，以进行一次性 OVS 数据路径处理。VF 端口代表程序功能，如物理"patch 面板"前端的软件版本。

有关创建新实例的更多信息，请参阅 第 8.8 节 “在 SR-IOV 或 OVS TC-flower 硬件卸载环境中创建实例”。

3. 在接口和代表端口上应用以下配置，以确保 TC 流程序在端口级别推送流编程：

```
$ sudo ethtool -K <device-name> hw-tc-offload on
```

4. 调整每个网络接口的频道数以提高性能。
频道包括中断请求(IRQ)以及触发 IRQ 的队列集合。当您将 **mlx5_core** 驱动程序设置为 **switchdev** 模式时，**mlx5_core** 驱动程序默认为一个组合频道，它可能无法提供最佳性能。

在物理功能(PF)代表器上，输入以下命令调整主机可用的 CPU 数量。

示例

在这个示例中，网络接口中的多用途频道数量被设置为 **3**，**eno3s0f0**:

```
$ sudo ethtool -L enp3s0f0 combined 3
```

其他资源

- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的创建 overcloud

- *命令行界面参考*中的 overcloud 部署
- 第 8.8 节 "在 SR-IOV 或 OVS TC-flower 硬件卸载环境中创建实例"
- **ethtool**的 man page
- **devlink**的 man page
- 在 *为实例创建 配置 Compute 服务中的在 Compute 节点上配置 CPU 固定*

8.7. 在 SR-IOV 或 OVS TC-FLOWER 硬件卸载环境中创建主机聚合

为了在 Red Hat OpenStack Platform (RHOSP) SR-IOV 或 OVS TC-flower 硬件卸载环境中提高性能，请部署具有 CPU 固定和巨页的客户机。您可以通过匹配聚合元数据和类别元数据，在主机子集中调度高性能实例。

先决条件

- 为 SR-IOV 或 OVS 硬件卸载环境配置的 RHOSP overcloud。
- 必须为 **AggregateInstanceExtraSpecsFilter** 配置您的 RHOSP overcloud。
  如需更多信息，请参阅 第 8.2 节 "为 OVS TC-flower 硬件卸载配置 PCI 透传设备" 。

流程

1. 创建聚合组，并添加相关主机。
   定义与定义的类别元数据匹配的元数据，如 **sriov=true**。

```
$ openstack aggregate create sriov_group
$ openstack aggregate add host sriov_group compute-sriov-0.localdomain
$ openstack aggregate set --property sriov=true sriov_group
```

2. 创建类别。

```
$ openstack flavor create <flavor> --ram <size_mb> --disk <size_gb> \
--vcpus <number>
```

3. 设置其他类别属性。
   请注意，定义的元数据 **sriov=true** 与 SR-IOV 聚合中定义的元数据匹配。

```
$ openstack flavor set --property sriov=true \
--property hw:cpu_policy=dedicated \
--property hw:mem_page_size=1GB <flavor>
```

其他资源

- *命令行界面参考中的 聚合*
- *命令行界面参考中的 flavor*

8.8. 在 SR-IOV 或 OVS TC-FLOWER 硬件卸载环境中创建实例

您可以使用多个命令在 Red Hat OpenStack Platform (RHOSP) SR-IOV 或 OVS TC-flower 硬件卸载环境中创建实例。

使用主机聚合来分隔高性能计算主机。如需更多信息，请参阅 第 8.7 节 “在 SR-IOV 或 OVS TC-flower 硬件卸载环境中创建主机聚合”。

注意

固定 CPU 实例可以位于与未固定实例相同的 Compute 节点上。如需更多信息，请参阅配置 Compute *服务以进行实例创建指南中的在 Compute* 节点上配置 CPU 固定。

先决条件

- 为 SR-IOV 或 OVS 硬件卸载环境配置的 RHOSP overcloud。
- 对于 OVS 硬件卸载环境，您必须有虚拟功能(VF)端口或 RHOSP 管理员的物理功能(PF)端口才能创建实例。
  OVS 硬件卸载需要绑定配置集来创建 VF 或 PF。只有具有 **admin** 角色的 RHOSP 用户才能使用绑定配置集。

流程

1. 创建类别。

```
$ openstack flavor create <flavor_name> --ram <size_mb> \
--disk <size_gb> --vcpus <number>
```

提示

您可以通过将额外的 spec **hw:pci_numa_affinity_policy** 添加到类别，为 PCI 透传设备和 SR-IOV 接口指定 NUMA 关联性策略。有关更多信息，请参阅配置 *计算服务以进行实例创建中的类别元数据* 。

2. 创建网络和子网：

```
$ openstack network create <network_name> \
--provider-physical-network tenant \
--provider-network-type vlan --provider-segment <vlan_id>

$ openstack subnet create <name> --network <network_name> \
--subnet-range <ip_address_cidr> --dhcp
```

3. 如果您不是具有 **admin** 角色的 RHOSP 用户，您的 RHOSP 管理员可以为您提供所需的 VF 或 PF 来创建实例。继续执行第 5 步。
4. 如果您是具有 **admin** 角色的 RHOSP 用户，您可以创建 VF 或 PF 端口：
   - VF 端口：

```
$ openstack port create --network <network_name> --vnic-type direct \
--binding-profile '{"capabilities": ["switchdev"]} <port_name>
```

   - 专用于单个实例的 PF 端口：

此 PF 端口是一个网络服务(neutron)端口，但不由网络服务控制，因此不能作为网络适配器可见，因为它是传递给实例的 PCI 设备。

```
$ openstack port create --network <network_name> \
--vnic-type direct-physical <port_name>
```

5. 创建一个实例。

```
$ openstack server create --flavor <flavor> --image <image_name> \
--nic port-id=<id> <instance_name>
```

其他资源

- *命令行界面参考中的 flavor create*
- *命令行界面参考*中的 network create
- *命令行接口参考*中的 subnet create
- *命令行界面参考*中的 port create
- *命令行接口参考*中的 server create

8.9. OVS TC-FLOWER 硬件卸载故障排除

当对使用 OVS TC-flower 硬件卸载的 Red Hat OpenStack Platform (RHOSP)环境进行故障排除时，请查看网络和接口的先决条件和配置。

先决条件

- Linux 内核 4.13 或更新版本
- OVS 2.8 或更新版本
- RHOSP 12 或更高版本
- iproute 4.12 或更新版本
- Mellanox NIC 固件，如 FW ConnectX-5 16.21.0338 或更新版本

有关支持的先决条件的更多信息，请参阅红帽知识库解决方案网络适配器 Fast Datapath 功能支持列表。

网络配置

在 HW 卸载部署中，您可以根据您的要求为网络配置选择以下场景之一：

- 您可以使用附加到绑定的相同接口集或每种类型的不同 NIC 集来在 VXLAN 和 VLAN 上基本客户虚拟机。
- 您可以使用 Linux 绑定绑定 Mellanox NIC 的两个端口。
- 您可以在 Mellanox Linux 绑定顶部的 VLAN 接口上托管租户 VXLAN 网络。

确保单个 NIC 和绑定是 ovs-bridge 的成员。

请参阅以下网络配置示例：

```
...
- type: ovs_bridge
  name: br-offload
  mtu: 9000
  use_dhcp: false
  members:
  - type: linux_bond
    name: bond-pf
    bonding_options: "mode=active-backup miimon=100"
    members:
    - type: sriov_pf
      name: p5p1
      numvfs: 3
      primary: true
      promisc: true
      use_dhcp: false
      defroute: false
      link_mode: switchdev
    - type: sriov_pf
      name: p5p2
      numvfs: 3
      promisc: true
      use_dhcp: false
      defroute: false
      link_mode: switchdev
...
 - type: vlan
   vlan_id:
     get_param: TenantNetworkVlanID
   device: bond-pf
   addresses:
   - ip_netmask:
       get_param: TenantIpSubnet
...
```

支持以下绑定配置：

- active-backup - mode=1
- active-active or balance-xor - mode=2
- 802.3ad (LACP) - mode=4

不支持以下绑定配置：

- xmit_hash_policy=layer3+4

接口配置

使用以下步骤验证接口配置。

流程

1. 在部署过程中，使用主机网络配置工具 **os-net-config** 启用 **hw-tc-offload**。
2. 每当您重启 Compute 节点时，在 **sriov_config** 服务上启用 **hw-tc-offload**。

3. 对于附加到绑定的 NIC，将 **hw-tc-offload** 参数设置为 **on**。

示例

```
$ ethtool -k ens1f0 | grep tc-offload

hw-tc-offload: on
```

接口模式

使用以下步骤验证接口模式。

流程

1. 将 eswitch 模式设置为用于 HW 卸载的接口的 **switchdev**。
2. 使用主机网络配置工具 **os-net-config** 在部署期间启用 **eswitch**。
3. 每当重新引导 Compute 节点时，在 **sriov_config** 服务上启用 **eswitch**。

示例

```
$ devlink dev eswitch show pci/$(ethtool -i ens1f0 | grep bus-info \
| cut -d ':' -f 2,3,4 | awk '{$1=$1};1')
```

注意

PF 接口的驱动程序被设置为 **"mlx5e_rep"**，以显示它是 e-switch uplink 端口的代表器。这不会影响功能。

OVS 卸载状态

使用以下步骤验证 OVS 卸载状态。

- 在 Compute 节点中启用 OVS 中的硬件卸载。

```
$ ovs-vsctl get Open_vSwitch . other_config:hw-offload

"true"
```

VF 代表端口名称

为确保 VF 的一致性命名器端口，**os-net-config** 使用 udev 规则重命名 <PF-name>_<VF_id> 格式的端口。

流程

- 部署后，验证 VF 代表器端口是否正确被命名。

示例

```
$ cat /etc/udev/rules.d/80-persistent-os-net-config.rules
```

输出示例

```
# This file is autogenerated by os-net-config

SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}!="",
ATTR{phys_port_name}=="pf*vf*", ENV{NM_UNMANAGED}="1"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", KERNELS=="0000:65:00.0",
NAME="ens1f0"
SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}=="98039b7f9e48",
ATTR{phys_port_name}=="pf0vf*", IMPORT{program}="/etc/udev/rep-link-name.sh
$attr{phys_port_name}", NAME="ens1f0_$env{NUMBER}"
SUBSYSTEM=="net", ACTION=="add", DRIVERS=="?*", KERNELS=="0000:65:00.1",
NAME="ens1f1"
SUBSYSTEM=="net", ACTION=="add", ATTR{phys_switch_id}=="98039b7f9e49",
ATTR{phys_port_name}=="pf1vf*", IMPORT{program}="/etc/udev/rep-link-name.sh
$attr{phys_port_name}", NAME="ens1f1_$env{NUMBER}"
```

网络流量流

HW 卸载网络流功能类似于具有特定于应用程序集成电路(ASIC)芯片的物理交换机或路由器。

您可以访问交换机或路由器的 ASIC shell，以检查路由表和进行其他调试。以下流程使用 Cumulus Linux 交换机中的 Broadcom 芯片组作为示例。替换适合您的环境的值。

流程

1. 要获取 Broadcom 芯片表内容，请使用 **bcmcmd** 命令。

```
$ cl-bcmcmd l2 show
```

输出示例

```
mac=00:02:00:00:00:08 vlan=2000 GPORT=0x2 modid=0 port=2/xe1
mac=00:02:00:00:00:09 vlan=2000 GPORT=0x2 modid=0 port=2/xe1 Hit
```

2. 检查流量控制(TC)层。

```
$ tc -s filter show dev p5p1_1 ingress
```

输出示例

```
…
filter block 94 protocol ip pref 3 flower chain 5
filter block 94 protocol ip pref 3 flower chain 5 handle 0x2
  eth_type ipv4
  src_ip 172.0.0.1
  ip_flags nofrag
  in_hw in_hw_count 1
        action order 1: mirred (Egress Redirect to device eth4) stolen
        index 3 ref 1 bind 1 installed 364 sec used 0 sec
        Action statistics:
        Sent 253991716224 bytes 169534118 pkt (dropped 0, overlimits 0 requeues 0)
        Sent software 43711874200 bytes 30161170 pkt
        Sent hardware 210279842024 bytes 139372948 pkt
```

```
        backlog 0b 0p requeues 0
        cookie 8beddad9a0430f0457e7e78db6e0af48
        no_percpu
```

3. 检查此输出中的 **in_hw** 标志和统计信息。词语 **hardware** 表示硬件处理网络流量。如果使用 **tc-policy=none**，您可以检查此输出或 tcpdump 在硬件或软件处理数据包时进行调查。当驱动程序无法卸载数据包时，您可以在 **dmesg** 或 **ovs-vswitch.log** 中看到对应的日志消息。

4. 对于 Mellanox，例如，在 **dmesg** 中类似复合消息的日志条目。

   输出示例

   ```
   [13232.860484] mlx5_core 0000:3b:00.0: mlx5_cmd_check:756:(pid 131368):
   SET_FLOW_TABLE_ENTRY(0x936) op_mod(0x0) failed, status bad parameter(0x3),
   syndrome (0x6b1266)
   ```

   在本例中，错误代码(0x6b1266)代表以下行为：

   输出示例

   ```
   0x6B1266 |  set_flow_table_entry: pop vlan and forward to uplink is not allowed
   ```

系统

使用以下步骤验证您的系统。

流程

1. 确保系统上启用了 SR-IOV 和 VT-d。
2. 通过在内核参数中添加 **intel_iommu=on** 在 Linux 中启用 IOMMU，例如使用 GRUB。

8.10. 调试 TC-FLOWER 硬件卸载流

如果您在 **ovs-vswitch.log** 文件中遇到以下信息，您可以使用以下步骤：

```
2020-01-31T06:22:11.257Z|00473|dpif_netlink(handler402)|ERR|failed to offload flow: Operation not
supported: p6p1_5
```

流程

1. 要在卸载模块上启用日志记录并获取此故障的额外日志信息，请在 Compute 节点上使用以下命令：

   ```
   ovs-appctl vlog/set dpif_netlink:file:dbg
   # Module name changed recently (check based on the version used
   ovs-appctl vlog/set netdev_tc_offloads:file:dbg [OR] ovs-appctl vlog/set
   netdev_offload_tc:file:dbg
   ovs-appctl vlog/set tc:file:dbg
   ```

2. 再次检查 **ovs-vswitchd** 日志，以查看此问题的更多详情。
   在以下示例日志中，因为不支持的属性标记，卸载会失败。

```
 2020-01-31T06:22:11.218Z|00471|dpif_netlink(handler402)|DBG|system@ovs-system:
put[create] ufid:61bd016e-eb89-44fc-a17e-958bc8e45fda
recirc_id(0),dp_hash(0/0),skb_priority(0/0),in_port(7),skb_mark(0),ct_state(0/0),ct_zone(0/0),ct
_mark(0/0),ct_label(0/0),eth(src=fa:16:3e:d2:f5:f3,dst=fa:16:3e:c4:a3:eb),eth_type(0x0800),ipv
4(src=10.1.1.8/0.0.0.0,dst=10.1.1.31/0.0.0.0,proto=1/0,tos=0/0x3,ttl=64/0,frag=no),icmp(type=0/
0,code=0/0),
actions:set(tunnel(tun_id=0x3d,src=10.10.141.107,dst=10.10.141.124,ttl=64,tp_dst=4789,flags(
df|key))),6

2020-01-31T06:22:11.253Z|00472|netdev_tc_offloads(handler402)|DBG|offloading attribute
pkt_mark isn't supported

2020-01-31T06:22:11.257Z|00473|dpif_netlink(handler402)|ERR|failed to offload flow:
Operation not supported: p6p1_5
```

调试 Mellanox NIC

Mellanox 提供了系统信息脚本，类似于 Red Hat SOS 报告。

https://github.com/Mellanox/linux-sysinfo-snapshot/blob/master/sysinfo-snapshot.py

运行此命令时，您可以创建一个相关日志信息的 zip 文件，这对支持问题单很有用。

流程

- 您可以使用以下命令运行这个系统信息脚本：

```
# ./sysinfo-snapshot.py --asap --asap_tc --ibdiagnet --openstack
```

您还可以安装 Mellanox Firmware Tools (MFT)、mlxconfig、mlxlink 和 OpenFabrics Enterprise Distribution (OFED)驱动程序。

有用的 CLI 命令

使用带有以下选项的 **ethtool** 工具来收集诊断信息：

- ethtool -l <uplink representor> : 查看频道数
- ethtool -I <uplink/VFs> : Check statistics
- ethtool -i <uplink rep> : View driver information
- ethtool -g <uplink rep> : Check ring size
- ethtool -k <uplink/VFs> : View enabled features

使用代表器和 PF 端口的 **tcpdump** 工具来类似检查流量流。

- 对代表器端口的链接状态所做的任何更改也会影响 VF 链接状态。
- Representor 端口统计也存在 VF 统计。

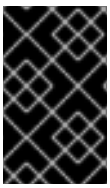
使用以下命令获取有用的诊断信息：

```
$ ovs-appctl dpctl/dump-flows -m type=offloaded
```

```
$ ovs-appctl dpctl/dump-flows -m

$ tc filter show dev ens1_0 ingress

$ tc -s filter show dev ens1_0 ingress

$ tc monitor
```

第 9 章 规划您的 OVS-DPDK 部署

要使用 NFV 的 Data Plane Development Kit (OVS-DPDK)部署优化 Open vSwitch，您应该了解 OVS-DPDK 如何使用 Compute 节点硬件(CPU、NUMA 节点、内存、NIC)以及根据 Compute 节点确定单个 OVS-DPDK 参数的注意事项。



重要

在使用 OVS-DPDK 和 OVS 原生防火墙（基于 conntrack 的有状态防火墙）时，您只能跟踪使用 ICMPv4、ICMPv6、TCP 和 UDP 协议的数据包。OVS 将所有其他类型的网络流量标记为无效。

重要

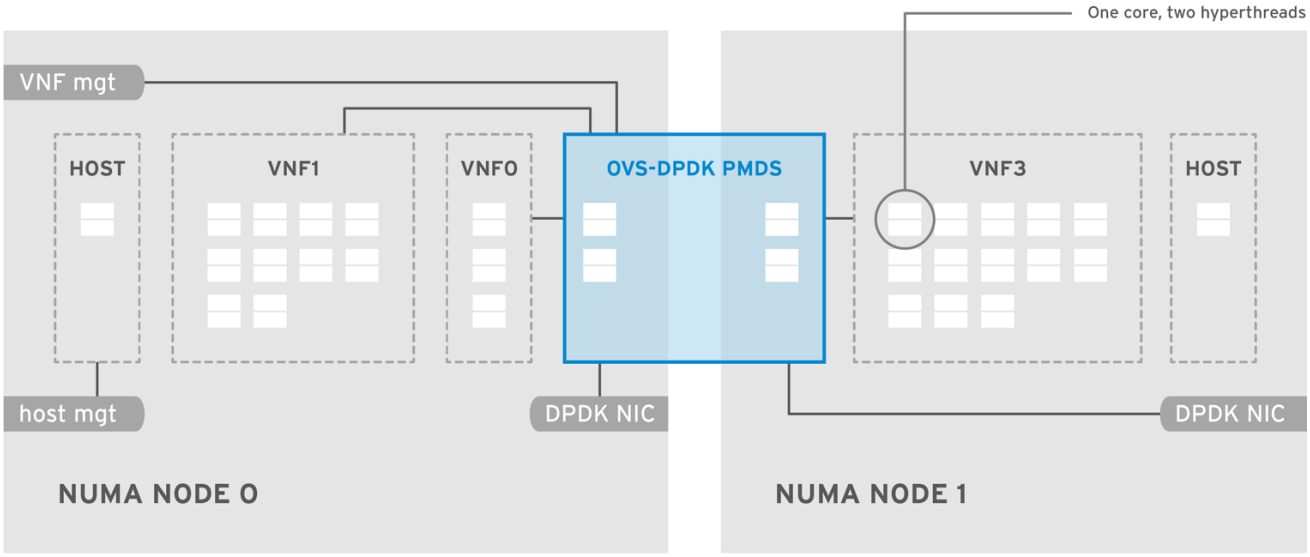
红帽不支持将 OVS-DPDK 用于非 NFV 工作负载。如果您需要用于非 NFV 工作负载的 OVS-DPDK 功能，请联系您的大客户经理(TAM)或打开客户服务请求案例，以讨论支持例外和其他选项。要创建一个客户服务请求问题单，请访问创建一个问题单并选择 **Account > Customer Service Request**。

9.1. 带有 CPU 分区和 NUMA 拓扑的 OVS-DPDK

OVS-DPDK 对主机、客户机本身的硬件资源进行分区。OVS-DPDK 轮询模式驱动程序(PMD)运行 DPDK 活跃的循环，这需要专用 CPU 内核。因此，您必须将一些 CPU 和巨页分配给 OVS-DPDK。

示例分区包括双插槽 Compute 节点上的每个 NUMA 节点 16 个内核。流量需要额外的 NIC，因为您无法在主机和 OVS-DPDK 之间共享 NIC。

图 9.1. NUMA 拓扑：带有 CPU 分区的 OVS-DPDK



注意

您必须在两个 NUMA 节点上保留 DPDK PMD 线程，即使 NUMA 节点没有关联的 DPDK NIC。

为获得最佳 OVS-DPDK 性能，请将本地内存块保留给 NUMA 节点。选择与用于内存和 CPU 固定的同一 NUMA 节点关联的 NIC。确保两个绑定接口都来自同一 NUMA 节点上的 NIC。

9.2. OVS-DPDK 参数

本节介绍 OVS-DPDK 如何使用 director **network_environment.yaml** heat 模板中的参数来配置 CPU 和内存以获得最佳性能。使用此信息来评估 Compute 节点上的硬件支持以及如何对硬件进行分区，以优化您的 OVS-DPDK 部署。

注意

在分配 CPU 内核时，始终将物理内核中的 CPU 同级线程或逻辑 CPU 组合在一起。

有关如何确定 Compute 节点上的 CPU 和 NUMA 节点的详情，请参阅发现 NUMA 节点拓扑。使用此信息映射 CPU 和其他参数，以支持主机、客户机实例和 OVS-DPDK 进程需求。

9.2.1. CPU 参数

OVS-DPDK 将以下参数用于 CPU 分区：

OvsPmdCoreList

提供用于 DPDK 轮询模式驱动程序(PMD)的 CPU 内核。选择与 DPDK 接口本地 NUMA 节点关联的 CPU 内核。将 **OvsPmdCoreList** 用于 OVS 中的 **pmd-cpu-mask** 值。对 **OvsPmdCoreList** 使用以下建议：

- 将同级线程配对在一起。
- 性能取决于为此 PMD Core 列表分配的物理内核数。在与 DPDK NIC 关联的 NUMA 节点上，分配所需的内核。
- 对于带有 DPDK NIC 的 NUMA 节点，根据性能要求确定所需的物理内核数，并包括每个物理内核的所有同级线程或逻辑 CPU。
- 对于没有 DPDK NIC 的 NUMA 节点，分配任何物理内核的同级线程或逻辑 CPU，但 NUMA 节点的第一个物理内核除外。

注意

您必须在两个 NUMA 节点上保留 DPDK PMD 线程，即使 NUMA 节点没有关联的 DPDK NIC。

NovaComputeCpuDedicatedSet

可以调度用于固定实例 CPU 的进程的逗号分隔列表或物理主机 CPU 范围。例如，**NovaComputeCpuDedicatedSet: [4-12,^8,15]** 保留来自 4-12 和 15 的核心，不包括 8。

- 从 **OvsPmdCoreList** 中排除所有内核。
- 包括所有剩余的内核。
- 将同级线程配对在一起。

NovaComputeCpuSharedSet

用来确定实例仿真程序线程的主机 CPU 号的逗号分隔列表或物理主机 CPU 范围。

IsolCpusList

与主机进程隔离的一组 CPU 内核。**IsolCpusList** 是 **cpu-partitioning-variable.conf** 文件中的 **isolated_cores** 值，用于 **tuned-profiles-cpu-partitioning** 组件。对 **IsolCpusList** 使用以下建议：

- 匹配 **OvsPmdCoreList** 和 **NovaComputeCpuDedicatedSet** 中的内核列表。
- 将同级线程配对在一起。

DerivePciWhitelistEnabled

要为虚拟机保留虚拟功能(VF)，请使用 **NovaPCIPassthrough** 参数创建传递给 Nova 的 VF 列表。列表中排除的 VF 仍可用于主机。
对于列表中的每个 VF，使用解析为 address 值的正则表达式填充 address 参数。

以下是手动列表创建过程的示例。如果在名为 **eno2** 的设备中启用 NIC 分区，使用以下命令列出 VF 的 PCI 地址：

```
[tripleo-admin@compute-0 ~]$ ls -lh /sys/class/net/eno2/device/ | grep virtfn
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn0 -> ../0000:18:06.0
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn1 -> ../0000:18:06.1
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn2 -> ../0000:18:06.2
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn3 -> ../0000:18:06.3
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn4 -> ../0000:18:06.4
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn5 -> ../0000:18:06.5
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn6 -> ../0000:18:06.6
lrwxrwxrwx. 1 root root    0 Apr 16 09:58 virtfn7 -> ../0000:18:06.7
```

在这种情况下，NIC Partitioning 的 **eno2** 使用 VF 0、4 和 6。手动配置 **NovaPCIPassthrough** 使其包含 VF 1-3、5 和 7，因此排除 VF 0、4 和 6，如下例所示：

```
NovaPCIPassthrough:
  - physical_network: "sriovnet2"
  address: {"domain": ".*", "bus": "18", "slot": "06", "function": "[1-3]"}
  - physical_network: "sriovnet2"
  address: {"domain": ".*", "bus": "18", "slot": "06", "function": "[5]"}
  - physical_network: "sriovnet2"
  address: {"domain": ".*", "bus": "18", "slot": "06", "function": "[7]"}
```

9.2.2. 内存参数

OVS-DPDK 使用以下内存参数：

OvsDpdkMemoryChannels

映射每个 NUMA 节点的 CPU 中的内存频道。**OvsDpdkMemoryChannels** 是 OVS 中的 **other_config:dpdk-extra="-n <value>"** 值。观察 **OvsDpdkMemoryChannels** 的以下建议：

- 使用 **dmidecode -t memory** 或您的硬件手册来确定可用的内存通道数。
- 使用 **ls /sys/devices/system/node/node* -d** 来确定 NUMA 节点的数量。
- 将可用内存通道的数量除以 NUMA 节点数。

NovaReservedHostMemory

为主机上的任务保留内存（以 MB 为单位）。**NovaReservedHostMemory** 是 **nova.conf** 中 Compute 节点的 **reserved_host_memory_mb** 值。观察以下 **NovaReservedHostMemory** 的建议：

- 使用静态推荐值 4096 MB。

OvsDpdkSocketMemory

指定每个 NUMA 节点从巨页池预先分配的内存量（以 MB 为单位）。**OvsDpdkSocketMemory** 是 OVS 中的 **other_config:dpdk-socket-mem** 值。观察 **OvsDpdkSocketMemory** 的以下建议：

- 以逗号分隔列表形式提供。
- 对于没有 DPDK NIC 的 NUMA 节点，请使用静态推荐 1024 MB (1GB)
- 从 NUMA 节点上的每个 NIC 的 MTU 值计算 **OvsDpdkSocketMemory** 值。
- 以下 equation 大约是 **OvsDpdkSocketMemory** 的值：
  - MEMORY_REQD_PER_MTU = (ROUNDUP_PER_MTU + 800) * (4096 * 64) Bytes
    - 800 是开销值。
    - 4096 * 64 是 mempool 中的数据包数量。
- 为 NUMA 节点上设置的每个 MTU 值添加 MEMORY_REQD_PER_MTU，并添加另一个 512 MB 作为缓冲区。将值设为 1024 的倍数。

Calculation 示例 - MTU 2000 和 MTU 9000

DPDK NIC dpdk0 和 dpdk1 位于同一 NUMA 节点 0 上，并且分别配置了 MTU 9000 和 2000。派生内存的计算示例如下：

1. 将 MTU 值舍入到最接近的 1024 字节。

```
The MTU value of 9000 becomes 9216 bytes.
The MTU value of 2000 becomes 2048 bytes.
```

2. 根据这些舍入字节值，计算每个 MTU 值所需的内存。

```
Memory required for 9000 MTU = (9216 + 800) * (4096*64) = 2625634304
Memory required for 2000 MTU = (2048 + 800) * (4096*64) = 746586112
```

3. 计算所需的组合内存总量，以字节为单位。

```
2625634304 + 746586112 + 536870912 = 3909091328 bytes.
```

此计算代表(MTU 为 9000)+(MTU 为 2000 的内存所需的内存)+(512 MB 缓冲区)。

4. 将所需的总内存转换为 MB。

```
3909091328 / (1024*1024) = 3728 MB.
```

5. 将该值向上舍入到最接近的 1024。

```
3724 MB rounds up to 4096 MB.
```

6. 使用这个值设置 **OvsDpdkSocketMemory**。

```
OvsDpdkSocketMemory: "4096,1024"
```

Calculation 示例 - MTU 2000

DPDK NIC dpdk0 和 dpdk1 位于同一 NUMA 节点 0 上，各自配置有 2000 的 MTU。派生内存的计算示例如下：

1. 将 MTU 值舍入到最接近的 1024 字节。

```
The MTU value of 2000 becomes 2048 bytes.
```

2. 根据这些舍入字节值，计算每个 MTU 值所需的内存。

```
Memory required for 2000 MTU = (2048 + 800) * (4096*64) = 746586112
```

3. 计算所需的组合内存总量，以字节为单位。

```
746586112 + 536870912 = 1283457024 bytes.
```

此计算代表(MTU 为 2000 的内存需要)+(512 MB 缓冲区)。

4. 将所需的总内存转换为 MB。

```
1283457024 / (1024*1024) = 1224 MB.
```

5. 将该值向上舍入到最接近的 1024 的倍数。

```
1224 MB rounds up to 2048 MB.
```

6. 使用这个值设置 **OvsDpdkSocketMemory**。

```
OvsDpdkSocketMemory: "2048,1024"
```

9.2.3. 网络参数

OvsDpdkDriverType

设置 DPDK 使用的驱动程序类型。使用 **vfio-pci** 的默认值。

NeutronDatapathType

OVS 网桥的 datapath 类型。DPDK 使用 **netdev** 的默认值。

NeutronVhostuserSocketDir

为 OVS 设置 vhost-user 套接字目录。将 **/var/lib/vhost_sockets** 用于 vhost 客户端模式。

9.2.4. 其他参数

NovaSchedulerEnabledFilters

提供一系列过滤器，供 Compute 节点用来为请求的客户机实例查找匹配的 Compute 节点。

VhostuserSocketGroup

设置 vhost-user 套接字目录组。默认值为 **qemu**。将 **VhostuserSocketGroup** 设置为 **hugetlbfs**，以便 **ovs-vswitchd** 和 **qemu** 进程可以访问共享的巨页和 unix 套接字，以配置 virtio-net 设备。这个值特定于角色，应应用到利用 OVS-DPDK 的任何角色。

重要

要使用参数 **VhostuserSocketGroup**，还必须设置 **NeutronVhostuserSocketDir**。如需更多信息，请参阅 第 9.2.3 节 “网络参数”。

KernelArgs

在引导时为 Compute 节点提供多个内核参数 **/etc/default/grub**。根据您的配置添加以下值：

- **hugepagesz** ：设置 CPU 上巨页的大小。这个值可能因 CPU 硬件而异。为 OVS-DPDK 部署设置为 1G (**default_hugepagesz=1GB hugepagesz=1G**)。使用此命令检查 **pdpe1gb** CPU 标记，以确认您的 CPU 支持 1G。

```
lshw -class processor | grep pdpe1gb
```

- **hugepages count** ：根据可用主机内存设置可用巨页数量。使用大多数可用内存，但 **NovaReservedHostMemory** 除外。您还必须在 Compute 节点的类别中配置巨页数值。
- **IOMMU**: 对于 Intel CPU，添加 **"intel_iommu=on iommu=pt"**
- **isolcpus** ：设置用于调优的 CPU 内核。这个值与 **IsolCpusList** 匹配。

有关 CPU 隔离的更多信息，请参阅红帽知识库解决方案 OpenStack CPU 隔离指南，以了解 RHEL 8 和 RHEL 9。

DdpPackage

配置动态设备个性化(DDP)，以在部署时将配置集软件包应用到设备，以更改设备的数据包处理管道。将以下行添加到 network_environment.yaml 模板，使其包含 DDP 软件包：

```
parameter_defaults:
  ComputeOvsDpdkSriovParameters:
    DdpPackage: "ddp-comms"
```

9.2.5. VM 实例类别规格

在 NFV 环境中部署虚拟机实例之前，创建一个使用 CPU 固定、巨面和仿真器线程固定的类别。

hw:cpu_policy

当此参数设置为 **专用** 时，客户机将使用固定 CPU。从带有此参数集的类别创建的实例具有有效过量使用比例 1:1。默认值为 **shared**。

hw:mem_page_size

将此参数设置为带有标准后缀的特定值的有效字符串（例如 **4KB**、**8MB** 或 **1GB**）。使用 1GB 与 **hugepagesz** 引导参数匹配。通过从引导参数中减去 **OvsDpdkSocketMemory**，计算虚拟机可用的巨页数量。以下值也有效：

- small （默认）- 使用最小页面大小
- large - 只使用大页面大小。(2MB 或 1GB 在 x86 构架上)

- any - 计算驱动程序可以尝试使用大型页面，但如果无可用，则默认为 small。

hw:emulator_threads_policy

将此参数的值设置为 **共享**，以便仿真程序线程锁定在 heat 参数 **NovaComputeCpuSharedSet** 中标识的 CPU。如果仿真程序线程在具有轮询模式驱动程序(PMD)或实时处理的 vCPU 上运行，您可能会遇到负面影响，如数据包丢失。

9.3. 在 OVS-DPDK 部署中保存电源

在 Red Hat Enterprise Linux 9 (RHEL 9)中引入了节能配置集 **cpu-partitioning-powersave**，现在在 Red Hat OpenStack Platform (RHOSP) 17.1.3 中提供。此 TuneD 配置集是在 RHOSP 17.1 NFV 环境中节省电源的基本构建块。

先决条件

- 访问 **stack** 用户的 undercloud 主机和凭据。
- 启用要实现节能的 CPU，以允许更高的 C-states。
  如需更多信息，请参阅 **tuned-profiles-cpu-partitioning (7)** 的 man page 中的 **max_power_state** 选项。

流程

1. 以 **stack** 用户的身份登录 undercloud。
2. Source **stackrc** 文件：

```
$ source ~/stackrc
```

3. 创建一个 Ansible playbook YAML 文件，如 **/home/stack/cli-overcloud-tuned-maxpower-conf.yaml**。
4. 在 **cli-overcloud-tuned-maxpower-conf.yaml** 文件中添加以下配置：

```
cat <<EOF > /home/stack/cli-overcloud-tuned-maxpower-conf.yaml
{% raw %}
---
#/home/stack/cli-overcloud-tuned-maxpower-conf.yaml
- name: Overcloud Node set tuned power state
  hosts: compute-0 compute-1
  any_errors_fatal: true
  gather_facts: false
  pre_tasks:
    - name: Wait for provisioned nodes to boot
      wait_for_connection:
        timeout: 600
        delay: 10
      connection: local
  tasks:
    - name: Check the max power state for this system
      become: true
      block:
        - name: Get power states
          shell: "for s in /sys/devices/system/cpu/cpu2/cpuidle/*; do grep . $s/{name,latency};
```

```
done"
        register: _list_of_power_states
      - name: Print available power states
        debug:
          msg: "{{ _list_of_power_states.stdout.split('\n') }}"
      - name: Check for active tuned power-save profile
        stat:
          path: "/etc/tuned/active_profile"
        register: _active_profile
      - name: Check the profile
        slurp:
          path: "/etc/tuned/active_profile"
        when: _active_profile.stat.exists
        register: _active_profile_name
      - name: Print states
        debug:
          var: (_active_profile_name.content|b64decode|string)
      - name: Check the max power state for this system
        block:
          - name: Check if the cstate config is present in the conf file
            lineinfile:
              dest: /etc/tuned/cpu-partitioning-powersave-variables.conf
              regexp: '^max_power_state'
              line: 'max_power_state=cstate.name:C6'
            register: _cstate_entry_check
{% endraw %}
EOF
```

5. 在角色数据文件中添加节能配置集。
   如需更多信息，请参阅 10.2。生成角色和镜像文件。

6. 将 **cli-overcloud-tuned-maxpower-conf.yaml** playbook 添加到裸机节点定义文件。
   如需更多信息，请参阅 10.5。创建裸机节点定义文件。

7. 确保在 NIC 配置模板中设置了队列大小。
   如需更多信息，请参阅 10.6。创建 NIC 配置模板。

其他资源

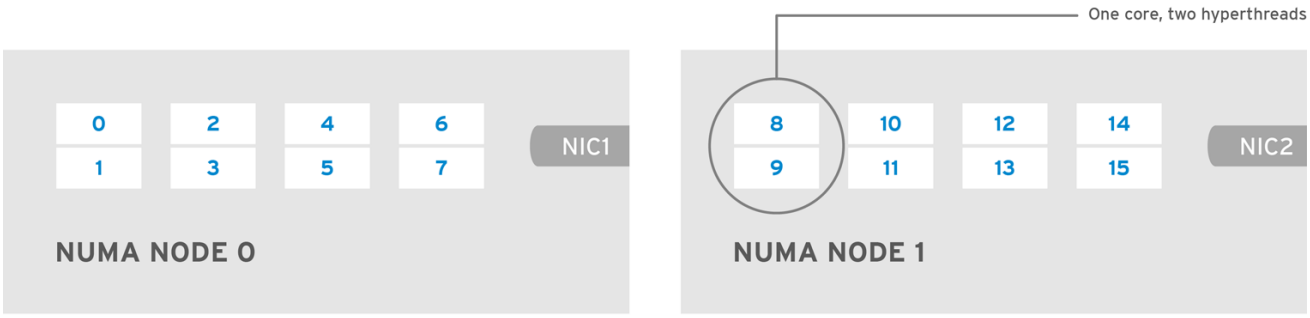
- **force_latency**

9.4. 两个 NUMA 节点示例 OVS-DPDK 部署

以下示例中的 Compute 节点包含两个 NUMA 节点：

- NUMA 0 具有内核 0-7。同级线程对为(0,1),(2,3),(4,5)和(6,7)
- NUMA 1 具有内核 8-15。同级线程对有(8,9),(10,11),(12,13)和(14,15)。
- 每个 NUMA 节点都连接到一个物理 NIC，即 NUMA 0 上的 NIC1，以及 NUMA 1 上的 NIC2。

图 9.2. OVS-DPDK：两个 NUMA 节点示例



注意

为非数据路径 DPDK 进程保留第一个物理内核或每个 NUMA 节点（0、1 和 89）的线程对。

这个示例还假设 1500 MTU 配置，因此所有用例的 **OvsDpdkSocketMemory** 都相同：

```
OvsDpdkSocketMemory: "1024,1024"
```

NIC 1 用于 DPDK，一个物理内核用于 PMD

在这种情况下，您可以为 PMD 在 NUMA 0 上分配一个物理内核。您还必须在 NUMA 1 上分配一个物理内核，即使该 NUMA 节点的 NIC 上未启用 DPDK。为客户机实例分配剩余的内核。生成的参数设置为：

```
OvsPmdCoreList: "2,3,10,11"
NovaComputeCpuDedicatedSet: "4,5,6,7,12,13,14,15"
```

NIC 1 用于 DPDK，两个物理内核用于 PMD

在这种情况下，您可以为 PMD 在 NUMA 0 上分配两个物理内核。您还必须在 NUMA 1 上分配一个物理内核，即使该 NUMA 节点的 NIC 上未启用 DPDK。为客户机实例分配剩余的内核。生成的参数设置为：

```
OvsPmdCoreList: "2,3,4,5,10,11"
NovaComputeCpuDedicatedSet: "6,7,12,13,14,15"
```

NIC 2 for DPDK，一个物理内核用于 PMD

在这种情况下，您可以为 PMD 在 NUMA 1 上分配一个物理内核。您还必须在 NUMA 0 上分配一个物理内核，即使该 NUMA 节点的 NIC 上未启用 DPDK。为客户机实例分配剩余的内核。生成的参数设置为：

```
OvsPmdCoreList: "2,3,10,11"
NovaComputeCpuDedicatedSet: "4,5,6,7,12,13,14,15"
```

NIC 2 用于 DPDK，两个物理内核用于 PMD

在这种情况下，您可以为 PMD 在 NUMA 1 上分配两个物理内核。您还必须在 NUMA 0 上分配一个物理内核，即使该 NUMA 节点的 NIC 上未启用 DPDK。为客户机实例分配剩余的内核。生成的参数设置为：

```
OvsPmdCoreList: "2,3,10,11,12,13"
NovaComputeCpuDedicatedSet: "4,5,6,7,14,15"
```

NIC 1和 NIC2 用于 DPDK，两个物理内核用于 PMD

在这种情况下，您可以在每个 NUMA 节点上为 PMD 分配两个物理内核。为客户机实例分配剩余的内核。生成的参数设置为：

```
OvsPmdCoreList: "2,3,4,5,10,11,12,13"
NovaComputeCpuDedicatedSet: "6,7,14,15"
```

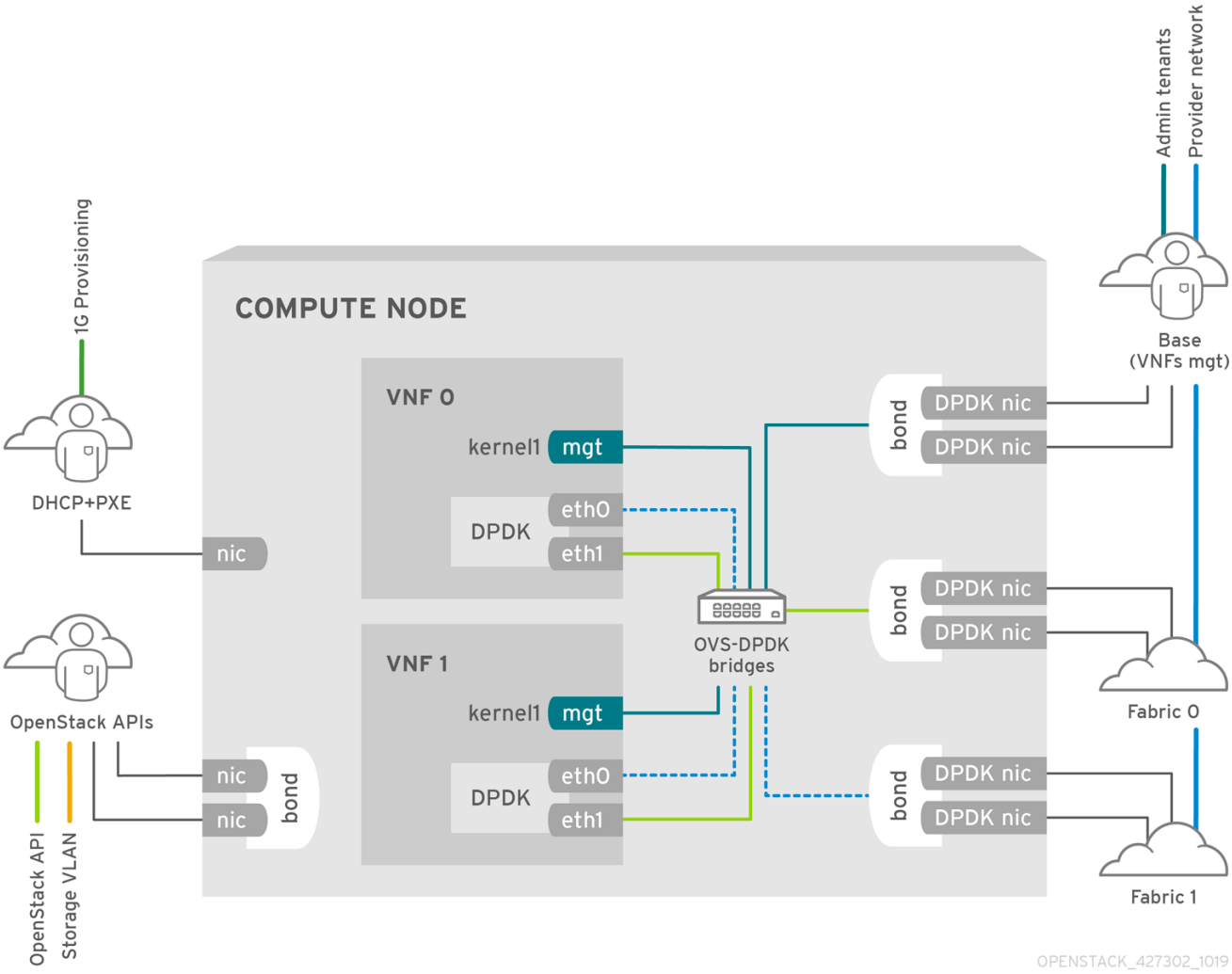
9.5. NFV OVS-DPDK 部署的拓扑

此示例部署显示了一个 OVS-DPDK 配置，它由两个虚拟网络功能(VNF)组成，每个接口有两个接口：

- 管理接口，由 **mgt** 表示。
- data plane 接口。

在 OVS-DPDK 部署中，VNF 通过支持物理接口的内建 DPDK 来运行。OVS-DPDK 在 vSwitch 级别启用绑定。为了提高 OVS-DPDK 部署中的性能，建议您将内核和 OVS-DPDK NIC 分开来。要分隔管理(**mgt**)网络，连接到虚拟机的基本提供商网络，请确保您有额外的 NIC。Compute 节点包含两个用于 Red Hat OpenStack Platform API 管理的常规 NIC，它们可以被 Ceph API 重复使用，但不能与任何 OpenStack 项目共享。

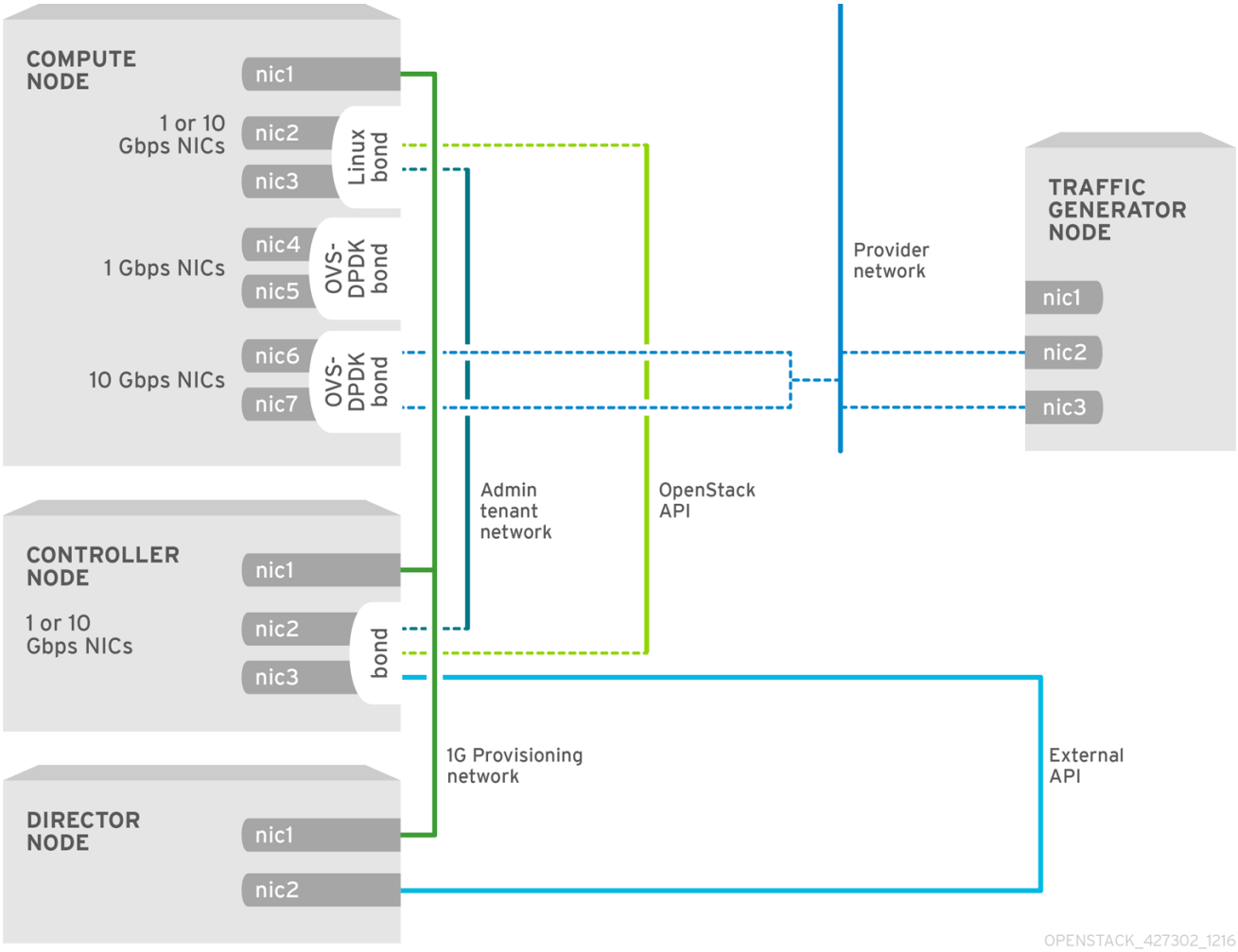
图 9.3. Compute 节点： NFV OVS-DPDK



NFV OVS-DPDK 拓扑

下图显示了 NFV 的 OVS-DPDK 的拓扑：它由具有 1 或 10 Gbps NIC 和 director 节点的 Compute 和 Controller 节点组成。

图 9.4. NFV 拓扑：OVS-DPDK

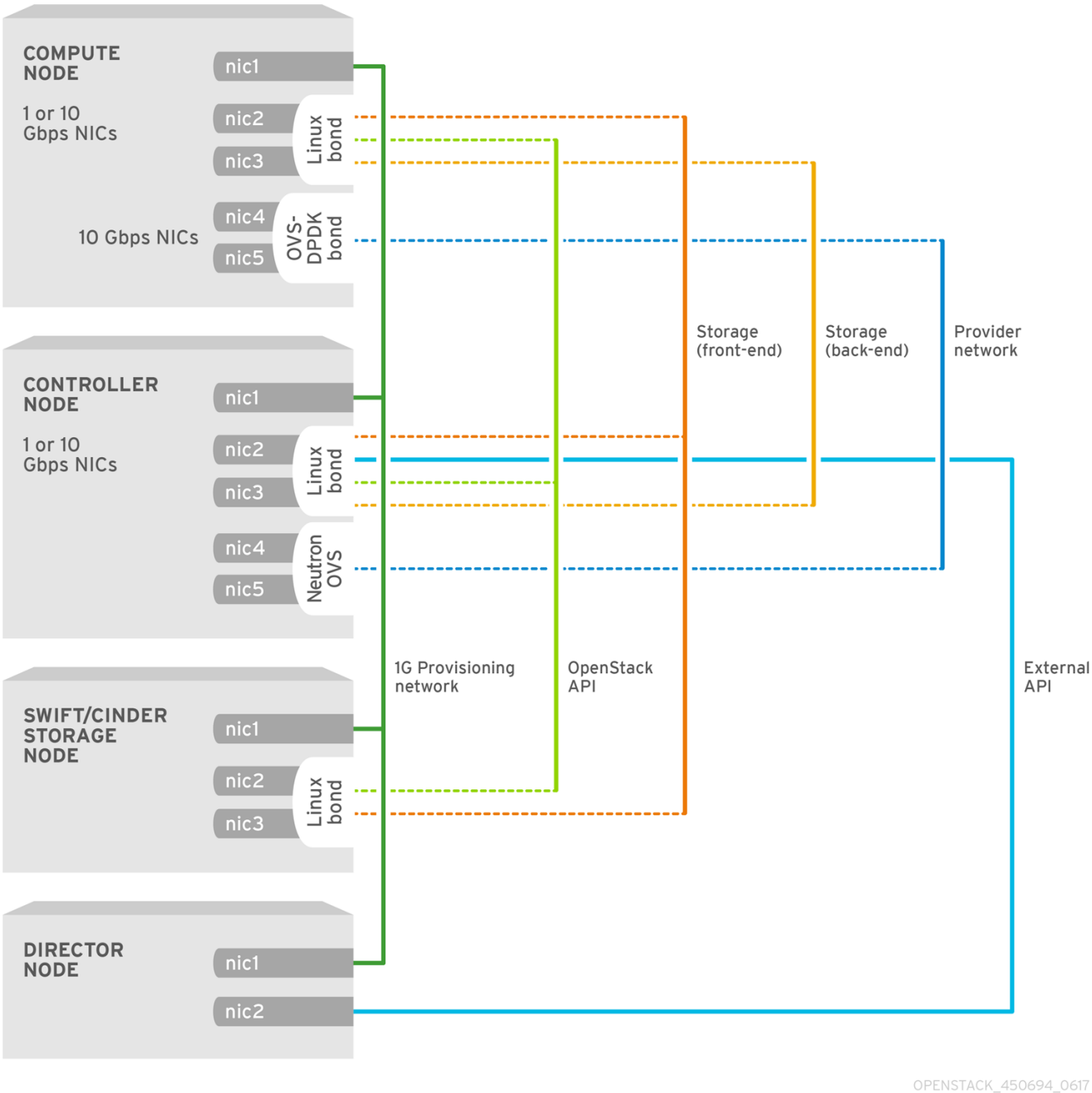


第 10 章 配置 OVS-DPDK 部署

本节论述了如何为 Red Hat OpenStack Platform (RHOSP)环境部署、使用和排除 Open vSwitch Data Plane Development Kit (OVS-DPDK)环境。RHOSP 在 OVS 客户端模式下运行，用于 OVS-DPDK 部署。

下图显示了一个 OVS-DPDK 拓扑，其 control plane 和数据平面有两个绑定端口：

图 10.1. OVS-DPDK 拓扑示例



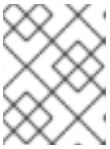
重要

本节包括必须为拓扑和用例修改的示例。如需更多信息，请参阅 NFV 的硬件要求。

先决条件

- RHOSP undercloud。

在部署 overcloud 之前，您必须安装和配置 undercloud。如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform*。



注意

RHOSP director 通过您在模板和自定义环境文件中指定的键值对修改 OVS-DPDK 配置文件。您不能直接修改 OVS-DPDK 文件。

- 访问 **stack** 用户的 undercloud 主机和凭据。

流程

使用 Red Hat OpenStack Platform (RHOSP) director 在 RHOSP 环境中安装和配置 OVS-DPDK。高级步骤有：

1. 检查 OVS-DPDK 的已知限制。
2. 生成角色和镜像文件。
3. 为您的 OVS-DPDK 自定义创建环境文件。
4. 为安全组 配置防火墙。
5. 创建裸机节点定义文件。
6. 创建 NIC 配置模板。
7. 设置 OVS-DPDK 接口的 MTU 值。
8. 为 OVS-DPDK 接口设置多队列。
9. 为节点置备 配置 DPDK 参数。
10. 置备 overcloud 网络和 VIP。
    如需更多信息，请参阅：
    - *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 配置和置备 overcloud 网络定义。
    - *使用 director 安装和管理 Red Hat OpenStack Platform 指南中的为 overcloud 配置和置备网络 VIP*。
11. 置备裸机节点。
    使用 director 安装和管理 Red Hat OpenStack Platform 指南中的 为 overcloud 置备裸机节点。
12. 部署 OVS-DPDK overcloud。

其他资源

- 第 10.11 节 “为 OVS-DPDK 创建类别和部署实例”
- 第 10.12 节 “对 OVS-DPDK 配置进行故障排除”

10.1. OVS-DPDK 的已知限制

在 Open vSwitch Data Plane Development Kit (OVS-DPDK)环境中配置 Red Hat OpenStack Platform 时观察以下限制：

- 将 Linux 绑定用于非 DPDK 流量和 control plane 网络，如内部、管理、存储、存储管理和租户。确保绑定中使用的 PCI 设备位于同一 NUMA 节点上以实现最佳性能。红帽不支持 Neutron Linux 网桥配置。
- 对于在带有 OVS-DPDK 的主机上运行的每个实例，您需要巨页。如果客户机中没有巨页，接口会出现，但无法正常工作。
- 使用 OVS-DPDK 时，使用 tap 设备（如分布式虚拟路由(DVR)）的服务的性能降低。生成的性能不适用于生产环境。
- 在使用 OVS-DPDK 时，同一 Compute 节点上的所有网桥都必须是 **ovs_user_bridge** 类型。director 可以接受配置，但 Red Hat OpenStack Platform 不支持在同一节点上混合 **ovs_bridge** 和 **ovs_user_bridge**。

后续步骤

- 继续 第 10.2 节 “生成角色和镜像文件”。

10.2. 生成角色和镜像文件

Red Hat OpenStack Platform (RHOSP) director 使用角色为节点分配服务。在 OVS-DPDK 环境中部署 RHOSP 时，ComputeOvsDpdk 是 RHOSP 安装提供的自定义角色，它除默认的计算服务外，还包括 **ComputeNeutronOvsDpdk** 服务。

undercloud 安装需要一个环境文件来确定从何处获取容器镜像以及如何存储它们。

先决条件

- **访问 stack 用户的 undercloud 主机和凭据。**

流程

1. **以 stack 用户的身份登录 undercloud。**
2. **Source stackrc 文件：**

```
$ source ~/stackrc
```

3. **生成一个新的角色数据文件，如 roles_data_compute_ovsdpdk.yaml，其中包含 Controller 和 ComputeOvsDpdk 角色：**

```
$ openstack overcloud roles generate \
-o /home/stack/templates/roles_data_compute_ovsdpdk.yaml \
Controller ComputeOvsDpdk
```

注意

如果您在 RHOSP 环境、OVS-DPDK、SR-IOV 和 OVS 硬件卸载中使用多种技术，则仅生成一个角色数据文件来包含所有角色：

```
$ openstack overcloud roles generate -o /home/stack/templates/\
roles_data.yaml Controller ComputeOvsDpdk ComputeOvsDpdkSriov \
Compute:ComputeOvsHwOffload
```

4. **可选：您可以使用 TuneD 配置集 cpu-partitioning-powersave，将 OVS-DPDK 配置为在没有转发数据包时进入睡眠模式。**

要配置 cpu-partitioning-powersave，请在生成的角色数据文件中将默认的 TuneD 配置集替换为节能 TuneD 配置集 cpu-partitioning-powersave ：

```
TunedProfileName: "cpu-partitioning-powersave"
```

示例

在这个生成的角色数据文件中，/home/stack/templates/roles_data_compute_ovsdpdk.yaml，TunedProfileName 的默认值被 cpu-partitioning-powersave 替换：

```
$ sed -i \
's/TunedProfileName:.*$/TunedProfileName: "cpu-partitioning-powersave"/' \
/home/stack/templates/roles_data_compute_ovsdpdk.yaml
```

5. **要生成镜像文件，请运行 openstack tripleo container image prepare 命令。需要以下输入：**

- **您在前面步骤中生成的角色数据文件，如 roles_data_compute_ovsdpdk.yaml。**
- **适合您的网络服务机制驱动程序的 DPDK 环境文件：**
  - **ML2/OVN 环境的 neutron-ovn-dpdk.yaml 文件。**

- ML2/OVS 环境的 neutron-ovs-dpdk.yaml 文件。

示例

在本例中，为 ML2/OVN 环境生成 overcloud_images.yaml 文件：

```
$ sudo openstack tripleo container image prepare \
  --roles-file ~/templates/roles_data_compute_ovsdpdk.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-dpdk.yaml \
  -e ~/containers-prepare-parameter.yaml \
  --output-env-file=/home/stack/templates/overcloud_images.yaml
```

6. 注意角色数据文件的路径和文件名以及您创建的镜像文件。您稍后会在部署 overcloud 时使用这些文件。

后续步骤

- 继续 第 10.3 节 “为您的 OVS-DPDK 自定义创建环境文件”。

其他资源

- 在 OVS-DPDK 部署中保存电源
- *使用 director 安装和管理 Red Hat OpenStack Platform*中的可组合服务和自定义角色
- *使用 director 安装和管理 Red Hat OpenStack Platform 中的准备容器镜像* https://access.redhat.com/documentation/zh-cn/red_hat_openstack_platform/17.1/html/installing_and_managing_red_hat_openstack_platform_with_director/assembly_preparing-for-director-installation#proc_preparing-container-images_preparing-for-director-installation

10.3. 为您的 OVS-DPDK 自定义创建环境文件

您可以在自定义环境 YAML 文件中使用特定的 Red Hat OpenStack Platform 配置值来配置 OVS-DPDK 部署。

先决条件

- 访问 stack 用户的 undercloud 主机和凭据。

流程

1. 以 stack 用户的身份登录 undercloud。
2. Source stackrc 文件：

```
$ source ~/stackrc
```

3. 创建自定义环境 YAML 文件，如 ovs-dpdk-overrides.yaml。
4. 在自定义环境文件中，确保 AggregateInstanceExtraSpecsFilter 位于 NovaSchedulerEnabledFilters 参数的过滤器列表中，用于过滤节点：

```
parameter_defaults:
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - AggregateInstanceExtraSpecsFilter
```

5. 将 OVS-DPDK Compute 节点的特定于角色的参数添加到自定义环境文件。

示例

```
parameter_defaults:
  ComputeOvsDpdkParameters:
    NeutronBridgeMappings: "dpdk:br-dpdk"
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1GB hugepages=64 iommu=pt
intel_iommu=on
isolcpus=2,4,6,8,10,12,14,16,18,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,23,25,27,
29,31,33,35,37,39"
    TunedProfileName: "cpu-partitioning"
    IsolCpusList:
"2,4,6,8,10,12,14,16,18,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,23,25,27,29,31,33,
```

```
35,37,39"
  NovaReservedHostMemory: 4096
  OvsDpdkSocketMemory: "4096,4096"
  OvsDpdkMemoryChannels: "4"
  OvsDpdkCoreList: "0,20,1,21"
  NovaComputeCpuDedicatedSet:
"4,6,8,10,12,14,16,18,24,26,28,30,32,34,36,38,5,7,9,11,13,15,17,19,27,29,31,33,35,37,39"
  NovaComputeCpuSharedSet: "0,20,1,21"
  OvsPmdCoreList: "2,22,3,23"
  OvsEnableDpdk: true
```

6. 如果您需要覆盖这些文件中的任何配置默认值，请将您的覆盖添加到在第 3 步中创建的自定义环境文件。

   RHOSP director 使用以下文件来配置 OVS-DPDK：

   - ML2/OVN 部署

     /usr/share/openstack-tripleo-heat-templates/environment/services/neutron-ovn-dpdk.yaml

   - ML2/OVS 部署

     /usr/share/openstack-tripleo-heat-templates/environment/services/neutron-ovs-dpdk.yaml

7. 请注意您创建的自定义环境文件的路径和文件名。您稍后会在部署 overcloud 时使用此文件。

后续步骤

- 继续 第 10.4 节 “为安全组配置防火墙”。

10.4. 为安全组配置防火墙

data plane 接口在有状态的防火墙中需要高性能。要保护这些接口，请考虑在 Red Hat OpenStack Platform (RHOSP) OVS-DPDK 环境中将电信级防火墙部署为虚拟网络功能(VNF)。

要在 ML2/OVS 部署中配置 control plane 接口，请在自定义环境文件的 parameter_defaults 下将 NeutronOVSFirewallDriver 参数设置为 openvswitch。在 OVN 部署中，您可以使用访问控制列表 (ACL)实施安全组。

您不能将 OVS 防火墙驱动程序与硬件卸载搭配使用，因为卸载路径中不支持流的连接跟踪属性。

先决条件

- 访问 stack 用户的 undercloud 主机和凭据。

流程

1. 以 stack 用户的身份登录 undercloud。
2. Source stackrc 文件：

```
$ source ~/stackrc
```

3. 打开您在 第 10.3 节 “为您的 OVS-DPDK 自定义创建环境文件” 中创建的自定义环境 YAML 文件，或创建新环境。
4. 在 parameter_defaults 下，添加以下键值对：

```
parameter_defaults:
...

  NeutronOVSFirewallDriver: openvswitch
```

5. 如果您创建了新的自定义环境文件，请记下其路径和文件名。您稍后会在部署 overcloud 时使用此文件。
6. 部署 overcloud 后，运行 openstack port set 命令，以禁用 data plane 接口的 OVS 防火墙驱动程序：

```
$ openstack port set --no-security-group --disable-port-security ${PORT}
```

后续步骤

- 继续 第 10.5 节 “创建裸机节点定义文件”。

其他资源

- *使用 director 安装和管理 Red Hat OpenStack Platform*中的可组合服务和自定义角色
- 为 NFV 测试的 NIC

10.5. 创建裸机节点定义文件

使用 Red Hat OpenStack Platform (RHOSP) director，您可以使用定义文件为 OVS-DPDK 环境置备裸机节点。在裸机节点定义文件中，定义您要部署并分配 overcloud 角色的裸机节点的数量和属性。另外，也定义节点的网络布局。

先决条件

- 访问 stack 用户的 undercloud 主机和凭据。

流程

1. 以 stack 用户的身份登录 undercloud。
2. Source stackrc 文件：

```
$ source ~/stackrc
```

3. 按照 *使用 director 的 Red Hat OpenStack Platform 置备裸机节点指南中所述，创建裸机节点定义文件，如 overcloud -baremetal-deploy.yaml。*
4. 在 overcloud-baremetal-deploy.yaml 文件中，向 Ansible playbook 添加声明 cli-overcloud-node-kernelargs.yaml。playbook 包含在置备裸机节点时要使用的内核参数。

```
- name: ComputeOvsDpdk
...
  ansible_playbooks:
```

```
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
...
```

5. 如果要在运行 playbook 时设置任何额外的 Ansible 变量，请使用 extra_vars 属性来设置它们。

如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 裸机节点置备属性。

注意

您添加到 extra_vars 的变量应当是之前添加到 OVS-DPDK 自定义的环境文件中的 OVS-DPDK Compute 节点的特定于角色的参数。

示例

```
- name: ComputeOvsDpdk
  ...
  ansible_playbooks:
  - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
    extra_vars:
      kernel_args: 'default_hugepagesz=1GB hugepagesz=1GB hugepages=64
iommu=pt intel_iommu=on
isolcpus=2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,21,23,
25,27,29,31,33,35,37,39'
      tuned_isolated_cores:
'2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,21,23,25,27,29,
31,33,35,37,39'
      tuned_profile: 'cpu-partitioning'
      reboot_wait_timeout: 1800
  - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-openvswitch-
dpdk.yaml
    extra_vars:
      pmd: '2,22,3,23'
      memory_channels: '4'
      socket_mem: '4096,4096'
      pmd_auto_lb: true
      pmd_load_threshold: "70"
      pmd_improvement_threshold: "25"
      pmd_rebal_interval: "2"
```

6. 可选： 您可以使用 TuneD 配置集 cpu-partitioning-powersave，将 OVS-DPDK 配置为在

没有转发数据包时进入睡眠模式。

要配置 cpu-partitioning-powersave，请在裸机节点定义文件中添加以下行：

```
...

tuned_profile: "cpu-partitioning-powersave"

...

  - playbook: /home/stack/ospd-17.1-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/playbooks/cli-overcloud-tuned-maxpower-conf.yaml
  - playbook: /home/stack/ospd-17.1-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/playbooks/overcloud-nm-config.yaml
    extra_vars:
      reboot_wait_timeout: 900

...

      pmd_sleep_max: "50"

...
```

示例

```
- name: ComputeOvsDpdk
  ...
  ansible_playbooks:
  - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
    extra_vars:
      kernel_args: default_hugepagesz=1GB hugepagesz=1GB hugepages=64 iommu=pt intel_iommu=on isolcpus=2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39
      tuned_isolated_cores: 2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39
      tuned_profile: cpu-partitioning
      reboot_wait_timeout: 1800
  - playbook: /home/stack/ospd-17.1-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/playbooks/cli-overcloud-tuned-maxpower-conf.yaml
  - playbook: /home/stack/ospd-17.1-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid/playbooks/overcloud-nm-config.yaml
    extra_vars:
      reboot_wait_timeout: 900
  - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-openvswitch-dpdk.yaml
    extra_vars:
      pmd: 2,22,3,23
      memory_channels: 4
```

```
    socket_mem: 4096,4096
    pmd_auto_lb: true
    pmd_load_threshold: "70"
    pmd_improvement_threshold: "25"
    pmd_rebal_interval: "2"
    pmd_sleep_max: "50"
```

7. 注意您创建的裸机节点定义文件的路径和文件名。稍后，在配置 NIC 时，并在置备节点时使用 此文件作为 overcloud 节点置备 命令的输入文件。

后续步骤

- 继续 第 10.6 节 “创建 NIC 配置模板”。

其他资源

- 在 OVS-DPDK 部署中保存电源
- *使用 director 安装和管理 Red Hat OpenStack Platform*中的可组合服务和自定义角色
- 为 NFV 测试的 NIC

10.6. 创建 NIC 配置模板

通过修改 Red Hat OpenStack Platform (RHOSP)附带的示例 Jinja2 模板的副本来定义您的 NIC 配置模板。

先决条件

- 访问 stack 用户的 undercloud 主机和凭据。

流程

1. 以 stack 用户的身份登录 undercloud。

2. **Source stackrc 文件：**

```
$ source ~/stackrc
```

3. **复制示例网络配置模板。**

从 /usr/share/ansible/roles/tripleo_network_config/templates/ 目录中的示例复制 NIC 配置 Jinja2 模板。选择最符合 NIC 要求的值。根据需要进行修改。

4. **在 NIC 配置模板中，如 single_nic_vlans.j2，添加您的 DPDK 接口。**

注意

在示例 NIC 配置模板 single_nic_vlans.j2 中，节点只使用一个网络接口作为 VLAN 的中继。原生 VLAN （未标记的流量）是 control plane，每个 VLAN 对应于 RHOSP 网络之一：内部 API、存储等。

示例

```
...
 - type: ovs_dpdk_bond
   name: dpdkbond0
   mtu: 9000
   rx_queue: 1
   ovs_extra:
     - set Interface dpdk0 options:n_rxq_desc=4096
     - set Interface dpdk0 options:n_txq_desc=4096
     - set Interface dpdk1 options:n_rxq_desc=4096
     - set Interface dpdk1 options:n_txq_desc=4096
   members:
   - type: ovs_dpdk_port
     name: dpdk0
     driver: vfio-pci
     members:
     - type: interface
       name: nic5
   - type: ovs_dpdk_port
     name: dpdk1
     driver: vfio-pci
     members:
     - type: interface
       name: nic6
...
```

5. 将自定义网络配置模板（如 single_nic_vlans.j2） 添加到裸机节点定义文件，如您在 第 10.5 节 “创建裸机节点定义文件” 中创建的 overcloud-baremetal-deploy.yaml。

示例

```
- name: ComputeOvsDpdk
  count: 2
  hostname_format: compute-%index%
  defaults:
    networks:
      - network: internal_api
        subnet: internal_api_subnet
      - network: tenant
        subnet: tenant_subnet
      - network: storage
        subnet: storage_subnet
    network_config:
      template: /home/stack/templates/single_nic_vlans.j2
...
```

6. 可选： 您可以使用 TuneD 配置集 cpu-partitioning-powersave，将 OVS-DPDK 配置为在没有转发数据包时进入睡眠模式。

要配置 cpu-partitioning-powersave，请确保在 NIC 配置模板中设置了队列大小。

示例

```
...
  - type: ovs_dpdk_bond
    name: dpdkbond0
    mtu: 9000
    rx_queue: 1
    ovs_extra:
      - set Interface dpdk0 options:n_rxq_desc=4096
      - set Interface dpdk0 options:n_txq_desc=4096
      - set Interface dpdk1 options:n_rxq_desc=4096
      - set Interface dpdk1 options:n_txq_desc=4096
```

```
    members:
    - type: ovs_dpdk_port
      name: dpdk0
      driver: vfio-pci
      members:
      - type: interface
        name: nic5
    - type: ovs_dpdk_port
      name: dpdk1
      driver: vfio-pci
      members:
      - type: interface
        name: nic6
...
```

7. 请注意您创建的 NIC 配置模板的路径和文件名。您稍后会在部署 overcloud 时使用此文件。

后续步骤

- 继续 第 10.7 节 “为 OVS-DPDK 接口设置 MTU 值”。

其他资源

- 在 OVS-DPDK 部署中保存电源

10.7. 为 OVS-DPDK 接口设置 MTU 值

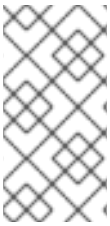
Red Hat OpenStack Platform (RHOSP)支持 OVS-DPDK 的巨型帧。要为巨型帧设置最大传输单元(MTU)值，您必须：

- 在自定义环境文件中为网络设置全局 MTU 值。
- 在 NIC 配置模板中设置物理 DPDK 端口 MTU 值。

  vhost 用户界面也会使用这个值。
- 在 Compute 节点上的任何客户机实例内设置 MTU 值，以确保您的配置末尾具有可比较的

MTU 值。

您不需要物理 NIC 的任何特殊配置，因为 NIC 由 DPDK PMD 控制，并且 NIC 配置模板设置了相同的 MTU 值。您不能设置大于物理 NIC 支持的最大值的 MTU 值。



注意

VXLAN 数据包在标头中包含额外的 50 字节。根据这些额外的标头字节计算您的 MTU 要求。例如，MTU 值 9000 表示 VXLAN 隧道 MTU 值是 8950，用于这些额外字节。

先决条件

- 访问 stack 用户的 undercloud 主机和凭据。

流程

1. 以 stack 用户的身份登录 undercloud。
2. Source stackrc 文件：

```
$ source ~/stackrc
```

3. 打开您在 第 10.3 节 “为您的 OVS-DPDK 自定义创建环境文件” 中创建的自定义环境 YAML 文件，或创建新环境。
4. 在 parameter_defaults 下，设置 NeutronGlobalPhysnetMtu 参数。

示例

在本例中，Neutron GlobalPhysnetMtu 设置为 9000 ：

```
parameter_defaults:
  # MTU global configuration
  NeutronGlobalPhysnetMtu: 9000
```

注意

确保 network-environment.yaml 文件中的 OvsDpdkSocketMemory 值足够大，以支持巨型帧。如需更多信息，请参阅 内存参数。

5. 打开您在 第 10.6 节 “创建 NIC 配置模板” 中创建的 NIC 配置模板，如 single_nic_vlans.j2。

6. 将网桥上的 MTU 值设置为 Compute 节点。

```
-
  type: ovs_bridge
  name: br-link0
  use_dhcp: false
  members:
    -
      type: interface
      name: nic3
      mtu: 9000
```

7. 设置 OVS-DPDK 绑定的 MTU 值：

```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          mtu: 9000
          members:
            - type: interface
              name: nic4
        - type: ovs_dpdk_port
          name: dpdk1
          mtu: 9000
          members:
            - type: interface
              name: nic5
```

8. 请注意 NIC 配置模板和自定义环境文件的路径和文件名。您稍后会在部署 overcloud 时使用这些文件。

后续步骤

- 继续 第 10.8 节 “为 OVS-DPDK 接口设置多队列”。

10.8. 为 OVS-DPDK 接口设置多队列

您可以配置 OVS-DPDK 部署，根据负载和队列使用情况自动将队列负载平衡到非隔离模式驱动程序 (PMD)。Open vSwitch 可以在以下情况下触发自动队列重新平衡：

- 您可以通过将 pmd-auto-lb 的值设置为 true 来启用基于 RX 队列的分配。
- 存在两个或多个非隔离 PMD。
- 有多个队列轮询至少一个非隔离 PMD。
- 聚合 PMD 的负载值在一分钟内超过 95%。

重要

多队列是实验性的，仅支持手动队列固定。

先决条件

- 访问 stack 用户的 undercloud 主机和凭据。

流程

1. 以 stack 用户的身份登录 undercloud。
2. Source stackrc 文件：

```
$ source ~/stackrc
```

3. 打开您在 第 10.6 节 “创建 NIC 配置模板” 中创建的 NIC 配置模板，如 single_nic_vlans.j2。

4. 为 Compute 节点上的 OVS-DPDK 中的接口设置队列数量：

```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
        - type: ovs_dpdk_port
          name: dpdk0
          mtu: 9000
          members:
            - type: interface
              name: nic4
        - type: ovs_dpdk_port
          name: dpdk1
          mtu: 9000
          members:
            - type: interface
              name: nic5
```

5. 请注意 NIC 配置模板的路径和文件名。您稍后会在部署 overcloud 时使用此文件。

后续步骤

- 继续 第 10.9 节 “为节点置备配置 DPDK 参数”。

10.9. 为节点置备配置 DPDK 参数

您可以配置 Red Hat OpenStack Platform (RHOSP) OVS-DPDK 环境，以自动负载平衡 Open vSwitch (OVS)轮询模式驱动程序(PMD)线程。为此，您可以通过编辑 RHOSP director 在裸机节点置备和 overcloud 部署期间使用的参数。

OVS PMD 线程为用户空间上下文切换执行以下任务：

- 持续轮询数据包的输入端口。
- 分类收到的数据包。
- 在分类后对数据包执行操作。

先决条件

- 访问 stack 用户的 undercloud 主机和凭据。

流程

1. 以 stack 用户的身份登录 undercloud。
2. Source stackrc 文件：

```
$ source ~/stackrc
```

3. 在 第 10.5 节 “创建裸机节点定义文件” 中创建的裸机节点定义文件中设置参数，如 overcloud-baremetal-deploy.yaml ：

pmd_auto_lb

设置为 true 以启用 PMD 自动负载平衡。

pmd_load_threshold

在触发 PMD 负载均衡前，其中一个 PMD 线程必须使用的处理周期百分比。整数，范围 0-100。

pmd_improvement_threshold

在触发 PMD 自动负载平衡的非隔离 PMD 线程中评估改进的最小百分比。整数，范围 0-100。

为计算预计改进，完成重新分配的空运行，与当前的差异相比估计的负载差异。默认为 25%。

pmd_rebal_interval

连续两个 PMD Auto Load Balance 操作之间的最短时间（以分钟为单位）。范围 0-20,000 分钟。

配置此值，以防止在流量模式可改变时触发频繁重新分配。例如，您可以每 10 分钟触发一次重新分配，或者每几分钟后触发一次。

示例

```
ansible_playbooks:
…
 - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-openvswitch-dpdk.yaml
  extra_vars:
   …
   pmd_auto_lb: true
   pmd_load_threshold: "70"
   pmd_improvement_threshold: "25"
   pmd_rebal_interval: "2"
```

4. 打开您在 第 10.3 节 “为您的 OVS-DPDK 自定义创建环境文件” 中创建的自定义环境 YAML 文件，或创建新环境。

5. 在自定义环境文件中，添加在第 3 步中设置的相同的裸机节点预置备值。使用这些等同的参数：

OvsPmdAutoLb

等同于 pmd_auto_lb 的 Heat。

设置为 true 以启用 PMD 自动负载平衡。

OvsPmdLoadThreshold

等同于 pmd_load_threshold 的 Heat。

在触发 PMD 负载均衡前，其中一个 PMD 线程必须使用的处理周期百分比。整数，范围

0-100。

OvsPmdImprovementThreshold

等同于 pmd_improvement_threshold 的 Heat。

在触发 PMD 自动负载平衡的非隔离 PMD 线程中评估改进的最小百分比。整数，范围 0-100。

为计算预计改进，完成重新分配的空运行，与当前的差异相比估计的负载差异。默认为 25%。

OvsPmdRebalInterval

等同于 pmd_rebal_interval 的 Heat。

连续两个 PMD Auto Load Balance 操作之间的最短时间（以分钟为单位）。范围 0-20,000 分钟。

配置此值，以防止在流量模式可改变时触发频繁重新分配。例如，您可以每 10 分钟触发一次重新分配，或者每几分钟后触发一次。

示例

```
parameter_merge_strategies:
  ComputeOvsDpdkSriovParameters:merge
...
parameter_defaults:
  ComputeOvsDpdkSriovParameters:
    ...
    OvsPmdAutoLb: true
    OvsPmdLoadThreshold: 70
    OvsPmdImprovementThreshold: 25
    OvsPmdRebalInterval: 2
```

6. 请注意 NIC 配置模板和自定义环境文件的路径和文件名。您稍后在置备裸机节点并部署 overcloud 时，您可以使用这些文件。

后续步骤

1. 置备您的网络和 VIP。

2. 置备裸机节点。

   确保使用裸机节点定义文件，如 overcloud-baremetal-deploy.yaml，作为运行 provision 命令的输入。

3. 继续 第 10.10 节 “部署 OVS-DPDK overcloud”。

其他资源

- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 配置和置备 overcloud 网络定义。
- *使用 director 安装和管理 Red Hat OpenStack Platform 指南*中的为 overcloud 配置和置备网络 VIP。
- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 为 overcloud 置备裸机节点。

10.10. 部署 OVS-DPDK OVERCLOUD

在 OVS-DPDK 环境中部署 Red Hat OpenStack Platform (RHOSP) overcloud 的最后一步是运行 openstack overcloud deploy 命令。命令的输入包括您构建的所有 overcloud 模板和环境文件。

先决条件

- 访问 stack 用户的 undercloud 主机和凭据。
- 您已执行了本节前面流程中列出的所有步骤，并编译了所有各种 heat 模板和环境文件，以用作 overcloud deploy 命令的输入。

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. 输入 openstack overcloud deploy 命令。

以特定顺序列出 openstack overcloud deploy 命令的输入非常重要。常规规则是首先指定默认的 heat 模板文件，后跟包含自定义配置的自定义环境文件和自定义模板，如覆盖默认属性。

按照以下顺序将输入添加到 openstack overcloud deploy 命令中：

a. 包含 overcloud 上 SR-IOV 网络规格的自定义网络定义文件，如 network-data.yaml。

如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 网络定义文件配置选项。

b. 包含 RHOSP director 用来部署 SR-IOV 环境的 Controller 和 ComputeOvsDpdk 角色的角色文件。

示例： roles_data_compute_ovsdpdk.yaml

如需更多信息，请参阅 第 10.2 节 “生成角色和镜像文件”。

c. 置备 overcloud 网络的输出文件。

示例： overcloud-networks-deployed.yaml

如需更多信息，请参阅*使用 director 安装和管理 Red Hat OpenStack Platform 指南*中的配置和管理 overcloud 网络定义。

d. 置备 overcloud VIP 的输出文件。

示例： overcloud-vip-deployed.yaml

如需更多信息，请参阅 *安装和管理 Red Hat OpenStack Platform* 指南中的 为 overcloud 配置和置备网络 VIP。

e. 置备裸机节点的输出文件。

示例： overcloud-baremetal-deployed.yaml

如需更多信息，请参阅：

- 第 10.9 节 “为节点置备配置 DPDK 参数”.
- *使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的 为 overcloud 置备裸机节点。

f. director 用来确定获取容器镜像的位置以及如何存储它们的镜像文件。

示例： overcloud_images.yaml

如需更多信息，请参阅 第 10.2 节 “生成角色和镜像文件”。

g. 您的环境使用的网络服务(neutron)机制驱动程序和路由器方案的环境文件：

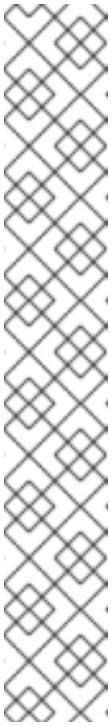
- ML2/OVN
  - 分布式虚拟路由(DVR)： neutron-ovn-dvr-ha.yaml
  -

- 集中式虚拟路由： neutron-ovn-ha.yaml

- ML2/OVS
  - 分布式虚拟路由(DVR)： neutron-ovs-dvr.yaml
  - 集中式虚拟路由： neutron-ovs.yaml

h. OVS-DPDK 的环境文件，具体取决于您的机制驱动程序：

- ML2/OVN
  - neutron-ovn-dpdk.yaml
- ML2/OVS
  - neutron-ovs-dpdk.yaml



注意

如果您也有一个 SR-IOV 环境，并希望在同一节点上定位 SR-IOV 和 OVS-DPDK 实例，请在部署脚本中包含以下环境文件：

- ML2/OVN

  neutron-ovn-sriov.yaml
- ML2/OVS

  neutron-sriov.yaml

i. 一个或多个包含您的配置的自定义环境文件：

- 覆盖 OVS-DPDK 环境的默认配置值。
- 防火墙作为虚拟网络功能(VNF)。
- 巨型帧的最大传输单元(MTU)值。

  示例： ovs-dpdk-overrides.yaml

  如需更多信息，请参阅：

- 第 10.3 节 “为您的 OVS-DPDK 自定义创建环境文件”.
- 第 10.4 节 “为安全组配置防火墙”.
- 第 10.7 节 “为 OVS-DPDK 接口设置 MTU 值”.

示例

示例 openstack overcloud deploy 命令摘录演示了使用 DVR 的 OVS-DPDK ML2/OVN 环境正确排序命令输入：

```
$ openstack overcloud deploy \
--log-file overcloud_deployment.log \
--templates /usr/share/openstack-tripleo-heat-templates/ \
--stack overcloud \
-n /home/stack/templates/network_data.yaml \
-r /home/stack/templates/roles_data_compute_ovsdpdk.yaml \
-e /home/stack/templates/overcloud-networks-deployed.yaml \
-e /home/stack/templates/overcloud-vip-deployed.yaml \
-e /home/stack/templates/overcloud-baremetal-deployed.yaml \
-e /home/stack/templates/overcloud-images.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-dvr-ha.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/services/\
neutron-ovn-dpdk.yaml \
-e /home/stack/templates/ovs-dpdk-overrides.yaml
```

-

4. 运行 **openstack overcloud deploy** 命令。

   完成 overcloud 创建后，RHOSP director 会提供帮助您访问 overcloud 的详细信息。

验证

- ***使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的执行验证 overcloud 部署中的步骤。**

后续步骤

- **如果您配置了防火墙，请运行 openstack port set 命令，以禁用 data plane 接口的 OVS 防火墙驱动程序：**

```
$ openstack port set --no-security-group --disable-port-security ${PORT}
```

其他资源

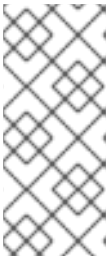
- ***使用 director 安装和管理 Red Hat OpenStack Platform* 指南中的创建 overcloud**
- ***命令行界面参考*中的 overcloud 部署**

10.11. 为 OVS-DPDK 创建类别和部署实例

在使用 NFV 为 Red Hat OpenStack Platform 部署配置 OVS-DPDK 后，您可以按照以下步骤创建类别并部署实例：

1. **创建聚合组，并为 OVS-DPDK 添加相关主机。定义与定义的类别元数据匹配的元数据，如 dpdk=true。**

```
# openstack aggregate create dpdk_group
# openstack aggregate add host dpdk_group [compute-host]
# openstack aggregate set --property dpdk=true dpdk_group
```



注意

固定 CPU 实例可以位于与未固定实例相同的 Compute 节点上。如需更多信息，请参阅配置 *计算服务以进行实例创建* 中的在 *Compute* 节点上配置 CPU 固定。

2. **创建类别。**

```
# openstack flavor create <flavor> --ram <MB> --disk <GB> --vcpus <#>
```

3. **设置类别属性。请注意，定义的元数据 dpdk=true 与 DPDK 聚合中定义的元数据匹配。**

```
# openstack flavor set <flavor> --property dpdk=true --property hw:cpu_policy=dedicated --
property hw:mem_page_size=1GB --property hw:emulator_threads_policy=isolate
```

有关性能改进的仿真程序线程策略的详情，请参考 *为实例创建配置计算服务* 中的配置 仿真程序线程。

4. **创建网络。**

```
# openstack network create net1 --provider-physical-network tenant --provider-network-type
vlan --provider-segment <VLAN-ID>
# openstack subnet create subnet1 --network net1 --subnet-range 192.0.2.0/24 --dhcp
```

5. **可选： 如果您将 multiqueue 与 OVS-DPDK 搭配使用，请在您要用于创建实例的镜像上设置 hw_vif_multiqueue_enabled 属性：**

```
# openstack image set --property hw_vif_multiqueue_enabled=true <image>
```

6. **部署实例。**

```
# openstack server create --flavor <flavor> --image <glance image> --nic net-id=<network
ID> <server_name>
```

10.12. 对 OVS-DPDK 配置进行故障排除

本节介绍对 OVS-DPDK 配置进行故障排除的步骤。

1. 检查网桥配置，并确认网桥具有 datapath_type=netdev。

```
# ovs-vsctl list bridge br0
_uuid               : bdce0825-e263-4d15-b256-f01222df96f3
auto_attach         : []
controller          : []
datapath_id         : "00002608cebd154d"
datapath_type       : netdev
datapath_version    : "<built-in>"
external_ids        : {}
fail_mode           : []
flood_vlans         : []
flow_tables         : {}
ipfix               : []
mcast_snooping_enable: false
mirrors             : []
name                : "br0"
netflow             : []
other_config        : {}
ports               : [52725b91-de7f-41e7-bb49-3b7e50354138]
protocols           : []
rstp_enable         : false
rstp_status         : {}
sflow               : []
status              : {}
stp_enable          : false
```

2. 另外，您可以查看错误的日志，如容器无法启动。

```
# less /var/log/containers/neutron/openvswitch-agent.log
```

3. 确认 ovs-dpdk 的 Poll Mode Driver CPU 掩码固定到 CPU。如果是超量线程，请使用同级 CPU。

例如，要检查 CPU4 的同级功能，请运行以下命令：

```
# cat /sys/devices/system/cpu/cpu4/topology/thread_siblings_list
4,20
```

CPU4 的同级是 CPU20，因此使用以下命令：

```
# ovs-vsctl set Open_vSwitch . other_config:pmd-cpu-mask=0x100010
```

显示状态：

```
# tuna -t ovs-vswitchd -CP
thread  ctxt_switches pid SCHED_ rtpri affinity voluntary nonvoluntary       cmd
3161 OTHER  0     6 765023       614 ovs-vswitchd
3219   OTHER  0     6      1         0    handler24
3220   OTHER  0     6      1         0    handler21
3221   OTHER  0     6      1         0    handler22
3222   OTHER  0     6      1         0    handler23
3223   OTHER  0     6      1         0    handler25
3224   OTHER  0     6      1         0    handler26
3225   OTHER  0     6      1         0    handler27
3226   OTHER  0     6      1         0    handler28
3227   OTHER  0     6      2         0    handler31
3228   OTHER  0     6      2         4    handler30
3229   OTHER  0     6      2         5    handler32
3230   OTHER  0     6 953538       431    revalidator29
3231   OTHER  0     6   1424258       976    revalidator33
3232   OTHER  0     6   1424693       836    revalidator34
3233   OTHER  0     6 951678       503    revalidator36
3234   OTHER  0     6   1425128       498    revalidator35
*3235   OTHER  0     4 151123        51         pmd37*
*3236   OTHER  0    20 298967        48         pmd38*
3164   OTHER  0     6  47575         0  dpdk_watchdog3
3165   OTHER  0     6 237634         0   vhost_thread1
3166   OTHER  0     6   3665         0        urcu2
```

第 11 章 调优 RED HAT OPENSTACK PLATFORM 环境

11.1. 固定仿真程序线程

仿真程序线程处理虚拟机硬件模拟的中断请求和非阻塞进程。这些线程在客户机用来处理的 CPU 间的浮点点。如果用于轮询模式驱动程序(PMD)或这些客户机 CPU 上运行的实时处理线程，您可能会遇到数据包丢失或丢失期限。

您可以通过将线程固定到自己的客户机 CPU，从而将仿真程序线程与虚拟机处理任务分开，从而提高性能。

要提高性能，请为托管仿真程序线程保留主机 CPU 子集。

流程

1. 使用为给定角色定义的 NovaComputeCpuSharedSet 部署 overcloud。NovaComputeCpuSharedSet 的值应用到该角色内主机的 nova.conf 文件中的 cpu_shared_set 参数。

```
parameter_defaults:
  ComputeOvsDpdkParameters:
    NovaComputeCpuSharedSet: "0-1,16-17"
    NovaComputeCpuDedicatedSet: "2-15,18-31"
```

2. 创建一个类别，以构建将仿真程序线程分隔到共享池中的实例。

```
openstack flavor create --ram <size_mb> --disk <size_gb> --vcpus <vcpus> <flavor>
```

3. 添加 hw:emulator_threads_policy 额外规格，并将值设置为 共享。使用此类别创建的实例将使用 nova.conf 文件中的 cpu_share_set 参数中定义的实例 CPU。

```
openstack flavor set <flavor> --property hw:emulator_threads_policy=share
```

注意

您必须在 nova.conf 文件中设置 cpu_share_set 参数，以便为这个额外规格启用共享策略。最好使用 heat，因为手动编辑 nova.conf 可能无法在重新部署后保留。

验证

1. 识别给定实例的主机和名称。

```
openstack server show <instance_id>
```

2. 使用 SSH 以 tripleo-admin 用户身份登录标识的主机。

```
ssh tripleo-admin@compute-1
[compute-1]$ sudo virsh dumpxml instance-00001 | grep `'emulatorpin cpuset'`
```

11.2. 在虚拟和物理功能之间配置信任

您可以在物理功能(PF)和虚拟功能(VF)之间配置信任，以便 VF 可以执行特权操作，如启用混杂模式或修改硬件地址。

先决条件

- Red Hat OpenStack Platform 的操作安装，包括 director

流程

完成以下步骤，在物理和虚拟功能之间配置和部署带有信任的 overcloud：

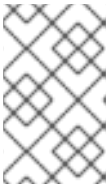
1. 在 parameter_defaults 部分中添加 NeutronPhysicalDevMappings 参数，以链接逻辑网络名称和物理接口。

```
parameter_defaults:
  NeutronPhysicalDevMappings:
    - sriov2:p5p2
```

2. 将新属性 trusted 添加至 SR-IOV 参数。

```
parameter_defaults:
  NeutronPhysicalDevMappings:
    - sriov2:p5p2
  NovaPCIPassthrough:
    - vendor_id: "8086"
```

```
    product_id: "1572"
    physical_network: "sriov2"
    trusted: "true"
```



注意

您必须在值 "true" 中包含双引号。

11.3. 使用可信 VF 网络

1. **创建类型为 vlan 的网络。**

```
openstack network create trusted_vf_network  --provider-network-type vlan \
 --provider-segment 111 --provider-physical-network sriov2 \
 --external --disable-port-security
```

2. **创建子网。**

```
openstack subnet create --network trusted_vf_network \
  --ip-version 4 --subnet-range 192.168.111.0/24 --no-dhcp \
 subnet-trusted_vf_network
```

3. **创建端口。将 vnic-type 选项设置为 直接，并将 binding-profile 选项设为 true。**

```
openstack port create --network sriov111 \
--vnic-type direct --binding-profile trusted=true \
sriov111_port_trusted
```

4. **创建一个实例，并将它绑定到之前创建的可信端口。**

```
openstack server create --image rhel --flavor dpdk  --network internal --port
trusted_vf_network_port_trusted --config-drive True --wait rhel-dpdk-sriov_trusted
```

验证

确认虚拟机监控程序上的可信 VF 配置：

1. **在创建实例的计算节点上，输入以下命令：**

```
# ip link
7: p5p2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9000 qdisc mq state UP mode
DEFAULT group default qlen 1000
    link/ether b4:96:91:1c:40:fa brd ff:ff:ff:ff:ff:ff
    vf 6 MAC fa:16:3e:b8:91:c2, vlan 111, spoof checking off, link-state auto, trust on,
query_rss off
    vf 7 MAC fa:16:3e:84:cf:c8, vlan 111, spoof checking off, link-state auto, trust off, query_rss
off
```

2. 验证 上 VF 的信任状态是否 信任。示例输出包含包含两个端口的环境的详细信息。请注意，vf 6 包含 上的文本信任。

3. 您可以在 Networking 服务(neutron)网络中设置了 port_security_enabled: false，或者在运行 openstack port create 命令时包含参数 --disable-port-security 来禁用 spoof 检查。

11.4. 通过管理 RX-TX 队列大小来防止数据包丢失

出于许多原因，您可能会遇到高于每秒 350 万条数据包的数据包丢失，例如：

- 一个网络中断
- a SMI
- 虚拟网络功能中的数据包处理延迟

要防止数据包丢失，请将队列大小从默认的 512 增加到最多 1024。

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. 查找 **stackrc undercloud** 凭证文件：

```
$ source ~/stackrc
```

3. 创建自定义环境 **YAML** 文件并在 **parameter_defaults** 下添加以下定义来提高 **RX** 和 **TX** 队列大小：

```
parameter_defaults:
  NovaLibvirtRxQueueSize: 1024
  NovaLibvirtTxQueueSize: 1024
```

4. 运行部署命令并包括核心 **heat** 模板、其他环境文件、包含 **RX** 和 **TX** 队列大小更改的环境文件：

示例

```
$ openstack overcloud deploy --templates \
-e <other_environment_files> \
-e /home/stack/my_tx-rx_queue_sizes.yaml
```

验证

1. 观察 **nova.conf** 文件中 **RX** 队列大小和 **TX** 队列大小的值。

```
$ egrep "^[rt]x_queue_size" /var/lib/config-data/puppet-generated/\
nova_libvirt/etc/nova/nova.conf
```

您应该看到以下内容：

```
rx_queue_size=1024
tx_queue_size=1024
```

2. 在 **Compute** 主机上 **libvirt** 生成的 **VM** 实例 **XML** 文件中检查 **RX** 队列大小和 **TX** 队列大小的值：

a. 创建新实例。

b. 获取 Compute 主机和实例名称：

```
$ openstack server show testvm-queue-sizes -c OS-EXT-SRV-ATTR:\
hypervisor_hostname -c OS-EXT-SRV-ATTR:instance_name
```

输出示例

您应该看到类似如下的输出：

```
+-------------------------------------+-----------------------------------+
| Field                               | Value                             |
+-------------------------------------+-----------------------------------+
| OS-EXT-SRV-ATTR:hypervisor_hostname | overcloud-novacompute-1.sales     |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000059                 |
+-------------------------------------+-----------------------------------+
```

c. 登录 Compute 主机并转储实例定义。

示例

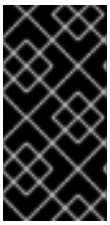
```
$ podman exec nova_libvirt virsh dumpxml instance-00000059
```

输出示例

您应该看到类似如下的输出：

```
...
  <interface type='vhostuser'>
    <mac address='56:48:4f:4d:5e:6f'/>
    <source type='unix' path='/tmp/vhost-user1' mode='server'/>
    <model type='virtio'/>
    <driver name='vhost' rx_queue_size='1024'   tx_queue_size='1024' />
    <address type='pci' domain='0x0000' bus='0x00' slot='0x10' function='0x0'/>
  </interface>
...
```

11.5. 配置 NUMA 感知 VSWITCH



重要

该功能在此发行版本中作为*技术预览*提供，因此不享有红帽的全面支持。它只应用于测试，不应部署在生产环境中。有关技术预览功能的更多信息，请参阅覆盖范围详细信息。

在实现 NUMA 感知 vSwitch 之前，请检查您的硬件配置的以下组件：

- 物理网络的数量。
- PCI 卡的放置。
- 服务器的物理架构。

内存映射 I/O (MMIO)设备（如 PCIe NIC）与特定的 NUMA 节点关联。当虚拟机和 NIC 位于不同的 NUMA 节点上时，性能会显著降低。要提高性能，请将 PCIe NIC 放置和实例处理对齐在同一 NUMA 节点上。

使用此功能确保共享物理网络的实例位于同一 NUMA 节点上。要优化数据中心硬件的利用率，您必须使用多个 physnet。

警告

要为最佳服务器利用率配置 NUMA 感知网络，您必须了解 PCIe 插槽和 NUMA 节点的映射。有关具体硬件的详情，请参考您的厂商文档。如果您无法正确规划或实现 NUMA 感知 vSwitch，可能会导致服务器只使用单个 NUMA 节点。

为防止跨 NUMA 配置，请通过提供 NIC 到 Nova 的位置，将虚拟机放置在正确的 NUMA 节点上。

先决条件

- 您已启用了过滤器 NUMATopologyFilter。

流程

1. 设置一个新的 NeutronPhysnetNUMANodesMapping 参数，将物理网络映射到与物理网络关联的 NUMA 节点。

2. 如果使用 VxLAN 或 GRE 等隧道，还必须设置 NeutronTunnelNUMANodes 参数。

```
parameter_defaults:
  NeutronPhysnetNUMANodesMapping: {<physnet_name>: [<NUMA_NODE>]}
  NeutronTunnelNUMANodes: <NUMA_NODE>,<NUMA_NODE>
```

示例

以下是一个将两个物理网络连接到 NUMA 节点 0 的示例：

- 一个与 NUMA 节点 0 关联的项目网络
- 没有关联性的管理网络

```
parameter_defaults:
  NeutronBridgeMappings:
    - tenant:br-link0
  NeutronPhysnetNUMANodesMapping: {tenant: [1], mgmt: [0,1]}
  NeutronTunnelNUMANodes: 0
```

- 在本例中，将名为 eno2 的设备的 physnet 分配给 NUMA 编号 0。

```
# ethtool -i eno2
bus-info: 0000:18:00.1

# cat /sys/devices/pci0000:16/0000:16:02.0/0000:18:00.1/numa_node
0
```

观察示例 heat 模板中的 physnet 设置：

```
NeutronBridgeMappings: 'physnet1:br-physnet1'
NeutronPhysnetNUMANodesMapping: {physnet1: [0] }
```

```
- type: ovs_user_bridge
            name: br-physnet1
            mtu: 9000
            members:
              - type: ovs_dpdk_port
                name: dpdk2
                members:
                  - type: interface
                    name: eno2
```

验证

按照以下步骤测试您的 NUMA 感知 vSwitch：

1. **观察 /var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf 文件中的配置：**

```
[neutron_physnet_tenant]
numa_nodes=1
[neutron_tunnel]
numa_nodes=1
```

2. **使用 lscpu 命令确认新配置：**

```
$ lscpu
```

3. **启动附加到适当网络的 NIC 的虚拟机。**

其他资源

- 发现 NUMA 节点拓扑
- 第 11.6 节 “NUMA 感知 vSwitches 已知的限制”

11.6. NUMA 感知 VSWITCHES 已知的限制

重要

该功能在此发行版本中作为*技术预览*提供，因此不享有红帽的全面支持。它只应用于测试，不应部署在生产环境中。有关技术预览功能的更多信息，请参阅覆盖范围详细信息。

本节列出了在 Red Hat OpenStack Platform (RHOSP) 网络功能虚拟化 (NFVi) 中实施 NUMA 感知 vSwitch 的限制。

- 如果您没有指定双节点客户机 NUMA 拓扑，则无法启动有两个 NIC，连接到不同 NUMA 节点上的 physnet。
- 如果您没有指定双节点客户机 NUMA 拓扑，则无法启动一个 NIC 连接到不同 NUMA 节点上的隧道网络的虚拟机。
- 如果您没有指定双节点客户机 NUMA 拓扑，则无法启动具有一个 vhost 端口和不同 NUMA 节点上的 VF 的虚拟机。
- NUMA 感知 vSwitch 参数特定于 overcloud 角色。例如，计算节点 1 和 Compute 节点 2 可以有不同的 NUMA 拓扑。
- 如果虚拟机的接口具有 NUMA 关联性，请确保该关联性仅用于单个 NUMA 节点。您可以在任何 NUMA 节点上找到任何没有 NUMA 关联性的接口。
- 为数据平面网络配置 NUMA 关联性，而不是管理网络。
- 隧道网络的 NUMA 关联性是适用于所有虚拟机的全局设置。

11.7. NFVI 环境中的服务质量 (QOS)

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在网络功能虚拟化(NFVi)网络中 Red Hat OpenStack Platform (RHOSP)网络上的出口和入口流量。

在 NFVi 环境中，QoS 支持仅限于以下规则类型：

- 如果厂商支持，SR-IOV 上的 最小带宽。
- SR-IOV 和 OVS-DPDK 出口接口的 带宽限制。

其他资源

- 配置服务质量(QoS)策略

11.8. 创建使用 DPDK 的 HCI OVERCLOUD

您可以通过共置（co-locating）并配置计算和 Ceph Storage 服务来优化资源使用量，通过部署带有超融合节点的 NFV 基础架构。

有关超融合基础架构(HCI)的更多信息，*请参阅部署超融合基础架构*。

以下章节提供了各种配置示例。

11.8.1. NUMA 节点配置示例

为提高性能，将租户网络和 Ceph 对象服务守护进程(OSD)放在一个 NUMA-0 中，如 NUMA-0，以及 VNF 和另一个 NUMA 节点上的任何非 NFV 虚拟机，如 NUMA-1。

CPU 分配：

NUMA-0	NUMA-1
Ceph OSD 数量 * 4 HT	VNF 和非 NFV 虚拟机的客户机 vCPU
DPDK lcore - 2 HT	DPDK lcore - 2 HT
DPDK PMD - 2 HT	DPDK PMD - 2 HT

CPU 分配示例：

	NUMA-0	NUMA-1
Ceph OSD	32,34,36,38,40,42,76,78,80,82,84,86	

	NUMA-0	NUMA-1
DPDK-lcore	0,44	1,45
DPDK-pmd	2,46	3,47
nova		5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,49,51,53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87

11.8.2. Ceph 配置文件示例

本节介绍 Red Hat Ceph Storage 配置文件示例。您可以通过替换适合您的 Red Hat OpenStack Platform 环境的值来对配置文件建模。

```
[osd]
osd_numa_node = 0                    # 1
osd_memory_target_autotune = true          # 2

[mgr]
mgr/cephadm/autotune_memory_target_ratio = 0.2 # 3
```

使用以下参数，为 Ceph Object Storage Daemons (OSD)进程分配 CPU 资源。此处显示的值是示例。根据您的工作负载和硬件调整值。

1

osd_numa_node ：将 Ceph 进程的关联性设置为 NUMA 节点，例如 0 代表 NUMA-0，1 代表 NUMA-1，以此类推。-1 将关联性设置为没有 NUMA 节点。

在本例中，osd_numa_node 设置为 NUMA-0。如 第 11.8.3 节 “DPDK 配置文件示例” 所示，IsolCpusList 在 NUMA-1 上包含奇数的 CPU，在 OvsPmdCoreList 元素被删除后。由于对延迟敏感的 Compute 服务(nova)工作负载托管在 NUMA-1 上，因此您必须在 NUMA-0 上隔离 Ceph 工作负载。本例假定 stroage 网络的磁盘控制器和网络接口都位于 NUMA-0 上。

2

osd_memory_target_autotune: 当设为 true 时，OSD 守护进程会根据 osd_memory_target 配置选项调整其内存消耗。

3

autotune_memory_target_ratio ：用于为 OSD 分配内存。默认值为 0.7。

系统中 RAM 总量的 70% 是起点，从中减去了非自动tuned Ceph 守护进程所消耗的任何内存。当所有 OSD 的 osd_memory_target_autotune 为 true 时，剩余的内存被除以 OSD。对于 HCI 部署，mgr/cephadm/autotune_memory_target_ratio 可以设置为 0.2，以便更多内存可用于计算服务。根据需要调整，以确保每个 OSD 至少有 5 GB 内存。

其他资源

- 第 11.8.6 节 “部署 HCI-DPDK overcloud”

11.8.3. DPDK 配置文件示例

```
parameter_defaults:
  ComputeHCIParameters:
    KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=240 intel_iommu=on
iommu=pt                                  # 1

isolcpus=2,46,3,47,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,49,51,53,55,57,59,61,6
3,65,67,69,71,73,75,77,79,81,83,85,87"
    TunedProfileName: "cpu-partitioning"
    IsolCpusList:                                 # 2
      ”2,46,3,47,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,49,51,
      53,55,57,59,61,63,65,67,69,71,73,75,77,79,81,83,85,87"
    VhostuserSocketGroup: hugetlbfs
    OvsDpdkSocketMemory: "4096,4096"                      # 3
    OvsDpdkMemoryChannels: "4"

    OvsPmdCoreList: "2,46,3,47"                       # 4
```

1

KernelArgs ：要计算 大页，请从总内存中减去 NovaReservedHostMemory 参数的值。

2

IsolCpusList：分配您要使用此参数与主机进程隔离的一组 CPU 内核。将 OvsPmdCoreList 参数的值添加到 NovaComputeCpuDedicatedSet 参数的值，以计算 IsolCpusList 参数的值。

3

OvsDpdkSocketMemory ：使用 OvsDpdkSocketMemory 参数指定从每个 NUMA 节点的巨页池中预先分配的内存量（以 MB 为单位）。有关计算 OVS-DPDK 参数的更多信息，请参阅 OVS-DPDK 参数。

4

OvsPmdCoreList：指定用于带有这个参数的 DPDK 轮询模式驱动程序(PMD)的 CPU 内核。选择与 DPDK 接口本地 NUMA 节点关联的 CPU 内核。为每个 NUMA 节点分配 2 个 HT 同级线程，

以计算 OvsPmdCoreList 参数的值。

11.8.4. nova 配置文件示例

```
parameter_defaults:
  ComputeHCIExtraConfig:
    nova::cpu_allocation_ratio: 16 # 2
    NovaReservedHugePages:                                  # 1
      - node:0,size:1GB,count:4
      - node:1,size:1GB,count:4
  NovaReservedHostMemory: 123904                            # 2
  # All left over cpus from NUMA-1
  NovaComputeCpuDedicatedSet:                                    # 3
['5','7','9','11','13','15','17','19','21','23','25','27','29','31','33','35','37','39','41','43','49','51','|
53','55','57','59','61','63','65','67','69','71','73','75','77','79','81','83','85','87
```

1

NovaReservedHugePages：通过 NovaReservedHugePages 参数，从巨页池中预先分配内存（以 MB 为单位）。它是与 OvsDpdkSocketMemory 参数的值相同的内存总量。

2

NovaReservedHostMemory: 为带有 NovaReservedHostMemory 参数的主机上的任务保留内存（以 MB 为单位）。使用以下准则计算您必须保留的内存量：

- 每个 OSD 5 GB。
- 每个虚拟机的 0.5 GB 开销。
- 4GB 用于常规主机处理。确保分配足够的内存以防止跨 NUMA OSD 操作导致潜在的性能下降。

3

NovaComputeCpuDedicatedSet ：列出 OvsPmdCoreList 中未找到的 CPU，或使用 NovaComputeCpuDedicatedSet 参数列出 Ceph_osd_docker_cpuset_cpus。CPU 必须与 DPDK NIC 位于同一个 NUMA 节点上。

11.8.5. 推荐的 HCI-DPDK 部署配置

表 11.1. HCI 部署的可调整参数

块设备类型	OSD、内存、vCPU 每个设备
NVMe	内存：5GB per OSD OSDs per device: 4 vCPUs per device: 3
SSD	内存：5GB per OSD OSDs per device: 1 vCPUs per device: 4
HDD	内存：5GB per OSD OSDs per device: 1 vCPUs per device: 1

将相同的 NUMA 节点用于以下功能：

- **磁盘控制器**
- **存储网络**
- **存储 CPU 和内存**

为 DPDK 提供商网络的以下功能分配另一个 NUMA 节点：

- **NIC**
- **PMD CPU**
- **套接字内存**

11.8.6. 部署 HCI-DPDK overcloud

按照以下步骤部署使用 DPDK 的超融合 overcloud。

先决条件

- **Red Hat OpenStack Platform (RHOSP) 17.1 或更高版本。**
- **最新版本的 Red Hat Ceph Storage 6.1。**

流程

1. **为 Controller 和 ComputeHCIOvsDpdk 角色生成 roles_data.yaml 文件。**

```
$ openstack overcloud roles generate -o ~/<templates>/roles_data.yaml \
Controller ComputeHCIOvsDpdk
```

2. **使用 openstack flavor create 和 openstack flavor set 命令创建和配置新类别。**

3. **使用 RHOSP director 和 Ceph 配置文件部署 Ceph。**

   示例

```
$ openstack overcloud ceph deploy --config initial-ceph.conf
```

4. **使用您生成的自定义 roles_data.yaml 文件部署 overcloud。**

   示例

```
$ openstack overcloud deploy --templates \
  --timeout 360 \
  -r ~/<templates>/roles_data.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/\
  cephadm/cephadm-rbd-only.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services-docker/neutron-ovs-
dpdk.yaml \
  -e ~/<templates>/<custom environment file>
```

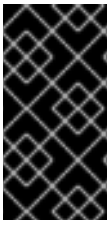
重要

本例在没有 Ceph RGW （对象存储）的情况下部署 Ceph RBD （块存储）。要在部署中包含 RGW，请使用 cephadm.yaml 而不是 cephadm-rbd-only.yaml。

其他资源

- *自定义 Red Hat OpenStack Platform 部署中的可组合服务和自定义角色*
- 第 11.8.2 节 “Ceph 配置文件示例”
- *与 director 一起部署 Red Hat Ceph Storage 和 Red Hat OpenStack Platform 中的配置 Red Hat Ceph Storage 集群。*

11.9. 将您的计算节点与 TIMEMASTER 同步



重要

该功能在此发行版本中作为*技术预览*提供，因此不享有红帽的全面支持。它只应用于测试，不应部署在生产环境中。有关技术预览功能的更多信息，请参阅覆盖范围详细信息。

使用时间协议在系统间保持一致的时间戳。

Red Hat OpenStack Platform (RHOSP)包括对 Precision Time Protocol (PTP)和网络时间协议(NTP)的支持。

您可以使用 NTP 在毫秒范围内同步网络中的时钟，您可以使用 PTP 将时钟同步到更高、子微秒的准确性。PTP 的用例示例是虚拟无线访问网络(vRAN)，其中包含多个异常，它提供了更高的吞吐量，并带来更多干扰风险。

timemaster 是一个使用 ptp4l 和 phc2sys 的程序，它结合 chronyd 或 ntpd 将系统时钟与 NTP 和 PTP 时间源同步。phc2sys 和 ptp4l 程序使用共享内存驱动程序(SHM)参考时钟将 PTP 时间发送到

chronyd 或 ntpd，这会比较时间源来同步系统时钟。

在 Red Hat Enterprise Linux (RHEL)内核中 PTPv2 协议的实现是 linuxptp。

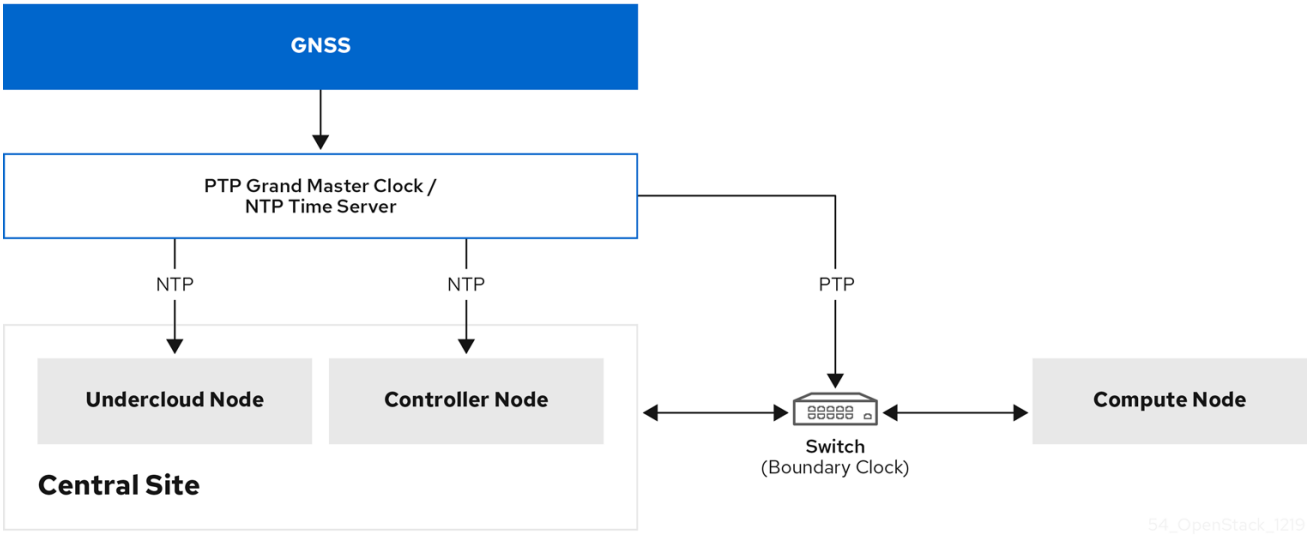
linuxptp 软件包包括 PTP 边界时钟和普通时钟同步的 ptp4l 程序，以及硬件时间戳的 phc2sys 程序。有关 PTP 的更多信息，请参阅 *Red Hat Enterprise Linux 系统管理员指南*中的 PTP 简介。

Chrony 是 NTP 协议的实现。Chrony 的主要组件是 chronyd，它是 Chrony 守护进程，以及 Chrony 命令行界面的 chronyc。

有关 Chrony 的更多信息，请参阅 *Red Hat Enterprise Linux 系统管理员指南中的*使用 Chrony 套件配置 NTP。

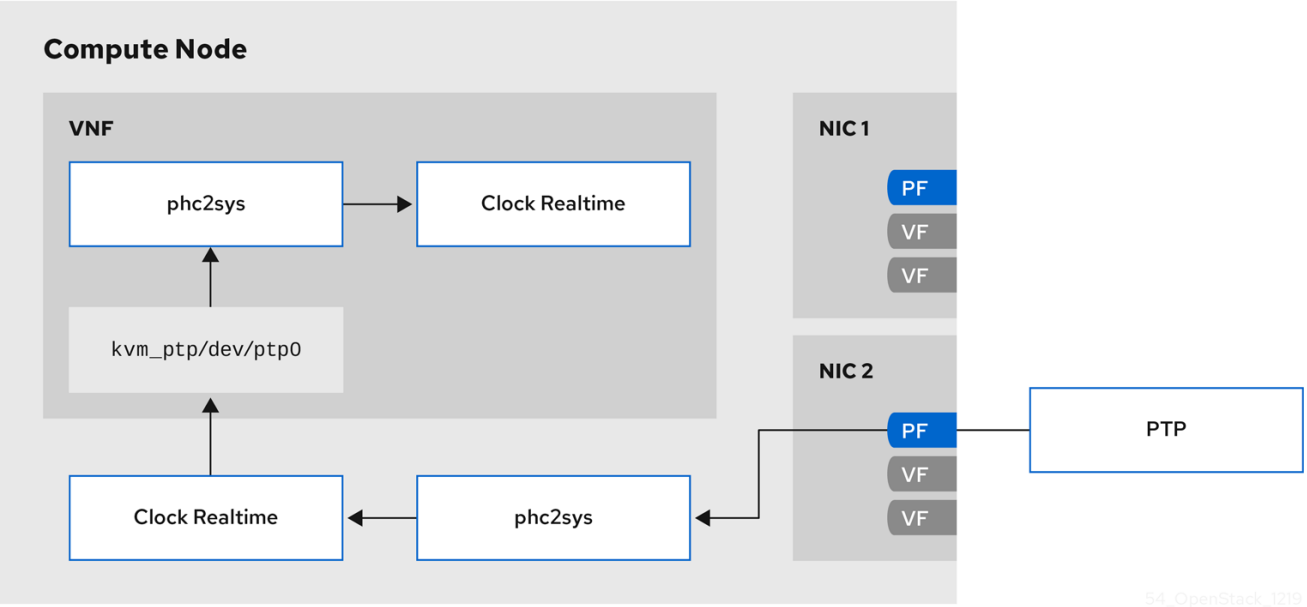
下图是 PTP 配置中数据包之旅的概述。

图 11.1. PTP 数据包之旅概述



下图是 PTP 配置中 Compute 节点的数据包之旅概述。

图 11.2. PTP 数据包之旅详情



11.9.1. Timemaster 硬件要求

确定您有以下硬件功能：

- 您已配置了具有硬件时间戳功能的 NIC。
- 您已将交换机配置为允许多播数据包。
- 您已将交换机配置为也充当边界或透明时钟。

您可以使用 ethtool -T <device> 命令来验证硬件时间戳。

```
$ ethtool -T p5p1
Time stamping parameters for p5p1:
Capabilities:
        hardware-transmit     (SOF_TIMESTAMPING_TX_HARDWARE)
        software-transmit     (SOF_TIMESTAMPING_TX_SOFTWARE)
        hardware-receive      (SOF_TIMESTAMPING_RX_HARDWARE)
        software-receive      (SOF_TIMESTAMPING_RX_SOFTWARE)
        software-system-clock (SOF_TIMESTAMPING_SOFTWARE)
        hardware-raw-clock    (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 6
Hardware Transmit Timestamp Modes:
        off                   (HWTSTAMP_TX_OFF)
```

```
        on                   (HWTSTAMP_TX_ON)
Hardware Receive Filter Modes:
        none                 (HWTSTAMP_FILTER_NONE)
        ptpv1-l4-sync        (HWTSTAMP_FILTER_PTP_V1_L4_SYNC)
        ptpv1-l4-delay-req   (HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ)
        ptpv2-event          (HWTSTAMP_FILTER_PTP_V2_EVENT)
```

您可以使用透明或边界时钟交换机来更好地准确性和较少的延迟。您可以将 uplink 开关用于边界时钟。边界时钟交换机在 PTPv2 标头上使用 8 位 修正Field 来更正延迟差异，并确保对最终时钟的准确性更大。在透明时钟交换机中，结束时钟会计算延迟变化，而不是 更正Field。

11.9.2. 配置 Timemaster

用于 overcloud 节点中时间同步的默认 Red Hat OpenStack Platform (RHOSP)服务是 OS::TripleO::Services::Timesync。

已知限制

- **为虚拟化控制器启用 NTP，并为裸机节点启用 PTP。**
- **VirtIO 接口不兼容，因为 ptp4l 需要兼容的 PTP 设备。**
- **对带有 SR-IOV 的虚拟机使用物理功能(PF)。虚拟功能(VF)不会公开 PTP 所需的注册，虚拟机使用 kvm_ptp 计算时间。**
- **带有多个源和多个网络路径的高可用性(HA)接口不兼容。**

流程

1. **要在属于您选择的角色的节点上启用 Timemaster 服务，请将包含 OS::TripleO::Services::Timesync 的行替换为该角色的 roles_data.yaml 文件部分中的 OS::TripleO::Services::TimeMaster。**

```
#- OS::TripleO::Services::Timesync
- OS::TripleO::Services::TimeMaster
```

2. **为您使用的 compute 角色配置 heat 参数。**

```
#Example
```

```
ComputeSriovParameters:
  PTPInterfaces: '0:eno1,1:eno2'
  PTPMessageTransport: 'UDPv4'
```

3. 在 openstack overcloud deploy 命令中包含新的环境文件，以及与您的环境相关的任何其他环境文件：

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

…
-e <existing_overcloud_environment_files> \
-e <new_environment_file1> \
-e <new_environment_file2> \
…
```

- 使用作为您现有部署一部分的环境文件列表替换 <existing_overcloud_environment_files>。

- 将 <new_environment_file> 替换为您要包含在 overcloud 部署过程中的新环境文件或文件。

验证

- 使用通过 ptp4linux 安装的命令 phc_ctl 来查询 NIC 硬件时钟。

```
# phc_ctl <clock_name> get
# phc_ctl <clock_name> cmp
```

11.9.3. timemaster 配置示例

```
$ cat /etc/timemaster.conf
# Configuration file for timemaster

#[ntp_server ntp-server.local]
#minpoll 4
#maxpoll 4

[ptp_domain 0]
interfaces eno1
#ptp4l_setting network_transport l2
#delay 10e-6

[timemaster]
ntp_program chronyd
```

```
[chrony.conf]
#include /etc/chrony.conf
server clock.redhat.com iburst minpoll 6 maxpoll 10

[ntp.conf]
includefile /etc/ntp.conf

[ptp4l.conf]
#includefile /etc/ptp4l.conf
network_transport L2

[chronyd]
path /usr/sbin/chronyd

[ntpd]
path /usr/sbin/ntpd
options -u ntp:ntp -g

[phc2sys]
path /usr/sbin/phc2sys
#options -w

[ptp4l]
path /usr/sbin/ptp4l
#options -2 -i eno1
```

11.9.4. timemaster 操作示例

```
$ systemctl status timemaster
● timemaster.service - Synchronize system clock to NTP and PTP time sources
   Loaded: loaded (/usr/lib/systemd/system/timemaster.service; enabled; vendor preset: disabled)
   Active: active (running) since Tue 2020-08-25 19:10:18 UTC; 2min 6s ago
 Main PID: 2573 (timemaster)
    Tasks: 6 (limit: 357097)
   Memory: 5.1M
   CGroup: /system.slice/timemaster.service
           ├─2573 /usr/sbin/timemaster -f /etc/timemaster.conf
           ├─2577 /usr/sbin/chronyd -n -f /var/run/timemaster/chrony.conf
           ├─2582 /usr/sbin/ptp4l -l 5 -f /var/run/timemaster/ptp4l.0.conf -H -i eno1
           ├─2583 /usr/sbin/phc2sys -l 5 -a -r -R 1.00 -z /var/run/timemaster/ptp4l.0.socket -t [0:eno1] -n
0 -E ntpshm -M 0
           ├─2587 /usr/sbin/ptp4l -l 5 -f /var/run/timemaster/ptp4l.1.conf -H -i eno2
           └─2588 /usr/sbin/phc2sys -l 5 -a -r -R 1.00 -z /var/run/timemaster/ptp4l.1.socket -t [0:eno2] -n
0 -E ntpshm -M 1

Aug 25 19:11:53 computesriov-0 ptp4l[2587]: [152.562] [0:eno2] selected local clock
e4434b.fffe.4a0c24 as best master
```

第 12 章 为 NFV 工作负载启用 RT-KVM

为了便于安装和配置 Red Hat Enterprise Linux Real Time KVM (RT-KVM)，Red Hat OpenStack Platform 提供以下功能：

- 为实时置备 Red Hat Enterprise Linux 的实时 Compute 节点角色。
- 额外的 RT-KVM 内核模块。
- 自动配置 Compute 节点。

12.1. 规划 RT-KVM COMPUTE 节点

在计划 RT-KVM Compute 节点时，请确保完成以下任务：

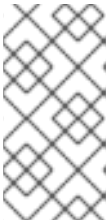
- 您必须将红帽认证的服务器用于 RT-KVM Compute 节点。

  如需更多信息，请参阅 Red Hat Enterprise Linux for Real Time 认证服务器。

- 注册 undercloud 并附加有效的 Red Hat OpenStack Platform 订阅。

  如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform 中的注册 undercloud 和 附加订阅*。

- 启用 undercloud 所需的存储库，如 RT-KVM 的 rhel-9-server-nfv-rpms 存储库，并将系统软件包更新至最新版本。

  注意

  您需要单独订阅 Red Hat OpenStack Platform for Real Time SKU，然后才能访问此软件仓库。

  如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform 中的 为*

undercloud 启用存储库。

构建实时镜像

1. **在 undercloud 上安装 libguestfs-tools 软件包以获取 virt-customize 工具：**

```
(undercloud) [stack@undercloud-0 ~]$ sudo dnf install libguestfs-tools
```

重要

如果在 undercloud 上安装 libguestfs-tools 软件包，请禁用 iscsid.socket，以避免在 undercloud 上与 tripleo_iscsid 服务冲突：

```
$ sudo systemctl disable --now iscsid.socket
```

2. **提取镜像：**

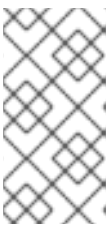
```
(undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-director-images/overcloud-hardened-uefi-full-17.1.x86_64.tar
(undercloud) [stack@undercloud-0 ~]$ tar -xf /usr/share/rhosp-director-images/ironic-python-agent-17.1.x86_64.tar
```

3. **复制默认镜像：**

```
(undercloud) [stack@undercloud-0 ~]$ cp overcloud-hardened-uefi-full.qcow2 overcloud-realtime-compute.qcow2
```

4. **注册您的镜像以启用与您的自定义相关的红帽软件仓库。将 [username] 和 [password] 替换为以下示例中的有效凭证。**

```
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'subscription-manager register --username=[username] --password=[password]' \
subscription-manager release --set 9.0
```



注意

为安全起见，您可以在命令提示符上使用凭据时从历史记录文件中删除凭据。您可以使用 history -d 命令后跟行号，删除历史记录中的个别行。

5. **从您的帐户的订阅中查找池 ID 列表，并将适当的池 ID 附加到您的镜像。**

```
sudo subscription-manager list --all --available | less
...
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'subscription-manager attach --pool [pool-ID]'
```

6. **添加带有 NFV 的 Red Hat OpenStack Platform 所需的存储库。**

```
virt-customize -a overcloud-realtime-compute.qcow2 --run-command \
'sudo subscription-manager repos --enable=rhel-9-for-x86_64-baseos-eus-rpms \
--enable=rhel-9-for-x86_64-appstream-eus-rpms \
--enable=rhel-9-for-x86_64-highavailability-eus-rpms \
--enable=ansible-2.9-for-rhel-9-x86_64-rpms \
--enable=rhel-9-for-x86_64-nfv-rpms
--enable=fast-datapath-for-rhel-9-x86_64-rpms'
```

7. **创建一个脚本来配置镜像的实时功能。**

```
(undercloud) [stack@undercloud-0 ~]$ cat <<'EOF' > rt.sh
 #!/bin/bash

 set -eux

 dnf -v -y --setopt=protected_packages= erase kernel.$(uname -m)
 dnf -v -y install kernel-rt kernel-rt-kvm tuned-profiles-nfv-host
 grubby --set-default /boot/vmlinuz*rt*
 EOF
```

8. **运行脚本来配置实时镜像：**

```
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-realtime-compute.qcow2 -
v --run rt.sh 2>&1 | tee virt-customize.log
```

注意

如果您在 rt.sh 脚本输出中看到以下行，"grubby fatal error: unable to find a suitable template"，您可以忽略此错误。

9. **检查前面命令导致的 virt-customize.log 文件，以使用 rt.sh 脚本检查正确安装的软件包。**

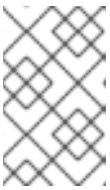
```
(undercloud) [stack@undercloud-0 ~]$ cat virt-customize.log | grep Verifying

  Verifying  : kernel-3.10.0-957.el7.x86_64                          1/1
  Verifying  : 10:qemu-kvm-tools-rhev-2.12.0-18.el7_6.1.x86_64       1/8
  Verifying  : tuned-profiles-realtime-2.10.0-6.el7_6.3.noarch       2/8
  Verifying  : linux-firmware-20180911-69.git85c5d90.el7.noarch      3/8
  Verifying  : tuned-profiles-nfv-host-2.10.0-6.el7_6.3.noarch       4/8
  Verifying  : kernel-rt-kvm-3.10.0-957.10.1.rt56.921.el7.x86_64     5/8
  Verifying  : tuna-0.13-6.el7.noarch                                6/8
  Verifying  : kernel-rt-3.10.0-957.10.1.rt56.921.el7.x86_64         7/8
  Verifying  : rt-setup-2.0-6.el7.x86_64                             8/8
```

10. **重新标记 SELinux：**

```
(undercloud) [stack@undercloud-0 ~]$ virt-customize -a overcloud-realtime-compute.qcow2 --selinux-relabel
```

11. **提取 vmlinuz 和 initrd：**

```
(undercloud) [stack@undercloud-0 ~]$ mkdir image
(undercloud) [stack@undercloud-0 ~]$ guestmount -a overcloud-realtime-compute.qcow2 -i --ro image
(undercloud) [stack@undercloud-0 ~]$ cp image/boot/vmlinuz-3.10.0-862.rt56.804.el7.x86_64 ./overcloud-realtime-compute.vmlinuz
(undercloud) [stack@undercloud-0 ~]$ cp image/boot/initramfs-3.10.0-862.rt56.804.el7.x86_64.img ./overcloud-realtime-compute.initrd
(undercloud) [stack@undercloud-0 ~]$ guestunmount image
```



注意

vmlinuz 和 initramfs 文件名中的软件版本与内核版本不同。

12. **上传镜像：**

```
(undercloud) [stack@undercloud-0 ~]$ openstack overcloud image upload --update-existing --os-image-name overcloud-realtime-compute.qcow2
```

现在，您有一个实时镜像，可用于所选 Compute 节点上的 ComputeOvsDpdkRT 可组合角色。

修改 RT-KVM Compute 节点上的 BIOS 设置

要减少 RT-KVM Compute 节点上的延迟，请在 Compute 节点 BIOS 设置中禁用以下参数的所有选项：

- 电源管理
- 超线程
- CPU 睡眠状态
- 逻辑处理器

12.2. 使用 RT-KVM 配置 OVS-DPDK

12.2.1. 为 Real-time Compute 设计节点

要为 Real-time Compute 指定节点，请创建一个新的角色文件来配置 Real-time Compute 角色，并使用 Real-time Compute 资源类配置裸机节点，以标记实时的 Compute 节点。

注意

以下流程适用于您尚未调配的新 overcloud 节点。要将资源类分配给已调配的现有 overcloud 节点，请缩减 overcloud 以取消置备节点，然后扩展 overcloud，以使用新的资源类分配来重新置备节点。有关更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 中的 扩展 overcloud 节点。

流程

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

```
[stack@director ~]$ source ~/stackrc
```

3. 根据 /usr/share/openstack-tripleo-heat-templates/environments/compute-real-time-example.yaml 文件，创建一个 compute-real-time.yaml 环境文件，为 ComputeRealTime 角色设置参数。

4. **生成一个名为 roles_data_rt.yaml 的新角色数据文件，其中包含 ComputeRealTime 角色，以及 overcloud 所需的任何其他角色。以下示例生成角色数据文件 roles_data_rt.yaml，其中包括角色 Controller, Compute, 和 ComputeRealTime：**

```
(undercloud)$ openstack overcloud roles generate \
-o /home/stack/templates/roles_data_rt.yaml \
ComputeRealTime Compute Controller
```

5. 为 **ComputeRealTime 角色更新 roles_data_rt.yaml 文件：**

```
###############################################################
# Role: ComputeRealTime                                       #
###############################################################
- name: ComputeRealTime
  description: |
    Real Time Compute Node role
  CountDefault: 1
  # Create external Neutron bridge
  tags:
    - compute
    - external_bridge
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
  HostnameFormatDefault: '%stackname%-computert-%index%'
  deprecated_nic_config_name: compute-rt.yaml
```

6. 为 **overcloud 注册 ComputeRealTime 节点，将它们添加到您的节点定义模板中：node.json 或 node.yaml。**

有关更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 中的 为 overcloud 注册节点。

7. 检查节**点硬件：**

```
(undercloud)$ openstack overcloud node introspect --all-manageable --provide
```

如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 中的 创建裸机节点硬件清单。

8. 使用自定义 **ComputeRealTime** 资源类标记您要为 **ComputeRealTime** 指定的每个裸机节点：

```
(undercloud)$ openstack baremetal node set \
 --resource-class baremetal.RTCOMPUTE <node>
```

将 **<node>** 替换为裸机节点的名称或 **UUID**。

9. 将 **ComputeRealTime** 角色添加到节点定义文件 **overcloud-baremetal-deploy.yaml** 中，并定义您要分配给节点的任何预先节点放置、资源类、网络拓扑或其他属性：

```
- name: Controller
  count: 3
  ...
- name: Compute
  count: 3
  ...
- name: ComputeRealTime
  count: 1
  defaults:
    resource_class: baremetal.RTCOMPUTE
    network_config:
      template: /home/stack/templates/nic-config/<role_topology_file>
```

- 将 **<role_topology_file >** 替换为用于 **ComputeRealTime** 角色的拓扑文件的名称，如 **myRoleTopology.j2**。您可以重复使用现有网络拓扑，或为角色创建新的自定义网络接口模板。

  如需更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform 中的定义自定义网络接口模板*。要使用默认网络定义设置，请不要在角色定义中包含 **network_config**。

  有关您可以在节点定义文件中配置节点属性的属性的更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 中的 裸机节点置备属性。

  有关节点定义文件的示例，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 中的 节点定义文件示例。

10. 创建以下 **Ansible playbook** 以在节点置备过程中配置内核，并将 **playbook** 保存为 **/home/stack/templates/fix_rt_kernel.yaml** ：

```
# RealTime KVM fix until BZ #2122949 is closed-
- name: Fix RT Kernel
  hosts: allovercloud
  any_errors_fatal: true
  gather_facts: false
  vars:
    reboot_wait_timeout: 900
  pre_tasks:
    - name: Wait for provisioned nodes to boot
      wait_for_connection:
        timeout: 600
        delay: 10
  tasks:
    - name: Fix bootloader entry
      become: true
      shell: |-
        set -eux
        new_entry=$(grep saved_entry= /boot/grub2/grubenv | sed -e s/saved_entry=//)
        source /etc/default/grub
        sed -i "s/options.*/options root=$GRUB_DEVICE ro $GRUB_CMDLINE_LINUX
$GRUB_CMDLINE_LINUX_DEFAULT/" /boot/loader/entries/$(</etc/machine-
id)$new_entry.conf
        cp -f /boot/grub2/grubenv /boot/efi/EFI/redhat/grubenv
  post_tasks:
    - name: Configure reboot after new kernel
      become: true
      reboot:
        reboot_timeout: "{{ reboot_wait_timeout }}"
      when: reboot_wait_timeout is defined
```

11. **在节点置备文件中的 ComputeOvsDpdkSriovRT 角色定义中包含 /home/stack/templates/fix_rt_kernel.yaml 作为 playbook：**

```
- name: ComputeOvsDpdkSriovRT
  ...
  ansible_playbooks:
    - playbook: /usr/share/ansible/tripleo-playbooks/cli-overcloud-node-kernelargs.yaml
      extra_vars:
        kernel_args: "default_hugepagesz=1GB hugepagesz=1G hugepages=64 iommu=pt
intel_iommu=on tsx=off isolcpus=2-19,22-39"
        reboot_wait_timeout: 900
        tuned_profile: "cpu-partitioning"
        tuned_isolated_cores: "2-19,22-39"
        defer_reboot: true
    - playbook: /home/stack/templates/fix_rt_kernel.yaml
      extra_vars:
        reboot_wait_timeout: 1800
```

有关您可以在节点定义文件中配置节点属性的属性的更多信息，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 中的 裸机节点置备属性。

有关节点定义文件的示例，*请参阅使用 director 安装和管理 Red Hat OpenStack Platform* 中的 节点定义文件示例。

12. 为您的角色置备新节点：

```
(undercloud)$ openstack overcloud node provision \
[--stack <stack> \ ]
[--network-config \]
--output <deployment_file> \
/home/stack/templates/overcloud-baremetal-deploy.yaml
```

- 可选：将 **<stack>** 替换为置备裸机节点的堆栈的名称。默认值为 **overcloud**。
- 可选：包含 **--network-config** 可选参数，为 **cli-overcloud-node-network-config.yaml Ansible playbook** 提供网络定义。如果您没有使用 **network_config** 属性定义网络定义，则使用默认网络定义。
- 将 **<deployment_file>** 替换为用于部署命令生成的 **heat** 环境文件的名称，如 **/home/stack/templates/overcloud-baremetal-deployed.yaml**。

13. 在一个单独的终端中监控置备进度。当置备成功时，节点状态将从 **available** 变为 **active** ：

```
(undercloud)$ watch openstack baremetal node list
```

14. 如果您在没有 **--network-config** 选项运行 **provisioning** 命令，请在 **network-environment.yaml** 文件中配置 **<Role>NetworkConfigTemplate** 参数以指向 NIC 模板文件：

```
parameter_defaults:
  ComputeNetworkConfigTemplate: /home/stack/templates/nic-configs/compute.j2
  ComputeAMDSEVNetworkConfigTemplate: /home/stack/templates/nic-
configs/<rt_compute>.j2
  ControllerNetworkConfigTemplate: /home/stack/templates/nic-configs/controller.j2
```

将 **&lt ;rt_compute >** 替换为包含 **ComputeRealTime** 角色的网络拓扑的文件的名称，如 **computert.yaml** 以使用默认网络拓扑。

15. 使用其他环境文件将环境文件添加到堆栈中，并部署 **overcloud** ：

-

```
(undercloud)$ openstack overcloud deploy --templates \
 -r /home/stack/templates/roles_data_rt.yaml \
 -e /home/stack/templates/overcloud-baremetal-deployed.yaml
 -e /home/stack/templates/node-info.yaml \
 -e [your environment files] \
 -e /home/stack/templates/compute-real-time.yaml
```

12.2.2. 配置 OVS-DPDK 参数

1. 在 **parameter_defaults** 下，将隧道类型设置为 **vxlan**，网络类型设置为 **vxlan,vlan** ：

```
NeutronTunnelTypes: 'vxlan'
NeutronNetworkType: 'vxlan,vlan'
```

2. 在 **parameters_defaults** 下，设置网桥映射：

```
# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings:
 - dpdk-mgmt:br-link0
```

3. 在 **parameter_defaults** 下，为 **ComputeOvsDpdkSriov** 角色设置特定于角色的参数：

```
 ############################
 # OVS DPDK configuration #
 ############################
 ComputeOvsDpdkSriovParameters:
   KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt
intel_iommu=on isolcpus=2-19,22-39"
   TunedProfileName: "cpu-partitioning"
   IsolCpusList: "2-19,22-39"
   NovaComputeCpuDedicatedSet: ['4-19,24-39']
   NovaReservedHostMemory: 4096
   OvsDpdkSocketMemory: "3072,1024"
   OvsDpdkMemoryChannels: "4"
   OvsPmdCoreList: "2,22,3,23"
   NovaComputeCpuSharedSet: [0,20,1,21]
   NovaLibvirtRxQueueSize: 1024
   NovaLibvirtTxQueueSize: 1024
```

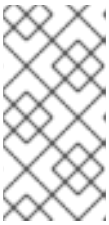
注意

要防止客户机创建过程中失败，在每个 **NUMA** 节点上至少分配一个带有同级线程的 **CPU**。在示例中，**OvsPmd CoreList** 参数的值表示来自 **NUMA 0** 的内核 **2** 和 **22**，以及 **NUMA 1** 中的内核 **3** 和 **23**。

注意

这些大页面由虚拟机使用，以及 OVS-DPDK 使用 OvsDpdkSocketMemory 参数，如此流程中所示。虚拟机可用的巨页数量是 引导参数 减去 OvsDpdkSocketMemory。

您还必须将 hw:mem_page_size=1GB 添加到与 DPDK 实例关联的类别。



注意

OvsDpdkMemoryChannels 是此流程所需的设置。对于 optimum 操作，请确保使用适当的参数和值部署 DPDK。

4. 为 SR-IOV 配置特定于角色的参数：

```
NovaPCIPassthrough:
  - vendor_id: "8086"
    product_id: "1528"
    address: "0000:06:00.0"
    trusted: "true"
    physical_network: "sriov-1"
  - vendor_id: "8086"
    product_id: "1528"
    address: "0000:06:00.1"
    trusted: "true"
    physical_network: "sriov-2"
```

12.3. 启动 RT-KVM 实例

执行以下步骤在实时启用的 Compute 节点上启动 RT-KVM 实例：

1. 在 overcloud 上创建 RT-KVM 类别：

```
$ openstack flavor create r1.small --id 99 --ram 4096 --disk 20 --vcpus 4
$ openstack flavor set --property hw:cpu_policy=dedicated 99
$ openstack flavor set --property hw:cpu_realtime=yes 99
$ openstack flavor set --property hw:mem_page_size=1GB 99
$ openstack flavor set --property hw:cpu_realtime_mask="^0-1" 99
$ openstack flavor set --property hw:cpu_emulator_threads=isolate 99
```

2. 启动 **RT-KVM** 实例：

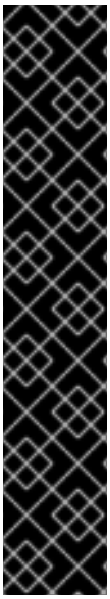
```
$ openstack server create  --image <rhel> --flavor r1.small --nic net-id=<dpdk-net> test-rt
```

3. 要验证实例是否使用分配的仿真程序线程，请运行以下命令：

```
$ virsh dumpxml <instance-id> | grep vcpu -A1
<vcpu placement='static'>4</vcpu>
<cputune>
  <vcpupin vcpu='0' cpuset='1'/>
  <vcpupin vcpu='1' cpuset='3'/>
  <vcpupin vcpu='2' cpuset='5'/>
  <vcpupin vcpu='3' cpuset='7'/>
  <emulatorpin cpuset='0-1'/>
  <vcpusched vcpus='2-3' scheduler='fifo'
  priority='1'/>
</cputune>
```

第 13 章 示例：使用 VXLAN 隧道配置 OVS-DPDK 和 SR-IOV

您可以使用 OVS-DPDK 和 SR-IOV 接口部署 Compute 节点。集群包括 ML2/OVS 和 VXLAN 隧道。



重要

在角色配置文件中，如 roles_data.yaml，在生成 overcloud 角色时注释掉或删除包含 OS::TripleO::Services::Tuned 的行。

```
ServicesDefault:
# - OS::TripleO::Services::Tuned
```

当您注释掉或移除了 OS::TripleO::Services::Tuned 时，您可以设置 TunedProfileName 参数以满足您的要求，如 "cpu-partitioning "。如果您没有注释掉或删除 OS::TripleO::Services::Tuned 行和重新部署，则 TunedProfileName 参数会获取默认值 "throughput-performance"，而不是您设置的任何其他值。

13.1. 配置角色数据

Red Hat OpenStack Platform 在 roles_data.yaml 文件中提供了一组默认角色。您可以创建自己的 roles_data.yaml 文件来支持您需要的角色。

在本例中，创建 ComputeOvsDpdkSriov 角色。

其他资源

- 在自定义 *Red Hat OpenStack Platform 部署时创建新角色*
- roles-data.yaml

13.2. 配置 OVS-DPDK 参数

1. 在 parameter_defaults 下，将隧道类型设置为 vxlan，网络类型设置为 vxlan,vlan ：

```
NeutronTunnelTypes: 'vxlan'
NeutronNetworkType: 'vxlan,vlan'
```

2. 在 parameters_defaults 下，设置网桥映射：

```
# The OVS logical->physical bridge mappings to use.
NeutronBridgeMappings:
 - dpdk-mgmt:br-link0
```

3. 在 parameter_defaults 下，为 ComputeOvsDpdkSriov 角色设置特定于角色的参数：

```
##########################
# OVS DPDK configuration #
##########################
ComputeOvsDpdkSriovParameters:
  KernelArgs: "default_hugepagesz=1GB hugepagesz=1G hugepages=32 iommu=pt
intel_iommu=on isolcpus=2-19,22-39"
  TunedProfileName: "cpu-partitioning"
  IsolCpusList: "2-19,22-39"
  NovaComputeCpuDedicatedSet: ['4-19,24-39']
  NovaReservedHostMemory: 4096
  OvsDpdkSocketMemory: "3072,1024"
  OvsDpdkMemoryChannels: "4"
  OvsPmdCoreList: "2,22,3,23"
  NovaComputeCpuSharedSet: [0,20,1,21]
  NovaLibvirtRxQueueSize: 1024
  NovaLibvirtTxQueueSize: 1024
```

注意

要防止客户机创建过程中失败，在每个 NUMA 节点上至少分配一个带有同级线程的 CPU。在示例中，OvsPmd CoreList 参数的值表示来自 NUMA 0 的内核 2 和 22，以及 NUMA 1 中的内核 3 和 23。

注意

这些大页面由虚拟机使用，以及 OVS-DPDK 使用 OvsDpdkSocketMemory 参数，如此流程中所示。虚拟机可用的巨页数量是 引导参数 减去 OvsDpdkSocketMemory。

您还必须将 hw:mem_page_size=1GB 添加到与 DPDK 实例关联的类别。

注意

OvsDpdkMemoryChannels 是此流程所需的设置。对于 optimum 操作，请确保使用适当的参数和值部署 DPDK。

4. **为 SR-IOV 配置特定于角色的参数：**

```
NovaPCIPassthrough:
  - vendor_id: "8086"
    product_id: "1528"
    address: "0000:06:00.0"
    trusted: "true"
    physical_network: "sriov-1"
  - vendor_id: "8086"
    product_id: "1528"
    address: "0000:06:00.1"
    trusted: "true"
    physical_network: "sriov-2"
```

13.3. 配置控制器节点

1. **为隔离的网络创建 control-plane Linux 绑定。**

```
- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  use_dhcp: false
  dns_servers:
    get_param: DnsServers
  members:
  - type: interface
    name: nic2
    primary: true
```

2. **将 VLAN 分配给此 Linux 绑定。**

```
- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet

- type: vlan
  vlan_id:
```

```
    get_param: StorageNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
      get_param: StorageIpSubnet

- type: vlan
  vlan_id:
    get_param: StorageMgmtNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
      get_param: StorageMgmtIpSubnet

- type: vlan
  vlan_id:
    get_param: ExternalNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
      get_param: ExternalIpSubnet
  routes:
  - default: true
    next_hop:
      get_param: ExternalInterfaceDefaultRoute
```

3. 创建 OVS 网桥，以访问 neutron-dhcp-agent 和 neutron-metadata-agent 服务。

```
- type: ovs_bridge
  name: br-link0
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic3
    mtu: 9000
  - type: vlan
    vlan_id:
      get_param: TenantNetworkVlanID
    mtu: 9000
    addresses:
    - ip_netmask:
        get_param: TenantIpSubnet
```

13.4. 为 DPDK 和 SR-IOV 配置 COMPUTE 节点

从默认的 compute.yaml 文件创建 computeovsdpdksriov.yaml 文件，并进行以下更改：

1. 为隔离的网络创建 control-plane Linux 绑定。

```
- type: linux_bond
  name: bond_api
  bonding_options: "mode=active-backup"
  use_dhcp: false
  dns_servers:
    get_param: DnsServers
  members:
  - type: interface
    name: nic3
    primary: true
  - type: interface
    name: nic4
```

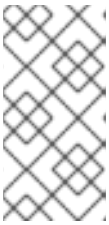
2. 将 VLAN 分配给此 Linux 绑定。

```
- type: vlan
  vlan_id:
    get_param: InternalApiNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet

- type: vlan
  vlan_id:
    get_param: StorageNetworkVlanID
  device: bond_api
  addresses:
  - ip_netmask:
      get_param: StorageIpSubnet
```

3. 使用 DPDK 端口设置网桥，以链接到控制器。

```
- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  ovs_extra:
    - str_replace:
        template: set port br-link0 tag=_VLAN_TAG_
        params:
          _VLAN_TAG_:
            get_param: TenantNetworkVlanID
  addresses:
    - ip_netmask:
        get_param: TenantIpSubnet
  members:
    - type: ovs_dpdk_bond
      name: dpdkbond0
      mtu: 9000
      rx_queue: 2
      members:
```

```
- type: ovs_dpdk_port
  name: dpdk0
  members:
    - type: interface
      name: nic7
- type: ovs_dpdk_port
  name: dpdk1
  members:
    - type: interface
      name: nic8
```



注意

要包含多个 DPDK 设备，请对要添加的每个 DPDK 设备重复 类型 代码部分。

注意

在使用 OVS-DPDK 时，同一 Compute 节点上的所有网桥都必须是 ovs_user_bridge 类型。Red Hat OpenStack Platform 不支持同一节点上的 ovs_bridge 和 ovs_user_bridge。

13.5. 部署 OVERCLOUD

- 运行 overcloud_deploy.sh 脚本：

第 14 章 升级带有 NFV 的 RED HAT OPENSTACK 平台

有关升级配置了 OVS-DPDK 的 Red Hat OpenStack Platform (RHOSP)的更多信息，请参阅 *Framework 中的 准备网络功能虚拟化(NFV) 以升级(16.2 到 17.1)* 指南。

第 15 章 DPDK SR-IOV YAML 和 JINJA2 文件示例

本节提供了示例 yaml 文件，作为在同一计算节点上添加单一根 I/O 虚拟化(SR-IOV)和数据平面开发套件(DPDK)接口的参考。

注意

这些模板来自完全配置的环境，包含与 NFV 无关的参数，它们可能不适用于您的部署。有关组件支持级别的列表，请参阅红帽知识库解决方案 支持 Graduation。

15.1. ROLES_DATA.YAML

- 运行 openstack overcloud roles generate 命令，以生成 roles_data.yaml 文件。

  根据您要在您的环境中部署的角色，在命令中包括角色名称，如 Controller、ComputeSriov、ComputeOvsDpdkRT、ComputeOvsDpdkSriov 或其他角色。

  示例

  例如，要生成包含 Controller 和 ComputeHCIOvsDpdkSriov 角色的 roles_data.yaml 文件，请运行以下命令：

```
$ openstack overcloud roles generate -o roles_data.yaml \
Controller ComputeHCIOvsDpdkSriov
```

```
###############################################################################
####
# File generated by TripleO
###############################################################################
####
###############################################################################
####
# Role: Controller                                                            #
###############################################################################
####
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  tags:
    - primary
    - controller
  networks:
    External:
```

```
    subnet: external_subnet
  InternalApi:
    subnet: internal_api_subnet
  Storage:
    subnet: storage_subnet
  StorageMgmt:
    subnet: storage_mgmt_subnet
  Tenant:
    subnet: tenant_subnet
# For systems with both IPv4 and IPv6, you may specify a gateway network for
# each, such as ['ControlPlane', 'External']
default_route_networks: ['External']
HostnameFormatDefault: '%stackname%-controller-%index%'
# Deprecated & backward-compatible values (FIXME: Make parameters consistent)
# Set uses_deprecated_params to True if any deprecated params are used.
uses_deprecated_params: True
deprecated_param_extraconfig: 'controllerExtraConfig'
deprecated_param_flavor: 'OvercloudControlFlavor'
deprecated_param_image: 'controllerImage'
deprecated_nic_config_name: 'controller.yaml'
update_serial: 1
ServicesDefault:
  - OS::TripleO::Services::Aide
  - OS::TripleO::Services::AodhApi
  - OS::TripleO::Services::AodhEvaluator
  - OS::TripleO::Services::AodhListener
  - OS::TripleO::Services::AodhNotifier
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::BarbicanApi
  - OS::TripleO::Services::BarbicanBackendSimpleCrypto
  - OS::TripleO::Services::BarbicanBackendDogtag
  - OS::TripleO::Services::BarbicanBackendKmip
  - OS::TripleO::Services::BarbicanBackendPkcs11Crypto
  - OS::TripleO::Services::BootParams
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CeilometerAgentCentral
  - OS::TripleO::Services::CeilometerAgentNotification
  - OS::TripleO::Services::CephExternal
  - OS::TripleO::Services::CephGrafana
  - OS::TripleO::Services::CephMds
  - OS::TripleO::Services::CephMgr
  - OS::TripleO::Services::CephMon
  - OS::TripleO::Services::CephRbdMirror
  - OS::TripleO::Services::CephRgw
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::CinderApi
  - OS::TripleO::Services::CinderBackendDellPs
  - OS::TripleO::Services::CinderBackendDellSc
  - OS::TripleO::Services::CinderBackendDellEMCPowermax
  - OS::TripleO::Services::CinderBackendDellEMCPowerStore
  - OS::TripleO::Services::CinderBackendDellEMCSc
  - OS::TripleO::Services::CinderBackendDellEMCUnity
  - OS::TripleO::Services::CinderBackendDellEMCVMAXISCSI
  - OS::TripleO::Services::CinderBackendDellEMCVNX
  - OS::TripleO::Services::CinderBackendDellEMCVxFlexOS
  - OS::TripleO::Services::CinderBackendDellEMCXtremio
```

```
    - OS::TripleO::Services::CinderBackendDellEMCXTREMIOISCSI
    - OS::TripleO::Services::CinderBackendNetApp
    - OS::TripleO::Services::CinderBackendPure
    - OS::TripleO::Services::CinderBackendScaleIO
    - OS::TripleO::Services::CinderBackendVRTSHyperScale
    - OS::TripleO::Services::CinderBackendNVMeOF
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderHPELeftHandISCSI
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Clustercheck
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::ContainerImagePrepare
    - OS::TripleO::Services::DesignateApi
    - OS::TripleO::Services::DesignateCentral
    - OS::TripleO::Services::DesignateProducer
    - OS::TripleO::Services::DesignateWorker
    - OS::TripleO::Services::DesignateMDNS
    - OS::TripleO::Services::DesignateSink
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::Ec2Api
    - OS::TripleO::Services::Etcd
    - OS::TripleO::Services::ExternalSwiftProxy
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GnocchiApi
    - OS::TripleO::Services::GnocchiMetricd
    - OS::TripleO::Services::GnocchiStatsd
    - OS::TripleO::Services::HAproxy
    - OS::TripleO::Services::HeatApi
    - OS::TripleO::Services::HeatApiCloudwatch
    - OS::TripleO::Services::HeatApiCfn
    - OS::TripleO::Services::HeatEngine
    - OS::TripleO::Services::Horizon
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::IronicApi
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::IronicInspector
    - OS::TripleO::Services::IronicPxe
    - OS::TripleO::Services::IronicNeutronAgent
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Keepalived
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::ManilaApi
    - OS::TripleO::Services::ManilaBackendCephFs
    - OS::TripleO::Services::ManilaBackendIsilon
    - OS::TripleO::Services::ManilaBackendNetapp
    - OS::TripleO::Services::ManilaBackendUnity
    - OS::TripleO::Services::ManilaBackendVNX
    - OS::TripleO::Services::ManilaBackendVMAX
    - OS::TripleO::Services::ManilaScheduler
    - OS::TripleO::Services::ManilaShare
    - OS::TripleO::Services::Memcached
    - OS::TripleO::Services::MetricsQdr
```

```
    - OS::TripleO::Services::MistralApi
    - OS::TripleO::Services::MistralEngine
    - OS::TripleO::Services::MistralExecutor
    - OS::TripleO::Services::MistralEventEngine
    - OS::TripleO::Services::Multipathd
    - OS::TripleO::Services::MySQL
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::NeutronApi
    - OS::TripleO::Services::NeutronBgpVpnApi
    - OS::TripleO::Services::NeutronSfcApi
    - OS::TripleO::Services::NeutronCorePlugin
    - OS::TripleO::Services::NeutronDhcpAgent
    - OS::TripleO::Services::NeutronL2gwAgent
    - OS::TripleO::Services::NeutronL2gwApi
    - OS::TripleO::Services::NeutronL3Agent
    - OS::TripleO::Services::NeutronLinuxbridgeAgent
    - OS::TripleO::Services::NeutronMetadataAgent
    - OS::TripleO::Services::NeutronML2FujitsuCfab
    - OS::TripleO::Services::NeutronML2FujitsuFossw
    - OS::TripleO::Services::NeutronOvsAgent
    - OS::TripleO::Services::NeutronVppAgent
    - OS::TripleO::Services::NeutronAgentsIBConfig
    - OS::TripleO::Services::NovaApi
    - OS::TripleO::Services::NovaConductor
    - OS::TripleO::Services::NovaIronic
    - OS::TripleO::Services::NovaMetadata
    - OS::TripleO::Services::NovaScheduler
    - OS::TripleO::Services::NovaVncProxy
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::OctaviaApi
    - OS::TripleO::Services::OctaviaDeploymentConfig
    - OS::TripleO::Services::OctaviaHealthManager
    - OS::TripleO::Services::OctaviaHousekeeping
    - OS::TripleO::Services::OctaviaWorker
    - OS::TripleO::Services::OpenStackClients
    - OS::TripleO::Services::OVNDBs
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::Pacemaker
    - OS::TripleO::Services::PankoApi
    - OS::TripleO::Services::PlacementApi
    - OS::TripleO::Services::OsloMessagingRpc
    - OS::TripleO::Services::OsloMessagingNotify
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rear
    - OS::TripleO::Services::Redis
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::Rsyslog
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::SaharaApi
    - OS::TripleO::Services::SaharaEngine
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::SwiftProxy
    - OS::TripleO::Services::SwiftDispersion
    - OS::TripleO::Services::SwiftRingBuilder
```

```
    - OS::TripleO::Services::SwiftStorage
    - OS::TripleO::Services::Timesync
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::Tuned
    - OS::TripleO::Services::Vpp
    - OS::TripleO::Services::Zaqar
##############################################################################
####
# Role: ComputeHCIOvsDpdkSriov                                  #
##############################################################################
####
- name: ComputeHCIOvsDpdkSriov
  description: |
    ComputeOvsDpdkSriov Node role hosting Ceph OSD too
  networks:
    InternalApi:
      subnet: internal_api_subnet
    Tenant:
      subnet: tenant_subnet
    Storage:
      subnet: storage_subnet
    StorageMgmt:
      subnet: storage_mgmt_subnet
  # CephOSD present so serial has to be 1
  update_serial: 1
  RoleParametersDefault:
    TunedProfileName: "cpu-partitioning"
    VhostuserSocketGroup: "hugetlbfs"
    NovaLibvirtRxQueueSize: 1024
    NovaLibvirtTxQueueSize: 1024
  ServicesDefault:
    - OS::TripleO::Services::Aide
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::BootParams
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephOSD
    - OS::TripleO::Services::CertmongerUser
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::ComputeCeilometerAgent
    - OS::TripleO::Services::ComputeNeutronCorePlugin
    - OS::TripleO::Services::ComputeNeutronL3Agent
    - OS::TripleO::Services::ComputeNeutronMetadataAgent
    - OS::TripleO::Services::ComputeNeutronOvsDpdk
    - OS::TripleO::Services::Docker
    - OS::TripleO::Services::IpaClient
    - OS::TripleO::Services::Ipsec
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::LoginDefs
    - OS::TripleO::Services::MetricsQdr
    - OS::TripleO::Services::Multipathd
    - OS::TripleO::Services::MySQLClient
```

```
    - OS::TripleO::Services::NeutronBgpVpnBagpipe
    - OS::TripleO::Services::NeutronSriovAgent
    - OS::TripleO::Services::NeutronSriovHostConfig
    - OS::TripleO::Services::NovaAZConfig
    - OS::TripleO::Services::NovaCompute
    - OS::TripleO::Services::NovaLibvirt
    - OS::TripleO::Services::NovaLibvirtGuests
    - OS::TripleO::Services::NovaMigrationTarget
    - OS::TripleO::Services::OvsDpdkNetcontrold
    - OS::TripleO::Services::ContainersLogrotateCrond
    - OS::TripleO::Services::Podman
    - OS::TripleO::Services::Rear
    - OS::TripleO::Services::Rhsm
    - OS::TripleO::Services::Rsyslog
    - OS::TripleO::Services::RsyslogSidecar
    - OS::TripleO::Services::Securetty
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timesync
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::OVNController
    - OS::TripleO::Services::OVNMetadataAgent
    - OS::TripleO::Services::Ptp
```

15.2. NETWORK-ENVIRONMENT-OVERRIDES.YAML

```
---
parameter_defaults:
  # The tunnel type for the tenant network (geneve or vlan). Set to '' to disable tunneling.
  NeutronTunnelTypes: "geneve"
  # The tenant network type for Neutron (vlan or geneve).
  NeutronNetworkType: ["geneve", "vlan"]
  NeutronExternalNetworkBridge: "'br-access'"
  # NTP server configuration.
  # NtpServer: ["clock.redhat.com"]
  # MTU global configuration
  NeutronGlobalPhysnetMtu: 9000
  # Configure the classname of the firewall driver to use for implementing security groups.
  NeutronOVSFirewallDriver: openvswitch
  SshServerOptionsOverrides:
    UseDns: "no"
  # Enable log level DEBUG for supported components
  Debug: true

  # From Rocky live migration with NumaTopologyFilter disabled by default
  # https://bugs.launchpad.net/nova/+bug/1289064
  NovaEnableNUMALiveMigration: true
  NeutronPluginExtensions: "port_security,qos,segments,trunk,placement"
  # RFE https://bugzilla.redhat.com/show_bug.cgi?id=1669584
  NeutronServicePlugins: "ovn-router,trunk,qos,placement"
  NeutronSriovAgentExtensions: "qos"

  ##############################
```

```
  #  Scheduler configuration #
  ##############################
  NovaSchedulerEnabledFilters:
    - AvailabilityZoneFilter
    - ComputeFilter
    - ComputeCapabilitiesFilter
    - ImagePropertiesFilter
    - ServerGroupAntiAffinityFilter
    - ServerGroupAffinityFilter
    - PciPassthroughFilter
    - NUMATopologyFilter
    - AggregateInstanceExtraSpecsFilter
  ComputeOvsDpdkSriovNetworkConfigTemplate: "/home/stack/ospd-17.0-geneve-ovn-dpdk-
sriov-ctlplane-dataplane-bonding-hybrid/nic-configs/computeovsdpdksriov.yaml"
  ControllerSriovNetworkConfigTemplate: "/home/stack/ospd-17.0-geneve-ovn-dpdk-sriov-
ctlplane-dataplane-bonding-hybrid/nic-configs/controller.yaml"
```

15.3. CONTROLLER.J2

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks if network not in 'Tenant,External' %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: interface
  name: nic1
  use_dhcp: false
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes:
  - ip_netmask: 169.254.169.254/32
    next_hop: {{ ctlplane_ip }}

- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu }}
  bonding_options: mode=active-backup
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  members:
  - type: interface
    name: nic2
    primary: true

{% for network in role_networks if network not in 'Tenant,External' %}
- type: vlan
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  device: bond_api
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
{% endfor %}
```

```
- type: ovs_bridge
  name: br-link0
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic3
    mtu: 9000
  - type: vlan
    vlan_id: {{ lookup('vars', networks_lower['Tenant'] ~ '_vlan_id') }}
    mtu: 9000
    addresses:
    - ip_netmask: {{ lookup('vars', networks_lower['Tenant'] ~ '_ip') }}/{{ lookup('vars',
networks_lower['Tenant'] ~ '_cidr') }}

- type: ovs_bridge
  name: br-dpdk0
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic4
    mtu: 9000

- type: ovs_bridge
  name: br-dpdk1
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic5
    mtu: 9000

- type: ovs_bridge
  name: br-sriov1
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic6
    mtu: 9000

- type: ovs_bridge
  name: br-sriov2
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic7
    mtu: 9000

- type: interface
  name: nic8
  use_dhcp: false
  defroute: false
```

```
- type: interface
  name: nic9
  use_dhcp: false
  defroute: false

- type: ovs_bridge
  name: br-access
  use_dhcp: false
  mtu: 9000
  members:
  - type: interface
    name: nic10
    mtu: 9000
  - type: vlan
    vlan_id: {{ lookup('vars', networks_lower['External'] ~ '_vlan_id') }}
    mtu: 9000
    addresses:
    - ip_netmask: {{ lookup('vars', networks_lower['External'] ~ '_ip') }}/{{ lookup('vars',
networks_lower['External'] ~ '_cidr') }}
    routes:
    - default: true
      next_hop: {{ lookup('vars', networks_lower['External'] ~ '_gateway_ip') }}
```

15.4. COMPUTE-OVS-DPDK.J2

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks if network not in 'Tenant,External' %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: interface
  name: nic1
  use_dhcp: false
  default: no

- type: interface
  name: nic2
  use_dhcp: false
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes:
  - ip_netmask: 169.254.169.254/32
    next_hop: {{ ctlplane_ip }}
  - default: true
    next_hop: {{ ctlplane_gateway_ip }}

- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu }}
  bonding_options: mode=active-backup
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
```

```
  members:
  - type: interface
    name: nic2
    primary: true

{% for network in role_networks if network not in 'Tenant,External' %}
- type: vlan
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  device: bond_api
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
{% endfor %}

- type: ovs_user_bridge
  name: br-link0
  use_dhcp: false
  ovs_extra: "set port br-link0 tag={{ lookup('vars', networks_lower['Tenant'] ~ '_vlan_id') }}"
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower['Tenant'] ~ '_ip') }}/{{ lookup('vars',
networks_lower['Tenant'] ~ '_cidr')}}
  members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    rx_queue: 1
    ovs_extra: "set port dpdkbond0 bond_mode=balance-slb"
    members:
      - type: ovs_dpdk_port
        name: dpdk0
        members:
          - type: interface
            name: nic7
      - type: ovs_dpdk_port
        name: dpdk1
        members:
          - type: interface
            name: nic8

- type: ovs_user_bridge
  name: br-dpdk0
  use_dhcp: false
  mtu: 9000
  rx_queue: 1
  members:
    - type: ovs_dpdk_port
      name: dpdk2
      members:
        - type: interface
          name: nic5

- type: ovs_user_bridge
  name: br-dpdk1
  use_dhcp: false
  mtu: 9000
  rx_queue: 1
```

```
  members:
    - type: ovs_dpdk_port
      name: dpdk3
      members:
        - type: interface
          name: nic6

- type: sriov_pf
  name: nic9
  mtu: 9000
  numvfs: 10
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true
  promisc: false

- type: sriov_pf
  name: nic10
  mtu: 9000
  numvfs: 10
  use_dhcp: false
  defroute: false
  nm_controlled: true
  hotplug: true
  promisc: false
```

15.5. OVERCLOUD_DEPLOY.SH

```
#!/bin/bash

tht_path='/home/stack/ospd-17.0-geneve-ovn-dpdk-sriov-ctlplane-dataplane-bonding-hybrid'
[[ ! -d "$tht_path/roles" ]] && mkdir $tht_path/roles
openstack overcloud roles generate -o $tht_path/roles/roles_data.yaml ControllerSriov
ComputeOvsDpdkSriov

openstack overcloud deploy \
  --templates /usr/share/openstack-tripleo-heat-templates \
  --ntp-server
clock.redhat.com,time1.google.com,time2.google.com,time3.google.com,time4.google.com \
  --stack overcloud \
  --roles-file $tht_path/roles/roles_data.yaml \
  -n $tht_path/network/network_data_v2.yaml \
  --deployed-server \
  -e /home/stack/templates/overcloud-baremetal-deployed.yaml \
  -e /home/stack/templates/overcloud-networks-deployed.yaml \
  -e /home/stack/templates/overcloud-vip-deployed.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-ha.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-
dpdk.yaml \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/neutron-ovn-
sriov.yaml \
  -e /home/stack/containers-prepare-parameter.yaml \
  -e $tht_path/network-environment-overrides.yaml \
  -e $tht_path/api-policies.yaml \
```

```
-e $tht_path/bridge-mappings.yaml \
-e $tht_path/neutron-vlan-ranges.yaml \
-e $tht_path/dpdk-config.yaml \
-e $tht_path/sriov-config.yaml \
--log-file overcloud_deployment.log
```